Instrumentation and Coverage Analysis of Cyber Physical System Models

by

Rahul Thekkalore Srinivasa

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2016 by the
Graduate Supervisory Committee:

Georgios Fainekos, Chair
Abdel Ra'ouf Mayyas
Hessam Sarjoughian

ARIZONA STATE UNIVERSITY

August 2016

ABSTRACT

A Cyber Physical System consists of a computer monitoring and controlling physical processes usually in a feedback loop. These systems are increasingly becoming part of our daily life ranging from smart buildings to medical devices to automobiles. The controller comprises discrete software which may be operating in one of the many possible operating modes and interacting with a changing physical environment in a feedback loop. The systems with such a mix of discrete and continuous dynamics are usually termed as hybrid systems. In general, these systems are safety critical, hence their correct operation must be verified. Model Based Design (MBD) languages like Simulink are being used extensively for the design and analysis of hybrid systems due to the ease in system design and automatic code generation. It also allows testing and verification of these systems before deployment. One of the main challenges in the verification of these systems is to test all the operating modes of the control software and reduce the amount of user intervention.

This research aims to provide an automated framework for the structural analysis and instrumentation of hybrid system models developed in Simulink. The behavior of the components introducing discontinuities in the model are automatically extracted in the form of state transition graphs. The framework is integrated in the S-TaLiRo toolbox to demonstrate the improvement in mode coverage.

ACKNOWLEDGMENTS

I would first like to express my sincere gratitude to my advisor and committee chair, Dr. Georgios Fainekos, for giving me an opportunity to work on this exciting topic and for providing his valuable ideas, support and patience in overseeing my research. I would like to thank my committee members, Dr. Hessam Sarjoughian and Dr. Abdel Mayyas for their insightful comments and feedbacks. I thank my fellow labmates Adel Dokhanchi, Erkan Tuncali, Bardh Hoxa, Kanjin Kim and Wei Wei for the stimulating discussions. Finally, I would like to express my profound gratitude to my parents for providing me with unfailing support and continuous encouragement through the process of researching and writing this thesis.

This thesis has been partially supported by NSF IIP-1361926 and NSF CNS-1319560, and Bosch and Toyota through the NSF I/UCRC Center for Embedded Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, Bosch, Toyota or the Center for Embedded Systems.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Motivation and Challenges

A Cyber Physical System (CPS) consists of a computer monitoring and controlling physical processes usually in a feedback loop. These systems are increasingly becoming part of our daily life ranging from smart buildings to medical devices to automobiles. In general, these systems have hard real time constraints and are safety critical. It is essential that these systems work correctly as failure can lead to catastrophic events and even loss of human life. Imagine a malfunction of a cruise control in an automobile or a failure in a flight control system which can have devastating effects. Hence it is of paramount importance to ensure their safe operation. Sometimes, specific system behavior is imposed by mandatory government regulations.

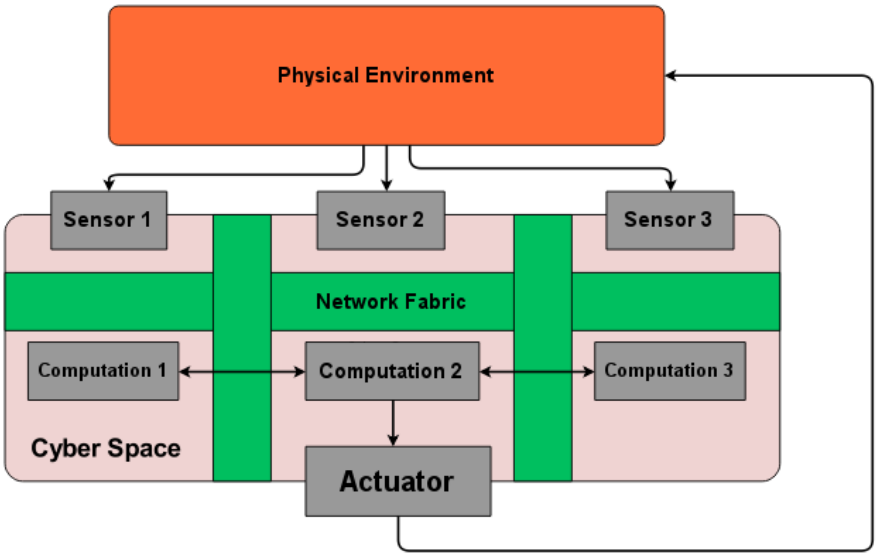

Figure 1 Architecture of a Cyber Physical System

The general architecture of a CPS is shown in Fig. 1 which is based on the figure

presented in [1]. The system consists of the following three main parts:

- Physical plant: it constitutes the physical environment part of the system and may include mechanical parts or human operators.
- Computation Platform: they are comprised of computers, sensors and actuators. They monitor the physical environment through sensors and based on the data may implement a control law that determines actions issued to the actuator.
- Network fabric: they enable communication between the computers in the system.

A CPS requires the combined understanding of the cyber and physical space interaction. The controller comprises discrete software which may be operating in one of the many possible operating modes and interacting with a changing physical environment in a feedback loop. The systems with such a mix of discrete and continuous dynamics are termed as hybrid systems. [1]

Although CPS have been in use for long, the recent advances in computing power, wireless communication and sensor technologies have enabled their deployment at low costs. The design, verification and validation of these has attracted immense research interests and has led to the formation of a separate academic discipline.

There has been progress in the control synthesis of hybrid systems [2] but in practice the problem still remains challenging. These systems are characterized by a large number of continuous state variables and interconnected components which may not be available for symbolic analysis. Currently, these methods are used for certain operating modes of the system. Rule tables and engineering experience is used to combine the

different operating modes. This specific combination approach is error prone and has resulted in recalls of many safety critical systems [3].

Hybrid system verification is seen as a potential alternative to the problem. The aim is to prove that no bad behaviors are possible to occur defined by the user. Some of the methods like reachability analysis [4, 5] and theorem provers [6] have been developed to address the problem. These methods can only be applied to large linear hybrid systems or in specific applications [7] and they cannot be extended in an automated manner to industrial size problems.

Alternatively, simulation guided methodologies [8] have been promising in analyzing and detecting errors in industrial size problems and complexity [9,10,11]. Specification robustness guided falsification is one specific class of such methods [12] [13] where carefully chosen tests are executed to violate a formal specification. The robustness semantics evaluate to positive values if the trajectory satisfies the specification and negative if it violates the specification. In other words, robustness is a measure of how well the behavior satisfies or violates the specification. A falsification algorithm essentially tries to generate tests that result in negative robustness values through the application of stochastic or deterministic optimization algorithms [11, 12].

In theory, the above technique is guaranteed to detect the system error but the number of tests required depends on the robustness landscape over the search space of the hybrid system. The discontinuities induced by the hybrid system make it hard to locate the regions of interest. However, the information about potential discontinuities in the hybrid system can be extracted and use it to narrow the search to such regions.

1.2 Contributions

This research aims to analyze and extract the discontinuities induced in hybrid systems models in the form of nonlinear blocks. The central focus of this work is to develop an automated framework to identify the nonlinear components and extract their behavior in the form of state transition graphs. Currently, the tool supports switch and saturation blocks but the modular workflow can be easily used to extend support for other blocks. Additionally, the framework is integrated with S-TaLiRo [14] for performing mode coverage and provide a platform for coverage based falsification. We experimentally demonstrate that coverage based testing can be achieved for closed loop systems.

Preliminary results of this thesis were published in [15]:

- A. Dokhanchi, A. Zutshi, R. T. Sriniva, S. Sankaranarayanan and G. Fainekos, "Requirements Driven falsification with coverage metrics," in *Embedded Software (EMSOFT)*, 2015.

CHAPTER 2

BACKGROUND

This chapter reviews some of the background concepts, definitions and notations based on which the thesis is built. The notations and terminologies will remain constant throughout the thesis. The thesis is built to be interfaced in a specification robustness guided testing framework. Hence, some concepts pertaining to convex optimization theory and requirement specification will be discussed to form a complete picture of the work.

2.1 Convex Set Theory

A set C is convex if the line segment between any two points in C lies in C, i.e., if for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have

$$\theta x_1 + (1- \theta x_2) \in C$$

The convex hull of a set C, denoted by conv C, is the set of all convex combinations of the points in C:

$$\text{conv } C = \{ \theta_1 x_1 + \ldots + \theta_k x_k \mid x_i \in C, \theta_i \geq 0, i = 1,\ldots,k, \theta_1 + \ldots + \theta_k = 1\}$$

The convex hull conv C is the smallest convex set that contains C.

A hyperplane is a set of the form

$$\{x \mid a^T x = b\},$$

where $a \in R^n$, $a \neq 0$, and $b \in R$. Here, R is the set of real numbers. Geometrically a hyperplane can be interpreted as the set of points with a constant inner product to a given vector a.

A hyperplane divides R^n into two halfspaces. A halfspace is a set of the form

$$\{x \mid a^T x \leq b\},$$

where $a \neq 0$, i.e., the solution set of one linear inequality. The halfspaces are convex but not affine. Figure 2 is presented to illustrate further based on the figure provided in [16]. The halfspace determined by $a^T x \geq b$ is the halfspace extending in the direction of the vector a. The shaded halfspace determined by $a^T x \leq b$ in the direction – a.

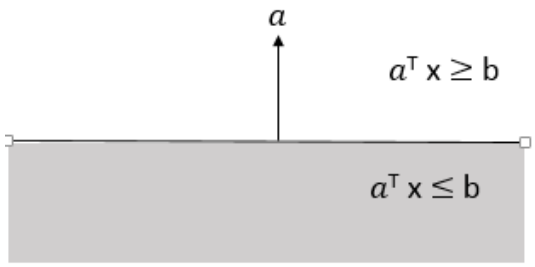

Figure 2 A Hyperplane Defined by $\boldsymbol{a^T x = b}$

A polyhedron is defined as the solution of a finite number of linear equalities and inequalities:

$$\mathcal{P} = \{ x \mid a_j^T x \leq b_j, j= 1, \ldots, m, c_j^T x = d_j, , j= 1, \ldots, p\}$$

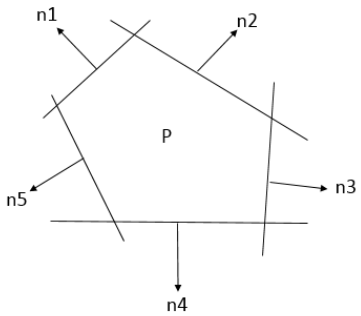

Figure 3 Polyhedron P is the Intersection of Five Halfspaces

In other words, polyhedron is the intersection of finite number of halfspaces and hyper planes. It can be seen that polyhedral sets are convex sets. A bounded polyhedron is termed as a polytope. A polyhedron formed by the intersection of five halfspaces is shown in Fig. 3 which is presented based on the figure provided in [16].

Alternatively, the polyhedron can be conveniently represented using the notation

$$\mathrm{P} = \{ \mathrm{x} \mid A\,\mathrm{x} \preccurlyeq b, C\,\mathrm{x} = d \}$$

where $A = [a_1{}^{\mathrm{T}} \ldots . a_{\mathrm{m}}{}^{\mathrm{T}}]^{\mathrm{T}}$, $C = [c_1{}^{\mathrm{T}} \ldots . c_{\mathrm{p}}{}^{\mathrm{T}}]^{\mathrm{T}}$ and the symbol $\preccurlyeq$ denotes component wise inequality in R^{m}

2.2 Model Based Design (MBD) using Simulink

MATLAB Simulink is a popularly used MBD tool in industry for the modelling and simulation of industrial control systems like automotive systems. The Simulink IDE make use of blocks to model the discrete and continuous parts of the hybrid models. The discrete components can be modelled using the Simulink Stateflow. The continuous part of the system can be discretized using sampling intervals. When the expected results are met by the simulations, a Real time workshop code generator can be used to obtain the implementation with certified code.

For the purpose of this thesis, the details of the switch and saturation blocks are highlighted in Fig. 4 and Fig. 5. These two blocks model if – else branching structure in the model. The terminologies corresponding to these blocks are introduced formally and will remain consistent throughout the thesis.

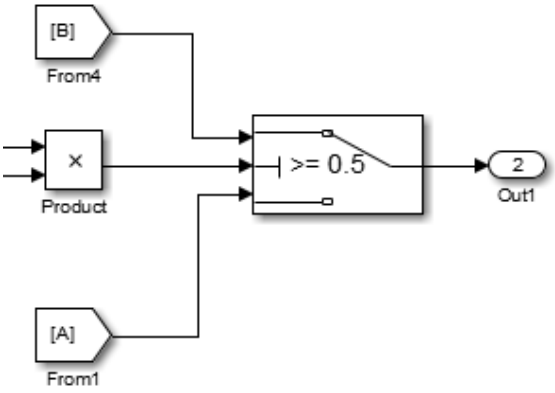

Figure 4 Simulink Switch Block

In Fig. 4 a switch block used in Simulink model is shown. The switch block has three inputs. The second input to switch decides which of the input is passed to the output of the switch. In this instance, if the output of the product block is greater than or equal to 0.5 the signal B is passed and for other case the signal A is passed. These two possible signals are termed as the operating mode or location of the switch. When the signal A is passed, the switch is said to be in mode (location) one and signal B corresponds to the mode (location) two.

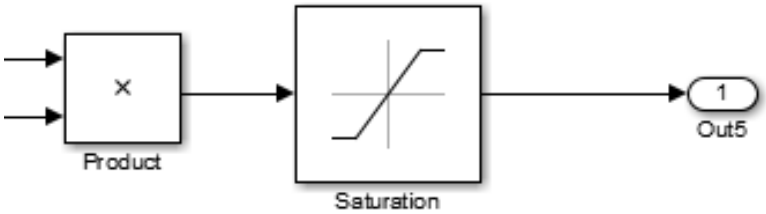

Figure 5 Simulink Saturation Block

In Fig. 5 the saturation block used in Simulink is illustrated. The block limits the input signal between a low threshold value and a high threshold value. Thus the

saturation block has three operating modes. The mode one corresponds to the low threshold value, mode 2 corresponds to the input signal and mode three corresponds to the high threshold value. At any given point the saturation block is operating in one these three modes.

