

# SANTA CLARA UNIVERSITY

Department of Computer Science and Engineering

Date: 8 May 2020

I HEREBY RECOMMEND THAT THE THESIS PREPARED  
UNDER MY SUPERVISION BY

**Jake Hedlund**

ENTITLED

**Prioritized Anomaly Catalog Generation Using Model-Based  
Reasoning**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF

MASTER OF SCIENCE IN  
COMPUTER SCIENCE AND ENGINEERING



Christopher Kitts, Thesis Advisor



Ahmed Amer, Thesis Reader

DocuSigned by:



96CE94A1A83A48A...

Nam Ling, Chairman of Department

Prioritized Anomaly Catalog Generation Using Model-Based  
Reasoning

By

Jake Hedlund

Submitted in Partial Fulfillment of the Requirements for the

Master of Science Degree in

Computer Science and Engineering in the

School of Engineering

Santa Clara University, 2020

Santa Clara, California

# Prioritized Anomaly Catalog Generation Using Model-Based Reasoning

Jake Hedlund

Department of Computer Engineering

Santa Clara University, 2020

## Abstract

Anomaly management—the detection, diagnosis, and resolution of anomalies in a system—is traditionally performed using experiential techniques which are quickly computed, but poorly structured. Newer model-based approaches are more systematic and higher performing but are computationally expensive, which is a particular challenge for execution in an operational environment. This paper builds on a novel system to pre-compute model-based anomaly symptoms to enable quick retrieval and diagnosis in operational settings. New additions to this system include a simplified model interface, anomaly likelihoods associated with each component, and easier interpretation of results. The implemented system has been used successfully to detect and diagnose anomalies in a baseline test circuit as well as in an operational satellite monitoring network. Results show that this approach is promising; with a thorough model, the diagnosis and resolution processes of anomaly management could be greatly improved for more complex remote systems such as university-operated nanosatellites and field robotic vehicles.

# Table of Contents

Table of Contents .....	iv
List of Figures .....	vi
List of Tables .....	vii
1. Introduction .....	1
1.1. Impact of Anomalies .....	1
1.2. Anomaly Management .....	2
1.3. Reasoning Approaches .....	3
1.3.1. Experiential systems .....	4
1.3.2. Model-based reasoning systems .....	5
1.3.3. Reasoning summary .....	7
1.4. Project Goals .....	7
2. System Architecture .....	9
2.1. Overview of Model-Based Reasoning .....	9
2.1.1. Types of anomalies .....	10
2.1.2. Full-adder system .....	12
2.2. Applying Model-Based Reasoning .....	12
2.2.1. Full-adder details .....	13
2.2.2. Anomaly example .....	14
2.3. Hybrid Reasoning System .....	15
2.3.1. Full-adder with anomalies .....	16
2.3.2. Anomaly representations .....	17
2.4. New Contributions .....	17
2.4.1. Adding anomaly confidence data .....	18
2.4.2. Hiding anomalies for some behaviors .....	19
2.5. The Anomaly Management Engine .....	20
2.5.1. Generating anomaly permutations .....	20
2.5.2. Efficiency improvements .....	22
2.6. The Anomaly Catalog .....	24
2.6.1. Anomaly detection .....	24
2.6.2. Anomaly diagnosis .....	25
2.7. Implementing Enhanced MBR .....	26
2.7.1. Modification of existing components .....	27
2.7.2. Passing data between blocks .....	27
2.7.3. The Anomaly Generator block .....	28
2.7.4. Anomaly input permutations block .....	29
2.7.5. Analyzing the anomaly catalog .....	30

2.8.	Multiple Anomalies .....	32
2.9.	Meaning of likelihoods .....	33
2.10.	Chapter Summary .....	35
3.	Applications and Results .....	37
3.1.	Full-Adder Results .....	37
3.1.1.	Improving simulation time .....	38
3.1.2.	Analysis results .....	38
3.1.3.	Querying with wildcards .....	40
3.2.	The Beacon Network .....	41
3.2.1.	Network architecture .....	42
3.2.2.	Motivations .....	42
3.2.3.	Modeling the Beacon Network .....	44
3.2.4.	Subsystems .....	46
3.3.	Testing the Model .....	47
3.3.1.	The O/OREOS experiment .....	47
3.3.2.	Experiment results .....	48
3.3.3.	Observation points .....	50
3.4.	Modeling limitations .....	51
3.4.1.	Feedback loops in certain simulation modes .....	51
3.4.2.	Performance inconsistencies .....	52
3.5.	Chapter summary .....	53
4.	Conclusion .....	54
4.1.	Future Work .....	55
4.2.	Current Utilization .....	57
4.3.	Final Thoughts .....	57
	References .....	59
	Appendix A – Supplementary Tables and Diagrams .....	61
	Appendix B – Code Samples .....	66

## List of Figures

Figure 1.1: Check engine lights on a car only give a clue to the underlying anomaly .....	4
Figure 1.2: Experiential anomaly management approach.....	5
Figure 1.3: Anomaly Management process steps.....	6
Figure 2.1: A logical AND gate functioning as expected. ....	9
Figure 2.2: The three types of anomalies using a logical AND gate as an example. ....	11
Figure 2.3: The basic full-adder circuit implemented in Simulink. ....	12
Figure 2.4: The full-adder circuit, implemented in Simulink using Matlab function blocks..	13
Figure 2.5: The two phases of the anomaly management process to diagnose and resolve anomalies.....	15
Figure 2.6: The enhanced full-adder circuit.....	19
Figure 2.7: Matlab code showing the nominal behavior of a logical AND gate. ....	20
Figure 2.8: Two approaches to simulation. ....	23
Figure 2.9: The anomaly-generator block injecting an anomaly into the XOR2 component.	28
Figure 2.10: Detail of the InputGenerator subsystem block. ....	29
Figure 3.1: Sample output after running the anomaly diagnosis script for the full-adder circuit. ....	39
Figure 3.2: Sample output with wildcards. ....	40
Figure 3.3: Beacon monitoring concept diagram. ....	41
Figure 3.4: A simplified block diagram of the beacon network.....	43
Figure 3.5: A very simple block diagram of a satellite. ....	44
Figure 3.6: The Simulink model of the Beacon Network. ....	45
Figure 3.7: The Simulink model of a single beacon node. ....	48
Figure 3.8: Results of filtering the Beacon Network anomaly catalog. ....	50
Figure 3.9: Solving the loopback issue in each beacon station. ....	52

## List of Tables

Table 2.1: The truth table for a logical AND gate. ....	10
Table 2.2: The truth table for the full-adder circuit.....	14
Table 2.3: Four rows of the anomaly permutations for the full-adder. ....	21
Table 2.4: A sample of the catalog generated by the anomaly management system showing four rows of the 448 unique permutations. ....	26
Table 2.5: The filtered output of the Catalog after searching for observed output. ....	31
Table 3.1: A small section of the catalog generated by the anomaly management system. ....	37

# **1. Introduction**

Modern engineering systems are incredibly complex entities and often require very experienced engineers and technicians to build and maintain them. When things go wrong, it can be time-consuming and costly to find and fix the issue. If, instead, there were intelligent and automated processes in place to detect and resolve problems, significant savings could be achieved related to maintenance and downtime costs. However, building such a system to monitor another system is not a trivial task; it is expensive and takes very knowledgeable engineers a significant amount of time to develop and validate such a challenging process. It is also very computationally expensive to systematically detect and diagnose anomalies in a given system due to its complexity.

## **1.1. Impact of Anomalies**

The cost spent on monitoring remote systems, especially in the aerospace industry, is extremely high. NASA employs dozens of people to monitor the health of the International Space Station (ISS) 24 hours a day, 365 days a year [1]. In the case that something goes wrong, response time is critical; any delay could cost the lives of the astronauts on board. While there are undoubtedly many automated systems to assist in the detection of errors, this is often where the automation stops – even NASA is still reliant on engineers to generate possible solutions to resolve the problem. Taking the detection process a step further, an automated process to suggest possible resolutions before the engineers even knew there was a problem would greatly accelerate the time needed to diagnose the anomaly.

Anomalies are also a constant source of disasters. In 1996, an Ariane 5 rocket self-destructed 30 seconds after launch, destroying its satellite payload. Despite numerous safeguards and checks that were in place, the flight control software systems failed to account for certain changes in the trajectory. This resulted in the rocket veering off-course causing an anomaly detection mechanism to trigger the self-destruct sequence. This was one of the most costly software bugs in history, but it could have been easily detected prior to the disaster if the systems onboard the rocket had been designed with more robust



error detection. The disaster resulted in a loss of more than \$320 million after \$7 billion was spent on its development, and it was an embarrassment for the European Space Agency [2].

Despite 18 years of progress, the space industry is still fighting error conditions in rockets. In 2014, a commercial rocket exploded just after launch, costing millions of dollars in damage and lost scientific experiments [3]. There are many, many other cases of anomalous conditions causing rocket or spacecraft failure; these two were chosen to emphasize the importance of having robust anomaly management systems onboard.

## **1.2. Anomaly Management**

As discussed above, millions of dollars are spent diagnosing and resolving anomalies in engineering systems. This emphasizes the importance of utilizing a reliable and efficient technique to assist human operators in the event of an anomaly. Anomaly management systems attempt to do this by creating structured methods for detecting and potentially resolving anomalies.

Formally, an anomaly is defined as an unexpected condition that occurs in a functional engineering system. Current theory recognizes three classes of anomalies: faults, hazards, and misconfigurations [4]. A fault is defined as a condition within a component that prevents it from performing as expected (such as an integrated-circuit with an internal structural disconnect); a hazard occurs when a component is utilized outside its defined operating range (such as the ambient temperature being too high given the component's specifications); and a misconfiguration occurs as a result of the settings to a component being incorrectly given (such as the radio frequency being wrong). These explicit definitions enable an anomaly management system to be more precise when identifying possible reasons that a system is behaving abnormally.

When things go wrong in a functional engineering system, it is absolutely critical that the problem is detected and fixed within a reasonable amount of time. In high pressure situations, sometimes the only approach is the natural one: let the engineers who know the system best diagnose and attempt to resolve the issue. This is both time-consuming and expensive; companies that could be inventing and innovating are instead spending

valuable time maintaining and fixing prior projects. An anomaly management system attempts to streamline this process by automating the tasks as much as possible [5].

Anomaly management systems typically implement three distinct steps of analysis. First, a symptom is detected, indicating that an anomalous condition exists. Next, possible reasons to explain the anomalies are generated in the diagnosis phase [6]. Finally, possible solutions are tested to attempt to resolve the problem. This last step may be automated if possible, or may necessitate technician intervention after the anomaly management system reports its findings. These processes are the same that are followed whether the anomaly management system is automated or implemented by a human team. An ideal anomaly management system eliminates the need for human intervention entirely; the next best approach is to assist the humans so that they can do their jobs more efficiently, utilizing fewer resources and therefore minimizing financial and time costs.

### **1.3. Reasoning Approaches**

The two main approaches to anomaly management are “experiential” and “model-based.” The experiential approach relies on the experience and knowledge of the experts who built the system; if an anomaly is detected, the experts analyze the problem and possible solutions are proposed. The model-based reasoning approach involves constructing a model or simulation of the system-under-test in order to detect possible anomalies and will be discussed in great detail throughout this paper.

For example, almost all modern cars contain a network of sensors that report their states to the main computer. When an anomaly is detected, the car notifies its operator in the form of a status light on the dashboard. A mechanic can then further diagnose the issue by communicating with the computer directly and reading the error codes. This is a form of the experiential approach, and depending on the experience of the mechanic, the anomaly will be fixed much more quickly than without an accurate error code. The mechanic follows the basic approach of experiential reasoning: he or she will attempt to diagnose and resolve the problem by using his or her own experience as a knowledgebase of problems and solutions; in some cases, the diagnostic system may also propose potential steps based on a database of prior experience.

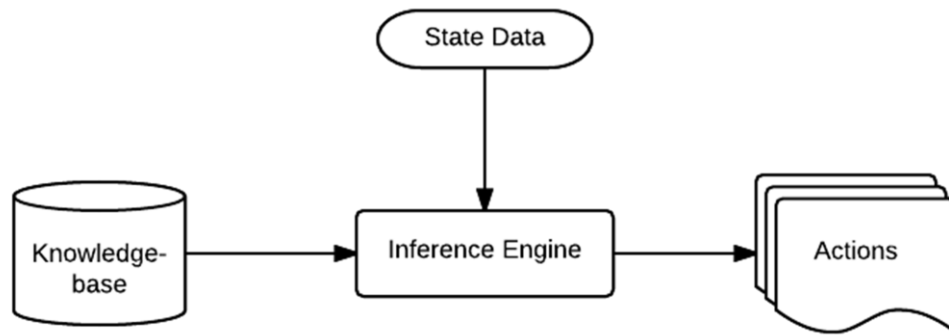


Figure 1.1: Check engine lights on a car only give a clue to the underlying anomaly [13].

For aerospace systems, the experiential process may be far more involved and time-consuming. For satellite operations centers, the process of diagnosing a symptom can take days or weeks since experts may have to be brought in from around the world to collaboratively evaluate the system. Furthermore, potential resolutions must often be exhaustively checked, simulated, and verified on hardware test platforms prior to being implemented (Kitts, personal communication).

### 1.3.1. Experiential systems

In an effort to streamline the experiential process, an automated “expert system” can be developed. This process attempts to consolidate all the knowledge and experience possessed by the experts and compiles it into a database. This knowledgebase is essentially a series of production rules in the form of “if-then” statements which the expert system uses to automatically implement or suggest solutions when problems occur [7]. While this method is very fast in the lookup phase, it is very time-consuming and error-prone to develop a robust database of all the knowledge possessed by the engineers. Furthermore, since this information is derived from their expertise, it is often the case that the true root cause is lost when the system diagnoses the issue. In addition, the knowledgebase is only applicable to situations that have been experienced; new situations and causes are typically unaddressed.



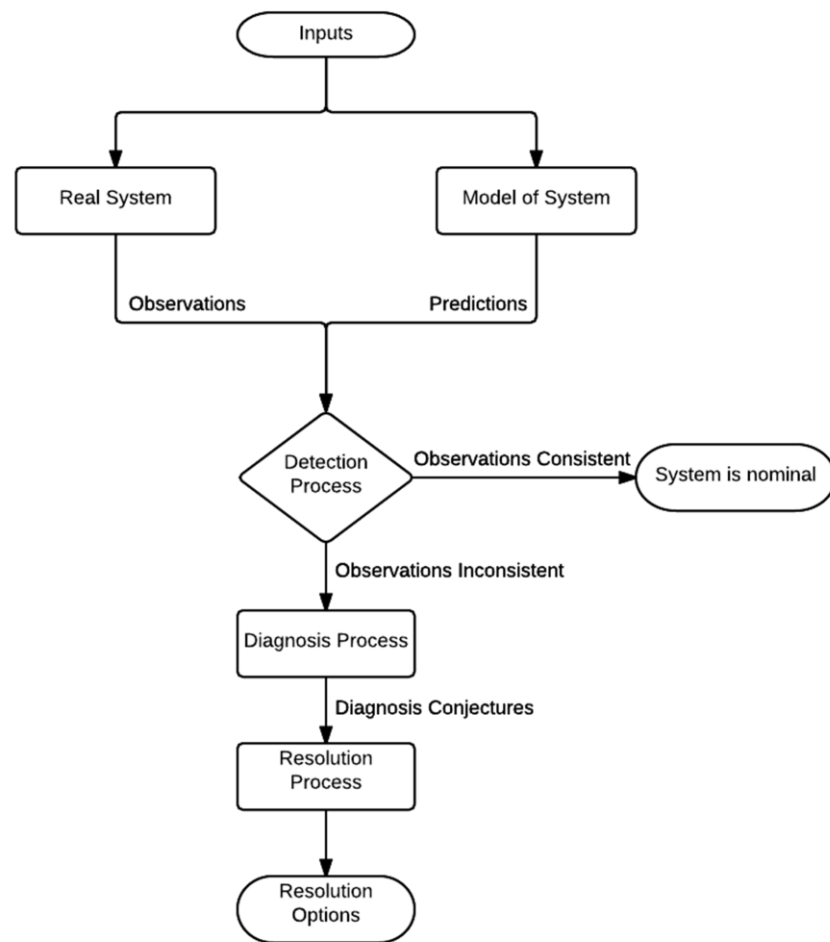
**Figure 1.2: Experiential anomaly management approach.** The inference engine compares the provided state data with possible scenarios in the knowledge-base to generate a list of potential resolutions [12].

A high-level representation of the experiential approach is shown in Figure 1.2, which illustrates how the state of the system is processed along with the compiled knowledgebase of information to create a list of proposed actions. To expand on the car mechanic example, an expert system would consist of a compiled list of all known anomalies, their symptoms, and their solutions as experienced by the mechanic. When analyzing a car, the symptoms of the anomaly (error code) would be provided to the expert reasoning system and a list of possible causes and solutions would be generated. Again, consulting the knowledgebase is efficient because it is simply a lookup table, but is limited to previously experienced – and accurately captured – scenarios.

### 1.3.2. Model-based reasoning systems

The more sophisticated approach, model-based reasoning (MBR), uses a functional model of the operational system to simulate its nominal and deduce its off-nominal behavior. The model is simulated alongside the real system, then the outputs are compared; if the results are inconsistent, an anomaly exists and must be diagnosed and resolved. Figure 1.3 shows an overview of this method.

When an anomaly is detected in the real system in the form of a symptom, the constraints on the model are relaxed until the output of the model matches the output of the system, which indicates a possible cause for the anomaly. In other words, anomalies are introduced into the simulation until its output matches the real system’s anomalous



**Figure 1.3: Anomaly Management process steps.** The model is simulated alongside the real system; outputs are compared and sent on to the subsequent processes. [4]

output. This is essentially a brute-force solution – all possible errors that could cause the observed symptoms are generated during the simulation phase. With a sufficiently detailed model, the simulation engine is able to generate every situation in which the anomaly occurs. Furthermore, with this approach it is fairly straightforward to change the model and re-run the simulation if something is changed in the real system. While a variety of extensions exist to avoid a true brute-force analysis, these techniques were not used as part of this thesis.

