INSTRUMENTATION, MODELING, AND SOUND METAMODELING FOUNDATIONS FOR COMPLEX HYBRID SYSTEMS

by

NATASHA AMELIA JARUS

A DISSERTATION

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

2021

Approved by:

Sahra Sedigh Sarvestani, Advisor
Ali R. Hurson, Co-Advisor
Minsu Choi
Jonathan Kimball
George Markowsky

ABSTRACT

Many of our critical infrastructures, from power grids to water distribution networks, are complex hybrid systems that use software to control their non-trivial physical dynamics. These systems must be able to capably serve their purpose, while also being reliable, dependable, safe, secure, and efficient. Representation and analysis of these features requires the creation of several distinct models. These models may encode design goals or be derived from collected instrumentation data, reflecting both how a system ought to operate and how it does operate. It is essential to ensure that all of these models consistently and accurately describe the same system. Adding or removing detail in one model may necessitate changes to several others.

This work focuses on system instrumentation, modeling, and metamodeling. Our instrumentation and modeling work studies the behavior of control systems when exposed to electromagnetic disturbances. These disturbances, which may lead to data corruption, system crashes, or hardware damage, present a challenge to engineers. We develop instrumentation for monitoring systems for such disturbances, methods for analyzing the data from our instrumentation, and models of system function which can detect electromagnetic disturbances, including many that do not cause user-visible failures.

Metamodeling offers a means of relating disparate models of a system, describing changes to models, and propagating those changes to other models. Our metamodeling work focuses on adding and removing detail from models — model *refinement* and *generalization*, respectively — and on connecting models that use different formalisms — model *transformation*. In order for these operations to produce meaningful results, we must ensure that they are *sound*; that is, they must produce models which describe, to the greatest extent possible, the same system as the models from which they are produced. We begin by creating a theory of abstract interpretation for system modeling. This theory defines a relationship between models and systems and enables verification of the soundness of our metamodeling operations. From this foundation, we create model refinement and generalization operations for specific modeling formalisms. Finally, we show how these operations can be used to perform sound model transformations.

ACKNOWLEDGMENTS

I extend my sincere gratitude and appreciation to my advisor, Dr. Sahra Sedigh Sarvestani. Her mentorship across two years of undergraduate research and seven years of doctoral research has been invaluable both in guiding my work and in developing my skills as a researcher, communicator, and educator. Her patience and kindness have softened the difficulties of research and of growing as a person. Her technical and social knowledge have broadened my perspective not only on the complexity of the systems that pervade our lives but also on the diversity of the people for whom we engineer those systems.

The depth of my gratitude to my loving wife, Hannah Goodman, and my dear metamour, Shasta Johnson, is inexpressible. Without your love and care, I would not be the person I am today. Your support through harrowing times and life-changing decisions has enabled me to pursue challenging goals and to achieve them. Thanks to your hard work and personal sacrifice, I have enjoyed the privilege of being able to prioritize my research and teaching responsibilities. Your thoughtful guidance and comments on every aspect of my work has enabled me to present polished work and navigate complex social situations. With all my being, thank you.

To my co-advisor, Dr. Ali Hurson, thank you for your insightful comments, your willingness to share your extensive knowledge, your friendly support, and your dry wit.

I thank my committee members, Dr. Jonathan Kimball, Dr. George Markowsky, and Dr. Minsu Choi, for giving your time and thought to my work, for offering fresh perspectives and specialized knowledge, and for your advice on academic matters and on my career.

My labmates deserve a special commendation. A Ph.D. dissertation is never the work of only one person; without your hard work and dedication, significant parts of this research would not have been completed. Thank you to Mark Woodard, Koosha Marashi, Ty Morrow, Madison Childress, Evan Hite, and Connor Jones. Special thanks to Michael Wisely for his support for many of my undertakings in graduate school. I extend my deepest appreciation to Laika Klingbeil and Joel Schott, who exerted incredible effort to achieve research accomplishments.

In addition, I thank my collaborators in the Missouri S&T Center for Electromagnetic Engineering, Dr. Chulsoon Hwang, Dr. Omid Hoseini Izadi, Runbing Hua, and Dr. Pratik Maheshwari, for sharing their expertise in electromagnetic engineering, access to their laboratory

equipment, and their time in planning, designing, and performing experiments. Jerry Tichenor and Kevin Hasner also deserve thanks for their assistance and advice in creating instrumentation for our work.

I am very grateful to my friend Michele White, whose knowledge of electrical engineering broadened my understanding of my research and whose ineffable enthusiasm and boundless energy made our collaboration a joy.