2.2.1 Model Flattening

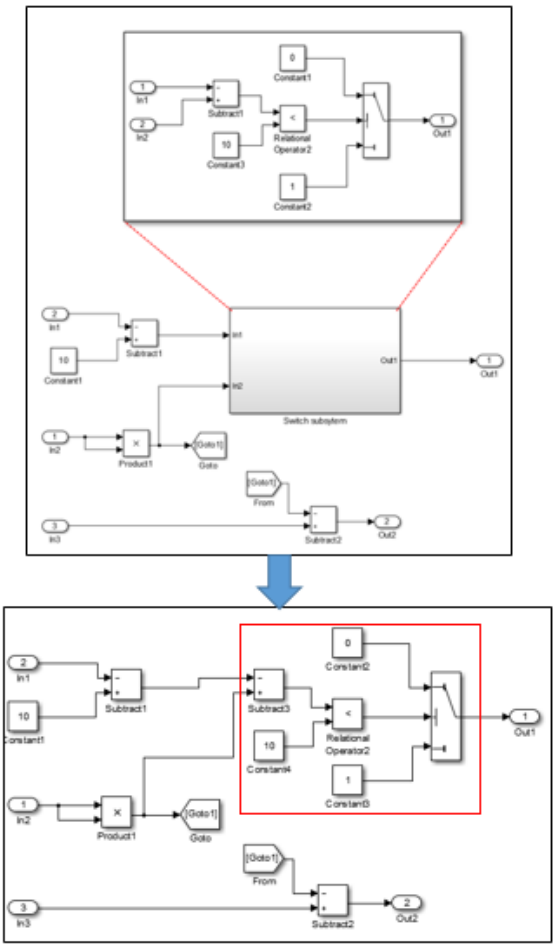

Figure 6 Model Flattening

The instrumentation framework developed in this work utilizes model flattening implemented using the tools in [22]. In their tool, the Simulink model to be flattened is loaded and all the block specific information like block type, parents and connectivity information is read. This information is used to identify blocks in sub systems and bring them out of their super blocks. The 'Goto'- 'From' Tags and the redundant input and output ports of the subsystems are discarded from the list of blocks. A new list consisting of only main blocks is generated as shown in Fig. 6 created based on their work.

The flattened version of the model is captured in the form of a directed graph where a directed edge represents a connection from a source block to destination block. The implementation is adopted with minor changes and integrated in the model instrumentation architecture.

2.3 S-TaLiRo Review

Simulink blocks have branching behaviors in the form of if- else constructs can be modeled as a Finite State Machine (FSM). A FSM, M is defined as a five tuple {I, O, S, Init, G} where

S is a set states,

I is a set of input valuations,

O is a set of output valuations,

Init is the set of initial states where $Init \subseteq S$

G is a transitional relation, $G: S \times I \rightarrow S \times O$

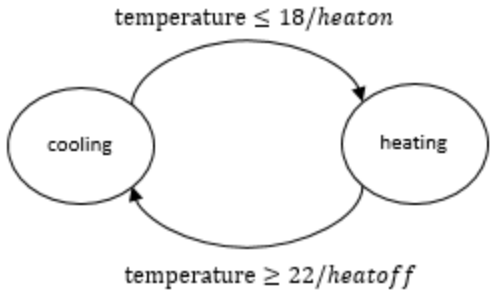

Figure 7 FSM Representation of a Thermostat System

An FSM can be represented using a graph where the vertices of the graph correspond to symbolic states and directed arcs corresponds to the transition relation G. Consider the Fig. 7 that illustrates an FSM representation of a simple thermostat system. The input to the system is temperature and the outputs are 'heatOn' and 'heatOff' which are Boolean signals. The system operates in any of the possible states – cooling and heating. The Guard set G determines the transition from the cooling state to heating state and vice versa. For this example, when the temperature is less than or equal to 18, there is a transition from the cooling state to heating state and the output 'heatOn' signal is true. Similarly there is a transition from heating to cooling state when the input temperature is less than or equal to 22 and the 'heatOff' is signal is true.

The FSM representation of the switch and saturation blocks are employed in S-TaLiRo during the test generation process. The details of this representation will be presented in the thesis and will be explained in Section 5.4 of Chapter 5.

For the purpose of this discussion, we will use the following notation:

Let R be the set of real numbers. A system Σ is viewed as a mapping from initial conditions X_0, system parameters P and input signals U^R to output signals Y^R. The output

space Y of the system Σ comprises of the original output space Y_Σ of the system, an auxiliary output space Y_A and a finite space Y_F, i.e., $Y = Y_\Sigma \times Y_A \times Y_F$. The output spaces Y_A and Y_F are introduced by the model instrumentation framework developed as part of this thesis and will be explained in Section 5.3 of Chapter 5. The output space Y_F captures the internal state of important components in the system. For instance, it can capture the state of switch/saturation blocks and state of state machines. The coverage metrics in this thesis are defined over space Y_F.

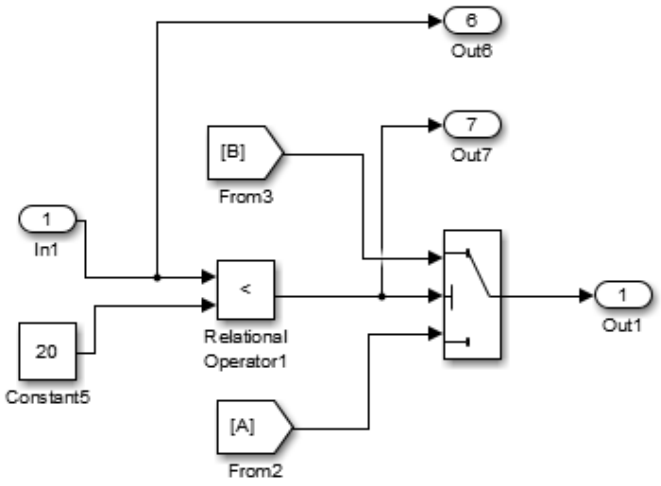

Figure 8 Switch Block Example to Demonstrate the Computation of Sets Y_A and Y_F

A simple illustration of the sets Y_A and Y_F for Simulink blocks is shown in Fig. 8. In Fig. 8, a simple switch block is presented which is an open loop system. The output port 'Out1' is part of the original output space Y_Σ. The output ports 'Out6' and 'Out7' are added to the system through instrumentation. Based on the result of the comparison of output port 'Out6' with the constant value 20, the corresponding state of the switch block is enabled. The value of 'Out6' is used to guide the search to the desired mode of the switch block. It is part of the set Y_A. The value of 'Out7' directly captures the current

state of the switch block and is part of the set Y_F. The same explanation can be extended to other Simulink blocks as well. In general, the output ports used for guiding the search to desired modes form the elements of the set Y_A. The output ports used only to obtain the mode information of the blocks are part of the set Y_F.

Using a metric **d** [17], a distance function can be defined that captures how far away a point y ∈ Y is from a set S ⊆ Y. The details of metric **d** are discussed in [11, 18]. When Y is a hybrid space, i.e., Y = R^n x Q with $Y_\Sigma = R^n$ and Y_F = Q, where Q is the set of states of single state chart in the model, then the generalized quasi-metric [11] $\mathbf{d}_h$ is defined as:

$$\mathbf{d}_h(\langle x, q\rangle, \langle x', q'\rangle) = \begin{cases} \langle 0, d(x', q')\rangle & if\ q = q' \\ \langle \pi(q, q'), \min\limits_{q \to q'' \xrightarrow{\pi(q,q')-1} q'} dist\ (x, G(t)(q, q''))\rangle & otherwise \end{cases}$$

where π is the shortest path metric on a graph, G(t) denotes the guard set that activates the transition from state q to state q' in the state chart. The min operator quantifies over all neighboring states q'' of q that are on the shortest path from q to q'. A more detailed discussion can be found in [11].

The idea behind the distance metric can be illustrated using an example on the Navigation benchmark [19]. Consider the example shown in Fig. 9. An automaton with nine states is considered where each cell corresponds to each state of the automaton. The variables x_1 and x_2 are the state variables corresponding to each state. Assume that the guards are simply the solid lines between each cell. The transitions are allowed from one cell to another in only orthogonal directions.

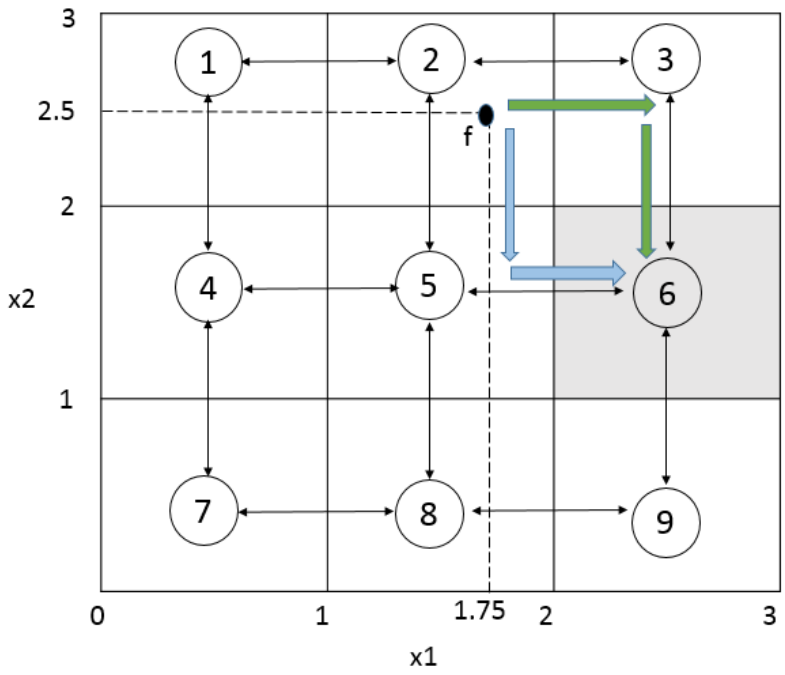

Figure 9 Navigation Benchmark to Demonstrate Distance Computation

Consider a point 'f' in the cell two with value x_1 = 1.75 and x_2 = 2.5. Let the unsafe set be present in the cell six (shaded grey in Fig. 9). The distance from 'f' to the unsafe state is the path distance between the two states weighted by the distance to the closest guard that will enable the transition to the next state. In this example, the path distance is two corresponding to the two highlighted paths in Fig. 9. The closest guard of all the paths with the same shortest path distance is the transition to cell three. Thus, the distance d_h is computed as (2, 0.25) where 2 is the path distance and 0.25 is the guard distance that enables a transition to the next state that reduces the path distance. This is essentially a heuristic that gives preference to shortest paths.

Metric Temporal Logic (MTL) Syntax: Let AP be the set of atomic propositions and *I* be any non-empty interval of $\mathbb{R}_+$. The set MTL of all well formed

MTL formulas is inductively defined as $\varphi ::= T \mid \pi \mid \neg \varphi \mid \varphi \vee \varphi \mid \varphi U_I \varphi$, where $\pi \in$ AP and T is true.

The MTL is a popular formal language used to state formal requirements for real time systems. Real time systems are characterized by quantitative temporal properties. We make use of metric temporal operators to express properties about the real time systems using MTL. A more detailed and formal discussion can be found in [20].

The formal specifications valuate using quantitative multi valued semantics. Given a system trajectory, the robust semantics evaluate to positive values if the trajectory satisfies the specification and to negative values if the trajectory violates the specification. The magnitude of the robust evaluations is a measure of how robustly the behavior satisfies or violates the specification.

MTL Falsification: For an MTL specification φ, the MTL falsification problem consists of finding an output signal y of the system Σ starting from some valid initial state from the set X_0 under a parameter vector p and an input signal u such that y does not satisfy φ. The test generation process makes use of the stochastic or deterministic optimization algorithms for the falsification problem [11, 12].

The above concepts can be further illustrated using a simple example. Consider a system defined by the differential equations:

$$\frac{dx}{dt} = x - y + 0.1t$$

$$\frac{dy}{dt} = y\cos(2\pi y) - xsin(2\pi x) + 0.1t$$

Let the initial conditions be the set [-1, 1] x [-1, 1] and the specification to be falsified be $G_{[0,2]} \neg a$ where O(a) = [-1.6, -1.4] x [-1.1, -0.9]. The specification is read as "Always Not a". This essentially captures the requirement of staying away from the unsafe set O(a).

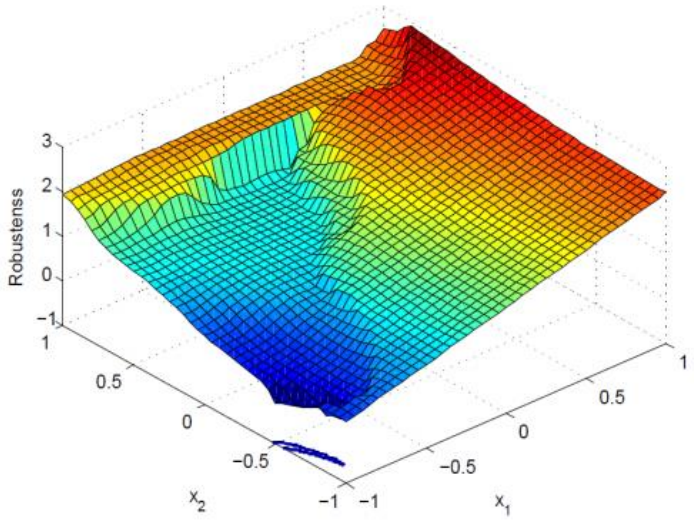

Figure 10 Robustness Landscape for the Specification $G_{[0,2]} \neg a$

The robustness landscape generated during the falsification of the above MTL formula is shown in Fig. 10. Here the system tries the generate tests that would guide the system trajectory to the unsafe set [-1.6, -1.4] x [-1.1, -0.9]. The positive values of the robustness indicate the specification is not falsified (did not enter unsafe set). The negative robustness shown as the blue region in the figure implies that the specification is falsified for the corresponding inputs, x_1 and x_2.

CHAPTER 3

PROBLEM DESCRIPTION

In this section, the problem is described formally and a brief solution overview is provided.