The simulation portion of the MBR process essentially generates the knowledgebase of the experiential approach – the simulation model embodies how the system is designed

and therefore can predict what the outputs look like when things go wrong. Instead of relying on expert engineers to create every possible condition which produces anomalous output, the simulation engine uses a model of the system to generate possible anomaly states systematically. The primary disadvantage of the MBR approach is that simulating a complex system in order to discover all potential anomalies is very computationally expensive and therefore slow, which may make it impractical to utilize in a time-critical operational setting.

### **1.3.3. Reasoning summary**

To review, there are two broad categories of anomaly management: the experiential approach, and the model-based reasoning (MBR) technique. The experiential approach is often disorganized and cumbersome to use, although it generally is very fast at diagnosing issues. The MBR approach attempts to solve some of the drawbacks of the experiential approach by organizing the system into a working model which can be simulated alongside the real system. The primary advantage this brings is that all possible outputs can be generated when necessary instead of relying on a narrow subset of captured, experienced cases that are hard-coded into the expert-reasoning system. Because the model-based process is deliberative, it can be very time-consuming to diagnose an anomaly compared to an expert system that has prior information on the same anomaly. Ultimately, a hybrid approach combining the benefits of these two approaches is of potential interest.

## **1.4. Project Goals**

The goal of this project is to enhance the typical model-based reasoning approach to anomaly management. In the first phase of the project, completed by a previous student, a hybrid approach was established which generated a catalog of anomalies and their potential causes. The system uses a model-based reasoning approach to pre-compute an exhaustive catalog of anomalies and associated symptoms prior to operation. Next, in the operational setting, improper behavior observed in the real system is used to search the catalog for consistent diagnoses, thereby dramatically improving response time when anomalies are detected.

The work reported in this thesis expands on the previous work in several ways. First, confidences were added in order to rank the confidence of each calculated anomaly. Second, the computation time was greatly reduced by better leveraging Matlab and Simulink's built-in functionality. Third, the augmented system was evaluated using both a baseline verification system as well as a real-world satellite monitoring network. Though the concepts explored in this paper are preliminary, the results are promising. We hope that the techniques presented can be extended and applied by future students to other, more elaborate systems to aid in the operation of their engineering projects.

## 2. System Architecture

In this chapter, the details of MBR are illustrated using several examples starting with basic Boolean logic gates. Next, the enhanced MBR approach is introduced and applied to a full-adder system as a proof-of-concept. Finally, implementation details of the novel system are examined and the benefits of using such a system are demonstrated.

### 2.1. Overview of Model-Based Reasoning

Model-based reasoning systematically detects, diagnoses, and resolves anomalies that occur in a functional engineering system. Broadly defined, a system is a collection of interconnected components. Inputs to the system result in signals that propagate through the system, ultimately producing output data or telemetry. The input configuration and the idealized model determine the correctness of the telemetry returned. The MBR framework used in this thesis is based on work by Dr. Kitts as seen in [4], which provides a formal review of its theory. This research simplifies the former implementation by utilizing Matlab's Simulink software as the MBR simulation engine among other modifications. The applicable theory will be defined in this chapter utilizing basic logic gates as the base components.

A component consists of several elements: inputs and outputs, a behavior constraint that specifies the value of an output as a function of inputs, and operating conditions that must be satisfied for the behavior statement to hold. For example, a typical logical AND gate, as shown in Figure 2.1, is defined as having two or more inputs and exactly one output; its behavior statement dictates that its nominal output,  $F$ , is the logical AND of its

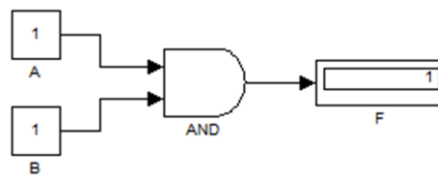


Figure 2.1: A logical AND gate functioning as expected.



**Table 2.1: The truth table for a logical AND gate.**

<b>A</b>	<b>B</b>	<b>F</b>
0	0	0
0	1	0
1	0	0
1	1	1

input ports, A and B. This is expressed as  $F = A \text{ AND } B$ . However, this behavior may only be prescribed when all operating constraints are met. One such constraint may require the component's temperature to be within a specified operating range. All possible inputs and outputs for the AND gate can be represented in a truth table as seen in Table 2.1.

### **2.1.1. Types of anomalies**

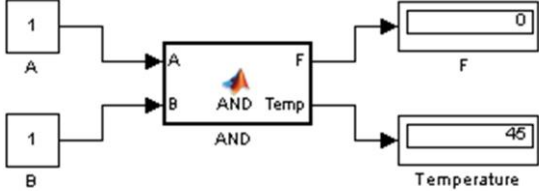
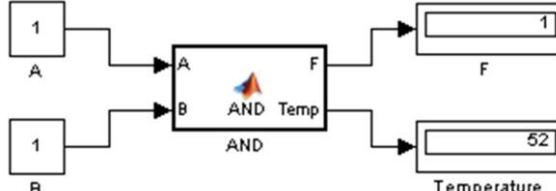
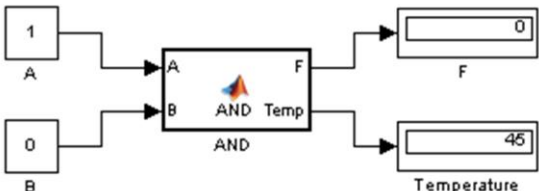
Any component in a system can experience an anomaly. Extended MBR theory defines three such cases: faults, hazards, and misconfigurations. Each of these manifests itself in a slightly different way as will be demonstrated in the following examples.

A fault is defined as the output of a given component being observed as incorrect for a given set of known inputs with the assumption that all operational conditions are satisfied. In the case of an AND gate, a fault would be detected if the inputs to the gate were both '1's but a '0' was observed on the output as shown in Figure 2.2(a). In essence, the component has failed and the output cannot be relied on until the fault is resolved; this may mean replacing the component with a new, working one. Furthermore, a fault may not be observed directly; in a multi-component system, only the overall inputs and outputs are known. The challenge is to narrow down the possibly anomalies through a reasoning process given this constraint.

A hazard anomaly is less intuitive: it occurs if a state violates one of the component's nominal operating conditions. For example, if the component's specification states that its operating range is 0 °C to 50 °C and the observed temperature is outside this range, a hazard anomaly would be reported even if the output was correct according to our observations. Note that another component (e.g. a temperature sensor) may be needed to

collect the data to determine this state. Although the output may still be correct, the system would still be in an anomalous state thus requiring further action to diagnose and resolve the anomaly.

The third type of anomaly, a misconfiguration, occurs if the inputs to the component or system are inconsistent with what has been specified. For example, in the case where a '0' is observed on the output, it could be that the gate has a fault (Figure 2.2(a)), or it could be that one of the inputs was configured with a '0' instead of the expected '1' as seen in Figure 2.2(c). This is an example of two distinct anomalies causing identical symptoms; the

Commanded inputs	Simulink model of an AND gate Actual inputs                      Observed Outputs	Anomaly type
A = 1 B = 1	 <p>(a)</p>	Fault
A = 1 B = 1	 <p>(b)</p>	Hazard
A = 1 B = 1	 <p>(c)</p>	Misconfiguration

**Figure 2.2: The three types of anomalies using a logical AND gate as an example.**(a) A fault in the component caused the output to read '0' when it should be '1'; (b) a hazard condition is present in the component (50°C temperature constraint violated); (c) a misconfiguration causes the output to read '0' when it should be '1'.

inclusion of anomaly likelihoods is largely motivated by the fact that multiple anomalies may result in identical symptoms. The enhanced MBR approach presented in this research differentiates between the two cases and can report each diagnosis with a different ranking.

### 2.1.2. Full-adder system

In order to demonstrate the application of MBR to a system, the full-adder circuit will be used as an example of a fully-functional engineering system. As can be seen in Figure 2.3, the full-adder consists of three system-level inputs, five interconnected components,

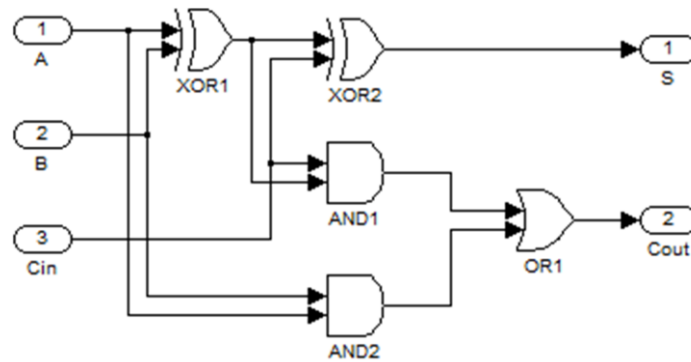


Figure 2.3: The basic full-adder circuit implemented in Simulink.

and two system-level outputs. The first XOR gate's output is connected to the inputs of the second XOR gate as well as the first AND gate; similar connections are made for the other gates. In the next sections, the full-adder will be further explored in the context of applying MBR to a system, and will be used as the prototypical engineering system to which the enhanced MBR approach will be applied as a proof-of-concept.

## 2.2. Applying Model-Based Reasoning

Before applying MBR to a complex system, the research herein began with a simplified model that serves as a proof-of-concept for the anomaly management system. For this, the binary full-adder was chosen. Matlab's Simulink software was used to model the circuit as seen in Figure 2.4. Logically identical to Figure 2.3, this is the basic computational system that was enhanced with confidence-based model-based reasoning. The model shown represents a real full-adder circuit that may be created in hardware using individual logic

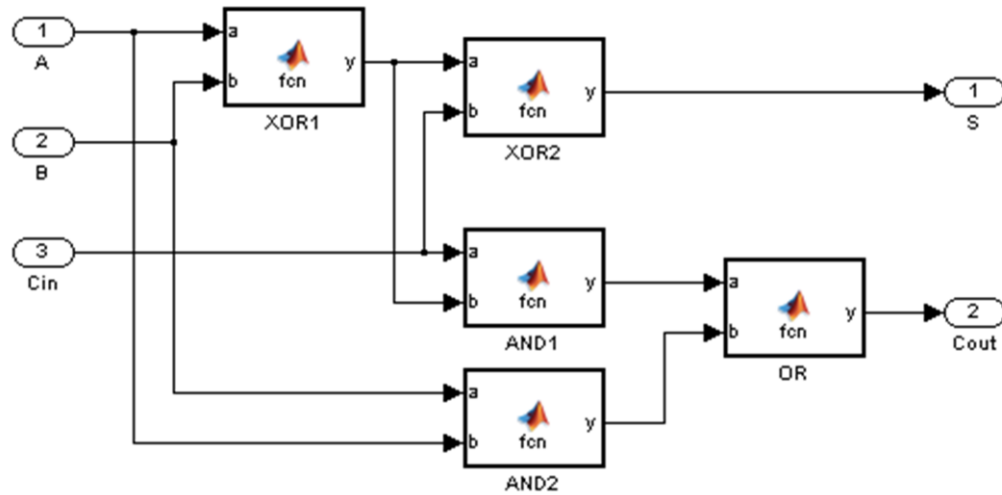


Figure 2.4: The full-adder circuit, implemented in Simulink using Matlab function blocks.

gates, each of which has its own characteristics, behaviors, and possible defects. The behavior functions, which map the block inputs to each gate's output, are contained inside each block and are implemented using the powerful Matlab scripting language. In this way, the system is very flexible, and the block definitions can be as simple or complex as desired, supplementing the standard logic-gate behavior.

### 2.2.1. Full-adder details

The full-adder contains three system-level inputs, each of which can take a binary value. Applying a logical '1' value to both the A and B inputs and a '0' to the  $C_{in}$  input, the expected result will yield a '0' on the Sum output and a '1' on the  $C_{out}$  output. For simplicity, the input is described as an array of three values:  $[1, 1, 0]$  corresponding to the three inputs,  $[A, B, C_{in}]$ . The nominal output associated with this particular input can then be written as  $[0, 1]$  corresponding to  $[S, C_{out}]$ . Since there are three binary inputs, there are a total of  $2^3 = 8$  unique input vectors which map to  $2^2 = 4$  unique output states. These can be concisely written in a truth table as seen in Table 2.2. When applying MBR to the full-adder, the nominal state is represented by this table; thus we can compare the output of the simulation to the corresponding output in the table to determine if an anomaly has occurred. This is the first step of anomaly management, detection.

**Table 2.2: The truth table for the full-adder circuit.**

Input			Output	
A	B	C <sub>in</sub>	S	C <sub>out</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

### 2.2.2. Anomaly example

In order to determine if an anomaly has occurred, an observer must compare the expected output of a system to the suspected anomalous output. For example, if we notice that our real full-adder circuit is outputting the values  $[S, C_{out}] = [1, 1]$  for the inputs  $[A, B, C_{in}] = [1, 1, 0]$ , then we can conclude that an anomaly has occurred somewhere in the system since the nominal output for this input is  $[0, 1]$ , as seen in the seventh row of inputs in Table 2.2. We can then proceed with the diagnosis processes from Figure 1.3 by simulating the model with different component constraints relaxed in order to discover the possible components that could have caused this failure. In this case, by examining Figure 2.3, we are able to determine that either XOR1 or XOR2 has likely faulted or is in a hazard condition, causing the incorrect 'S' output. Another possibility is that the input from  $C_{in}$  is misconfigured, and is using the value '1' instead of the expected '0'. In a classic MBR approach, all three of these possibilities would be presented as equally likely. Also note that for this case, we are only considering single anomalies, that is, only one anomaly existing in the system at any given time. Multiple simultaneous anomalies can occur and are examined in Section 2.8. The enhanced approach presented in this paper attempts to rank the anomalies in a useful order based on the individual likelihoods of each potential anomaly. This example demonstrates the principles of model-based reasoning; the specifics will be examined in the subsequent sections.

## 2.3. Hybrid Reasoning System

As discussed in Chapter 1, MBR simulation is typically slow due to its computational complexity, while using a rule-based expert system is very fast for looking up previously-encountered anomalous conditions. In an effort to leverage both the speed of the expert-reasoning system and the systematic nature of the model-based approach, the enhanced system presented pre-computes possible anomalous states and saves them to a catalog. This catalog then acts as a lookup table that the anomaly management engine uses in time-sensitive scenarios when anomalies must be diagnosed and resolved as quickly as possible. This two-phase process is shown in Figure 2.5. Phase 1 is pre-realtime operations during which the model is created/verified and then executed for every possible combination of

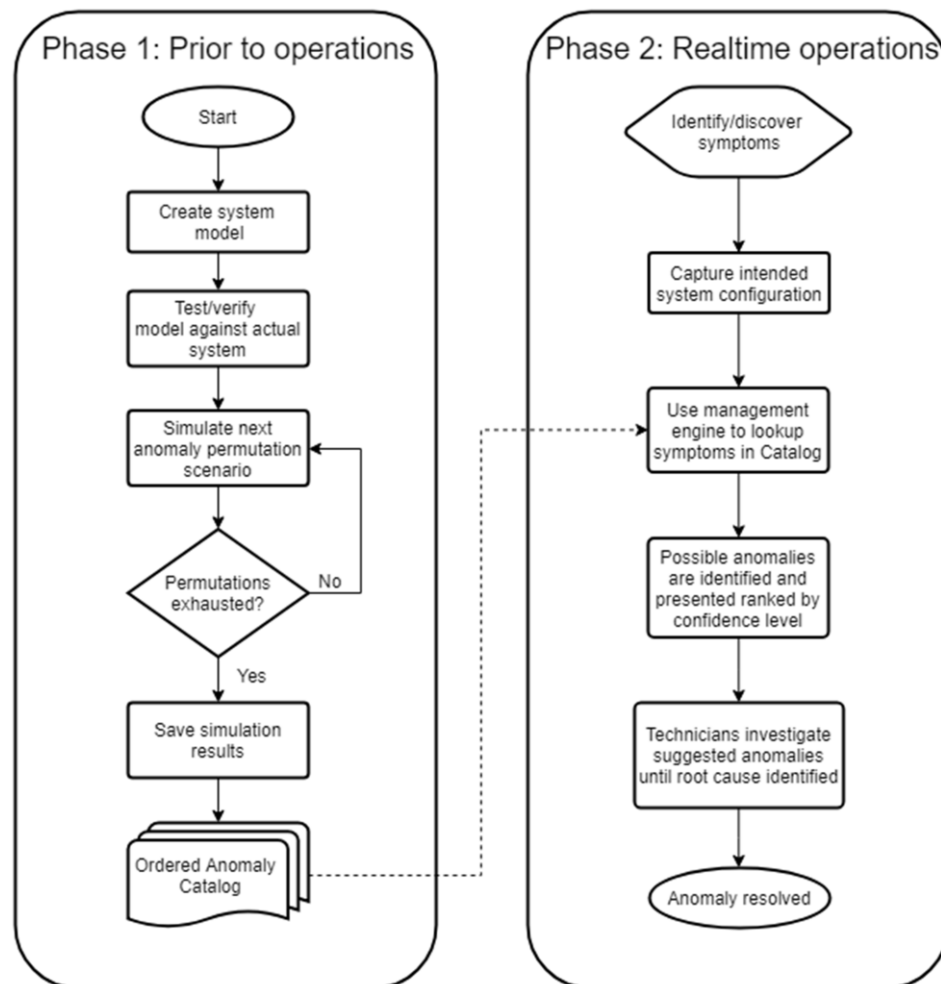


Figure 2.5: The two phases of the anomaly management process to diagnose and resolve anomalies.

input configuration and anomaly case (or at least as many as can be simulated in the time available), with the results of each scenario stored in the Anomaly Catalog. Then, during realtime operations when timing is critical, a symptom identified by the operator along with the current system configuration is used to search the catalog and identify all consistent anomaly scenarios.<sup>1</sup> A sample result set is shown and discussed in greater detail in the following sections.