Three of my course instructors deserve particular praise. Dr. Lindgren Johnson, thank you for piquing my interest in a career in writing and research and for your encouragement to publish my first research paper. Dr. John Seiffertt, you taught me to understand the language of mathematics in a completely new way. And especial thanks to Dr. Ilene Morgan for inspiring me to study abstract algebra, for her thoughtful comments on my research work, and for her friendship and support during my graduate career.

Finally, I thank my chickens, Gracie, Pig, Hank, Annie, Bertha, and Lion for the companionship, love, and joy they brought me during my study. I am especially grateful to Bertha for her support through a particularly dark time and for her efforts hengineering our instrumentation.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

APPENDICES

LIST OF TABLES

1. INTRODUCTION

The expansion of computers' capabilities, especially their ability to replace labor costs with capital acquisition, has precipitated a proliferation of complex systems incorporating digital control. Most critical infrastructure is an example of this: power grids containing sensors and software-controlled circuit breakers and other power flow control mechanisms, water distribution networks with automated valves, and even vehicles which contain complex networks of computers and are guided by computerized traffic control systems. These systems are built to a variety of design requirements. Not only do they need to be capable of meeting operational requirements but they also must be dependable, safe, secure, sustainable, and easy to repair and upgrade. We refer to these systems, characterized by significant interdependence between non-trivial physical dynamics and digital control, as *complex hybrid systems* [1].

Models are used to better understand, predict, and control these complex systems. Designers may want to evaluate the performance of a system or predict how it will fail. Instrumentation and monitoring software use models to determine whether a system is operating as intended. During maintenance and upgrades, models assist in identifying failures and ensuring changes are compatible with existing infrastructure. Models may encode a design goal for a system, such as a model power grid which assumes linear operation of components, or they may be derived from empirical data gathered experimentally. A myriad of modeling formalisms have been created to capture aspects of system operation.

While this work is not exclusive to them, we take a particular focus on models of system failure. We construct models of a system's reliability, a probabilistic measure of how long it will remain functional; and resilience, a proportional measure of the extent of performance degradation and speed of recovery from a failure. This leads to an investigation of generalization and refinement for reliability models, exploring in depth the information present in a particular reliability modeling formalism and the connections between different reliability models of the same system. In addition, we construct approximate models of system function which are used to identify operational anomalies. These anomalies arise from electromagnetic interference acting on computing hardware. No analytical model of the effect of this interference on software have been developed, so models are constructed entirely from empirically gathered data. These studies demonstrate the variety of perspectives included in dependability modeling: interdependencies among components, failure recovery procedures, and interactions between low-level physics and high-level control software.

The other main focus of this work is a study of relationships between models of a system. Firm distinctions between instrumentation-based modeling and design-based modeling cannot be drawn: conclusions drawn from instrumentation, design, and modeling all inform each other. As understanding of a system improves, models change to incorporate this knowledge and changes to instrumentation or system design may also be made. We propose work toward explicitly identifying the connections among models and correctly propagating changes through the models of a system.

1.1. INSTRUMENTATION AND MODELING OF THE EFFECTS OF ELECTROMAGNETIC INTERFERENCE AND ELECTROSTATIC DISCHARGE ON SOFTWARE

For critical infrastructure and other safety-critical applications, it is essential to understand the failure modes of control electronics and to improve their dependability when possible. Two threats which predominantly affects control systems are electrostatic discharge (ESD) and electromagnetic interference (EMI). Exposure to large electric, magnetic, or electromagnetic fields can induce spurious voltages or currents in computer hardware. These faults can lead to display or audio glitches, corrupted data, program crashes or mis-execution, system restarts, or permanent hardware damage. Thus, understanding and mitigating the effects of ESD and EMI (henceforth, electromagnetic disturbance, or EMD) on digital controllers is essential.

A significant challenge in developing a comprehensive understanding of how EMD impacts control systems is determining how hardware-level EMD alter software operation. Though a control system is not typically what is imagined as a complex hybrid system, it is true that control systems themselves couple digital control with the non-trivial physics of their hardware realizations. An additional complexity is the challenge of instrumentation itself: placing hardware instrumentation can be invasive and expensive, but software instrumentation is inherently unreliable as it is not decoupled from the system and thus can also be affected by EMD. Consequently, most studies of EMD characterize either hardware-level operation of a test system or high-level software operation of commercially available systems.