3.1 Problem Overview

This research aims to provide an automated framework for analyzing and extracting information about the blocks that introduce nonlinearities, e.g., switch and saturation blocks in Simulink models. A secondary goal is perform test generation for coverage analysis of these blocks with an existing specification guided testing framework, for example, S-TaLiRo [14].

In essence, the problem can be formulated as: Given a Simulink model:

- Identify the blocks with branching structure in the model and introduce output ports that become part of the auxiliary output space of the system Y_A.
- Automatically extract the state transition graphs for the branching blocks which will be utilized by S-TaLiRo.
- Integrate this framework in S-TaLiRo and use the S-TaLiRo test generation process to perform coverage analysis.

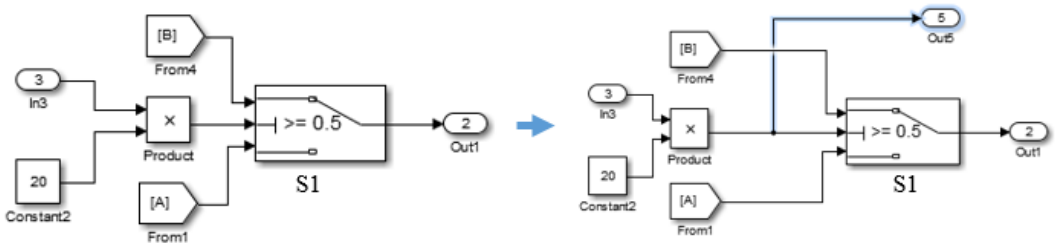

Figure 11 Scenario 1: Left – Switch Block Before Instrumentation. Right – Switch Block After Instrumentation.

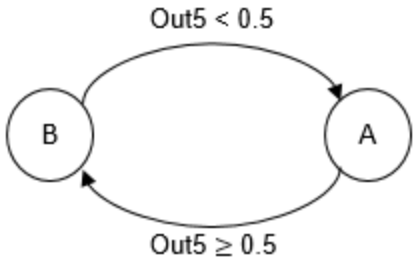

Figure 12 The FSM that Corresponds to the Switch Block S1 in Fig. 11. State B Corresponds to Signal B and State A Corresponds to Signal A Selection.

An instance of the problem is illustrated in Fig. 11. The switch block chooses either signal A or signal B based on the switching condition provided in the middle port. A framework is to be developed that can introduce the output port 'Out5' to the set Y_A for the switch block shown in Fig. 11. Additionally in an outer harness, the value of 'Out5' is used to compute the current mode information and used in the set Y_F. The corresponding statechart (Fig. 12) is to be extracted automatically from the the model.

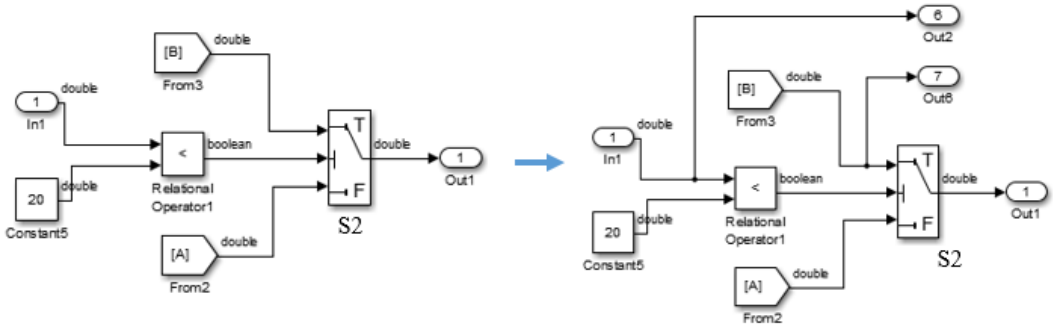

Figure 13 Scenario 2: Left - Switch Block Before Instrumentation. Right - Switch Block After Instrumentation

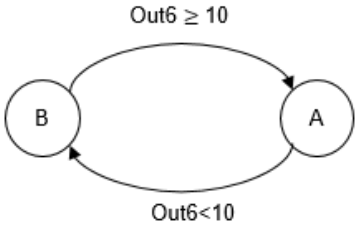

Figure 14 The FSM that Corresponds to the Switch Block S2 in Fig. 13. State B Corresponds to Signal B and State A Corresponds to Signal A Selection

Another example of the problem is depicted in Fig. 13. The framework is required to introduce the output ports ‘Out6’ and ‘Out7’. The output port ‘Out6’ along with the state chart in Fig. 14 can be used to compute the distance metric $\mathbf{d}_h$. The output port ‘Out6’ is part of the auxiliary output space Y_A. The output port ‘Out7’ forms an element of the set Y_F. It can be seen that the value of output port ‘Out7’ captures the state of the switch block.

In both state charts presented, the switch block is in state B when the input signal B is chosen and it is in state A, if signal A is chosen. State B corresponds to mode (location) two and state A corresponds to mode one of the switch.

The problem of coverage analysis is to determine the possibility of accessing the different modes (Signal A or Signal B in Fig. 11 and Fig. 13) using specification guided falsification performed by S-TaLiRo [20]. In a way, part of this problem relates to the branch and condition coverage criteria of software testing [21]. A similar problem description holds for any other blocks that introduce branching behaviors in the model, e.g., saturation blocks, statecharts, etc.

3.2 Solution Overview

Figure 15 shows the workflow of the model instrumentation framework proposed as a solution to the problem described above. A modular approach is used to develop the solution which enables easy maintenance and simple to incorporate future extensions. The first component is Model Flattening which produces a directed graph with the block connectivity information. Subsequently the Model Analysis component annotates the model and identifies the blocks with branching behaviors. Once the blocks are identified the block specific instrumentation module performs the instrumentation of these blocks.

The current framework supports instrumentation of switch and saturation blocks. The approach can be extended to other blocks. The instrumented blocks are represented as FSM. Finally the information is written in a MATLAB file as a set of commands to form the black box file interface to S-TaLiRo. The detailed implementation of these modules is described in Chapter 5. Further details can also be found in the help files of the m-functions distributed with S-TaLiRo [14].

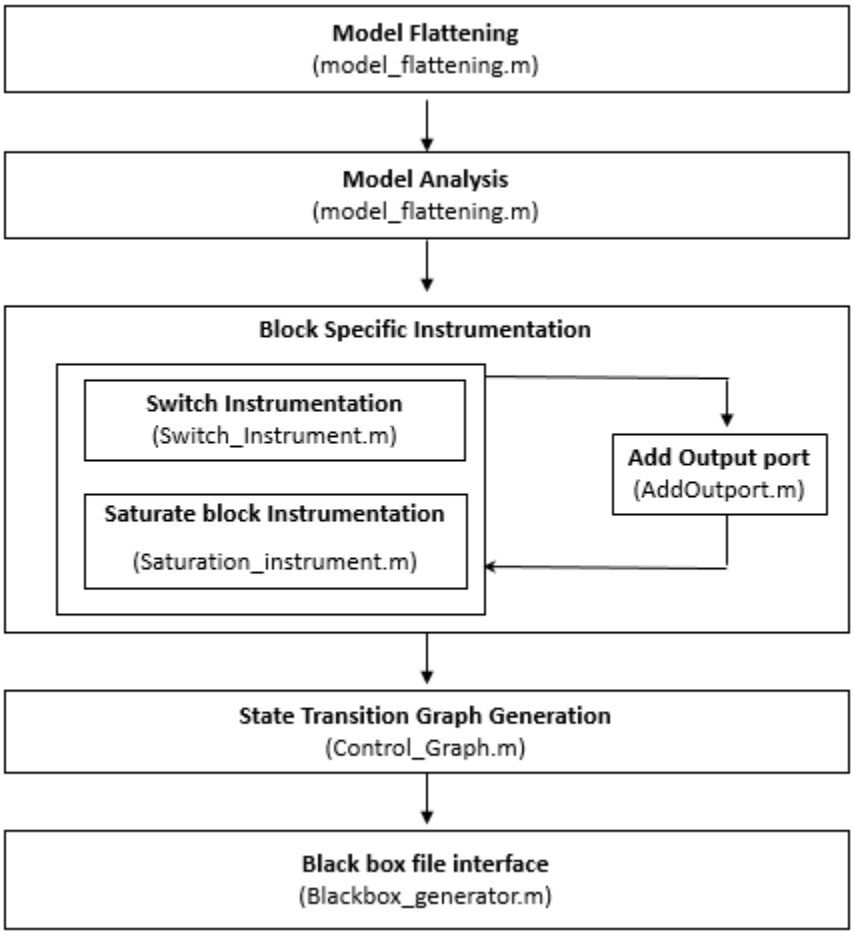

Figure 15 Workflow for Model Instrumentation Along with MATLAB m-Functions Implementing Each Step of the Process

CHAPTER 4

RELATED WORK

Simulation guided technologies is an emerging field for testing and it has attracted immense research interest. S-TaLiRo [14] is one such tool that it has been the focus of this work. Additionally, there are many other tools that focus on simulation guided testing which are similar to S-TaLiRo in some ways and differ in some other ways [29]. This section discusses some of the emerging tools for testing.

In [13], the authors present Breach - a MATLAB toolbox which aims to provide simulation based techniques to analyze models of hybrid systems. It performs approximate reachability analysis and parameter synthesis using sensitivity information for parameter variations. The tool uses Signal Temporal Logic (STL) for specifying the requirements. The tool makes use of a nonlinear optimizer based on the Nelder-Mead algorithm whereas S-TaLiRo makes use of a number of different optimization algorithms. Moreover, Breach does not support hybrid metrics that were first presented in [30].

In [25], a testing framework based on the rapidly exploring Random Trees (RRT) algorithm (popular motion planning algorithm) is proposed. A star discrepancy notion is used as a measure of test coverage for the continuous state space of a CPS. The approach is orthogonal to the coverage based testing employed in the S-TaLiRo [14]. The tool is still under development and its applicability to industrial size models is yet to be determined.

The Simulink Design Verifier from Mathworks [26] uses SAT- solving techniques to generate test inputs to maximize coverage criteria. The tool is useful for

open loop analysis of discrete time models and it does not support continuous closed - loop models. Tools like SpaceEX [5] and UPAAL [27] can perform exhaustive verification rather than test generation, but they can verify only certain classes of hybrid systems.

Another promising falsification based approach for closed loop systems is proposed in [28]. A technique called trajectory splicing is used which uses local optimization to minimize the gap between disconnected trajectory segments, in essence, splicing the trajectories to form a concrete trajectory. The initial results seem promising on the benchmark applications but their application on practical Simulink models is still under development.

CHAPTER 5

MODEL INSTRUMENTATION WORKFLOW

In this section, each component of the model instrumentation framework introduced in the previous chapter is discussed in detail.

5.1 Model Flattening

The first and foremost step is to flatten the given Simulink model. The model flattening is discussed in Section 2.2.1. The implementation is adopted from [22] with minor changes and integrated in the model instrumentation architecture. The Simulink model to be instrumented is provided as the input and a new list containing of only main blocks is generated. This list is used to form a directed graph that captures the flattened version of the model where a directed edge represents a connection from a source block to destination block.

5.2 Model Analysis

This component identifies blocks with branching structure present in the model. Currently, model analysis annotates the model for switch and saturation blocks. Since there can be a large number of switch and saturation blocks and all of them might not be significant for coverage analysis, some of them have to be excluded. The module is designed to accept a list of blocks that are to be excluded from the instrumentation and coverage analysis. Additionally, model analysis automatically excludes switch blocks that are not good candidates for the coverage problem. For example, one such scenario is illustrated in Fig. 16.

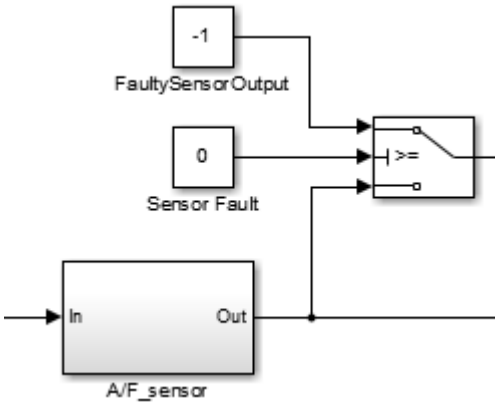

Figure 16 Switch Block Triggered by Constant Value

In Fig. 16, the switch block is triggered by a constant value. This implies that at all times only one of the possible modes is enabled. This type of hard coded values are often used by designers to manually activate only certain behaviors and such blocks add no value to the coverage problem. Hence, such blocks must be excluded from being further considered for instrumentation.

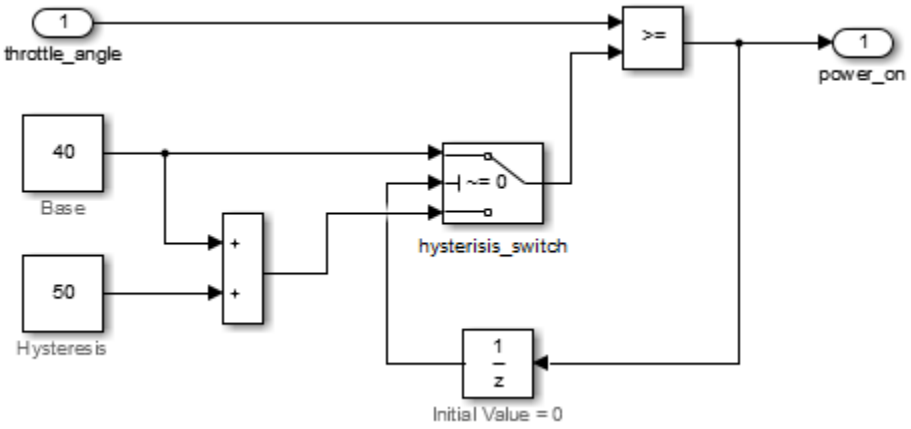

Figure 17 Switch with Delay Block (from [24])

There can be other scenarios for block exclusion as for example in Fig. 17. The subsystem models a hysteresis loop for the throttle angle. A simulation trace of the subsystem is shown in Fig. 18. The hysteresis range is determined by the sum of the base

constant value (40) and the hysteresis constant value (90). Initially the delay value is zero and the third input of the switch block, i.e., the value 90 is passed to the comparator block. When the throttle angle is greater than equal or equal to 90, the 'power_on' signal evaluates to one. Consequently, after the time delay specified by the delay block, the first input of the switch is passed to the comparator block. During this delay time, any change in the throttle angle input will not change the mode of the switch. After the delay time, the first input of the switch, i.e., the value 40 is passed to the comparator block. Now, when the throttle angle goes below 40, the new hysteresis constant of 90 is passed to the comparator after the delay time and the cycle continues.