### **2.3.1. Full-adder with anomalies**

Expanding on the full-adder example from the previous section, we can add information about possible anomalies to the system. By computing the output of the system for each anomalous case, we generate a database<sup>2</sup> (the anomaly catalog) of system states and outputs to be used in the diagnosis phase of the process. For example, one such case may be that the XOR1 gate has faulted. Simulating this scenario for the input  $[A, B, C_{in}] = [1, 1, 0]$  results in the output  $[S, C_{out}] = [0, 1]$  in the nominal case, but with the additional knowledge that XOR1 is in a faulted state, we can determine that the output of XOR2 and AND1 may also be affected by this fault thus potentially changing the value of the 'S' output. In the simulation engine, an anomalous state is represented as NaN, to indicate that the output value has been affected by an anomaly and thus can take on any possible value. When building the catalog, all possible scenarios can potentially be generated, although the computational time may be extreme for very complex systems. For modest systems, there will likely be more than enough time to evaluate all 1-, 2-, and 3-anomaly cases given that there will be weeks, months, or even years to prepare the catalog before a field mission. In the diagnosis phase, the catalog will be consulted to determine which anomalous condition(s) contributed to the observed output.

---

<sup>1</sup> In follow-on work, there is no longer the need for the operator to identify a symptom. The configuration and observed data is used to identify consistent state scenarios, anomalous or not.

<sup>2</sup> In this case the term "database" is used loosely. The catalog is currently left as a Matlab matrix variable inside the Matlab workspace; this variable can be saved to and loaded from a file using Matlab's user interface. Integrating an RDBMS such as MySQL has been left as future work.

### 2.3.2. Anomaly representations

During the simulation of a model, flags are set to indicate the current state of each component, either a fault, hazard, or valid state. If a given block is in an anomalous state, its output will be a special value representing the type of anomaly being simulated. The special value indicating a **fault** has occurred is '`Inf`'; the **hazard** case is indicated by '`NaN`'. Both values are built-in Matlab numerals representing infinity and not-a-number respectively. These values should not occur otherwise during normal use of the system.

In the fault anomaly scenario, the `Inf` value represents an invalid value on the output of the component. In other words, the component is known to have failed in a way such that its output is not correct, or there is no output at all. Conversely, in a hazard anomaly (`NaN`), the output is not known to be invalid – just that the component is in a state which might produce incorrect data. In both cases, the output continues to the next components' inputs. This gives the subsequent components the capability to act on known anomalous inputs; however in most cases this value will simply pass through each component and eventually be observed on the system output(s). The output of the system is then used as part of the anomaly catalog as discussed in detail in subsequent sections.

### 2.4. New Contributions

In an operational system, not all anomalies have the same likelihood of occurrence. For example, a car's brakes may have a higher chance of failure than the car's engine. In the full-adder, the AND gates may have different observed failure rates than the OR gates because of the manufacturing process or proximity to hot components, among other possibilities. Most MBR anomaly management approaches do not incorporate these likelihoods into the model; there are usually only two states, nominal and faulted. This enhancement is implemented in the model by adding a confidence property to each modeling constraint to represent the likelihood of each specific anomaly occurring for that component. This information is then used to rank possible diagnoses in order of likelihood of occurrence.



### 2.4.1. Adding anomaly confidence data

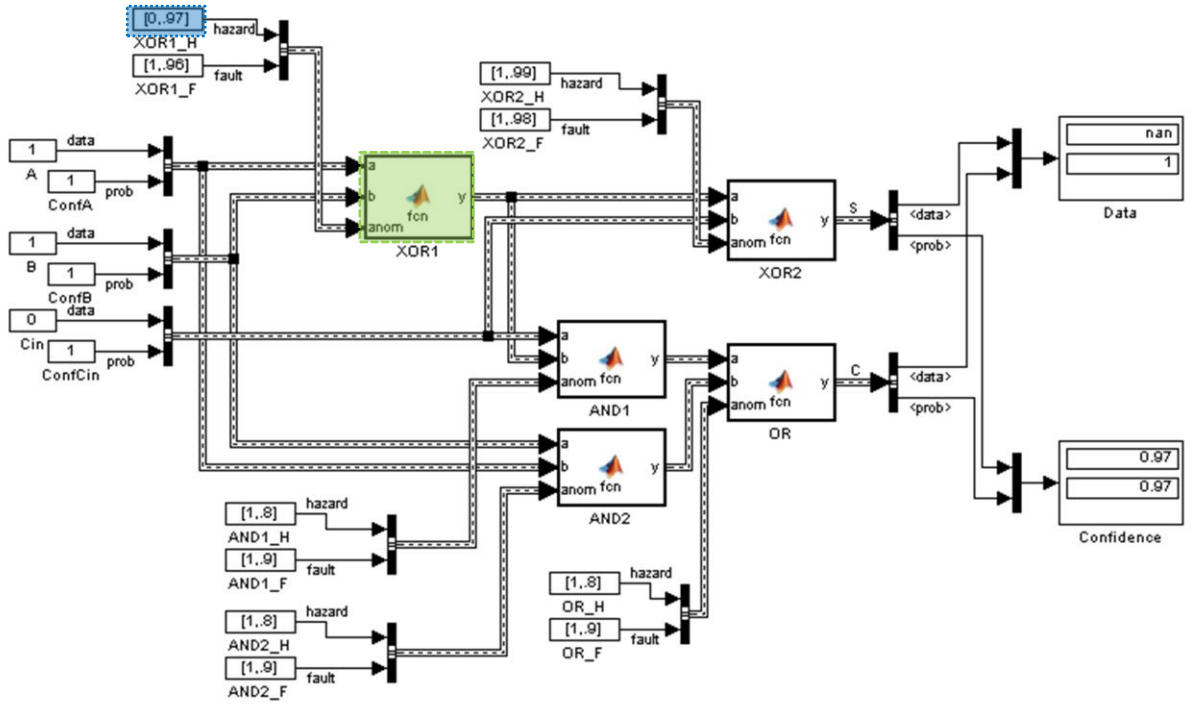
Again using the full-adder circuit, we introduce anomaly inputs that will serve to model anomalous conditions when testing the simulation against the faulted system. These appear in Figure 2.6 as Matlab 'Constant' blocks<sup>3</sup>. Each pair of numbers indicates the state of the failure mode and the confidence of this component not experiencing the given anomaly. For example, examining the anomaly input to 'XOR1' (highlighted in green), we see that 'XOR1\_H' (highlighted in blue) is set to [0, 0.97]. This signifies that a *hazard* condition is being simulated (indicated by the '0') in the component with a 97% chance that it will *not* occur in the actual system, e.g. a 97% confidence the output will be valid (or a 3% chance that an anomaly will occur). The data contained in the anomaly blocks are passed onto the next component until it arrives at the system-level output<sup>4</sup>. The information (anomaly state, inputs and outputs) for each step of the simulation is then saved to generate the catalog of conditions which lead to each possible anomalous state.

At the output of the model in Figure 2.6, the 'S' and 'C' full-adder bits can be seen with their respective Data and Confidence displays. In this case, the 'NaN' value on the *s* output indicates that an anomaly has occurred somewhere along the data path causing the output constraints to be relaxed. Again, we can see from the anomaly blocks that 'XOR1\_H' has been set to a hazard state with the associated confidence of 97%. This influences the 'XOR1' block which passes its relaxed value to 'XOR2' and 'AND1'. The subsequent results depend on each component's behavior along with the values of their other inputs; however in most cases the anomalous condition will be passed through the 'XOR2' gate, along with its 97% confidence rating. The other possible anomaly value, 'Inf' can also be a result caused by a hazard anomaly and will be seen in subsequent sections.

---

<sup>3</sup> In subsequent sections, these will be hidden behind custom 'AnomalyGenerator' blocks.

<sup>4</sup> If the simulation is running a multi-anomaly case and a subsequent component detects an anomaly on one of its inputs, the anomaly likelihood values are multiplied together. More detail is provided in section 2.8.



**Figure 2.6: The enhanced full-adder circuit.** Anomaly inputs are included in the logic gate blocks in order to assist with the diagnosis process of the simulation.

### 2.4.2. Hiding anomalies for some behaviors

Certain components may hide simulated anomalies despite the anomaly appearing on one of its input ports. This may seem like an error since a known anomaly is not being observed on the system output despite it being set in the simulation. However, this can be the correct behavior for components like the logical AND gate, whose behavior dictates that the output will be '1' if and only if all inputs are '1', and will output '0' otherwise. This implies that if *any* input is '0', the output will also be '0' even if there is an anomaly seen on another of its inputs. Therefore, we can ignore all other inputs to the AND gate, and the output becomes a valid logical '0' when one of its inputs is '0'. In the example shown in Figure 2.6, the output from 'XOR1' is carrying an anomalous condition on the 'B' input of 'AND1'. However, since the input (from  $C_{in}$ ) on the 'A' input is '0', the output of 'AND1' becomes a valid logical '0' and moves on to the 'OR1' gate. In Matlab, this becomes a special case that must be specifically accounted for due to the use of Simulink's double and "bus" datatypes in this project. The behavior is shown programmatically in Figure 2.7.

```

% AND gate nominal behavior.
function f = defaultAction(aIn,bIn)
    dataIn = [aIn.data, bIn.data];
    % If any of the inputs are 0, the output will be 0.
    if any( dataIn == 0 )
        f = 0;
    elseif any(isnan(dataIn)) || any(isinf(dataIn))
        f = NaN;
    else
        f = double (and(aIn.data,bIn.data));
    end
end

```

**Figure 2.7: Matlab code showing the nominal behavior of a logical AND gate.** This snippet demonstrates the case when an input is '0': if any of the inputs are '0', the output is also valid '0'.

Furthermore, the inverse is true (although not demonstrated in this example) for logical OR gates: if any input to an OR gate is '1', the output will be a valid '1' as well. These behaviors are built into the function of the logic gates and are dependent on the behavior of the components being modeled; other real-life components may behave similarly and hide anomalies that do not affect the output of the system.

## 2.5. The Anomaly Management Engine

The previous section introduced the high-level architecture of the enhanced model-based reasoning system with the example of the full-adder circuit. This and subsequent sections will go into more detail about how the reasoning system is implemented in order to compute a result-set for every permutation and combination of anomalous components. While Simulink is used as the “simulation engine,” there is some setup work done in the background in order to optimize the computation time of the model.

### 2.5.1. Generating anomaly permutations

In a typical scenario, one would like to simulate all combinations of anomalies in the system: i.e. the case when only the 'XOR1' gate has a fault, or only the 'AND2' gate has a hazard but the other gates are working nominally, or the case when 'XOR1' is valid but 'AND1' is faulted; etc. As part of the setup process, all possible states of every component is combined to form a vector of Boolean values describing each component's anomalous condition for the current iteration of the simulation. All of these state vectors are

**Table 2.3: Four rows of the anomaly permutations for the full-adder.** The '0' values represent anomalies that are active for the current iteration of the simulation, while the '1's symbolize the anomaly being dormant.

Anomaly Permutation Matrix									
XOR1-fault	XOR1-hazard	XOR2-fault	XOR2-hazard	AND1-fault	AND1-hazard	AND2-fault	AND2-hazard	OR1-fault	OR1-hazard
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1

combined into one matrix (table) of values, with each column representing an anomaly for a different component. A sample of this matrix for the full-adder is shown in Table 2.3.

After the model is created, the simulation is executed for each set of anomalous conditions corresponding to each row in the matrix<sup>5</sup>. With two possible anomalous states for each block, and five gates in the model, the anomaly input matrix has ten columns, two for each component. For example, the last row of the anomaly permutations shown in Table 2.3 is as follows: [1, 0, 1, 1, 1, 1, 1, 1, 1, 1]. Referring back to Figure 2.6, it can be seen how each pair of values corresponds to a different component. In this case, this row is interpreted as 'XOR1' being in a hazard state while the rest of the gates are valid. Similarly, the row [0, 1, 1, 1, 1, 1, 1, 1, 1, 1] indicates that 'XOR1' is in a faulted state, while the rest of the gates are valid. This table of anomaly states is then used by the anomaly management system to determine which component(s) is experiencing an anomaly during the current iteration of the simulation. These state values are presented along with the system outputs as part of the catalog as described in subsequent sections.

As can be expected, the set of anomaly-state vectors quickly grows in size with the number of components in order to cover all possible cases. In the case of the full-adder, the circuit has five logic gates with two anomaly inputs each. Using combinatorial math, we can calculate how many anomalous states can exist given a quantity of simultaneous

---

<sup>5</sup> For demonstration purposes, misconfigurations have been excluded from the set of possible anomalies; only faults and hazards are shown.

anomalies<sup>6</sup>. For example, if we allow up to two simultaneous anomalies to occur in the simulation the number of permutations is given as  $(_{10}C_2) + (_{10}C_1) + (_{10}C_0) = 56$  permutations. Including each possible input yields  $2^3 * 56 = 448$  possible combinations. This is generalized in equation (1).

$$s = 2^n \sum_{k=0}^b \binom{2m}{k} \quad (1)$$

In (1),  $m$  is the number of independent components in the model,  $n$  is the number of binary system-level inputs,  $b$  is the number of simultaneous anomalies that are allowed to occur in any given iteration, and  $k$  is an index iterating from 0 to  $b$ . The result,  $s$ , is the total number of anomalous states the system can assume. In the case of the full-adder being discussed,  $b = 2$ ,  $n = 3$ , and  $m = 5$  which, as above, yields  $s = 448$ . Note that this equation holds for binary system inputs; while the general architecture of the simulation can support it, adapting the system for non-binary data inputs has been left as future work.

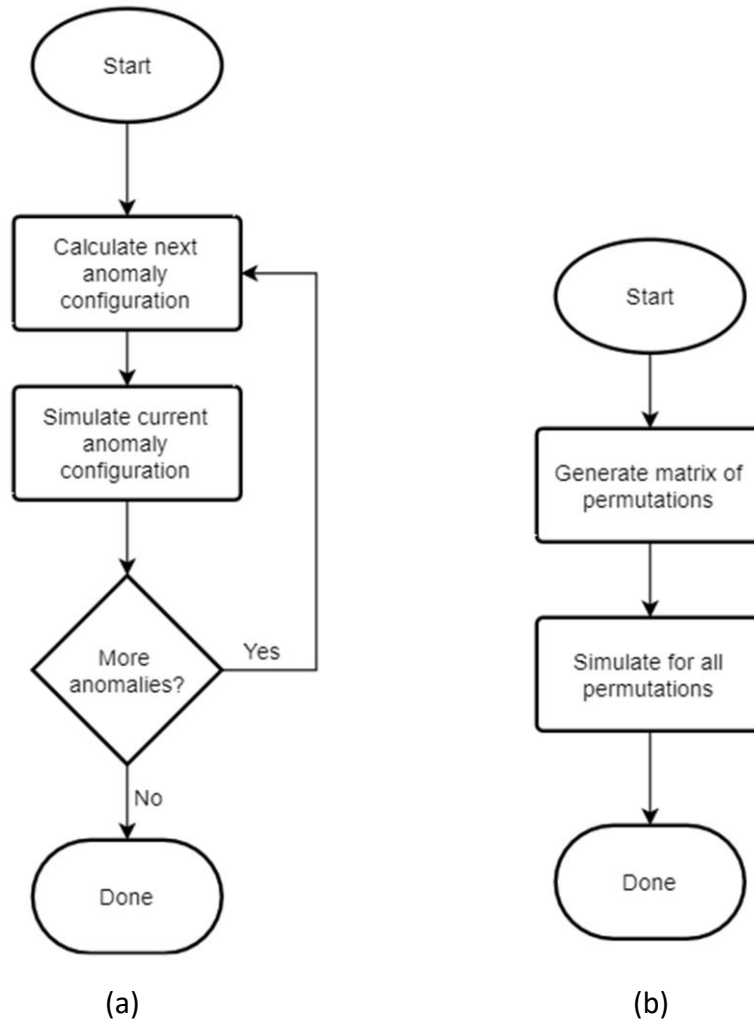
The matrix of permutations described above is only a part of the simulation process – all permutations are generated prior to running the simulation to form the full matrix. During the simulation, each row is used to set the state of the possibly-anomalous components. Fortunately, the generation of the matrix does not contribute significantly to the overall simulation time.

### 2.5.2. Efficiency improvements

In previously existing work involving a similar anomaly management system, the computation was implemented in a less-efficient manner. Instead of calculating permutations upfront, the system generated each permutation “on-the-fly” and re-ran the simulation for each step. While the end result should be the same, this method ran much

---

<sup>6</sup> Multiple simultaneous anomalies will be addressed in Section 2.8.



**Figure 2.8: Two approaches to simulation.** Approach (a) incurs a significant delay at the start of each simulation iteration; approach (b) does away with this delay by generating all permutations upfront and running the simulation just once.

more slowly due to the need to restart the simulation for every permutation instead of running the simulation once with the permutation matrix as a parameter. This is illustrated in Figure 2.8.

For example, in a complex system containing 200 components ( $m = 200$ ) with one or two possible simultaneous anomalies ( $k = 2$ ) there are more than 80,000 possible anomalous states that must be simulated. Since the simulation must iterate through each state, it is desirable to minimize the execution time required. The previous work executed Simulink's model solver once per iteration. However, a drawback of using Simulink is that there is approximately 1.5 to 2.0 seconds of compilation time each time the simulation

starts, with an actual execution time of a small fraction of a second. This adds up very quickly when there are many thousands of permutations to simulate.