EMD mitigation is viewed through two distinct perspectives. In the case that a system is being designed to operate in a fixed environment over a long period of time, the electromagnetic characteristics of the environment can be determined and the system designed and verified to withstand the EMD present. While this approach can be very effective, upgrades to the system require high-effort re-verification. Furthermore, in situations where the environment is not well understood or has uncontrolled aspects — for instance, an operator plugging a device into a USB

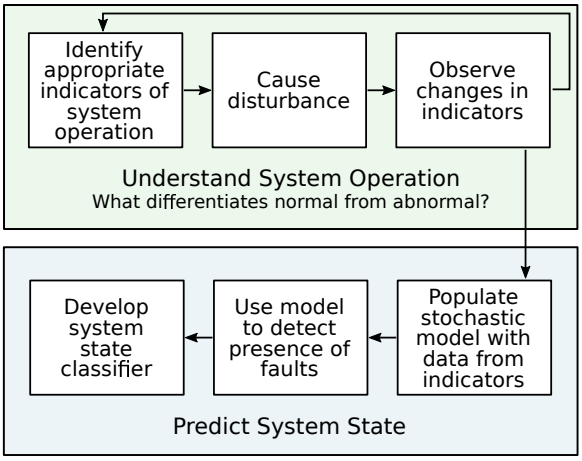


Figure 1.1. EMI instrumentation and modeling approach

port — determining whether a system is sufficiently shielded can be challenging. On the other hand, the EMD resilience perspective [2] proposes using instrumentation to continually monitor systems in the field, detect when their operation is impacted by EMD, and determine whether additional EMD mitigation is needed on an ongoing basis.

We propose an instrumentation and modeling approach designed to enable EMD resilience with fine-grained software instrumentation of system peripherals as depicted in Figure 1.1. This approach provides a low-overhead means of observing system peripheral operation, giving further insight into the effects of EMD on software which may not be visible to an end-user. These effects cannot be inferred from either software or hardware specifications of a system, nor do any analytical models exist to predict their occurrence. Consequently, we use experimental data to derive models of "normal" peripheral operation, allowing us to detect changes in operation that may be attributable to EMD. Furthermore, we propose statistical methods of validating the ability of our instrumentation to capture such operational anomalies.

1.2. COMPLEX HYBRID SYSTEM METAMODELING

Designers develop such systems through a process of careful modeling and simulation of multiple aspects of a system. The combination of continuous-time physics and discrete-time control creates a system whose performance is challenging to model. Furthermore, non-functional modeling perspectives overlap with each other and with performance models. For example, a reliability model may incorporate the timing of control inputs to a system, safety models and reliability models may both contain information about component failures, and security and safety models are often

created in tandem to prevent safety overrides from being overlooked by security measures [3]. All of these models have complementary perspectives on the system they model. Our work is focused on identifying overlaps between the various ways of modeling a system, or *modeling formalisms*, and using this information to more efficiently create and refine models.

Quantifying the information common to multiple models presents several challenges. Fundamentally, models approximate the operation of the system they describe [4]; even if an unambiguous modeling formalism was available, modelers would not necessarily benefit from fully specifying a system's operation. The modeling process is necessarily iterative; it starts with a general, high-level specification, then more detail is added as needed. Furthermore, modelers planning maintenance or upgrades to existing systems often work with incomplete technical specifications and components in unknown condition. In identifying overlapping information among models, we need to account for partial, approximate knowledge of the system being modeled.

Models may have differing perspectives on their shared knowledge — they represent the same aspects of a system differently. For example, a performance model of a power grid may represent each transmission line as an inductor-capacitor-resistor circuit, but the corresponding reliability model may view that transmission line as a single component. Alternatively, the performance model may not include circuit breakers as they are irrelevant to "ideal" performance, but a reliability model may incorporate them into a more precise model of cascading failures in the system. Even when information (e.g., transmission line dynamics or circuit breaker locations) is known about a system, it may not appear in all models of that system. Again, we must account for the fact that models represent only an approximation of the system they model.

We attempt to address these challenges by creating *metamodeling* formalisms that reduce the effort required to model a system and ensure that various models of a system are consistent. Models aim to study a system or phenomenon through an abstract representation; metamodeling is the study of these representations and can be considered a higher level of abstraction [5]. More detail can be added to a model by refining it, or extraneous detail removed by generalizing it. Such detail might be removed because it happened to be an incorrect or undesirable assumption or to reduce the complexity of a model to make it tractable for evaluation. Furthermore, we can transform the information in one model to another model expressed in a different modeling formalism. This can be used to add new modeling constructs to an existing modeling formalism, such as transforming a reliability model of a water distribution network into a reliability model that incorporates computer-controlled

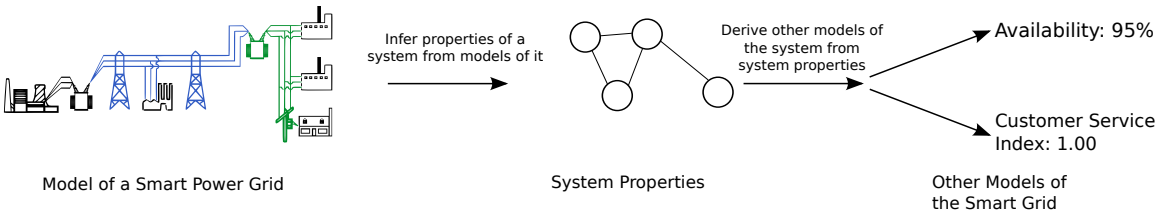


Figure 1.2. Relationship between properties and models of a system

valves intended to mitigate failures. It could also be used to produce, say, a partial specification of a reliability model from a model of a power grid's architecture and performance. We believe such a metamodeling process, illustrated in Figure 1.2, can be created.