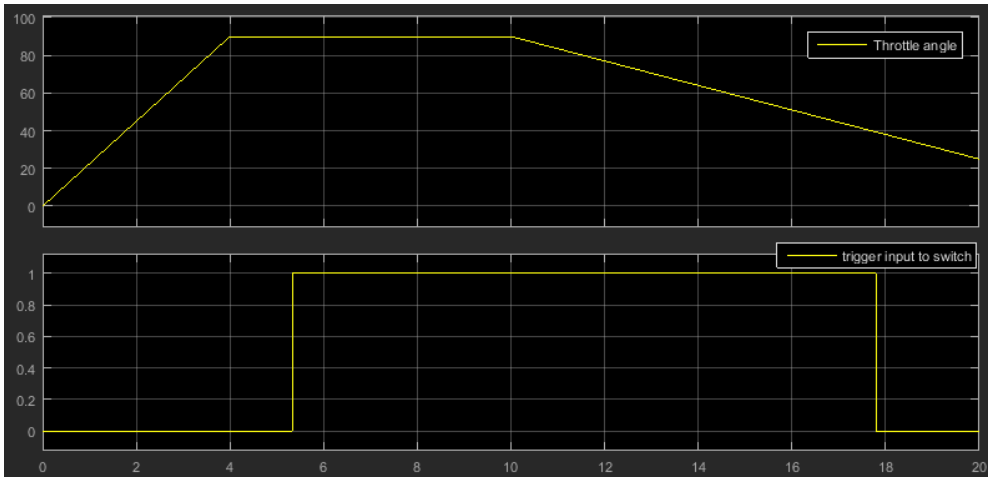

Figure 18 Simulation Trace of the Hysteresis Subsystem

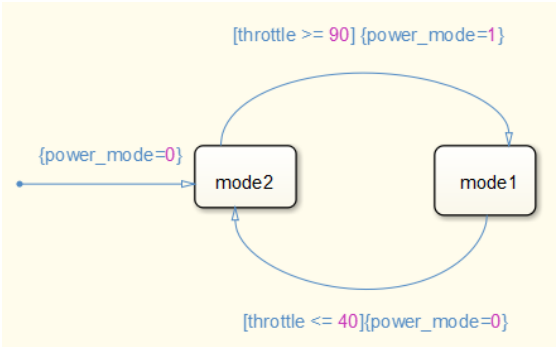

Figure 19 Stateflow that Corresponds to 'hysterisis_switch' in Fig. 17. Mode 2 Corresponds to Input Value of 90 and Mode 1 Corresponds to Input Value of 40.

The FSM that corresponds to ‘hysterisis_switch’ is shown in Fig. 19. The State mode1 corresponds to the input value of 40 and the State mode2 corresponds to the input value of 90.

As such, it is not meaningful to instrument the switch block for coverage guidance. However, note that the comparator block in Fig. 17 should be instrumented if it is controlling another switch block downstream. The subsystem can be supported if the FSM corresponding to the hysteresis loop is provided. The key to analyzing such components is that either they need to be modeled as Stateflow charts or we need to develop model analysis tools that extract such information. The latter is not a trivial task.

The model analysis component is designed with the user interface in mind. It can be called independently out of the model instrumentation framework to obtain useful information about the model before running a full blown coverage analysis on the model. In Chapter 6, potential extensions for model analysis are discussed which enable the user to have vital block information in the model.

5.3 Block Instrumentation

The block instrumentation module performs the instrumentation of the blocks with branching behavior identified by the model analysis section. Currently, only switch and saturation blocks are supported and the approach can be extended to other blocks. The instrumentation of the switch and saturation blocks is handled separately due to the different scenarios that decide their behavior.

5.3.1 Switch Instrumentation

The instrumentation of switch blocks was briefly introduced in the problem statement of Chapter 3. The two broad scenarios illustrated in Section 3.1 are reviewed

here for a more detailed discussion. The switch block in Fig. 20 is triggered by a real valued (double precision variable) input and the one in Fig. 21 is triggered by a Boolean value. These two cases are handled separately as the methodology of their analysis differs slightly.

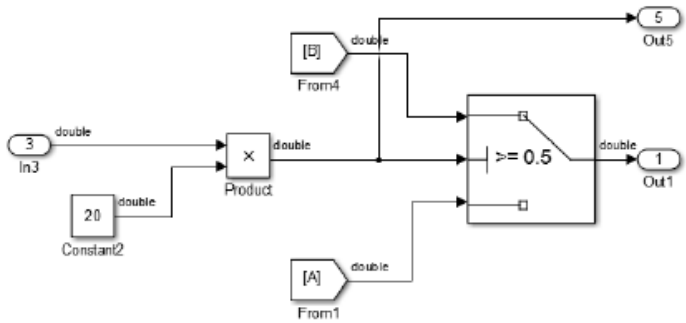

Figure 20 Switch Block Triggered by Double-Precision Input (Already Instrumented)

The switch blocks which are triggered by double-precision input signal variables are trivial to handle. Consider Fig. 20 which shows the switch block after instrumentation. The output port 'Out5' is automatically added by the instrumentation process. The output port 'Out5', which is part of the set Y_A, is utilized to compute the distance metric used in the optimization engine of S-TaLiRo. In this particular example, the value of 'Out5' can be compared with the threshold value of the switch, which in this case is 0.5. The difference between these two values provides a measure of how far or near we are to each of the state. This output port value forms the crux for obtaining the state transition graph of the above switch block. A more detailed explanation on the generation of state transition graph is provided in the next section (Section 5.4).

The switch blocks triggered by Boolean input values are analyzed more rigorously. The Boolean input to the middle port of the switch blocks maybe the output of a Boolean circuit. An example is illustrated in Fig. 21. In this case, the Boolean circuit

consists of four compare blocks and three logical operators. The output of the comparison blocks are the result of the comparison made between the input block and the constant blocks present in the left end of Fig. 21. These outputs are evaluated by a Logical OR block and then by a Logical AND block to determine the trigger input to the switch block. When the output of the 'AND' block evaluates to 'TRUE', signal B (mode 2) is passed. Similarly, signal A (mode 1) is passed when the output is 'FALSE'.

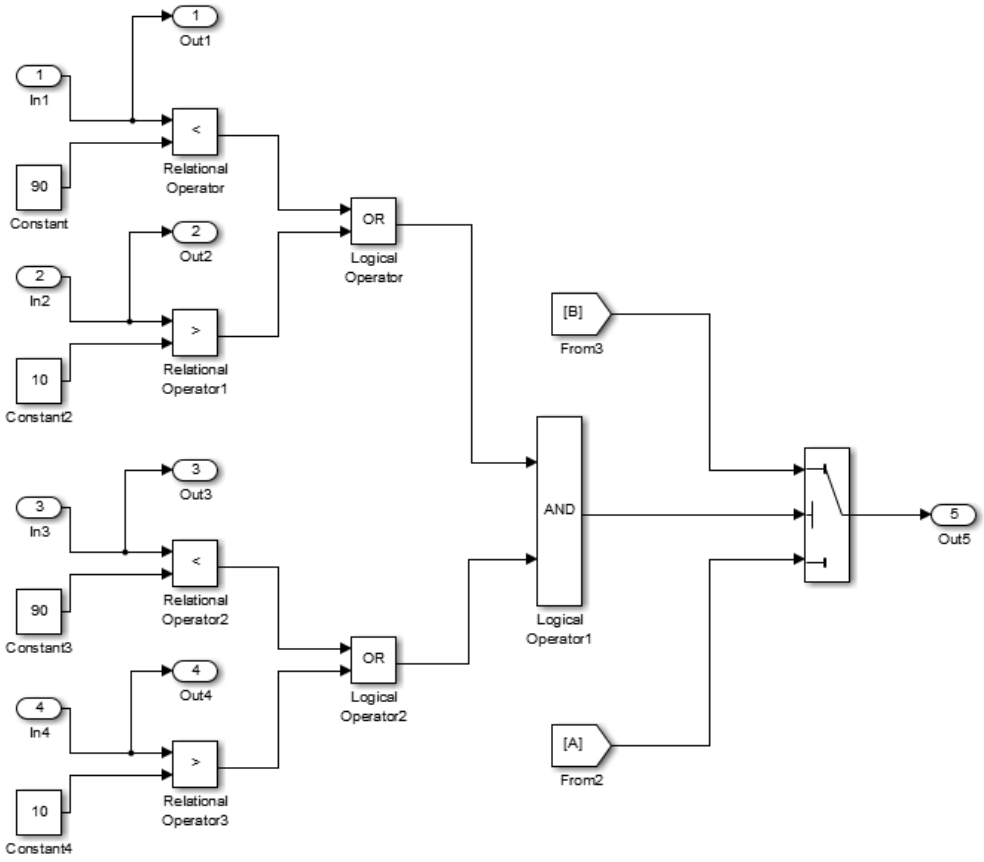

Figure 21 Example of Switch Block Triggered by a Boolean Function

It can be seen that mode coverage of the switch block can be obtained by the conditional coverage of the Boolean circuit. Hence, the auxiliary output ports are added at the relation operator blocks which can be used to compute the distance metric d_h. These outputs are termed as numerical output ports and are part of the set Y_A. These

output ports will be used in formulating the guards for the state transition of the switch. Additionally, an output port is added at the trigger port of the switch to determine whether the switch is in state A or B. The latter output ports are termed “location output ports” and are part of the set Y_F.

Once the output ports have been added, the instrumentation process also automatically extracts the Boolean function that decides the state of the switch. The graph information obtained from the model flattening is used for this process. It is possible that the Boolean function may be implemented in a separate subsystem and failure to flatten the model may lead to improper instrumentation. From the main flattened graph, a small subset graph establishing the connectivity information of the switch block is obtained. A DFS search is performed on this graph to identify the relational blocks and Logical blocks and the Boolean function is extracted in the form of a string. For the above example, the instrumentation will produce a string as:

$$((O1 < 90) \,||\, (O2 > 10)) \;\&\&\; ((O3 < 90) \,||\, (O4 > 10))$$

Then, the string is processed to obtain a mapping of the predicates to Boolean literals. The predicates (O1 < 90), (O2 > 10), (O3 < 90) and (O4 > 10) are replaced by Boolean literals, say p1, p2, p3 and p4 respectively and the Boolean string ‘(p1 | | p2) && (p3 | | p4)’ is used for further analysis. A predicate table is generated with the mapping information of the predicates and the corresponding Boolean literals. However, note that the Boolean function may not be in a Disjunctive Normal Form (DNF). The DNF representation is convenient for obtaining the state machine representation of the switch and, consequently, it is easier to perform distance computation under the metric d_h [15]. Hence, the Boolean formula is converted to its equivalent DNF representation.

5.3.1.1 DNF Conversion Methodology

A Boolean formula is in Disjunctive Normal Form (DNF) if it is a disjunction of conjunctive clauses or, in other words, if it is in Sum of Products (SOP) form. When any one of the conjunctive clauses is evaluated to true, the whole Boolean formula evaluates to true value. Now each of the conjunctive clauses provides a sense of distance from the transition state A to state B of the switch (Fig. 21). Hence the conditional coverage on the conjunctive clauses will result in mode coverage of the switch.

The main steps involved in the DNF conversion are illustrated through an example for the switch block shown Fig. 21. First, an expression tree is built for the formula '(p1 | | p2) && (p3 | | p4)' using a stack based approach as shown in Fig. 22. The Boolean literals form the leaves of the tree data structure and the Boolean operators '| | 'and '&&' constitute the parent nodes.

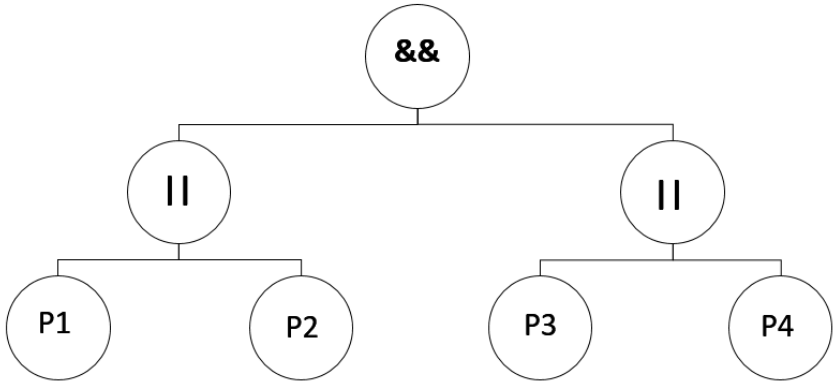

Figure 22 Expression Tree for the Boolean Formula '(p1 | | p2) && (p3 | | p4)'

The formula is converted to DNF by iteratively applying logical equivalences, such as double negation elimination, De Morgan's law and the distributive law throughout the tree [23]. The steps involved in the conversion can be summarized as:

- The negation is moved inwards recursively. In the process, De Morgan's law and double negation are applied.
- The AND operator is distributed over the OR operator recursively in the tree.
- The clauses that always evaluate to false, if any are removed.

The above steps are applied to the tree shown in Fig. 22. The result of the algorithm is shown in Fig. 23 and Fig. 24. Figure 23 illustrates the intermediate step where the '&&' (AND) operator is distributed over the '| |' (OR) operator. Finally, Fig. 24 illustrates the tree in DNF after recursively applying the algorithm.