The novel solution developed creates a matrix of system-state permutations (as seen above in Table 2.3) so that the simulation executes only once, using the next row of pre-calculated inputs at each time-step. This makes much better use of Simulink's capabilities and made the overall simulation much more efficient. Modifying the model used in the previous work, which contained 260 components, reduced the runtime of the simulation from 8 minutes to less than 10 seconds. This order of magnitude reduction in simulation time will allow more complex design models to be used and will allow deeper anomaly catalogs to be generated in a given amount of time.

## **2.6. The Anomaly Catalog**

When the model is simulated in Simulink, a matrix of all possible permutations of anomalies for the system is generated prior to evaluating the model<sup>7</sup>. The output of the simulation is each of these permutations concatenated with the associated input and output of the system; in other words, the system-level output for each state is reported alongside its corresponding input and anomaly configuration. This creates a catalog of possible states that serves as a lookup table in the detection and diagnosis processes: for any observed output, the set of input and anomaly states that are consistent with the output are looked up in the catalog. The catalog of possible states could potentially be hundreds of thousands of rows for a sufficiently complex system, but the time spent looking up rows in the table is trivial compared to the simulation computation time.

### **2.6.1. Anomaly detection**

The MBR system presented in this paper was designed under the assumption that there would be a separate process – whether human or computer – to detect anomalies, after which the MBR system would be used to filter the Catalog in order to determine the likely anomalies that could have occurred. However, towards the end of the project it was

---

<sup>7</sup> For readers with Simulink experience, this was implemented using the Simulink callback function 'InitFcn'.

realized that detection could be handled automatically by simply inputting the real system outputs into a Matlab script which would search the Catalog for matching cases. Then, the Catalog would automatically be queried to return the most likely anomalies that could cause the observed output. This feature was not implemented as part of this thesis work but has since been implemented by other researchers on the team.

In the automatic detection process, the outputs of the actual system are compared to the simulated outputs in the catalog database. In the nominal case, the system will return exactly one row matching the specified inputs and valid outputs. In the anomalous case, no rows will be returned since the anomalous output does not match any of the simulated outputs for the given input. In this case the anomaly management system would iteratively relax the constraints on each non-matching system output to generate a subset of the catalog in which the rows correspond with anomalies that could cause the anomalous output.

For example, consider the full-adder configured with the input  $[1, 1, 1]$ . If the output was observed to be  $[S, C] = [0, 1]$  (while  $[S, C] = [1, 1]$  is the expected output), this would indicate an anomaly on the Sum bit because this is the actual output that does not match the simulated output. Since in this case there is only one anomalous result, the Catalog would be queried automatically for all rows in which the carry bit is equal to '1' while relaxing the constraints on the Sum bit output: e.g. searching for  $[S, C] = [X, 1]$  (where X is "don't care"). The catalog would then return results indicating XOR1 or XOR2 is experiencing an anomalous condition, or that one of the inputs is misconfigured such that it is set to '0' instead of the expected '1'. After the anomaly is detected and all possible anomalies are found, the possible diagnoses are returned in a sorted tabular format for use by the operators of the system.

### **2.6.2. Anomaly diagnosis**

When an anomaly is detected by the anomaly detection process (this could be a human operator or a future implementation of the enhanced MBR system), an operator uses a standalone Matlab script to search the previously generated Catalog for potential causes of the observed anomaly. This script searches for the provided observations and returns a



sorted table containing only the Catalog rows that match the system output. Each row contains a different permutation of anomalies that could occur, depending on whether the catalog was generated to simulate multiple possible anomalies. The table is ranked by the confidence value of the anomalies as shown in Table 2.4. In the center columns (anomaly states), a ‘1’ indicates nominal operation for the corresponding component, while a ‘0’ indicates a fault, hazard, or misconfiguration has occurred. Since the table is ranked and ordered, the most likely scenarios can be evaluated by the operator first. This process is similar to the “expert reasoning” approach discussed in Chapter 1 – the lookup table (Catalog) is consulted in order to determine possible resolutions. However, since the table is enhanced with the addition of likelihood values, the process of determining probable diagnoses will be accelerated. This enhanced MBR approach can be viewed as combining the best parts from the experiential and model-based reasoning systems – the accuracy and structure of the model-based approach is combined with the rapid lookup of the

**Table 2.4: A sample of the catalog generated by the anomaly management system showing four rows of the 448 unique permutations.** Several of the anomaly states have been hidden for simplicity as they do not affect the output. The system-level inputs are shown in the leftmost columns, while the outputs are shown on the right. The “\_conf” columns represent the confidences associated with each system output.

System Inputs			Anomaly Inputs						Outputs			
A	B	C <sub>in</sub>	XOR1-fault	XOR1-hazard	XOR2-fault	XOR2-hazard	...	OR-hazard	S	C <sub>out</sub>	S_conf	C <sub>out_conf</sub>
1	1	1	1	1	1	1		1	1	1	1	1
0	1	1	1	1	1	1		1	0	1	1	1
1	1	1	0	1	1	1		1	Inf	1	0.99	0.99
1	1	1	1	0	1	1		1	NaN	1	0.985	0.985

experiential expert reasoning method in order to be as efficient as possible when and where it counts.

## 2.7. Implementing Enhanced MBR

This section will discuss the specific implementation details of how the aforementioned features are applied to a basic Simulink model. The addition of anomaly confidence data has already been presented in section 2.4.1; in this section the details of the “anomaly generator” block will be shown, as well as the “input permutations” block.

Together, these blocks enable the execution time improvements discussed in section 2.5.2. In addition to these two blocks, the Simulink model file must also contain callback functions<sup>8</sup> which create the anomaly permutation matrix and provides some additional functionality. For example code, see Appendix B .

### **2.7.1. Modification of existing components**

Starting from a Simulink model representing a given system, changes to each functional block are necessary. The MBR system provides a standard template to implement each functional behavior component: after an auxiliary input for the anomaly confidence data is added to the block, the template function checks the data inputs to detect the presence of anomalies both on component inputs as well as the anomaly likelihood input. If no anomalies are detected for the current iteration of the simulation, the Matlab function representing the component's nominal behavior is executed. The component's output is then its expected output for nominal behavior.

When an anomaly is present on the anomaly likelihood input, this indicates that the component is “experiencing” an anomaly (for the current iteration of the simulation). The component's output will then take the value of ‘NaN’ or ‘Inf’ depending on the type of anomaly present. For example, if an AND gate is being simulated with a fault anomaly, its output will be ‘Inf’. This indicates that the output could be anything, including nothing (a “flatline”). In the case of a hazard anomaly, the value ‘Inf’ will be the output, which will be sent to any connected downstream components. For both cases, the nominal behavior function will not be executed.

### **2.7.2. Passing data between blocks**

In order to keep track of the anomaly confidence of previous components, Simulink's bus datatype was used to pass data and confidence values between components. This is essentially a structure datatype that can encapsulate various other types of data and allows

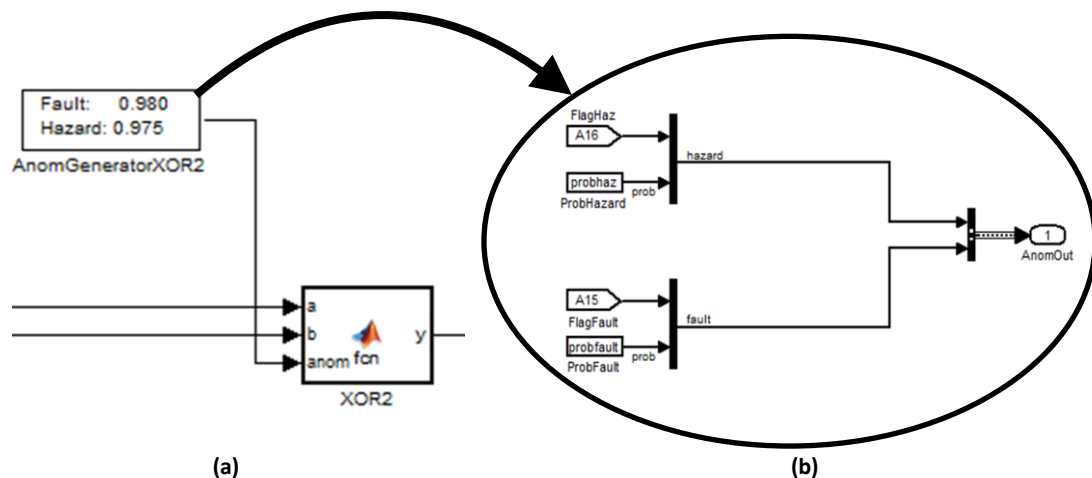
---

<sup>8</sup> These callback functions are generic and should work for any model as long as the correct blocks are included.

the confidence value to be tracked alongside the data propagating through the model<sup>9</sup>. If a previous component has experienced an anomaly, the data input to the current component will have a confidence value associated with it. This confidence is then used to calculate the overall confidence value if the current component also experiences an anomaly. Usually this will be a simple product of the two confidence values.

### 2.7.3. The Anomaly Generator block

Once each functional block has been modified to include the capability for simulating anomalies, several blocks must be added. First is the `AnomalyGenerator` block. This block is an interface to set the anomaly confidence parameters of its corresponding system block and is shown in Figure 2.9(a). During a simulation, the simulation engine will activate the corresponding anomalies based on the current permutation being simulated. The `AnomalyGenerator` block's output is connected to the supplementary input of each component that could experience an anomaly. The component is then responsible for computing its output state in the case of an anomaly.



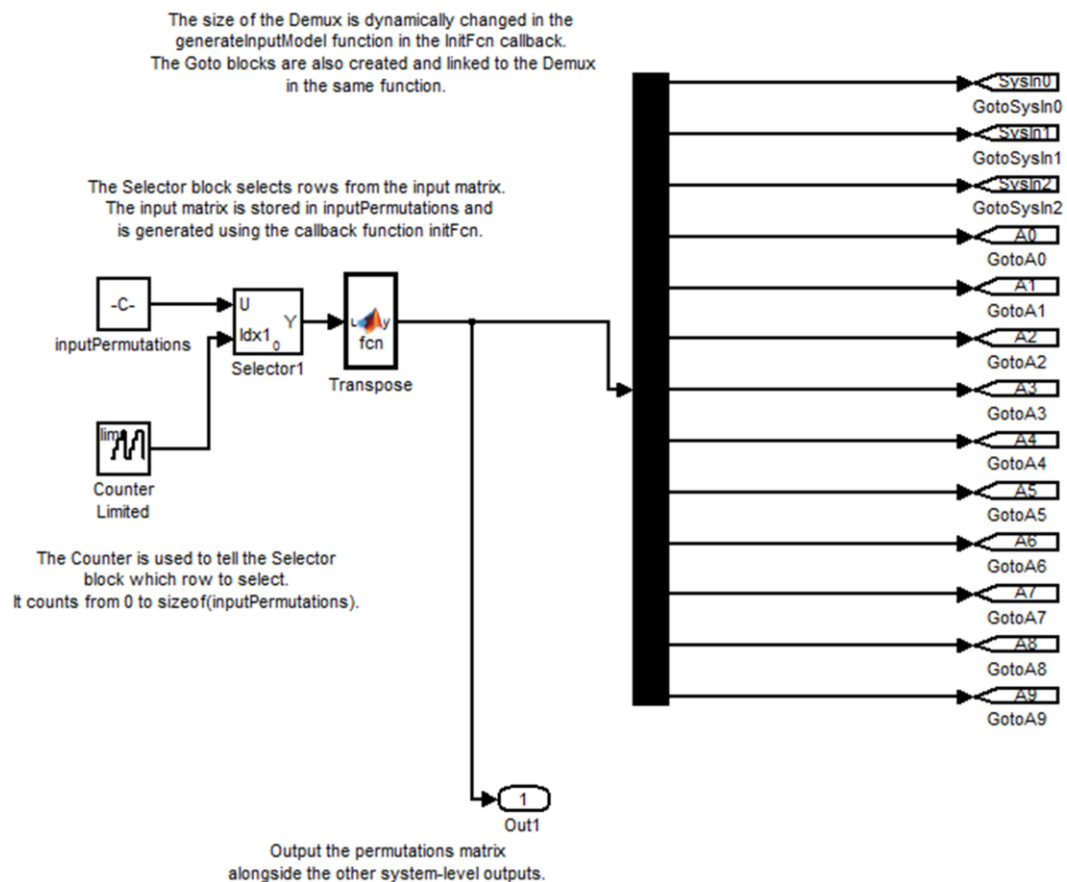
**Figure 2.9: The anomaly-generator block injecting an anomaly into the XOR2 component.** The AnomalyGenerator block in (a) is a mask which hides the components shown in (b). The component XOR2 then implements the behavior of the anomaly when applicable.

<sup>9</sup> In the implementation presented, the structure simply contains the Boolean data along with the confidence value. The structure could be modified to contain more data to be used by the components, but this has been left as future work.

The generator block output consists of a Boolean flag and a decimal confidence value for both hazard and fault anomalies. The two values are combined into a single output using Simulink's Multiplexer and Bus Creator blocks (as shown in Figure 2.9(b)). This data then is processed by the component itself; usually, the data will simply be forwarded to the next connected component as `NaN` or `Inf` indicating the presence of an anomaly.

#### 2.7.4. Anomaly input permutations block

The next addition that must be included in the Simulink model is the `InputGenerator` block detailed in Figure 2.10. This block simply reads each line of the anomaly permutation matrix (as described in section 2.5.1) and sends each Boolean value to its corresponding `AnomalyGenerator` block to set the state of each anomaly. Simulink's modeling engine



**Figure 2.10: Detail of the InputGenerator subsystem block.** The input generator block iterates through all possible anomaly permutations and passes the state to the `AnomalyGenerator` blocks using `Goto` blocks.

takes care of the rest: each component's behavioral script is executed to determine its output given the inputs and anomalous state until the final system output is computed. The inputs and outputs, along with the corresponding anomaly permutations, are then saved as a new row in the Anomaly Catalog.

When the simulation runs, an initialization function is executed to setup several parameters that will be used by the model. Given that the model has been constructed correctly using the anomaly generator blocks, the initialization function will create the table of all anomaly permutations based on the number of anomaly generator blocks in the model. It then automatically generates the `Goto` blocks shown in Figure 2.10. These `Goto` blocks simply send the anomaly states to each `AnomalyGenerator` block. Figure 2.10 shows the block as seen for the full-adder example. More complex models will have many more blocks to send each state to the corresponding component in the system.

#### 2.7.5. Analyzing the anomaly catalog

After the Catalog has been generated by the simulation, the anomaly diagnosis process can begin. In a system that does not utilize any kind of anomaly management engine, this would typically be done by experienced operators of the system such as engineers or trained technicians. It would involve a tedious process of identifying any and all anomalies that occurred, then examining each possible configuration of the system in order to determine what may have caused the observed anomaly. This could take hours, days, or even weeks. The focus of this research is ultimately to accelerate this timeline. By simulating the system then comparing the observed output to the simulated output, the simulation engine will report the possible faults, hazards, and misconfigurations that could cause the observed outputs. This is implemented as a Matlab script,

`findAnoms(inputs, catalog, symptoms)`<sup>10</sup>, which returns rows corresponding to the generated anomaly catalog, and the system's inputs and outputs. Given these items, the script will find the rows (anomaly states) that are consistent with the observed output.

---

<sup>10</sup> The full function definition is `[results, anoms] = findanoms(sysInput, catalog, symptoms, fname)`, indicating two outputs will be returned. This script is described in more detail in Section 3.1.2.

In a future form of this work, the `findAnoms` script could be enhanced with an automatic detection feature. This feature would determine if an anomaly was present given a set of system outputs, and then proceed with the diagnosis phase automatically. The results returned would be identical, however it would save the step of manually querying the Catalog.

### **Manual Detection**

This project assumes that the anomaly detection process occurs prior to the diagnosis procedure, usually by a human operator. In this case, the `findAnoms` script can be run by specifying exact outputs for which the operator would like to search, including a flag to indicate the suspected anomalous output. Using the full-adder catalog as an example, one might want to search for all cases where the ‘S’ output is anomalous, but the ‘C<sub>out</sub>’ output is a valid ‘0’. To do this, the operator would run the `findAnoms` script as so:

`findAnoms([0,1,0], yout, [inf, 0])`. This function takes as input a set of system level inputs (in this case `[0,1,0]`; optionally the empty set `[]` as a wildcard), the Catalog generated previously by the simulation engine (`yout`), and a vector corresponding to the observed outputs (`[Inf, 0]`). The output `[Inf, 0]` indicates that an anomaly has been detected on the Sum bit of the full-adder, and therefore can take on any value in its domain. The function returns a subset of catalog entries, ranked by the calculated total confidence output as shown in Table 2.5<sup>11</sup>. With these results, the operator or technician

**Table 2.5: The filtered output of the Catalog after searching for observed output.** The data has been sorted by ‘Overall Likelihood’, with the most likely case on top. As can be seen, XOR1 and XOR2 are the most likely candidates to cause the anomalous output  $[S, C_{out}] = [Inf, 0]$  given the input  $[0, 1, 0]$ .

'A'	'B'	'Cin'	'AND1 Fault'		'XOR1 Fault'	'XOR1 Haz'	'XOR2 Fault'	'XOR2 Haz'	'S'	'Cout'	'S_conf'	'Cout_conf'	Overall Confidence
0	1	0	1	...	1	1	0	1	Inf	0	0.975	1	0.975
0	1	0	1		0	1	1	1	Inf	0	1	1	0.985
0	1	0	1		0	1	0	1	Inf	0	0.99963	1	0.9999
0	1	0	1		1	0	0	1	Inf	0	0.99975	1	0.9999

<sup>11</sup> While these results demonstrate possible points of failure in the full-adder model, in this case the confidence values associated with each gate shown are arbitrary—they were simply chosen to demonstrate the concept.

can examine the real system, starting at XOR1 and XOR2, since these are the components at the top of the list of failures, having both experienced faults. In fact, in this case they are the only two components of the full-adder system that can fail in a way to generate the output `[Inf, 0]` from the input `[0, 1, 0]`. If we were to run the script specifying the wildcard input (`[]`) and a different output (for example, `[Inf, 1]`), different results would be returned, again with the most likely scenarios at the top of the list.

### ***Automatic detection***

As discussed in Section 2.6.1, the system could have the capability to automatically detect anomalies as well as diagnose them. In this case, the real system output would be sent to the `findAnoms` script as the `symptoms` parameter in much the same way as for manual detection. However, when the empty-set is found, this indicates that for no input in the catalog will the given output be valid. In other words, no nominal system input maps to the provided system output. Since the simulated output does not match the provided real-system output, this indicates an anomaly is present. Furthermore, it can be reasoned that the system output which does not match the simulated result for the given input is anomalous. In this case, the script would continue execution using this detected anomaly as a symptom and report results as described above and shown in Table 2.5. Further interpretation of the results will be discussed in Chapter 3.

## **2.8. Multiple Anomalies**

In any engineering system, there is a chance that more than one component could experience an anomaly simultaneously. Fortunately in the general case we can assume that all components fail independently [7], so the total likelihood of the scenario is a simple product of the two (or more) anomaly likelihoods. However, there are two separate cases that the MBR system considers: the case of two (or more) anomalies occurring along one datapath, which only affects a single system output; and two (or more) anomalies occurring in distinct, independent parts of the system (overall likelihood). In the first case, when two or more anomalies occur on the same path to the output, the corresponding confidences are simply multiplied together to calculate the anomaly likelihood associated

with that output. For example, if both XOR gates in the full-adder (Figure 2.6) experience an anomaly, the two likelihoods are combined and the likelihood output associated with the 'S' bit will assume the product of likelihoods for each gate.

The second scenario occurs when two anomalous components are not on the same datapath. For example, if XOR2 and AND2 are both faulted the likelihoods will not be multiplied together since they are not on the same path to the output. Instead, each output whose value is influenced by the anomalous component will report a faulted state with its associated likelihood. Then, the anomalous component likelihoods are multiplied together to give an overall likelihood for the current state. This overall likelihood is calculated for both the inline and independent multiple anomaly scenarios. However, in the first case the overall likelihood value will match the system output's likelihood while in the second case it will be different.

This is a subtle difference: the former scenario will calculate the product of the likelihoods if the failed components are *in series* along the datapath; the latter likelihood accounts for *all* anomalies everywhere in the system and reports the overall likelihood for the given anomalous state. In other words, the anomaly management system will generate and categorize the overall likelihood of a given state as well as the likelihood that each system output is valid.

If a system contains components whose anomaly likelihoods are dependent on each other, the behavior of the simulation would have to be modified to account for the more complex calculations. This scenario, however, is beyond the scope of this research and has been left as future work.

## **2.9. Meaning of likelihoods**

Thus far, the meaning of the likelihoods or confidences associated with each anomaly has been deliberately left abstract. It is tempting to refer to these numbers as "probabilities," but doing so implies that there exists experimental knowledge in order to precisely calculate these numbers. In the case of this research, "likelihood" is the "probability" of an anomaly occurring; "confidence" is the "probability" of the component



behaving nominally (it is not experiencing an anomaly). The two terms are inverses of each other: the likelihood is just one minus the confidence value.

Although the anomaly management system does not disallow using them as such, it has been developed with an alternative approach in mind. Instead of necessitating the use of precise numbers for the chance of an anomaly occurring, the designer can assign relative “ranks” to each anomaly and assign likelihood numbers based on the ranking. This relies on the designer to have some prior experience with the system and its components, but does not require extensive, time-consuming experiments to calculate the actual rates of failure.

For example, if there is a simple system that has three components which can be affected by anomalies A, B, and C, we can assign values based on the system designers’ experience. Based their observations, they may observe that Anomaly A occurs much more frequently than Anomaly B, but Anomaly C occurs just a little less frequently than Anomaly B. Then, the confidences could be assigned as follows (on an interval from 0.0 to 1.0):

- Anomaly A: 0.60
- Anomaly B: 0.80
- Anomaly C: 0.85

Recall from section 2.4 that a confidence closer to 0 means that the output has a lower chance of being valid; therefore Anomaly A is the most likely to occur. As described above, these anomaly confidences are interpreted as a relative number, and should *not* be interpreted as “Component A does not fail 6 out of 10 times the system is executed.” It should be read as “Anomaly A has been observed much more frequently than Anomaly B; Anomaly C occurs the least frequently, and only slightly less often than Anomaly B.” In this way, experienced users can build in more meaning than precise probabilities would have without the need to derive these probabilities for each individual component. The same methodology would then be applied to the other anomaly components in the system.

When using the confidences as described, the calculation for multiple anomalies changes slightly. In classical probability theory, the probability of two independent events occurring simultaneously is the product of the two events occurring separately. However,

since the system uses confidence values associated with anomalies (closer to 0 is more likely) rather than probabilistic likelihood values (closer to 0 is less likely to occur), the formula is inverted. For example, using the above confidence values the anomalies A and B multiplied together give 0.48. This would imply that the likelihood of two anomalies occurring would be much more likely than either one occurring independently which is intuitively incorrect: a single anomaly should always have a higher likelihood than multiple anomalies occurring simultaneously. Therefore, when we combine the confidence values we must multiply their inverses together to give a final answer. The equation in this case becomes:

$$\begin{aligned}C &= 1 - (1 - A)(1 - B) \\C &= 1 - 0.4 * 0.2 \\C &= 0.92\end{aligned}\tag{2}$$

Thus, the likelihood of both A and B occurring at the same time is much less likely than any of the three example anomalies occurring independently. In other words, we are 92% confident that the output will be valid; there is an 8% likelihood that two anomalies will occur.

## **2.10. Chapter Summary**

The new enhanced MBR system demonstrated in this chapter is a logical extension of previously created model-based reasoning systems and can be easily be added to supplement existing system models. Confidence values have been added to each anomaly in each component in order to generate a catalog of ranked anomalies to aid in the diagnosis phase when anomalies are detected. The MBR system has been designed to generate the Anomaly Catalog (a potentially time-consuming process) prior to the operational phase of an engineering system. The Catalog then serves as a lookup table to be utilized by operators and technicians after an anomaly has been detected. The anomaly ranking system allows more likely scenarios to be examined first, thus accelerating the diagnosis and resolution process to correct the anomaly in a timely manner. The model-

based simulation and the resulting catalog can be seen as a hybrid MBR-experiential approach meant to leverage the best parts of each methodology: the structured approach of MBR, combined with the rapid execution of an experiential rule-based system.

### 3. Applications and Results

This chapter describes the application of this research starting with the full-adder circuit, then demonstrating the system applied a more complex system utilized by Santa Clara University's Robotics Systems Lab (RSL) in its satellite operations program. First, the results of validation testing the full-adder system will be presented. Next, the design and implementation of the enhanced MBR system as applied to the Beacon Network system will be introduced, followed by the results of experimental tests on the system.

#### 3.1. Full-Adder Results

As detailed in the previous chapter, the full-adder circuit was used to initially validate the new model-based reasoning system. Each logic gate of the circuit was supplemented with an anomaly input which determined the anomalous condition of the component and its apparent likelihood of occurring. When the simulation completes, the output is the Anomaly Catalog as described in the previous chapter. A sample of the Catalog generated by the full-adder simulation has been reproduced in Table 3.1, while a more verbose version appears in Appendix A .

**Table 3.1: A small section of the catalog generated by the anomaly management system.** The catalog shows the configuration of each anomaly in the system, with each row corresponding to a unique permutation. A more extensive excerpt can be found in Appendix A

extensive excerpt can be found in Appendix A

System Inputs			Anomaly Inputs (0 indicates anomaly present)					Outputs (NaN/Inf indicates anomaly detected)					
A	B	C <sub>in</sub>	XOR1-fault	XOR1-hazard	XOR2-fault	XOR2-hazard	...	OR1-hazard	S	C <sub>out</sub>	S conf	C <sub>out</sub> conf	Overall Conf
1	1	1	1	1	1	1		1	1	1	1	1	1
0	1	1	1	1	1	1		1	0	1	1	1	1
1	1	1	0	1	1	1		1	NaN	1	0.99	0.99	0.99
1	1	1	1	0	1	1		1	Inf	1	0.985	0.985	0.985

In addition to an anomaly confidence value for each output, each state in the Catalog also has an associated "overall confidence" value which numerically represents the likelihood of the given state occurring in the system based on *all* of the anomaly confidence values in the model. As before, the confidence is on the interval  $[0, 1]$  and conveys how

likely the component's behavior will be valid. That is, a number closer to 1.0 means that the anomaly is less likely to occur and should be considered the least likely candidate when moving on to the resolution phase. A confidence value of exactly 1.0 represents the nominal output for states where no anomalies are present.

### **3.1.1. Improving simulation time**

As mentioned in the previous chapter, a novel technique of simulation was utilized which decreased simulation time by a factor of nearly 50 from the previous iteration of this research, reducing the execution time of a simulation involving 200 components from approximately eight minutes to less than ten seconds. In the case of the full-adder system with just five components, the entire simulation process for two simultaneous anomalies finishes in less than five seconds. Simulating three simultaneous anomalies takes less than ten seconds. The catalog generated for each of these scenarios is 448 and 1408 rows, respectively. The lookup time to analyze the catalog to determine which components could have caused the observed output is less than a second, where the previous technique needs several minutes to evaluate the same system with two anomalies. This accelerated simulation speed allows for more frequent and deeper analysis of a given system.

The more complicated system with 200 components modeled generates 30,000 permutations, and the simulation time for two anomalies is close to 10 minutes. Simulating three simultaneous anomalies yields almost 3 million permutations which required approximately 83 minutes. Searching and filtering the catalog is again inconsequential – looking up values in a table is several orders of magnitude faster than executing the full simulation. For comparison, the previous iteration of this research required multiple days to simulate three anomalies for a similar system.

### **3.1.2. Analysis results**

After running the simulation, there are several results that are output as variables in the Matlab workspace for later analysis. The primary computational output is the catalog described previously. It is created as both a raw Matlab matrix variable (for use in scripting/analysis operations), as well as an operator-friendly version which includes

column headers and component names. Once the catalog is generated, the `findAnoms` script can be executed using the catalog as a parameter. As described in Section 2.7.5, the script will return a sorted subset of data from the catalog, and also print human-readable information (shown in Figure 3.1) to inform the user of the most likely anomalies that created the observed output. The technician or operator can then utilize this list as part of the resolution phase in order to bring the faulted system back to a nominal state.

The results shown in Figure 3.1 were created by the `findAnoms` script after the catalog was generated by the full-adder model simulation, which was configured to simulate two simultaneous anomalies. The simulation and script took less than a second to execute. In this case, the script is looking for all anomalies in the system which could have caused the output `[1, NaN]`: i.e. the sum bit is a valid '1' while the observed carry-out bit is inconsistent with the simulation's output for the given input. The results are further filtered by the parameter specifying the system input as `[1, 0, 0]`, which refers to the A, B, and C inputs taking on the respective values. The full command is

`findAnoms([1, 0, 0], yout, [1, nan])`. Looking at the results, we see that the 'OR1' gate

```
0.9500: 'full_adder/OR1Haz'
0.9500: 'full_adder/OR1Fault', 'full_adder/OR1Haz'
0.9550: 'full_adder/AND2Fault'
0.9600: 'full_adder/AND2Haz'
0.9600: 'full_adder/AND2Fault', 'full_adder/AND2Haz'
0.9650: 'full_adder/AND1Fault'
0.9700: 'full_adder/AND1Haz'
0.9700: 'full_adder/AND1Fault', 'full_adder/AND1Haz'
0.9984: 'full_adder/AND1Fault', 'full_adder/AND2Fault'
0.9986: 'full_adder/AND1Fault', 'full_adder/AND2Haz'
0.9987: 'full_adder/AND1Haz', 'full_adder/AND2Fault'
0.9988: 'full_adder/AND1Haz', 'full_adder/AND2Haz'
0.9999: 'full_adder/AND2Fault', 'full_adder/OR1Haz'
0.9999: 'full_adder/AND2Haz', 'full_adder/OR1Haz'
0.9999: 'full_adder/AND1Fault', 'full_adder/OR1Haz'
1.0000: 'full_adder/AND1Haz', 'full_adder/OR1Haz'
```

**Figure 3.1: Sample output after running the anomaly diagnosis script for the full-adder circuit.** The output after running `findAnoms([1 0 0], yout, [1, nan])`; which is showing all anomalies that, when active, produced the output `[1, NaN]`. This output reflects the observation of the Sum output having a valid output of '1', while the Carry output is faulted. The results show each anomalous component with its full model path. The full-adder is a flat model, so every block is on the top level in this case. More complex models would show the full path of the component in order to easily locate it within the system. The confidence values are sorted in order to assist the operator in diagnosis/resolution. Note that the value of '1.000' is a result of Matlab rounding 0.99999 to 1.0. Safe to say, these scenarios are very unlikely to occur according to the model.

with a hazardous condition is reported first as the most likely candidate to have failed. The next row implies that 'OR1' with both a hazard and simultaneous fault condition is the next most likely candidate. However, this is intuitively much less likely to occur and therefore should have a confidence value closer to 1.0. In this case, this row can safely be ignored – the system does not currently account for simultaneous anomalies within the same component. In cases where this occurs, the hazard confidence value overwrites the fault value and this propagates through the system. This behavior could be changed in future iterations of this research. Note that this also occurs for the AND1 and AND2 gates.

### 3.1.3. Querying with wildcards

In many cases, the operator may wish to identify all components that could have caused any anomalous output using any system input. For example, if the symptoms observed are that the Sum bit was '1', and the Carry bit was anomalous, this can be specified using '-1' as in `[1, -1]`. This has the advantage of finding both hazards and faults in the output. The specified input can also be specified as a wildcard by specifying an empty-set as `[]`. Providing these parameters to the `findAnoms` script, the command becomes `findAnoms([], yout, [1 -1])`. However, specifying a wildcard input parameter has the disadvantage that many more results are returned, some of which are duplicate scenarios where only the system inputs (configurations) are different<sup>12</sup>. The

```
>> [res anom] = findAnoms([], yout, [1,-1]);
0.9450: [1,1,1] 'full_adder/OR1Fault
0.9500: [1,1,1] 'full_adder/OR1Haz
0.9550: [1,1,1] 'full_adder/AND2Fault
0.9600: [1,1,1] 'full_adder/AND2Haz
0.9650: [0,0,1] 'full_adder/AND1Fault
0.9700: [0,0,1] 'full_adder/AND1Haz
Elapsed time: 0.042242
```

**Figure 3.2: Sample output with wildcards.** The output after running `findAnoms([], yout, [1, -1]);` which is showing all anomalies that, when active, produced an anomaly on the carry-bit output. This list has had duplicates removed; the full list would show every input that would produce the output with a sum-bit of '1' and an anomalous carry-bit.

---

<sup>12</sup> These duplicates have been left as-is to allow for future flexibility in interpretation and analysis of the results.

results are shown in Figure 3.2. It can be seen that the anomalous components in this list match the results shown in the previous figure, as they should: the components that could have caused an anomaly on the carry-bit output are `AND1`, `AND2`, and `OR1`. Note that although `XOR1` does influence the carry-bit output, if it was anomalous the sum-bit would also show a symptom. Since the sum-bit was specified as being a valid '1', the system has correctly diagnosed that `XOR1` and `XOR2` cannot be in an anomalous condition. Therefore, any of the three components could have been in a faulted or hazardous state to cause the observed symptoms.

These initial results for simulating the full-adder proved that the system was effective at a small-system scale. The next step was to apply the enhanced MBR techniques to a larger, more complex system to test the validity and ease-of-use of the anomaly management engine in a real system.

### 3.2. The Beacon Network

Until this section, we have been discussing the theoretical application of model-based reasoning to the ideal full-adder system as a proof-of-concept. That is, we haven't applied MBR to a real engineering system. For demonstration purposes, we will look at how this anomaly management system can be applied to SCU's satellite beacon monitoring network operated by the RSL. This and subsequent sections will introduce the architecture of the beacon network and show its Matlab/Simulink implementation. Next, a simulated-anomaly experiment will be presented. Finally, the results of applying the anomaly management system to the beacon network and using it to diagnose anomalous behavior will be



**Figure 3.3: Beacon monitoring concept diagram.** The satellite orbiting the earth broadcasts its health status to one of several groundstations, which forwards the data to mission control [14].



discussed.