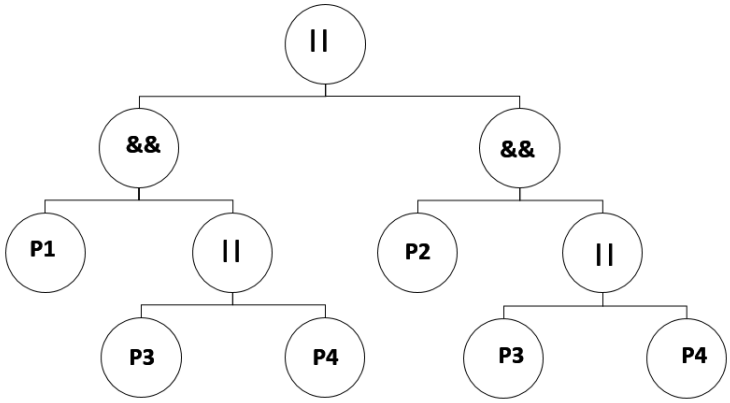

Figure 23 Intermediate Step of DNF Conversion Algorithm

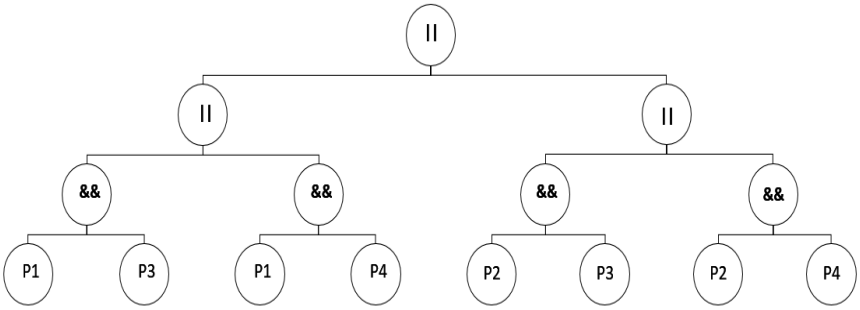

Figure 24 Final Expression Tree in DNF

The tree in DNF representation is read in infix order to obtain the Boolean formula in its equivalent DNF. Hence, the conversion of the formula '(p1 | | p2) && (p3 | | p4)' results in '(p1 && p3) | | (p1 && p4) | | (p2 && p3) | | (p2 && p4)'. Similarly, the negation of the formula after conversion results in '(!p1 && !p2) | | (!p3 && !p4) '. The Boolean formula in DNF along with the predicate mapping are used to develop the state transition graph for the switch. The details are discussed in the state transition graph generation section (Section 5.4). The complete implementation is done in C language and interfaced with MATLAB using the MEX compiler.

5.3.2 Saturation Instrumentation

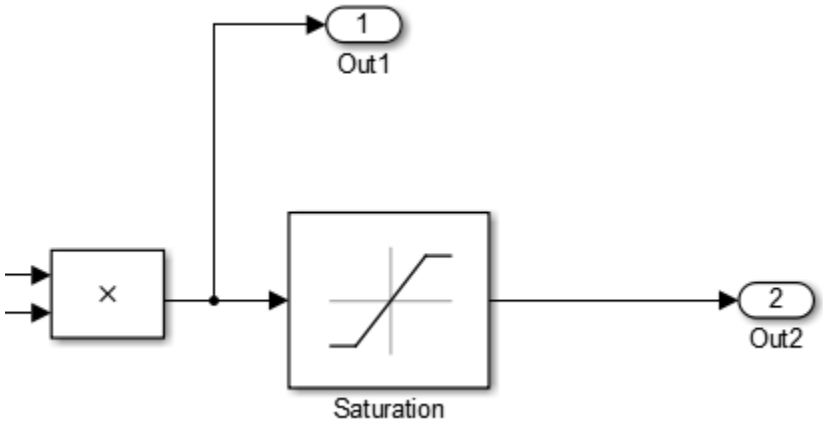

Figure 25 Saturation Block Instrumentation

A saturation block imposes upper and lower limits on the input signal. The input signal is passed to the output of the block when it falls between the lower and upper limits. In this case, there are three possible modes that the block may operate at any given point in time: lower saturation, upper saturation or no saturation. The instrumentation of the saturation blocks is similar to the switch blocks triggered by double precision valued input signals. Figure 25 shows the instrumentation of a saturation block. The output port

‘Out1’ is added at the input port of the block. The value of the output port ‘Out1’ can be used to compute the distance from the different operating modes of the saturation block and it is part of the set Y_A. Additionally, in an outer harness the value of ‘Out1’ can be used to determine the mode of the saturation block which is added to set Y_F.

Remark: It must be noted that these blocks with branching structure maybe present inside subsystems of the model. The instrumentation is performed recursively and the output ports are brought to the upper most layer of the model.

5.3.3 Avoiding Redundancy during Instrumentation

During the instrumentation of a block, it is possible that the output port already exists and the addition of a new output port can result in redundancy. The block instrumentation takes into account this possibility by checking for the existence of the output port. In case the output port already exists, it simply returns the id of the existing output port which can be used when generating the corresponding FSM. In essence, there is no duplication in the sets Y_Σ, Y_A and Y_F. The instrumentation fails to avoid redundancy when the existing output port in a subsystem is brought out to the upper most layer of the model with a different name. In this case, the instrumentation issues a warning and adds a redundant output port in either of the set Y_A or Y_F. It must be noted that the redundancy does not affect the correctness of the framework.

5.3.4 Selection of Blocks for Instrumentation

The current framework performs instrumentation of all the supported blocks with branching behavior. This may not be a practically scalable approach as the combinations for the mode coverage increase exponentially. The current methodology relies on some

inputs from the user to exclude some blocks from the instrumentation process. In addition, few redundant blocks are identified automatically and excluded from the instrumentation as described in Section 5.2. In general, certain heuristics must be developed that can disregard blocks that need not be considered for instrumentation to address this issue. This is left as future work.

5.4 State Transition Graph Generation

The automatic extraction of the FSM for the blocks is the most important aspect of the automation process. Prior to implementation of this framework, the user had to manually go through the model to identify the branching blocks and analyze them to obtain their FSM. This entire process can now be done at the click of a button.

This module obtains the state transition graph for the instrumented switch and saturation blocks. In particular, a switch block can be modeled as a finite state machine with two states s1 and s2 and a transition guard $G(s1, s2) \subseteq R^n = Y$, where Y is the output space of the system Σ. A transition from s1 to s2 is enabled if and only if $y(t) \in G(s1, s2)$. In S-TaLiRo, a data structure called "Guard" is used to capture the FSM representation. The Guard which captures the transition relation G, captures the inequality form $A\, y \preccurlyeq b$ where y is vector of the output space of Σ of size $\#(Y_\Sigma) + \#(Y_A)$ where #(Y) indicates the dimensionality of the space Y and A is a matrix of size m x $(\#(Y_\Sigma) + \#(Y_A))$ where m is the number of constraints that define the polyhedron G(s1,s2). In addition to the Guard structure, an adjacency graph structure 'CLG' (Control Location Graph) is used to represent the possible transitions from a given state to another.

The details of obtaining the ‘Guard’ and ‘CLG’ structure are illustrated in this section. First, a simple example of a double precision input triggered switch block is presented. Then, a slightly complex example of a Boolean value triggered switch block will be discussed.

For convenience, the switch block shown in Fig. 11 and its FSM representation are reproduced from Section 3.1. The FSM is depicted with state numbers one and two according to the convention introduced in Section 3.1. State 1 corresponds to operating mode one of the switch block where signal A is passed to the output. State 2 corresponds to the mode two of the switch block where signal B is passed.

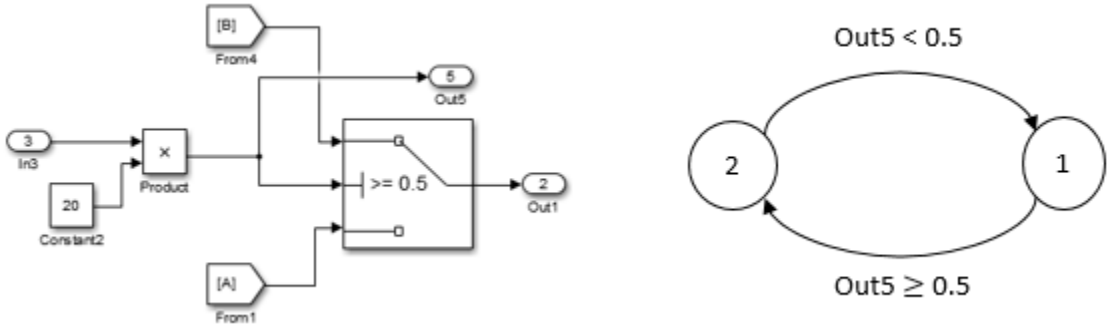

Figure 26 Left: Double Precision Input Value Triggered Switch Block; Right: the Corresponding FSM Where State 2 Corresponds to Signal B and State 1 Corresponds to Signal A Selection

It can be seen that the guard relation for the transition from State 1 to 2 is captured by the equation ‘Out5 ≥ 0.5’. The ‘Guard’ structure employed in S-TaLiRo uses the matrices ‘A’ and ‘b’ to capture the transition relation. For this example the ‘Guard’ is formulated as:

Guard (1, 2). A = [0 0 0 0 -1 0 . . . . 0] Guard (1, 2). b = - 0.5

where A is row vector of size #(Y_{Σ}) + #(Y_A), i.e., the original output ports plus the auxiliary output ports added by instrumentation.

The matrix A has a value -1 in the fifth position and the value of b is -0.5 which represents the guard relation in the form '- Out5 ≤ - 0.5' which is equivalent to the equation 'Out5 ≥ 0.5' since it is the convention to represent the guards in A y ≼ b form.

Similarly the guard relation for the transition from state 2 to 1 based on the equation 'Out5 < 0.5' is obtained as:

Guard (2, 1). A = [0 0 0 0 1 0 . . . . 0] Guard (1, 2). b = 0.5

Remark: The data structure "Guard" actually captures the inequality "Out5 ≤ 0.5" and not "Out5 < 0.5". However, this is not an issue since the guard is used to compute the robustness value and not whether the guard is activated. For example, when Out5 = 0, the robustness of both "Out5 ≤ 0.5" and "Out5 < 0.5" is the same, i.e., 0.5.

The adjacency graph 'CLG' for the switch contains the possible transitions from State 1 and the possible transitions from State 2. In case of a switch block, there is only one possible transition that can be taken from a given state and the 'CLG' can be simply expressed as:

CLG {1} = [2];

CLG {2} = [1];

In general, 'CLG' is a cell array whose length is equal to the number of states of the automaton. The contents of each cell is a vector containing the possible transition from that state. For a switch block, the only possible transition from state 1 is to state 2. Hence, CLG{1} is single vector with value 2.

Now, a more complex switch block triggered by a Boolean value input is considered for illustrating the process of obtaining the FSM. The switch block is the one considered in Fig. 21 in Section 5.3.1. The figure is reproduced here for convenience (Fig. 27). The previous instrumentation module obtains the Boolean formula in DNF that triggers the switch. For this example, the Boolean formula in DNF for the 'TRUE' condition (Signal A) is obtained as '(p1 && p3) || (p1&&p4) || (p2&&p3) || (p2&&p4)' and the formula in DNF for the 'FALSE' condition (Signal B) is '(!p1 && !p2) || (!p3&&!p4)'. The Boolean literals, i.e., p1, p2, p3 and p4 are mapped to the predicates of each compare block. For example, p1 corresponds to the predicate 'Out1 < 90'. Table 1 shows the mapping of predicates to the corresponding Boolean literals.

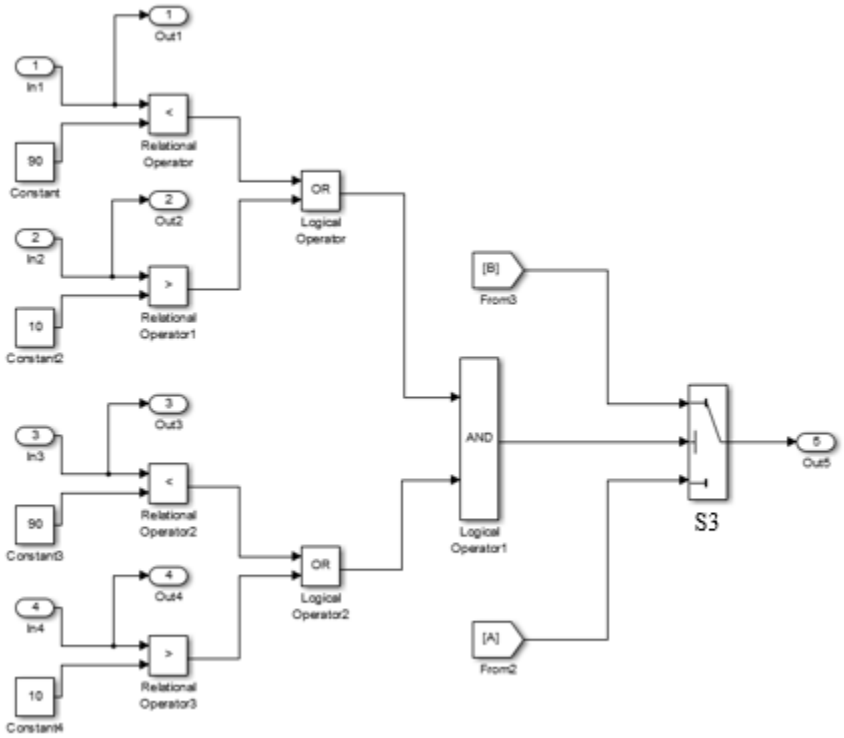

Figure 27 Switch Block S3 Triggered by Boolean Value Input

Predicate in the model	Boolean literal
Out1 < 90	p1
Out2 > 10	p2
Out3 < 90	p3
Out4 > 10	p4

Table 1 Mapping of Predicates to the Corresponding Boolean Literal

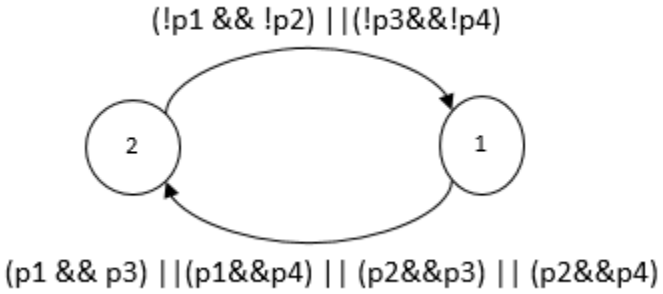

Figure 28 The FSM that Corresponds to the Switch Block S3 Shown in Fig. 27

The FSM for the switch block S3 is depicted in Fig. 28. The convention for the state is the same as the one introduced in Section 5.3.1. The State 1 and 2 in the FSM correspond to the signal A and signal B being passed to the output. The guard that corresponds to the transition from mode one to two is given by the DNF formula when it evaluates to 'TRUE'. Hence the formula '(p1 && p3) || (p1&&p4) || (p2&&p3) || (p2&&p4)' represents the guard for the transition from mode one to two. Similarly, the guard for transition from mode two to one corresponds to the expression '(!p1 && !p2) || (!p3&&!p4)'.