### **3.2.1. Network architecture**

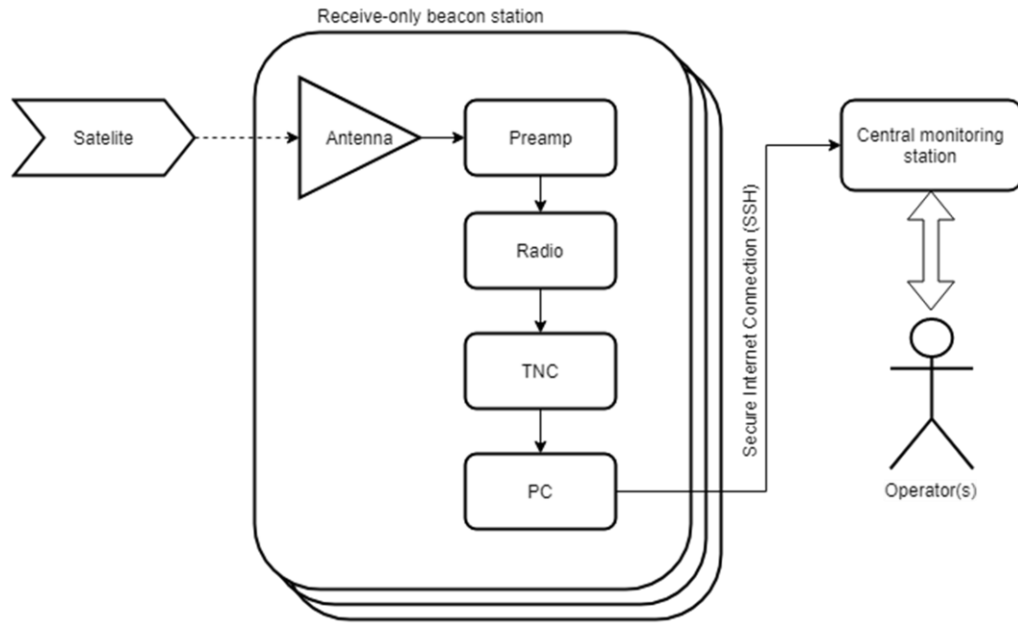
The beacon network exists in order to assist with the operation and maintenance of small satellites orbiting the Earth by providing a periodic summary of the satellite's health. The network is a set of computers distributed around the country, each of which is connected to an antenna and radio along with several other supporting components. Each beacon node is a receive-only station which receives the radio packet that is transmitted by the satellite being observed. This message follows the AX.25 packet protocol (detailed in [8]) and is decoded by a TNC (terminal node controller) which sends the short message via serial cable to the computer. The protocol represents binary data as ASCII-encoded text so it can be parsed and translated by a human operator viewing the terminal if necessary. An example of the message from the O/OREOS nanosatellite is as follows [9]:

```
OOREOS.org      6307260093011D021F6A0219029992B4010D5F094300204B46
```

After the computer receives the data from the radio, it then forwards the message over a secure Internet connection to the central beacon monitoring system, physically located at SCU [9]. This dataflow is shown for a single node in Figure 3.4. While there are several other software components that must be running on the associated computers, the details of the software operation are out of the scope of this research.

### **3.2.2. Motivations**

The beacon network automates many of the mundane but necessary tasks that are crucial to maintaining nominal operation of satellite. The network monitors data such as temperature, battery voltage, and solar panel currents and has the ability to send automated emails if any of these parameters are out of range as described in [9]. While this is a very useful tool during the operational phase of a satellite, the network is not without its issues. Each computer station currently requires manual setup and a constant and stable connection to the central computer located at SCU. This secure internet connection (SSH



**Figure 3.4: A simplified block diagram of the beacon network.** This diagram shows the network from the perspective of an individual node. The satellite data is forwarded from the station to the central monitoring computer via a secure internet channel.

tunnel) is terminated if any part of the network’s route is compromised. Furthermore, the computers require monitoring software to run constantly in the background in order to receive, log, and forward data packets. Unfortunately, this means that there are many points of failure at each beacon node. One of the primary motivations for this research is to develop a tool which can assist in the diagnosis of the beacon network system to maintain a functional system semi-autonomously.

A major factor in determining the status of a particular station is the state of the satellite being monitored since many things could influence the successful reception of a data packet. For example, the radio may have been automatically turned off to conserve power, the vehicle may be spinning in such a way that its antenna is not pointed towards the earth, or a more serious anomaly may have occurred causing the entire system to fail. Although there may be no anomalies in the groundstation itself, the state of the satellite plays an important role in detecting faults in the system and is therefore included in the model. A simple block diagram of a satellite is shown in Figure 3.5.

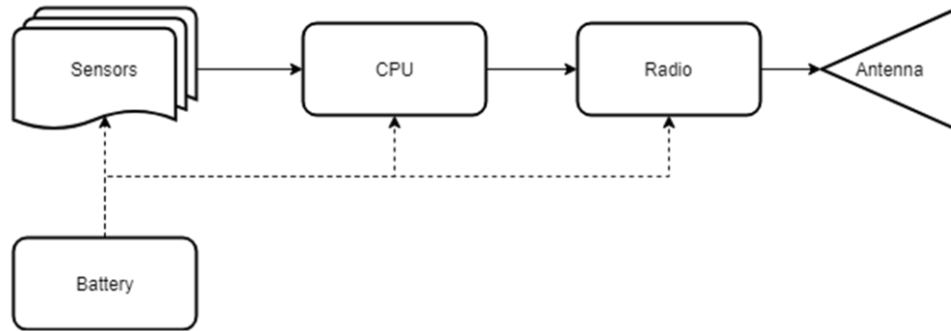


Figure 3.5: A very simple block diagram of a satellite.

The other components of the system include the station's computer waiting to receive the transmission, the radio which is tuned by the computer to the satellite's Doppler-shifted frequency, and the Internet networks connecting the remote system to the SCU operations computer. All of these pieces must be operational in order to receive data packets from the satellite and determine its health. When any component fails or a program crashes, the station has entered an anomalous state and will no longer be able to relay received packets to the operations computer. Without this data, the health of the satellite is unknown and the mission is at risk. Thus the beacon stations play a crucial role in monitoring the satellite and must be revived as soon as possible after an anomaly is detected.

### 3.2.3. Modeling the Beacon Network

The Simulink model of the beacon network system is shown in Figure 3.6. It consists of a block for each high-level component, each of which contains several more internal components. The model illustrates the dataflow previously presented: the **satellite** block creates a data packet which is received by the **beacon station** at SCU or one of the stations at collaborating universities around the country (Baylor University, Worcester Polytechnic Institute [WPI], and St. Luis University [SLU]). When the data is received by one or more stations, it is forwarded via the **Internet** to the primary **operations console** at SCU. In nominal operation, this console is running software which parses the packet and inserts it into a database of all received packets. The Matlab scripts which parse the packet also do rudimentary bounds checking of important parameters such as battery voltage and

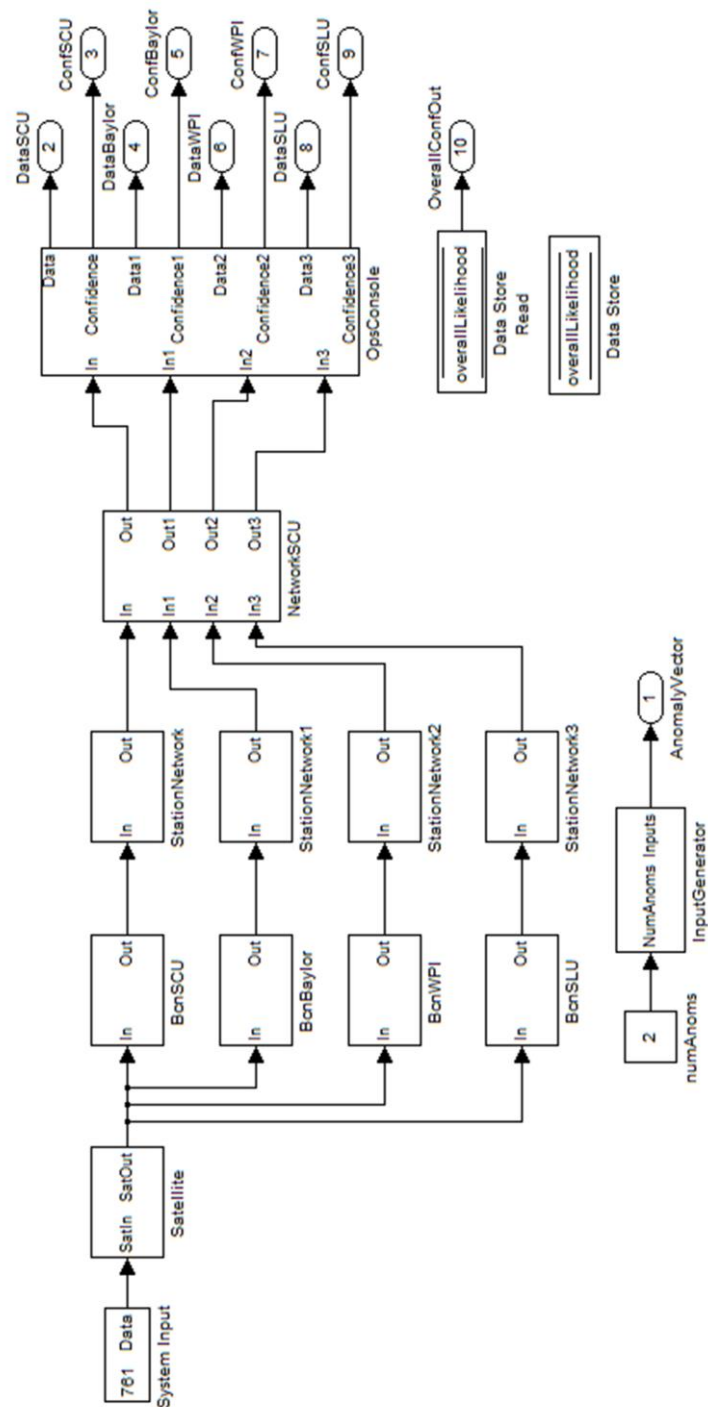


Figure 3.6: The Simulink model of the Beacon Network. Each block is its own independent subsystem containing its own components. Anomaly confidence data is entered for individual components within each subsystem shown.

temperature [9]. This is an example of an expert system as described in Chapter 1 – there is a set of rules that the script checks in order to determine if the satellite is in an anomalous or dangerous state, which has been determined by expert operators and/or designers of the system. While this is a very helpful diagnostic tool during operations, it only can accommodate a limited set of rules, and only diagnoses potential issues with the spacecraft. The advantage of using a model-based approach is that previously unseen anomalies could be detected and presented as possibilities in order to diagnose observed output.

At a system level, the beacon network model has a single input and a single output representing the packet received by the each station. Each block in the diagram is a different subsystem and contains its own components, each of which must be working and configured correctly for the system to be operational. Inside each ‘Bcn...’ block (seen in Figure 3.7), there is an antenna (which must be pointed correctly), a radio pre-amp (which must be enabled), the radio itself (which must be powered and tracking the satellite’s frequency), the TNC (which must be configured correctly), and the computer (which must be running the correct software). Of course, all of this equipment must also be connected to a reliable power source. In order for the station to be fully operational, each of these must be configured and functional.

#### **3.2.4. Subsystems**

Each subsystem of the beacon network has several configuration options. The radio, for example, must be tuned to the correct frequency in order to receive the signal. The target frequency is constantly changing due to the Doppler shift effect from the satellite, and so must be calculated on-the-fly by the connected computer. However, since the calculation is dependent on time, if the computer’s clock has drifted by a few seconds the radio might receive an attenuated signal (or no signal at all) because it has been set to an incorrect frequency. All of these configurations are accounted for by the model, and each is a potential point of failure to consider.

One of the strengths of using model-based reasoning is that each component has its own behavior functions built-in. That is, each component handles its own functionality and

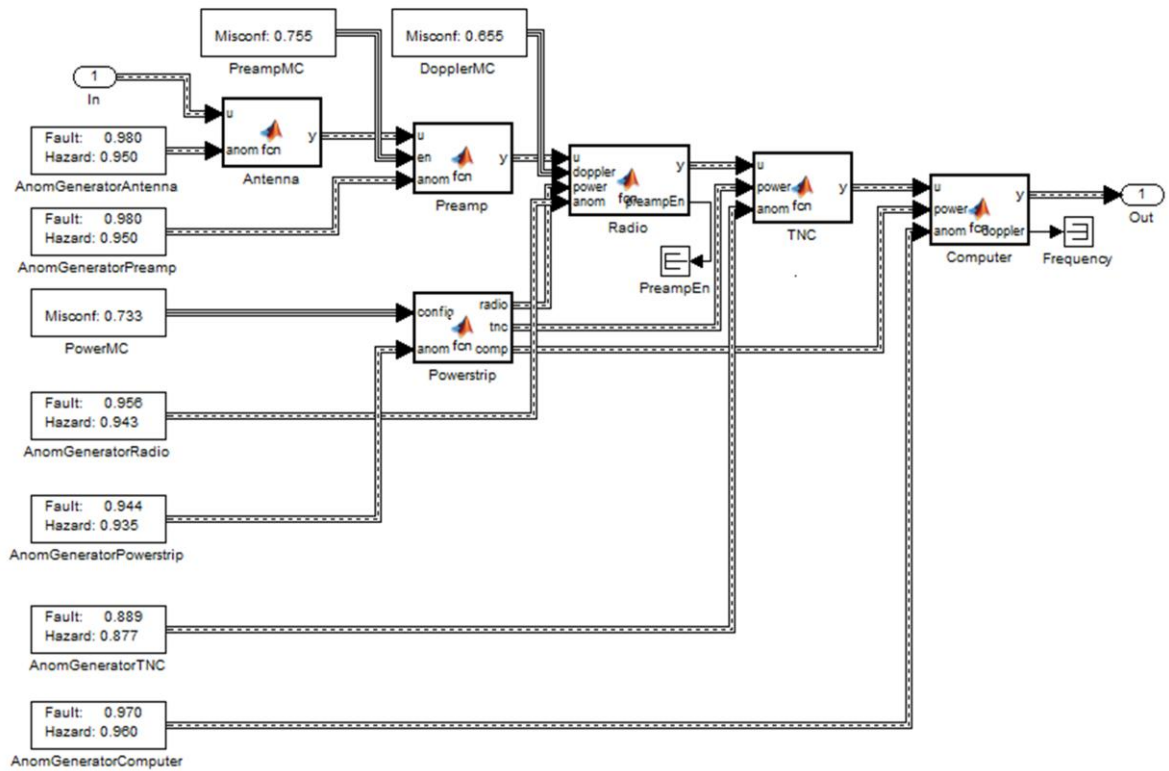
determines what happens for each anomalous case. Using the model of the satellite as an example, if the satellite's battery is under-charged it may lead to a known anomalous behavior that can be integrated into the components' behavior for more accurate predictions regarding its state. This is also a major advantage of using a graphical environment such as Simulink: while representing a basic overview of the model at the highest level, it allows the user to make the model as intricate as he or she wants in order to make the model as accurate as possible by modifying only the relevant parts while the rest remains unchanged.

### **3.3. Testing the Model**

Once the model for the beacon network was constructed it needed to be validated against the actual beacon network. In order to do this, an experiment was performed in which an anomaly was deliberately injected into the system in order to observe its real-life behavior. Prior to this, the anomaly management simulation was executed in order to generate a table of possible results with predicted confidences. In this case, the model was configured to generate results for a single anomaly. After the simulation completed, the catalog of anomalous states was consulted to determine the most likely scenario which caused the observed output. These results were then used to diagnose the actual system in order to resolve the injected anomaly.

#### **3.3.1. The O/OREOS experiment**

The O/OREOS (Organism/Organic Exposure to Orbital Stresses) nanosatellite was launched in 2010 after being designed at the NASA Ames research center located in Mountain View, California. Day-to-day operation of the satellite was performed by SCU graduate students interested in the project. The groundstation itself, consisting of redundant 3-meter (10 feet) diameter satellite dishes on pan/tilt units and a dual Yagi UHF/VHF receiving antenna, is located on the roof of SCU's engineering building. In order to test and validate associated receiving components of the groundstation, a benchtop unit was developed alongside the satellite. This unit is an identical copy of the orbiting satellite with only its experimental payload omitted. Thus it could be utilized to test and verify that



**Figure 3.7: The Simulink model of a single beacon node.** This model shows how the beacon data flows from reception by the antenna to output from the computer to the central network.

the SCU segment of the beacon network was fully functional even when the satellite is not overhead. It also provides a convenient means to realistically test this anomaly management system.

### 3.3.2. Experiment results

The validation experiment setup consisted of the aforementioned benchtop satellite system operating alongside the Beacon Network node at SCU. Before running the experiment, it was first established that the beacon network system was correctly receiving packets and observed the correct output in the form of the beacon data being displayed and parsed on the central receiving computer at 30-second intervals. This established that the entire node system from the antenna to the radio to the computer network was functioning correctly. In Figure 3.6, this corresponds with the topmost segment containing the 'BcnSCU' block. The other stations were assumed to be operational for this experiment,

as they were too far away to receive a packet from the benchtop satellite unit. However, they were assumed to be operational in the diagnosis phase of the experiment.

To test the system, the Ethernet cable connecting the beacon node computer to the Internet was unplugged, simulating a network failure anomaly at the station location. As expected, the groundstation did not receive any packets for the duration of the experiment. This corresponds with the simulation outputting no data from the SCU node, symbolized by 'NaN' output. The system output in the case of this model is a numeral representing the satellites data packet flowing through the components in the network. For simplicity, in these tests the number 761 was arbitrarily chosen to represent the packet. The output of the system is this number in the nominal case, and either 'NaN' or 'Inf' in the off-nominal case. Since each station transmits the message back to the central computer station, the outputs of the system can be represented by a vector containing the data<sup>13</sup> from the four stations: [SCU, Baylor, WPI, SLU]. Given this definition, the observed output of the system when the SCU station is not receiving packets is given as [Nan, 761, 761, 761].