The 'Guard' structure captures the transition relation corresponding to each of the conjunctive clauses in the DNF expression. Consider the guard representation for the

transition from mode one to two and the corresponding first conjunction expression '(p1 && p3)'. The structure is obtained as:

Guard (1, 2). A {1} = $\begin{bmatrix} 1\ 0\ 0\ 0\ 0 \ldots 0 \\ 0\ 0\ 1\ 0 \ldots 0 \end{bmatrix}$ Guard (1, 2). b {1} = $\begin{bmatrix} 90 \\ 90 \end{bmatrix}$

Now the field 'A' is a cell vector with each cell corresponding to a disjunctive clause in the expression (in this example there are 4 clauses). The first element in the cell A, i.e., A{1} and in the cell b, i.e., b{1} capture the predicate 'Out1 < 90' (mapped as p1). The second element captures the expression 'Out3 < 90'. In essence, the Guard shown above captures the information that the transition from mode one to two is enabled when 'Out1 < 90' and 'Out3 < 90' simultaneously. In a similar way, the Guard data corresponding to the other conjunctive clauses are obtained.

Along the same lines, the Guard for the transition from mode two to mode one can be derived easily. In this case, the expression '(!p1 && !p2) || (!p3&&!p4)' is utilized to obtain the Guard. The size of the cell 'A' and 'b' will be two corresponding to the two conjunctive clauses (!p1 && !p2) and (!p3&&!p4).

Each of the guards that correspond to an individual conjunctive clause in the DNF formula represent a different set. The disjunction of the linear inequalities is used to set up a convex optimization problem by the optimization engine of S-TaLiRo.

The same approach is employed for saturation blocks to obtain the equivalent FSM. In this case there are three states s1, s2 and s3. The states s1 and s3 correspond to the lower and upper threshold value of the saturation block. The Guard structure obtains the transition relation between each of these states in the same manner as illustrated for the switch blocks.

5.5 Black Box File Interface

The creation of the black box file is the final component of the model instrumentation process. The black box file forms an interface to the S-TaLiRo test engine. It is an executable MATLAB function file (m-function). The structure of the file is shown in Fig. 29. The CLG and Guard structures corresponding to the instrumented blocks obtained from the State Transition Generation module (Section 5.4) are written in the first section of the file. However, the CLG and Guard structures can also be initialized in advance rather than when the black box file is executed. The MATLAB class that could be used for the advanced initialization is discussed in Chapter 7.

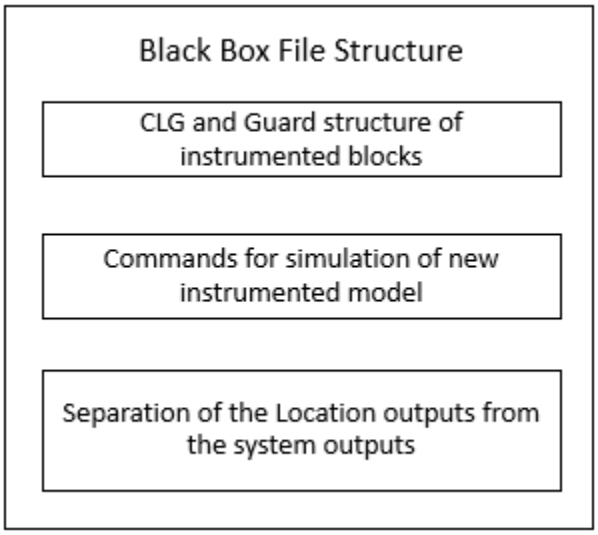

Figure 29 Black Box File Structure

In the second section of the file, the commands corresponding to the simulation of the new instrumented Simulink model are written. We simulate the system using the MATLAB 'sim' function for Simulink models. When this file is executed, the new instrumented model is simulated and the output trajectory of the system is obtained, i.e., system output trajectories and auxiliary outputs introduced by the instrumentation process.

In particular, this output trajectory will now also contain the information related to the modes of the switch and saturation blocks (Y_F) due to the output ports added during the instrumentation process.

In the third and final section of the black box file, the output ports corresponding to the locations are separated from the output ports used for the computation of the distance metric (Y_A). The values of the location output ports are used to determine the modes of the block.

Let 'sim_model' is the model to be instrumented. Then, a black box m-file with the name 'BlackBox_sim_model' is created with the following input- output interface:

[T, XT, YT, LT, CLG, Grd] = BlackBox_sim_model (simT, TU, U)

Inputs:

simT – simulation time for the model

TU – the discrete time steps in the simulation time 'simT'

U – The control input to the system

Outputs:

T – Time vector returned from the MATLAB sim command

XT – The state returned from the MATLAB sim command

CLG – Control Location Graph possible state transition for each instrumented block.

Grd – Guard structure of each instrumented block

LT – Location value of each instrumented block

YT – Output space after the location output ports are separated from the system output space.

Hence, after the execution of the black box file in S-TaLiRo, the following three pieces of information are obtained:

- The state transition graph and the guard relations for the blocks with branching structure in the model.
- The mode (location) information of the blocks with branching structure, Y_F.
- The auxiliary output space of the system Y_A which is used to compute the distance metric $\mathbf{d}_h$.

CHAPTER 6

EXPERIMENTS AND RESULTS

This chapter presents the results for two experiments that were conducted to illustrate the model instrumentation framework. In the first experiment, model instrumentation is performed on an air- fuel control system from the powertrain control benchmark (AbstarctFuelControl) [24]. Then, the instrumented model is used to demonstrate the mode coverage by integrating the framework in S-TaLiRo. In the second experiment, a toy example is considered to demonstrate the usage of model instrumentation and S-TaLiRO framework for coverage guided falsification.

6.1 Experimental Results for Powertrain Control Benchmark

In this section, two sets of results will be discussed. First, the output of running the model instrumentation framework on the benchmark is illustrated in Section 6.1.2. Second, a comparison is made between the uniform random sampling approach and the specification guided approach for a particular example of mode coverage.

The model is slightly modified, so that we can use it to demonstrate the mode coverage. The minor modification is discussed first before presenting the experimental results.

6.1.1 Model Modification

The hysteresis subsystem in the model is modified for the mode coverage experiments. The modified subsystem is shown in Fig. 30. The ‘hysteresis’ constant value is changed from 50 to 48 (highlighted blue in Fig. 30). In the original model, the ‘power_on’ signal is turned on only when the input throttle angle is greater than or equal

to 90. However, the input range for the throttle angle is the set [0, 90], i.e., the input value of exactly 90 has to be chosen to activate the ‘power_on’ signal. It is practically impossible for S-TaLiRo to choose this value from the search space. Since our aim is to demonstrate mode coverage, we modify the ‘hysteresis’ constant value from 50 to 48. Consequently, the ‘power_on’ signal is activated when input throttle angle is greater than or equal to 88. This improves the sampling probability and enables us to demonstrate mode coverage. An alternative would be to increase the input range to [-5, 95] and saturate the input to the range [0, 90]. This would render the probability of sampling the value 90 non zero.

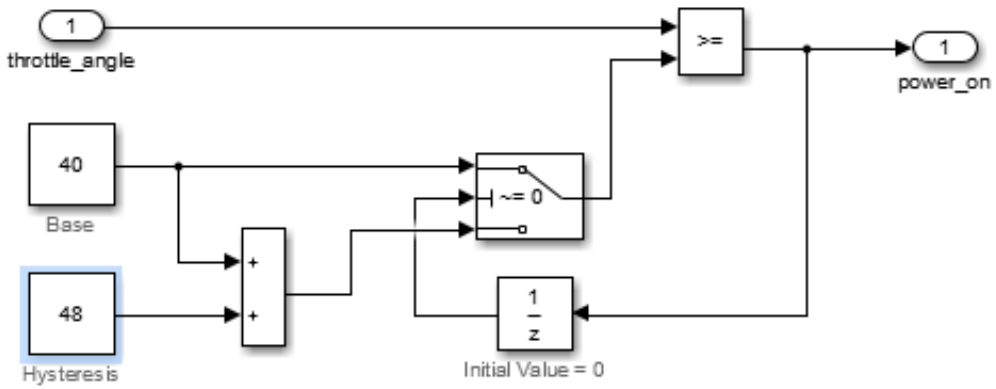

Figure 30 Modified Hysteresis Subsystem

6.1.2 Model Instrumentation Results

The air-fuel control benchmark has two inputs: - pedal angle and engine rpm and three verification measurements as outputs of the system. Additionally, the model contains a total of four switch blocks and two saturation blocks which are potential candidates for coverage analysis.

The model instrumentation algorithm disregards any switch block triggered by constant value since it is useless to perform coverage on this block since only one mode is activated at all times. In addition, another switch block triggered by delay block in feedback is not considered for instrumentation as explained in Section 5.2 of Chapter 5. Finally, two switch blocks and two saturation blocks are considered for instrumentation shown in Fig. 31, Fig. 32 and Fig. 33 respectively.

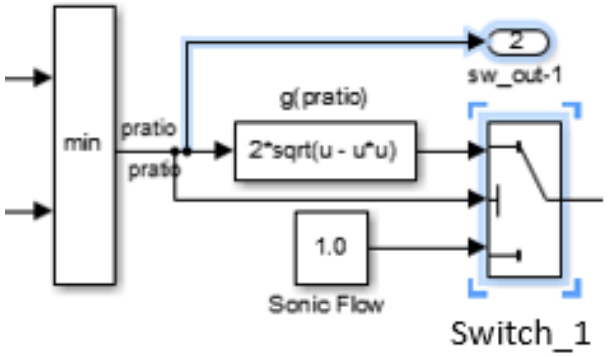

Figure 31 Double Precision Input Value Triggered Switch Block with the Added Output Ports Indicated by Blue Lines

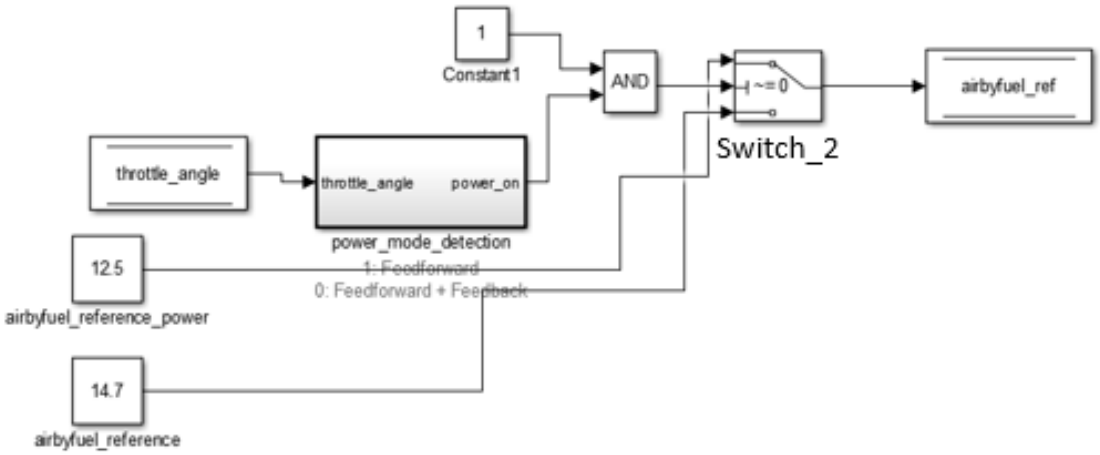

Figure 32 A Boolean Value Triggered Switch Block

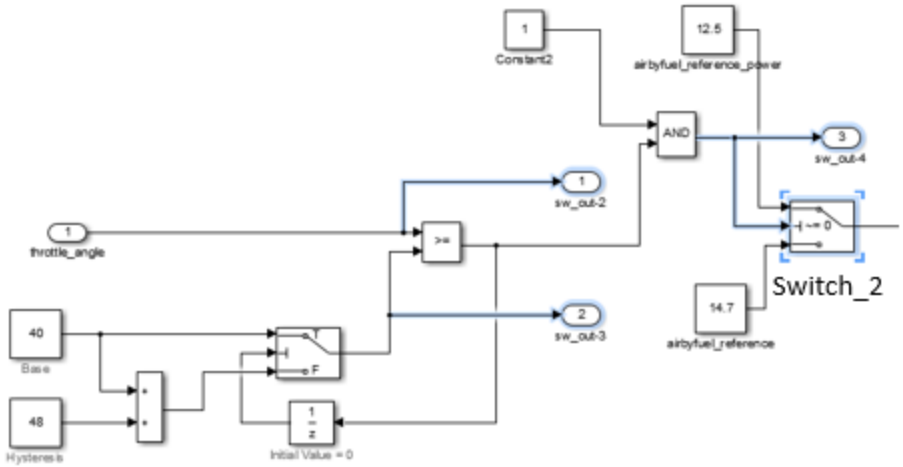

Figure 33 An Equivalent Visualization of Switch_2 in Fig. 32 After Flattening with the Added Output Ports Indicated by Blue Lines

The basic steps are illustrated for the switch block triggered by a Boolean value (Fig. 32). First the model flattening is performed and can be visualized as shown in Fig. 33. The blocks inside the power mode subsystem block are brought out and connected to the corresponding blocks in the upper level of the model. The numerical output ports are added to relational operator block and location output port is added at the trigger port of the switch. The instrumentation is performed recursively as explained in Section 5.3.2 and the output ports are brought to the upper most layer of the model. The added output ports are indicated by the blue lines in Fig. 33.