In the diagnosis phase we can provide these observed symptoms as the input to the `findAnoms` script (introduced in Section 2.6). The script returns the filtered data from the Catalog shown in Figure 3.8. As can be seen, all components that could have failed are associated with the SCU branch of the beacon network model (Figure 3.6). Intuitively this makes sense: we know that data is being received correctly from the other stations so we can conclude that the satellite is operating nominally. Based on the results returned in Figure 3.8, we can conclude that the most likely scenario is that an anomaly has occurred in the radio component in the SCU Beacon Station. As part of the resolution phase the operator would then check each of the possibilities listed in order. The first anomaly to check, `DopplerMC` refers to the configuration of the Doppler-shift calculation component in the beacon-station radio subsystem. The fact that it is listed first (and has the lowest confidence value) implies that it is the most likely to fail, and therefore should be the first

---

<sup>13</sup> This data usually is gathered over multiple orbits. In this case, the parameter can represent the latest data from each station.



```

>> [res anom] = findAnoms([], yout, [-1, 761 761 761]);
0.6550: 'beacon_new/BcnSCU/DopplerMC/FlagMC
0.7330: 'beacon_new/BcnSCU/PowerMC/FlagMC
0.9430: 'beacon_new/BcnSCU/AnomGeneratorRadio/FlagHaz
0.9500: 'beacon_new/BcnSCU/AnomGeneratorPreampl/FlagHaz
0.9560: 'beacon_new/BcnSCU/AnomGeneratorRadio/FlagFault
0.9800: 'beacon_new/BcnSCU/AnomGeneratorPreampl/FlagFault
0.9850: 'beacon_new/SCUNetwork/FlagFaultNet
0.9893: 'beacon_new/BcnSCU/AnomGeneratorComputer/FlagHaz
0.9920: 'beacon_new/BcnSCU/AnomGeneratorComputer/FlagFault
0.9930: 'beacon_new/SCUNetwork/FlagHazNet
0.9975: 'beacon_new/BcnSCU/AnomGeneratorAntenna/FlagHaz
0.9996: 'beacon_new/BcnSCU/AnomGeneratorAntenna/FlagFault
1.0000: 'beacon_new/BcnSCU/AnomGeneratorTNC/FlagFault
1.0000: 'beacon_new/BcnSCU/AnomGeneratorTNC/FlagHaz
Elapsed time: 0.007237

```

**Figure 3.8: Results of filtering the Beacon Network anomaly catalog.** This figure shows the results of running the findAnoms() script on the output of the Beacon Network simulation.

thing checked. In this case, the configuration is verified by logging onto the station computer via Microsoft's Remote Desktop client, which connects over the network to the station's computer. However, when the operator tried to connect as part of the experiment, the connection failed due to the network anomaly that disconnected the station from the internet. Since no remote connection was possible, the diagnosis of a network failure became the leading failure candidate needing resolution. Upon physical examination of the components in the station, the unplugged network cable was noticed and plugged back in. Thus, the anomaly was correctly diagnosed and resolved, even though the system initially reported a configuration anomaly regarding the Doppler-shift calculation was the most likely candidate. After the station was brought back online, it was verified that the central receiving station was now receiving packets from the SCU station, effectively resolving the anomaly.

### 3.3.3. Observation points

In previous research presented in [9], the notion of "Observation Points" was used in order to narrow down the results further. These meta-components could be inserted at strategic points in the model to indicate where data was valid based on human observation. For example, an observation point could be enabled at the output of the radio

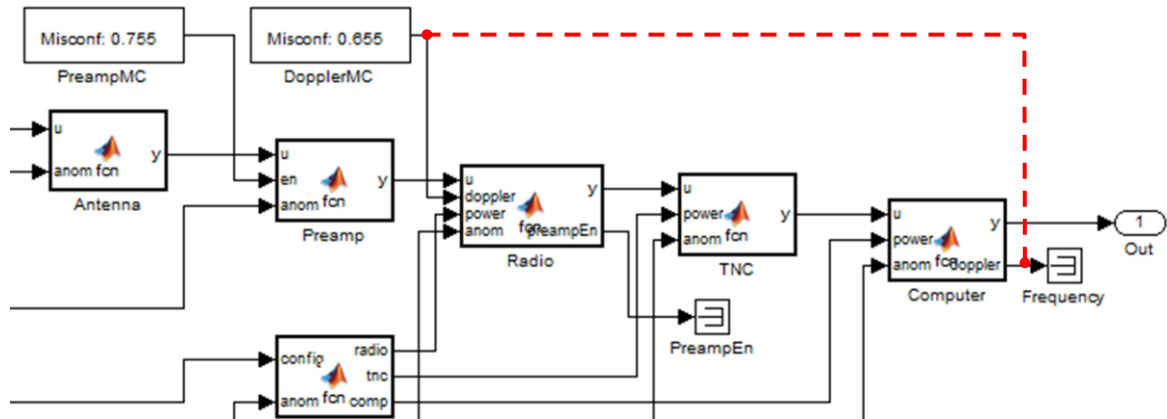
to indicate that the data packet was received correctly from the satellite, ground antenna, and radio, but was interrupted between the computer and central receiving station. This narrows the scope of where anomalies could occur to the station network component between the station computer and the SCU network (see Figure 3.6). In the current iteration of this research, observation points could be implemented easily by adding system outputs at the points to be observed, and then the filtering script `findAnoms` would be called specifying the state of the data at each observation point. However, these points are really just system outputs in disguise. If an operator or technician deems it necessary to observe the data along the way, this can be implemented as intermediate system outputs. Thus it can be easier to determine the state of the system at a given point rather than just looking at the final output data.

### **3.4. Modeling limitations**

During the development of the models for this research, several minor drawbacks of using Simulink/Matlab were discovered. This brief section explains two of the issues found and their workarounds.

#### **3.4.1. Feedback loops in certain simulation modes**

The first constraint of using Simulink to build the model-based system is seen when one component provides inputs or feedback to a previous (upstream) component. While this situation is very common for simulating motion control systems that rely on feedback, Simulink does not allow feedback for certain simulation solvers, including the “fixed-step discrete” mode being utilized for the anomaly management system. This means that if the system in question contains a component with an output port that provides the input to an upstream component, the simulation will not run, and an error will be thrown. This kind of loop occurs in each node of the beacon network at the point where the computer provides the radio with the Doppler-shifted frequency to receive data as indicated by the red dashed line in Figure 3.9. However, the computer also receives data from the radio via the TNC. This creates a loop in the model which is unsolvable without knowing information about the previous state. Since each state is discrete and independent, each block must be



**Figure 3.9: Solving the loopback issue in each beacon station.** The problematic loop is shown as a red dashed line. Instead of the computer controlling the frequency input to the radio as in the actual system, the frequency configuration is logically split from the computer. This solves the problem by splitting the configuration and operational logic into two discrete blocks. The careful reader will also observe the same technique has been utilized for the connection of the Radio [PreampEn] and the Preamp components.

executed before proceeding to evaluate the next block. Although initially this seems like a severe limitation, the solution is relatively simple: the computer can be logically separated in the model into two components (behaviors). The first component represents the handling of the data from the TNC/radio, while the second handles the configuration of any upstream components (i.e. the radio).

The model shown in Figure 3.9 demonstrates this logical separation of behaviors. The configuration component of the computer has been separated from the operational behavior logic and is symbolized by a configuration anomaly block (labeled `DopplerMC`) indicating whether the component's configuration is in a valid or invalid state. In a more detailed model, these configuration blocks could become entire components with more complex behavior to determine the desired configuration, such as the time-dependent Doppler-shifted frequency, or the component might represent the software that is calculating the frequency. Even though they are part of the same physical component, the logical separation would still hold because ultimately the goal is to model the behavior of the system, not necessarily its physical layout and connections.

### 3.4.2. Performance inconsistencies

Another limitation of using Simulink/Matlab as a simulation platform is due to its nature as an interpreted language. The Simulink modeling software was chosen because it

is intuitive and easy to use for new users of the anomaly management framework who would likely have previous Matlab experience. However, since the simulation is executed inside the Matlab environment, there are a lot of behind the scenes processes that the user does not have control over and can lead to the simulation time varying unpredictably. For example, simply having the “Variable Editor” window open to observe Matlab variables as the simulation executes can lead to an order of magnitude increase in computation time. One simulation ran in approximately 30 seconds normally, but increased 2000% to approximately 10 minutes with the variable window open. Performance issues like this can be hard to track down and eliminate, especially when they involve features that are sometimes seen as fundamental to the operation of the application. Furthermore, since Simulink runs in a virtual machine environment on top of the Windows operating system, there are restrictions that may not be obvious regarding allocated RAM and CPU resources. However, the benefits of using an easy-to-understand graphical interface outweigh most performance disadvantages. Furthermore, the system has been structured in such a way as to eliminate the necessity of very fast simulation speed.

### **3.5. Chapter summary**

This chapter first discussed the application of the enhanced anomaly management system to the full-adder and showed how to interpret the simulation results in order to provide meaningful information to the operators of the system. Next, the Beacon Network was introduced as a demonstration of how to apply the MBR system to a more complex, real-life system. An experiment was then performed on the beacon network in order to validate the results of the simulation. Finally, several limitations of the system imposed by the Matlab environment were discussed.

## 4. Conclusion

In this research program we have explored an enhancement to classic model-based reasoning diagnosis systems. The primary enhancement is to use a model to systematically generate a complete anomaly catalog computed prior to system operation, thereby reducing the operation-time analysis to simply querying the catalog for anomaly scenarios consistent with the system's configuration and identified symptoms. Contributions to this effort include re-architecting the manner in which anomaly permutations are executed as well as the manner in which models are established and integrated. Another enhancement involves the introduction of a confidence rating capability that can be used to sort diagnosis conjectures in order to focus operator attention on the most probable anomaly scenarios once troubleshooting commences. The technique was demonstrated using a full-adder model as a simple proof-of-concept and then applied to the more complex SCU beacon network. The system was designed to be easy to use, easily extensible, and flexible enough to be applied to other complex systems in the future without extensive knowledge of how model-based reasoning systems are implemented. Furthermore, the enhanced system provides the benefits of both model-based reasoning and expert systems in order to effectively diagnose anomalous behaviors.

In addition to providing a framework to integrate anomaly likelihoods into a system model, the proposed reasoning system enhances the computation time necessary to analyze a system model. This allows for a much more thorough understanding of how anomalies influence the output since much less time is needed between simulations. The building blocks are provided to build an anomaly reasoning system which is easily maintainable and delivers results quickly to help keep complex systems operating nominally. The utilization of Simulink's graphical interface allows faster design and iteration than would be possible with a purely code-based system, yet is flexible enough to allow for complex behavioral processes to be modeled. In this way, a more complete model can be developed in order to resolve anomalies in a systematic manner, especially in remote

engineering systems. Furthermore, the ranking of anomalies has been shown to make the anomaly resolution phase much easier and faster.

#### **4.1. Future Work**

The MBR approach explored in this research lays the foundation and serves as a proof-of-concept for future students to apply the technique to other engineering systems. The models presented serve as relatively simple examples of how to utilize the anomaly management engine, but its robustness and flexibility must be tested by applying the technique to other systems. While there may exist some optimizations and improvements, the framework that has been established can be utilized in the future with minimal effort to directly support complex systems.

In addition to utilizing the system to model more complex systems, its scalability should be tested. While Matlab and Simulink handle models that generate several thousand permutations very well, in some cases it was noticed that the simulation time increases disproportionately when the number of anomaly permutations reaches around 100,000. While this is not a major drawback (the simulation results are ideally generated long before they are utilized anyway), it is currently unknown if this will be an issue for larger systems. There may be techniques to optimize the simulation speed in Simulink, but again, since the simulation is meant to be executed in the pre-operational phase, the simulation time should be of little concern. It has also been observed that simulation times can vary as much as 400%, even for simple systems such as the full-adder. This is likely due to background work done by Matlab and Simulink prior to the simulation running, or simply how the operating system assigns resources to the applications during execution. While this is not a major limitation, there may be some optimizations that can be done in order to reduce this variance. Furthermore, other tools like parallel computing systems or distributed computing platforms could be utilized to enhance the computations.

##### ***Automatic anomaly detection***

This project was implemented with the assumption that there would be a separate detection process which would provide an operator with the information needed to query

the Catalog for possible diagnoses. Towards the end of the project, it was realized that the automatic detection of anomalies could be implemented as an extension to the `findAnoms` script by simply comparing provided system outputs to the simulated outputs stored in the Catalog. This would enable the system to be used to automatically detect then diagnose anomalies in realtime. Unfortunately, this feature has been left as future work due to time constraints.

With the above enhancement and a few other features, the system could even send results outside the Matlab workspace, for example to a relational database such as MySQL. Then, coupled with a website frontend, the MBR system could run in the background and autonomously detect and report anomalies as they occur. This would enable fully autonomous anomaly management and would be the ultimate form of this work.

#### ***Overcoming technique limitations***

An inherent drawback of the current approach is that the system is defined by only the component behaviors and connections between them. That is, we analyze possible issues that occur within defined behaviors, but not unforeseen outside influences or connections. For example, an electrical short-circuit is a common fault seen in the field that could be caused by moisture intrusion, corrosion, or simply a loose wire. A more subtle unforeseen connection could be extreme thermal characteristics of one component influencing another, causing the second to exceed its operating specifications and cause undefined behavior. In this model-based approach, these kinds of unforeseen connections are not able to be analyzed by the system unless explicitly defined in the model.

Furthermore, by design, this approach focuses on enumerating the behavioral complexities defined in the model and systematically evaluating the outputs in a discrete variable domain. As future system behaviors are modeled with more complexity, there is the possibility of modeling non-linear or chaotic systems. However, analyzing these highly state-dependent systems can quickly become very difficult, if not computationally infeasible, to evaluate. Future iterations of this project could explore other modeling techniques to integrate very complex behaviors and predict unforeseen connections between components.

### ***Hybrid systems***

Given that there are advantages and disadvantages to every formal reasoning approach, whether model-based, experiential, data-driven, or something entirely novel, the Robotics Systems Lab at SCU is interested in developing a hybrid system that can combine the best parts of each approach to analyze a system from multiple perspectives and reveal new insights into anomaly management. This project developed a model-based reasoning approach to prioritize possible anomaly categories with the intent of accelerating the diagnosis and repair of faulted systems. This approach, combined with other reasoning approaches, would make a very powerful comprehensive tool for the diagnosis and repair of complex mechatronic systems.

#### **4.2. Current Utilization**

At the time of this writing, the model-based reasoning system discussed in this paper is already being put to use by students in the RSL, using a remotely controlled vehicle built specifically for testing anomaly management approaches. The system consists of a three-wheeled triangular robot with an onboard microcontroller, radio transceiver, and physical toggle switches to simulate anomalies. It is controlled by a joystick attached to a computer which then transmits commands to the robot via the radio transceiver pair. There is also a redundant motor driver that can be activated to resolve anomalies autonomously after one is manually simulated and automatically detected. Then, using the enhanced MBR system, the operator can consult the ranked list of possible anomalies to determine the point of failure in the robot. After an initial learning curve, the anomaly management framework was used to build the model of the robotic system. The entire system is still a work in progress, but the enhanced MBR framework seems to be functioning as expected. In the future, this could be expanded upon to include more automation features and learn how it can be best applied to other systems as well.

#### **4.3. Final Thoughts**

The techniques explored in this paper demonstrate the usefulness of model-based reasoning approaches enhanced with individual component failure likelihoods. While this



work provides the framework implementing this concept, there is still plenty of room for growth and expansion. Furthermore, there is a growing opportunity in many different industries for an easy-to-implement anomaly management system. While proprietary, one-off anomaly management systems are often incorporated into engineering systems such as satellites, these systems are often exclusively designed to work with a single system [10]. A generalized, model-based approach similar to the work presented in this research would enable much more robust and rapid resolution of anomalies as they occur in a system. This research is just the starting point; there is still much to be done.

## References

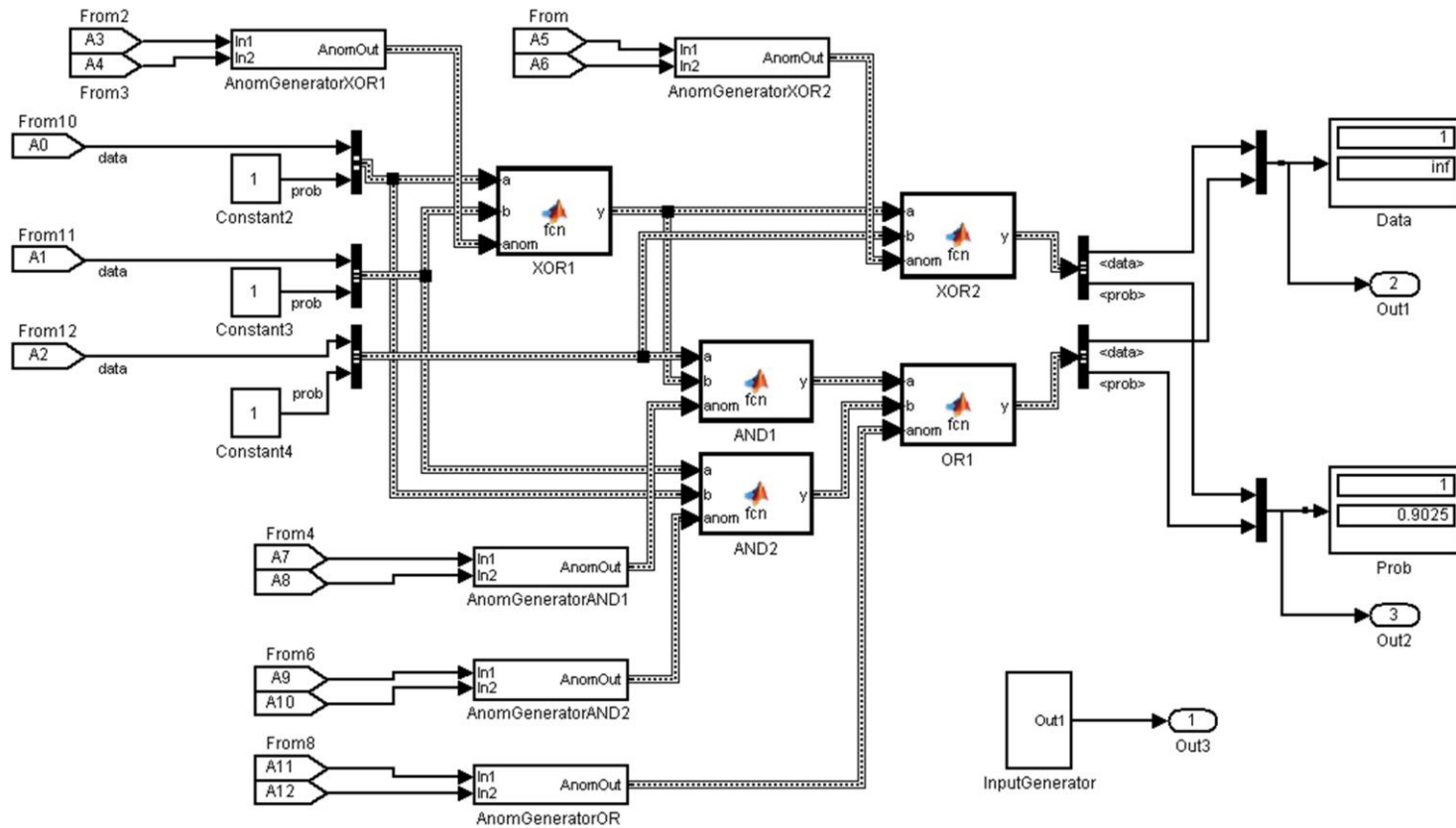
- [1] NASA, "NASA Mission Control Fact Sheet," 1 May 2015. [Online]. Available: [https://www.nasa.gov/centers/johnson/pdf/160406main\\_mission\\_control\\_fact\\_sheet.pdf](https://www.nasa.gov/centers/johnson/pdf/160406main_mission_control_fact_sheet.pdf). [Accessed 2015].
- [2] J. Gleick, "A Bug and a Crash," *New York Times Magazine*, 1 December 1996.
- [3] D. Rice, "Unmanned Antares Rocket Explodes on Launch," 29 October 2014. [Online]. Available: <http://www.usatoday.com/story/tech/2014/10/28/nasa-rocket-explodes-wallops-island/18080871/>. [Accessed 18 May 2015].
- [4] C. Kitts, "Managing Space System Anomalies Using First Principles Reasoning," *IEEE Robotics & Automation Magazine*, 2006.
- [5] B. J. Feder, "BUSINESS TECHNOLOGY; Repairing Machinery From Afar," 30 January 1991. [Online]. Available: <http://www.nytimes.com/1991/01/30/business/business-technology-repairing-machinery-from-afar.html>. [Accessed 5 May 2015].
- [6] R. Davis, "Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence," H. E. Shrobe, Ed., San Mateo, Morgan Kaufmann, 1988, pp. 297-340.
- [7] W. J. Clancey, "The Epistemology of A Rule-Based Expert System: A Framework for Explanation," Stanford University, Stanford, 1981.
- [8] W. A. Beech, D. E. Nielsen and J. Taylor, "AX.25 Link Access Protocol for Amateur Packet Radio, v. 2.2," July 1998. [Online]. Available: <https://www.tapr.org/pdf/AX25.2.2.pdf>. [Accessed 2015].
- [9] A. Young, "Initial Flight Results for an Automated Satellite Beacon Health Monitoring Network," *SSC10-XII-1*, 2010.
- [10] P. Zoetewij, J. Pietersma, R. Abreu, A. Feldman and A. J. van Gemund, "Automated Fault Diagnosis in Embedded Systems," Delft University of Technology, Delft, 2007.
- [11] S.-H. Liao, "Expert system methodologies and applications—a decade review from 1995 to 2004," *Expert Systems with Applications*, 2004.
- [12] A. Ajith, "Rule-based Expert Systems," Oklahoma State University, Stillwater, OK, 2005.

- [13] 2015. [Online]. Available:  
<http://www.autorepair.ae/dubai/images/inside/diagnostic/check%20engine%20Dubai.jpg>.  
[Accessed 1 June 2015].
- [14] C. A. Kitts and M. A. Swartout, "Beacon Monitoring: Reducing the Cost of Nominal Spacecraft Operations," *Journal of Reducing Space Mission Cost*, vol. 1, no. 4, pp. 305-338, 2002.
- [15] "Small Satellite Missions - OOREOS," 2 March 2011. [Online]. Available:  
[https://www.nasa.gov/mission\\_pages/smallsats/ooreos/main/](https://www.nasa.gov/mission_pages/smallsats/ooreos/main/). [Accessed 2 June 2015].
- [16] B. C. Williams and J. de Kleer, "Diagnosing Multiple Faults," *Artificial Intelligence*, no. 32, 1987.

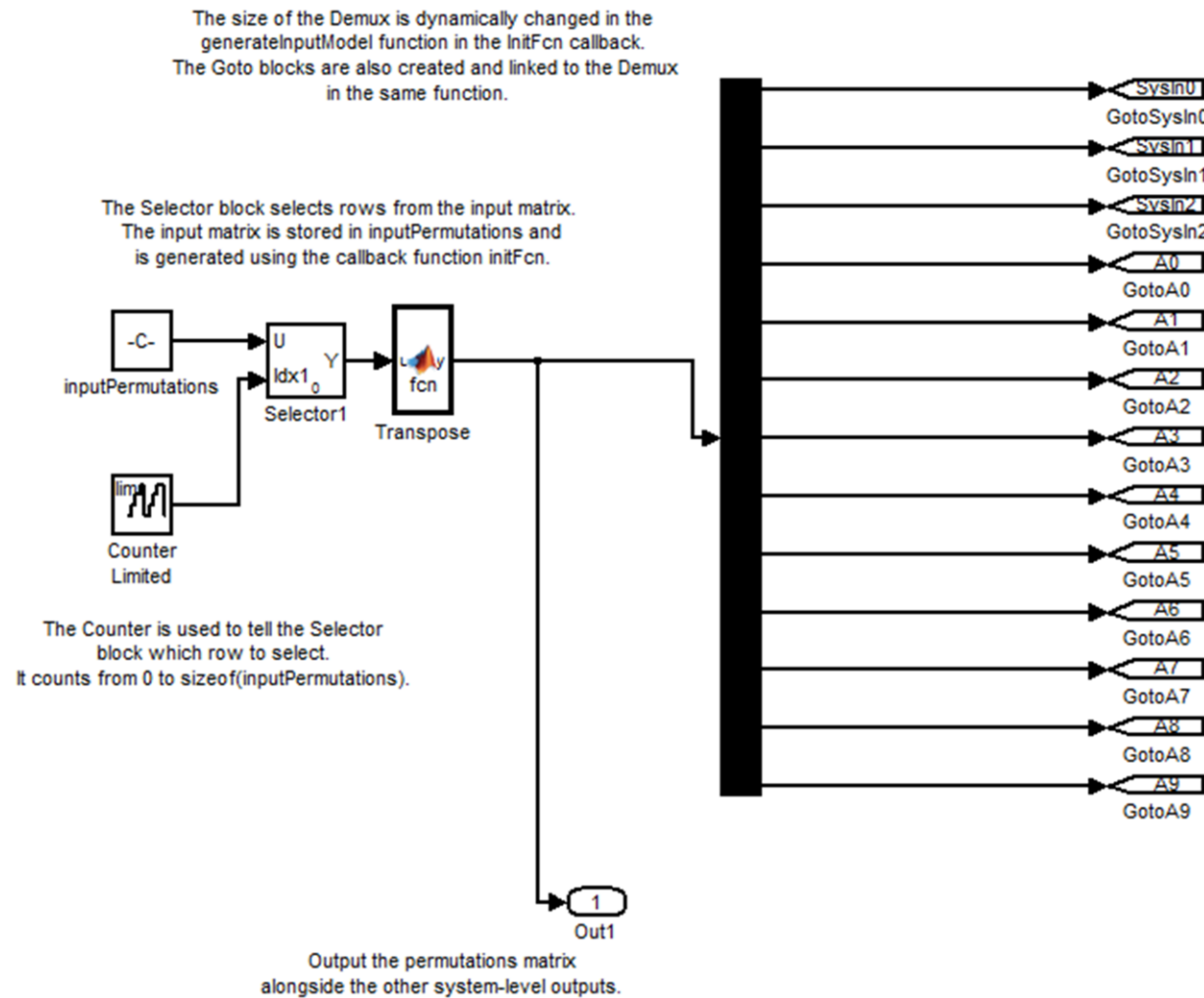
## Appendix A – Supplementary Tables and Diagrams

**Table A-1: An excerpt from the Catalog of the full-adder model.** The first three columns are the typical inputs to the full adder, while the next ten are anomaly permutations associated with each anomaly block in the model. The next four columns are the full-adder outputs with associated confidences. Each confidence is on the interval [0,1], with a higher number corresponding to a less likely scenario. The last column is the overall confidence of the associated permutation of anomalies.

System Inputs			Anomaly Inputs										Outputs				
A	B	C <sub>in</sub>	XOR1-fault	XOR1-hazard	XOR2-fault	XOR2-hazard	AND1-fault	AND1-hazard	AND2-fault	AND2-hazard	OR1-fault	OR2-hazard	S	C <sub>out</sub>	S-conf	C <sub>out</sub> -conf	Overall-conf
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	NaN	1	0.990	0.990	0.990
1	1	1	1	0	1	1	1	1	1	1	1	1	Inf	1	0.985	0.985	0.985
1	1	1	1	1	0	1	1	1	1	1	1	1	NaN	1	0.98	1	0.980
1	1	1	1	1	1	0	1	1	1	1	1	1	Inf	1	0.975	1	0.975
1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	0.970	0.970
1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0.965	0.965
1	1	1	1	1	1	1	1	1	0	1	1	1	1	NaN	1	0.960	0.960
1	1	1	1	1	1	1	1	1	1	0	1	1	1	NaN	1	0.955	0.955
1	1	1	1	1	1	1	1	1	1	1	0	1	1	NaN	1	0.950	0.950
1	1	1	1	1	1	1	1	1	1	1	1	0	1	Inf	1	0.945	0.945



**Figure A-1: The full Simulink model of the full-adder circuit.** The “From” blocks are used to receive the anomaly input permutations from the corresponding “Goto” blocks in the Input Generator block shown in Figure A-2. The likelihoods are set inside the AnomalyGenerator blocks.



**Figure A-2: The “InputGenerator” subsystem.** This block is used to send each permutation (row) of the anomaly input matrix to the corresponding “From” blocks attached to the anomaly input blocks.

**Figure A-3: The Simulink model of the Beacon Network.** This figure shows the system-level input and outputs as well as the auxiliary components to implement the anomaly management system.

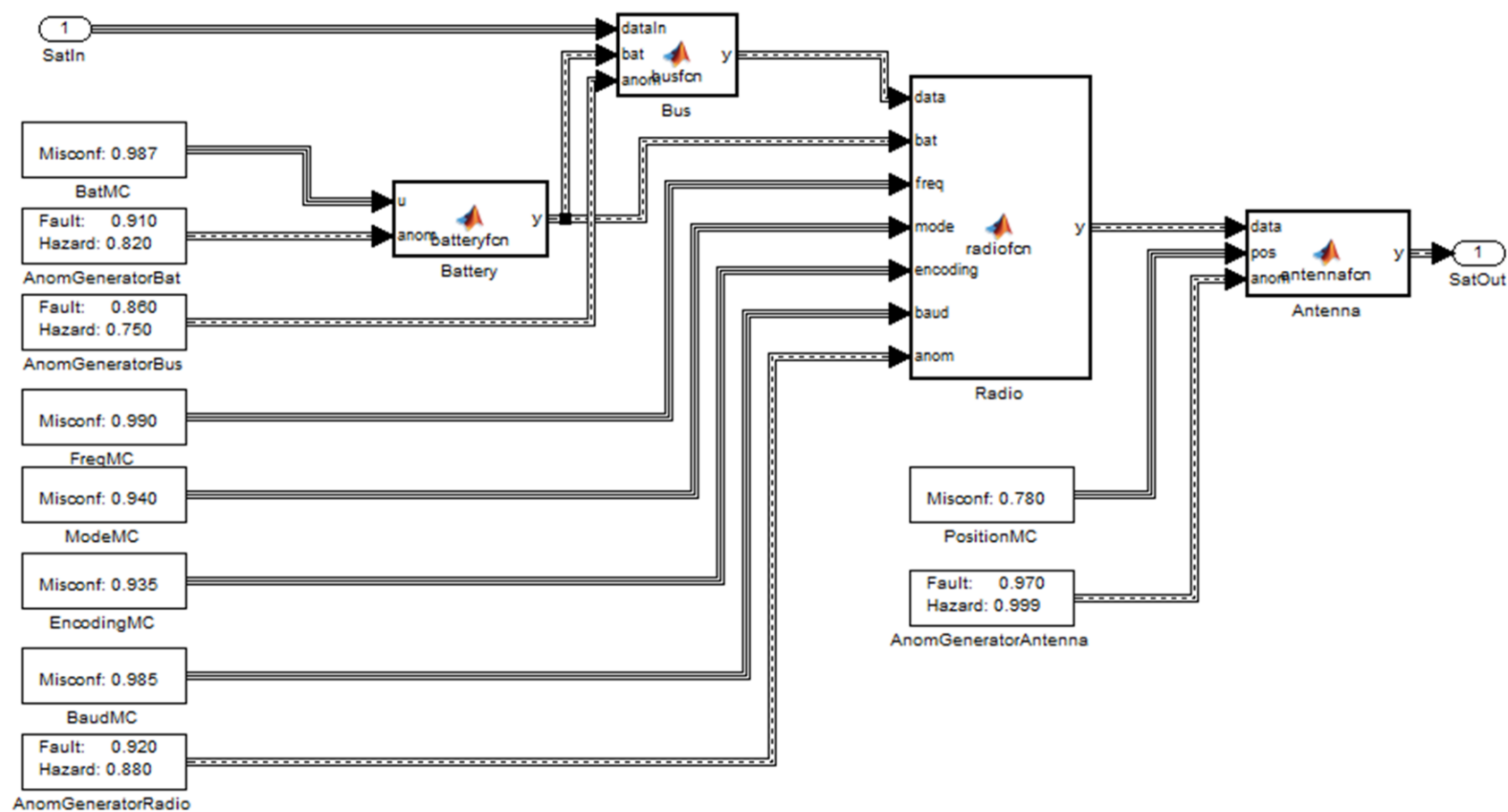


Figure A-4: The Simulink model representing the O/OREOS satellite as part of the Beacon Network model.



## Appendix B – Code Samples

**Snippet 1: The code for an AND gate.** The function takes its standard inputs (a,b) and the anomaly structure input (anom), then calls doAnsis(), which determines if an anomaly has occurred. This line is common for all blocks; function handles to specific behavior are passed in and called by the doAnsis() function. The calcProb() and defaultAction() functions can implement any operation in order to make the model more accurate. For example, defaultAction() could implement different behaviors for a fault or a hazard coming from the previous components.

```
function y = fcn(a,b,anom)
%#codegen

y = doAnsis(@defaultAction, @calcProb, anom, a,b);
gate_name = 'AND';

end

% Calculate the confidence based on the three values (2 inputs and the
% fault/hazard value.
function p = calcProb( anomProb, inputs)
    p = inputs(1) * inputs(2) * anomProb;
end

% The nominal case function takes the inputs and returns a Boolean value.
function f = defaultAction(aIn,bIn)
    dataIn = [aIn.data, bIn.data];
    % Short-circuit if there's an anomaly previously down the line.
    if any( dataIn == 0 )
        f = 0;
    elseif any(isnan(dataIn)) || any(isinf(dataIn))
        f = NaN;
    else
        f = double( and(aIn.data,bIn.data) );
    end
end
```

**Snippet 2: doAnsis.m.** This function is called by each component with different function handles passed as arguments, centralizing the anomaly detection logic in one function. This enables each block to implement a common interface which assists development of the model.

```
function y = doAnsis( fcndefault, fcnpob, anom, varargin )
%#codegen

if nargin <= 4
    fprintf('Incorrect usage! Need 4 or more inputs. ');
    y = varargin{1};
    return;
end

% preallocate to satisfy the compiler.
y = varargin{1};

% An array to hold the index of the varargin that has an anomaly.
inputAnomIndex = [];
inputData = [];
inputAnom = 0; % boolean for if the inputs have an anomaly.

% An anomaly on the anomaly input.
anomStatus = ~anom.hazard(1) || ~anom.fault(1);

% Preallocate the probabilities array. The size is the size of the varargin
% array.
probabilities = ones(1,numel(varargin));
% Loop through the inputs and set the probabilities. Also detect if there
% has been an anomaly on one of the inputs.
for k = 1:numel(varargin)
    probabilities(k) = varargin{k}.prob;
    % append the anomaly index to the anomalies array.
    if varargin{k}.prob ~= 1
        inputAnomIndex = [inputAnomIndex k];
    end
    inputData = varargin{k}.data;
end

% If the inputData array is empty, there is no anomaly on the data inputs.
if ~isempty(inputAnomIndex)
    inputAnom = 1;
end
```

[doAnsis.m continued]

```
% Both anomaly inputs are valid and working correctly (no anomalies) and
% there are no anomalies on the input lines.
% Case 1: no anomalies anywhere.
if ~anomStatus && ~inputAnom

    % Call the "valid" action with all the inputs.
    y.data = double(fcndefault(varargin{:}));
    y.prob = 1;

% Case 2: anomaly in block, none in inputs
elseif anomStatus && ~inputAnom
    if ~anom.hazard(1)
        y.data = NaN;
        y.prob = anom.hazard(2);
    elseif ~anom.fault(1)
        y.data = Inf;
        y.prob = anom.fault(2);
    end

% Case 3: no anomaly in block, anomaly in inputs.
elseif ~anomStatus && inputAnom
    y.data = fcndefault(varargin{:});
    y.prob = fcnpb(1,probabilities);

% Case 4: Anomaly in an input and current block.
elseif anomStatus && inputAnom
    if ~anom.hazard(1)
        y.data = NaN;
        y.prob = fcnpb(anom.hazard(2),probabilities);
    elseif ~anom.fault(1)
        y.data = Inf;
        y.prob = fcnpb(anom.fault(2),probabilities);
    end

else % should never ever reach here.
    y.data = 666;
    y.prob = 123;
end

end
```