The two saturation blocks after instrumentation are shown in Fig. 34. The saturation blocks are highlighted by the blue square boxes and the output ports added by model instrumentation are shown using the blue lines. Similar to the switch blocks, the output ports are recursively added to bring them to the upper most layer.

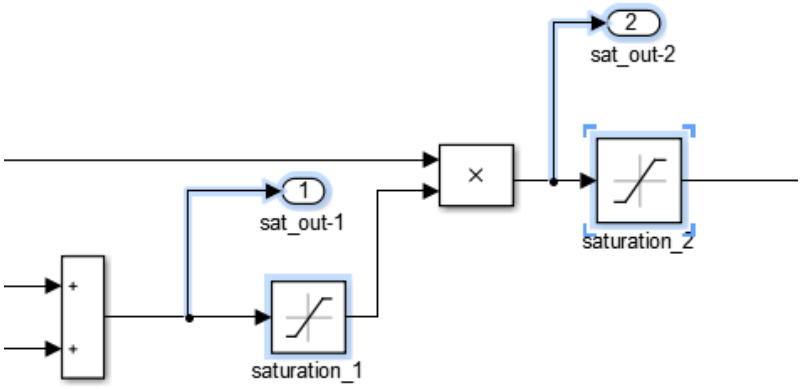

Figure 34 Two Saturation Blocks Chosen for Instrumentation with the Added Output Ports Indicated by Blue Lines

The final instrumented model after recursive addition of output ports is shown in Fig. 35. The output ports ‘sat_out-1’ and ‘sat_out-2’ correspond to the two saturation blocks and the output ports ‘sw_out-1’ to ‘sw_out-4’ correspond to the switch blocks. This instrumented model containing the discontinuities (switch and saturation blocks) is used by the S-TaLiRo engine for simulation.

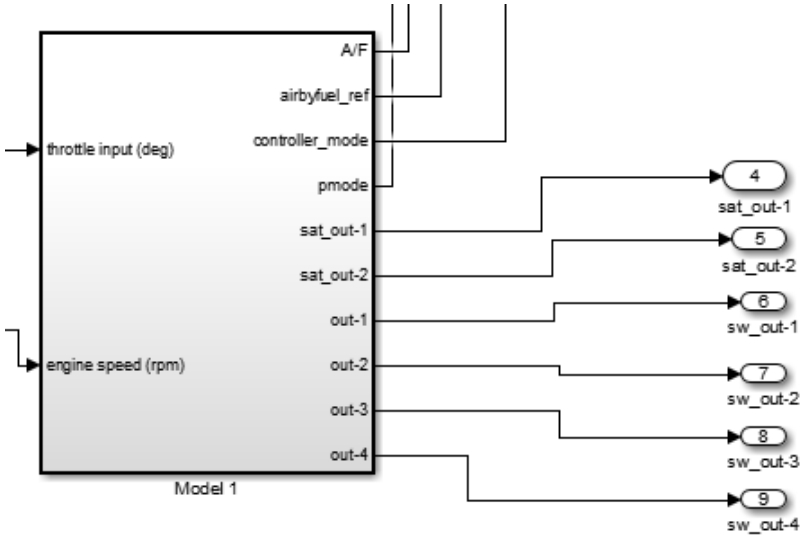

Figure 35 Auxiliary Output Ports Added at the Top Layer of the Instrumented Model

Black Box File Section	Figure Number	Description
1	36	State transition graph for switch_1
	37	State transition graph for switch_2
	38	State transition graph for saturation_1
	39	State transition graph for saturation_2
2	40	Simulation of instrumented model
3	41	Separation of location output ports of switch blocks
	42	Separation of location output ports of saturation blocks

Table 2 Mapping of Fig. 36 to Fig. 42 to the Corresponding Black Box File Section

The black box file generated by the model instrumentation framework is depicted in Fig. 36 to Fig. 42. The mapping of these figures to the corresponding sections in the block box file (Chapter 5, Section 5.5) is shown in Table 2.

The state transition graphs corresponding to the switch and saturation blocks are shown in Fig. 36 to Fig. 39. In particular, consider the Boolean value triggered switch block (see Fig. 32). The corresponding Boolean formula for the switch block is extracted and the DNF conversion is performed resulting in the state transition graph representation shown in Fig. 37. In this case, the Boolean function that triggers the switch block is ‘sw_out-3 – sw_out-2 < 0‘. This formula is captured with a value of ‘-1’ and ‘1’ in the matrix A of the guard specifying the transition from mode one to two of the switch block. Along the same lines the guards for the other blocks are specified.

```
guards(1,2).b = -0.5;
guards(2,1).b = 0.5;
guards(1,2).A = [0 0 0 0 0 -1 0 0];
guards(2,1).A = [0 0 0 0 0 1 0 0];
```

Figure 36 State Transition Guard Representation for Double Precision Input Value Triggered Switch_1 Block in Fig. 31 Which Capture the Equations 'sw_out-1 $\geq \mathbf{0.5}$' and ' sw_out-1 $\leq \mathbf{0.5}$'

```
guards(1,2).A{1,1} = [0 0 0 0 0 0 -1 1];
guards(1,2).b{1,1} = 0;
guards(2,1).A{1,1} = [0 0 0 0 0 0 1 -1];
guards(2,1).b{1,1} = 0;
```

Figure 37 State Transition Guard Representation of Switch_2 Block Shown in Fig. 32 Which Capture the Equations 'sw_out-3 - sw_out-2 $\leq \mathbf{0}$' and ' sw_out-2 - sw_out-2 $\leq \mathbf{0}$'

```
guards(1,2).A(1,:) = [0 0 0 1 0 0 0 0];
guards(1,2).A(2,:) = [0 0 0 -1 0 0 0 0];
guards(1,2).b(1,:) = Inf;
guards(1,2).b(2,:) = 0;
guards(2,1).A = [0 0 0 1 0 0 0 0];
guards(2,1).b = 0;
guards(1,3).A = [0 0 0 -1 0 0 0 0];
guards(1,3).b = -Inf;
guards(3,1).A = [0 0 0 1 0 0 0 0];
guards(3,1).b = 0;
guards(2,3).A = [0 0 0 -1 0 0 0 0];
guards(2,3).b = -Inf;
guards(3,2).A(1,:) = [0 0 0 1 0 0 0 0];
guards(3,2).A(2,:) = [0 0 0 -1 0 0 0 0];
guards(3,2).b(1,:) = Inf;
guards(3,2).b(2,:) = 0;
```

Figure 38 State Transition Guard Representation of Saturation_1 Block in Fig. 34 Which Capture the Equations '$0 \leq$ sat_out-1$\leq \infty$', 'sat_out-1 ≤ 0' and 'sat_out-1 $\geq \infty$'

```
guards(1,2).A(1,:) = [0 0 0 0 1 0 0 0];
guards(1,2).A(2,:) = [0 0 0 0 -1 0 0 0];
guards(1,2).b(1,:) = 1.66;
guards(1,2).b(2,:) = -0.13;
guards(2,1).A = [0 0 0 0 1 0 0 0];
guards(2,1).b = 0.13;
guards(1,3).A = [0 0 0 0 -1 0 0 0];
guards(1,3).b = -1.66;
guards(3,1).A = [0 0 0 0 1 0 0 0];
guards(3,1).b = 0.13;
guards(2,3).A = [0 0 0 0 -1 0 0 0];
guards(2,3).b = -1.66;
guards(3,2).A(1,:) = [0 0 0 0 1 0 0 0];
guards(3,2).A(2,:) = [0 0 0 0 -1 0 0 0];
guards(3,2).b(1,:) = 1.66;
guards(3,2).b(2,:) = -0.13;
```

Figure 39 State Transition Guard Representation of Saturation_2 Block in Fig. 34 Which Capture the Equations '$0 \leq$ sat_out-2$\leq \infty$', 'sat_out-2 ≤ 0' and 'sat_out-2 $\geq \infty$'

```
model = 'AbstractFuelControl_M1_ex_ouports_added';
warning off;
simopt = simget(model);
simopt = simset(simopt, SaveFormat ','Structure');
[T, XT, YTstruct] = sim(model,[0 simT],simopt,[TU U]);
for i = 1:9
          Temp = YTstruct.signals(i).values;
          YT = [YT, Temp];
end
```

Figure 40 Simulation of Instrumented Model

The second section of the black box file sets up the simulation of the instrumented model and captures the output values of the system (Fig. 40). It must be noted that as a result of the model instrumentation, the output values now contain information about the current status (mode) of the switch/saturation blocks and also the distance from a given mode of any of the blocks. The structure YT shown in the figure corresponds to the output trace of the systems which is essentially the set $Y_{\Sigma,}$ Y_A and Y_F.

```
LT_bool = YT(:,9:9);
LT_bool = LT_bool+1;
LT_float = YT(:,6:6);
switch_threshold_mat = 0.5;
switch_criteria_mat = 1;
szltf = size(LT_float);
for i = 1:szltf(2)
          for j = 1:szltf(1)
                    if switch_criteria_mat(i) == 1
                              if(LT_float (j,i)>= switch_threshold_mat(i))
                                        LT_float(j,i)  = 2;
                              else
                                        LT_float(j,i) = 1;
                              end
                    else
                              if(LT_float (j,i) > switch_threshold_mat(i))
                                        LT_float (j,i) = 2;
                              else
                                        LT_float(j,i) = 1;
                              end
                    end
          end
end
LT_switch = [LT_float LT_bool];
```

Figure 41 Compute the Mode History for Switch_1 and Switch_2 Blocks from the Output Trajectories

```
LT_sat = YT(:,4:5);
szltsb = size(LT_sat);
for i = 1:szltsb(2)
          for j = 1:szltsb(1)
                    if(LT_sat(j,i) <= sat_low_mat(i))
                              LT_sat(j,i) = 1;
                    elseif(LT_sat(j,i)> sat_low_mat(i)) &&
(LT_sat(j,i)< sat_high_mat(i))
                              LT_sat(j,i) = 2;
                    elseif(LT_sat(j,i)> sat_high_mat(i))
                              LT_sat(j,i) = 3;
                    end
          end
end
LT = [LT_switch LT_sat];
LT = double(LT);
YT(:,9:9) =  [];
end
```

Figure 42 Compute the Mode History for Saturation_1 and Saturation_2 Blocks from Output Trajectories

Finally, the third section computes the location information, Y_F from the output signals (YT) as shown in Fig. 41 and Fig. 42. The location values are obtained by indexing into the corresponding rows of the output signal matrix (YT in Fig. 40). The location information (LT in Fig. 40) forms the set Y_F which can be used to define the coverage metrics. These location values are removed from YT to form the set Y_Σ and Y_A used to compute the distance metric d_h.

6.1.3 Integration in S-TaLiRo for Mode Coverage

```
FALSIFIED at sample ,361!
  Running time of run 1: 3893.6316 sec
Stop
Elapsed time is 9383.799115 seconds.
Coverage information
List of all visited locations:
          2     1     1     1
          2     1     2     2
          1     1     2     2
          2     1     2     3
          1     1     2     3
          2     2     2     2
          2     2     2     3
          1     1     2     1

List of location predicates that are tested:
    []    [2]    []    [3]

List of location predicates that are visited:
    []    [2]    []    [3]

List of location predicates that are not visited:
List is empty
```

Figure 43 Coverage Analysis of a Sample Location of the Switch and Saturation Blocks Explained in Section 6.1.2

The model instrumentation package is integrated in S-TaLiRo [14] to completely automate the process of instrumentation and test generation for mode coverage. In this experiment setup, the coverage is performed to simultaneously reach location two of switch block shown in Fig. 32 and location three of saturation block named ‘saturation_2’ in Fig. 34. This requirement corresponds to activating the power mode subsystem and

reaching the upper limit of the corresponding saturation block. The MTL specification $\varphi = \neg F (P_{Des})$ is automatically generated where F is the temporal operator "eventually" and P_{Des} corresponds to the desired location to be covered. The specification in natural language means 'do not eventually reach P_{Des}'. The falsification of the specification will result in the specific mode coverage of the blocks.

The result of the experiment of one of the falsifying runs is shown if Fig. 43. The specification is falsified, i.e., the desired location is reached in 361 tests. Additionally, all the other locations that were reached before falsifying the specification are also reported as 'List of all visited locations'. For this experiment, each run of S-TaLiRo executes up to 1000 tests. If the specification is not falsified within 1000 tests then the test is considered to have failed. However, the smallest robustness value is reported to the user.

The robustness guided mode coverage is compared with the uniform random sampling approach. The comparison is performed in order to demonstrate that the instrumentation improves coverage. The robustness guided approach employs Simulated Annealing to guide the test generation process based on the robustness value. In uniform random sampling, each input sample for testing are independently chosen at random without being guided by the robustness value. The results are shown in Table 3.

	Uniform random sampling	Robustness Guided
Total Runs	30	30
No. of time mode was reached	20	30
Avg. tests for falsified cases	45	109

Table 3 Mode Coverage Comparison between Uniform Random Sampling and Robustness Guided Approach on the AbstarctFuelControl Model

It can be inferred from Table 2 that uniform random sampling approach falsifies the specification only twenty out of thirty times, i.e., the modes of the switch and saturation block could not be reached within 1000 tests. In comparison, the robustness guided approach is able to falsify the specification for all the thirty runs and hence reach the desired locations. The potential application of this mode coverage approach to verify system requirements is discussed in Section 6.2 using a challenging toy model.

6.2 Coverage Guided Falsification

In this section, a simple example is considered to illustrate the process of using model instrumentation to perform coverage guided falsification. The model used for the experiment is shown in Fig. 44 and it was provided by Toyota. The model consists of a switch block triggered by a Boolean network. The output of the Boolean network is determined by the comparison between the eight input blocks with the respective constant values. When the Boolean formula evaluates to 'TRUE', Signal B is passed to the output of the switch. Signal A is passed through when the formula evaluates to 'FALSE'. The Signals A and B correspond to the functions '$y = x^2 - 10$' and '$y = x^2$', respectively, using the input nine shown in the top right side of Fig. 44. The inputs one to eight take the values from the set [0,100] and the input nine takes the values from the set [-5,5].

Let an unsafe region be defined as one when the output of the switch is less than minus eight, i.e., 'Out1 < -8', i.e., the area below the red dotted line in Fig. 45. It can be seen from Fig. 45 that this unsafe region can be reached only when signal A (mode one of switch) is passed to the output of the switch. This is the case when the Boolean formula

evaluates to ‘FALSE’. For this to happen, the input signals should satisfy the following constraints:

In1, In3, In5, In7 $\geq$ 90 and In2, In4, In6, In8 $\leq$ 10

The probability for the above constraint is 10^{-8}, which is very low. Hence, it is practically impossible to reach the unsafe region by just random sampling. Falsification of the requirement requires that the system is first guided to reach mode one of the switch block and then eventually reach the unsafe region.

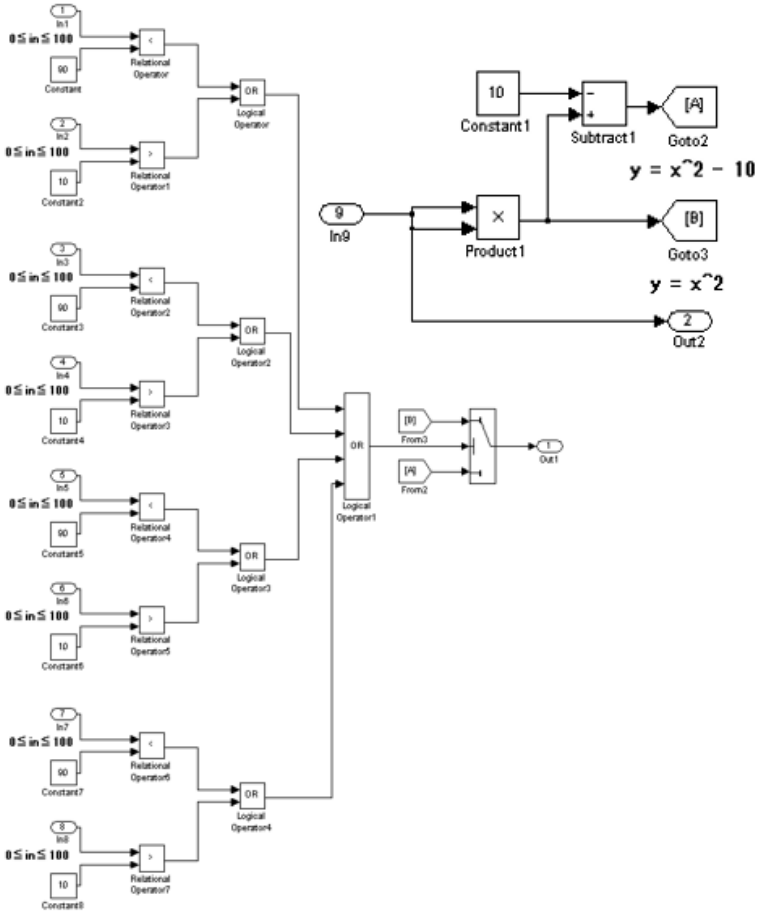

Figure 44 Example Model to Illustrate Coverage Guided Falsification (Provided by Toyota)

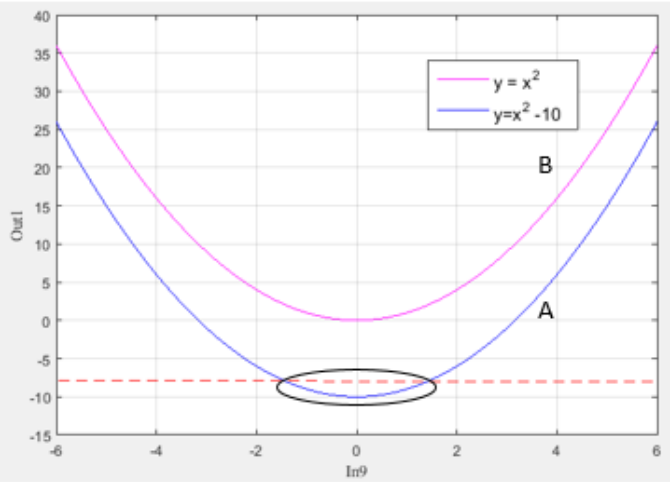

Figure 45 Trajectory of the Signal A and Signal B (Provided by Toyota)

The idea is illustrated by performing two sets of experiments on the model. In the first scenario, the falsification is performed without exposing the information related to the switch block. The falsification process looks at the model as a black box with nine inputs and an output with a requirement to be falsified. The algorithm tries to find the input values that will make the output reach the unsafe set.

In the second scenario, model instrumentation is performed to expose the state of the switch block. The compare blocks are instrumented and used to compute the distance metric d_h. Now the coverage metric is introduced in the falsification problem. The MTL formula is formulated as

$$\varphi' = \varphi \vee \neg F(pDes)$$

Here, φ corresponds to the original specification to be falsified (Out1 < -8) and the new atomic proposition $pDes$ corresponds to the desired combination of modes. In this example, $pDes$ corresponds to the mode one of the switch block. Hence, the falsification algorithm indirectly tries to push the system to go to the desired modes since

its robustness will be minimized as well. The falsification algorithm guides the system to mode one of the switch block (signal A) and, in turn, it falsifies the original requirement.

	Without Coverage	Coverage guided
Total Runs	25	25
No. of falsifications	0	21
Avg. tests for falsified cases	NA	540

Table 4 Falsification Results with and Without Coverage Guidance

Table 4 presents the results for falsification achieved with and without the coverage guided approach. Each run of the falsification algorithm constitutes a maximum of 1000 tests. It can be seen that without any coverage, S-TaLiRo fails to falsify in all the 25 runs whereas the coverage guided approach is able to falsify 21 out of 25 runs.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this work, a MATLAB based package for automatic analysis and instrumentation of industrial size Simulink models is presented. A hierarchical and a modular approach is proposed to extract the blocks with branching behavior and obtain their equivalent FSM representation. The framework is integrated in the robustness guided falsification tool S-TaLiRo [14], and it is utilized in order to perform mode coverage for CPS models. The robustness guided approach for mode coverage achieved better performance compared to the uniform random sampling approach. In addition, the potential for coverage guided falsification is illustrated on a challenging toy example.

However, the model instrumentation tool developed is still in its infancy. Currently, the tool provides only a basic framework for analyzing selected blocks. There are a few areas that can be explored in the future to further improve the effectiveness of the tool for CPS testing.

Some of the future extensions can be carried in the following areas:

- The automated analysis support can be extended to more blocks like state charts, Lookup tables, absolute value blocks etc. The challenge in the automatic analysis of Stateflow diagrams lies in the fact that analysis might be required to extract the guard transitions between the various states. Stateflow allows for arbitrary program expressions to be stated inside the Stateflow states and/or transitions. The current approach is to manually analyze them and obtain their equivalent state machine

representation. An alternative would be to impose certain syntactic restrictions on the allowed Stateflow charts in order to support automatic instrumentation.

- The current approach performs instrumentation of all the branching blocks identified in the model or indicated by the user. With an increasing number of blocks, there is exponential increase in the combinations for mode coverage. Some heuristics have to be explored to automatically (without the user intervention) disregard certain blocks from the instrumentation process. For example, one possible approach is to only consider nonlinear blocks in the controller section and neglect the blocks used to model the environment.
- The use of tools like Simulink Design Verifier (SDV) can be explored to analyze the components that require discrete time domain analysis. SDV performs well for open loop systems and fails for complex close loop dynamics. The trajectory generated by SDV for an open loop system can be used by S-TaLiRo for the test generation process of the closed loop systems.
- In [15], coverage guided falsification is discussed in detail. The approach is briefly illustrated on a toy example in Section 6.2. The approach needs to be extensively studied on standard benchmarks to establish the effectiveness of coverage guided falsification.

In addition to the above mentioned areas, changes in the interface can be done to improve the model instrumentation tool. Currently, the state transition graph information

is written in the blackbox file. This can be provided to the user in the form of a MATLAB class. The class termed as “blackbox” class in S-TaLiRo [14] has the following fields:

- Model_fcnptr – MATLAB function pointer to the blackbox model file.
- CLG – adjacency graph corresponding to each branching structure considered in the model instrumentation.
- GRD – The guard structure representing the transition relation from one state to another.

The ‘blackbox’ class can be used to initialize the ‘Guard’ and ‘CLG’ structures in advance before their execution in the blackbox file. This is particularly useful for models in which the guards do not change with time. So the repeated execution of the corresponding commands in blackbox file can be avoided for every simulation.

REFERENCES

[1] E. A. Lee and S.A. Seshia, “Introduction to embedded systems - A cyber-physical systems approach,” [Online] Edition 1.5, LeeSeshia.org, 2014.

[2] P. Tabuada, “Verification and control of Hybrid Systems: A Symbolic Approach,” Springer 2009.

[3] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk and J. Raman, “Analysis of Safety-Critical Computer Failures in Medical Devices,” in IEEE Security & Privacy 11.4 (2013): 14-26.

[4] X. Chen , E. Abraham and S. Sankaranarayanan, “Flow: An Analyzer for non linear hybrid systems,” in Computer Aided Verification, 2013.

[5] G. Frehse, C. L. Guernic, A. Donze, S. Cotton, R. Ray, O. Labeltel, R. Ripado, A. Girard, T. Dang and O. Maler, “SpaceEx: Scalable verification of hybrid systems,” in Computer Aided Verification, 2011.

[6] A. Platzer, “Logical Analysis of Hybrid Systems:Proving theorems for Complex Dynamics,” Springer-Heidelberg, 2010.

[7] J. B. Jeanin , K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt and E. Z. A. Platzer, “A formally verified hybrid systems for the next generation airborne collision avoidance system,” in International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pages 21-36. Springer, 2015.

[8] G. Fainekos, S. Sankaranarayanan, K. Ueda and H. Yazarel, “Verification of automotive control applications using s-taliro,” in American Control Conference, 2012.

[9] B. Hoxha, H. Abbas and G. Fainekos, “Using s-taliro on industrial size automotive models,” in Applied Verification for Continous and Hybrid Systems, 2014.

[10] X. Jin, A. Donze, J. Deshmukh and S. Seshia, “Mining requirements from closed loop control models,” in Hybrid Systems: Computation and Control. ACM Press, 2013.

[11] H. Abbas, G. E. Fainekos, S. Sankaranarayanan, F. Ivancic and A. Gupta, “Probabilistic temporal logic falsification of cyber-physical systems,” in ACM Transactions on Embedded Computing Systems (TECS), 12(s2), May 2103.

[12] Y. S. R. Annapureddy and G. E. Fainekos, “Ant Colonies for temporal logic falsification of hybrid systems,” in 36th Annual Conference of IEEE Industrial Electronics, pages 91-96, 2010.

[13] A. Donze, “Breach, a toolbox for verification and parameter synthesis of hybrid systems,” in Computer Aided Verification , pages 167-170. Springer, 2010.

[14] “TaLiRo Tools,” [Online]. Available: https://sites.google.com/a/asu.edu/s-taliro/s-taliro.

[15] A. Dokhanchi, A. Zutshi, R. T. Sriniva, S. Sankaranarayanan and G. Fainekos, “Requirements driven falsification with coverage metrics,” in Embedded Software (EMSOFT), 2015.

[16] S. Boyd and L. Vandenberghe, “Convex Optimizations,” Cambridge university press, 2004.

[17] A. K. Seda and P. Hitzler, “Generalised distance functions in the theory of computation,” in The Computer Journal, 53(4): bxm108443-464, 2008.

[18] G. Fainekos and G. J. Pappas, “Robustness of Temporal Logic specifications for continous-time signals,” in Theoretical Computer Science, 410(42):4262-4291, 2009.

[19] A. Fehnker and F. Ivančić, “Benchmarks for hybrid systems verification,” in International Workshop on Hybrid Systems: Computation and Control (pp. 326-341). Springer Berlin Heidelberg, 2004.

[20] R. Koymans, “Specifying real-time properties with metric temporal logic,” in Real-time systems, 2(4), 255-299, 1990.

[21] P. Ammann and J. Offutt, “Introduction to Software Testing,” Cambridge University Press, 2008.

[22] C. E. Tuncali, G. Fainekos and Y.-H. Lee, "Automatic Parallelization of Simulink Models for Multi-core Architectures," in 2015 IEEE 12th International Conference on Embedded Software and Systems (ICESS).

[23] S.J. Russell, P. Norvig, J. F. Canny, J. M. Malik, D.D. Edwards, “Artificial intelligence: a modern approach,” Vol. 2. Upper Saddle River: Prentice hall, 2003.

[24] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda and K. Butts, “Powertrain Control Verification Benchmark,” in the 17th international conference on Hybrid systems: Computation and Control, 2014.

[25] T. Dang and T. Nahhal, “Coverage-guided test generation for continuous and hybrid systems,” in Formal Methods in System Design, Volume 34, Issue 2, pp 183–213, 2009.

[26] F. Leitner and S. Leue, “Simulink Design Verifier vs. SPIN – a Comparative Case Study,” in Formal Methods for Industrial Critical Systems, 2008.

[27] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi and M. Hendriks, “UPPAAL 4.0,” in Third International Conference on the Quantitative Evaluation of Systems - (QEST'06), 2006.

[28] A. Zutshi, S. Sankaranarayanan and J. Deshmukh, "A trajectory splicing approach to concretizing counterexamples for hybrid systems," in 52nd IEEE Conference on Decision and Control, pages 3918 - 3925, 2013.

[29] J. Kapinski, J. Deshmukh, X. Jin, H. Ito and K. Butts, “Simulation-guided approaches for verification of automotive powertrain control systems,” in 2015 American Control Conference (ACC) (pp. 4086-4095). IEEE.

[30] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta and G.J. Pappas, “Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems,” in 2010 13th ACM international conference on Hybrid Systems: Computation and Control (pp. 211-220).