The concepts of *transformation*, *refinement*, and *generalization* are of great interest from a scientific perspective. The process of defining a modeling formalism and refinements and generalizations for it consists of understanding that formalism's perspective of a system, encoding that perspective into a language, and determining the operations that can be represented in that language. Doing so can lead to a better understanding of the limitations of a given formalism and the assumptions it makes about system operation. Model transformation translates a formalism's perspective (and its operations) into the language of a different model formalism. By creating transformations between formalisms we can, for example, determine the effect that changing the topology of a system has on that system's reliability.

1.3. RESEARCH PROJECTS AND CONTRIBUTIONS

My doctoral research has focused on modeling and metamodeling of complex systems. This has taken the form of several interrelated projects.

The first project was focused on creating software instrumentation and analytical models to detect, understand, and mitigate the effects of electromagnetic and electrostatic interference in embedded system peripherals [6–9]. We were able to successfully detect electrostatic interference in a USB bus control chip using only software instrumentation executing on the system to which the bus controller was attached. This approach presented several challenges: the bus controller's operation needs to be monitored frequently enough that anomalies are detected before being overwritten by its driver, but this precise monitoring needs to be accomplished with very low overhead and without interfering with or negatively affecting the system's operation. Our instrumentation approach is unique in its ability to illuminate the minutiae of how a failure affects the operation of a system.

Developing this approach requires an understanding of the effect of interference on the hardware of a system, as well as understanding of how these effects propagate to the software executing on that hardware.

A second project focused on dependability modeling of smart power grids [10]. We applied the Markov Imbeddable Structure (MIS) reliability modeling technique to the IEEE 9-bus system to determine its reliability. An intelligent control device, a static synchronous series compensator, was added to the system and the reliability model updated to incorporate the new device. We were thus able to determine how the addition of an intelligent control device affects the system's overall reliability. Furthermore, we modeled the resilience achieved by two different recovery strategies for a particular failure and compared them based on two different quantitative metrics.

The final project aims to create a sound mathematical basis for metamodeling complex systems [11–14]. To achieve this goal, we create a framework for defining transformations, refinements, and generalizations in a way that meaningfully relates them to model fidelity and enables the definition of properties of these operations, which in turn enable reasoning about them. This approach is based on *Abstract Interpretation* [15], a mathematical formalism initially created by Radhia and Patrick Cousot for static program analysis. Abstract interpretation provides a way of translating a potentially intractable problem, e.g., determining the set of states reachable by a computer program, into a tractable problem by abstracting (eliminating some detail from) that problem. For instance, consider a computer program that takes an integer as input and produces an integer as output. Rather than exactly predicting the program's output for a specific input, we might abstractly interpret that program as one that takes as input a range of numbers and produces as output a range of numbers. The result of our abstract analysis might overestimate the range of results from the program, but the abstract interpretation technique ensures that our result is *sound*; that is, it includes all possible outputs of the program.[1]

We think of models as abstractions of a system's semantics — its structure and operation. We propose a theory of abstract interpretation for system modeling, which enables relating the abstract, approximate operation of models to the exact operation of the system they represent. With this metamodeling perspective, our modeling goals become faithful transformations of system semantics. Properties of the system being modeled are used to connect various modeling formalisms. For each modeling formalism, we specify what these properties are, as well as how to translate them

[1]This problem is tractable in spite of the halting problem because the abstract interpretation can return the range of "all integers" — in effect, it is allowed to say "I don't know." Achieving more specific results requires more detailed abstract interpreters.

into models and vice versa. Finally, by showing that these properties and this translation satisfy certain constraints, we can derive transformation, refinement, and generalization operators for each modeling formalism.

I have also conducted an extensive literature survey on modeling of dependability attributes of cyber-physical systems. The survey has been submitted to the ACM Transactions on Cyber-Physical Systems [16].

My research contributions and publications to date are the following. The seventh publication relates to two different research contributions and is listed under both.

- Software-based detection and analysis of electromagnetic and electrostatic interference:

    1. A. Sabatini, N. Jarus, P. Maheshwari, and S. Sedigh, "Software instrumentation for failure analysis of USB host controllers," in *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, May 2013.
    2. N. Jarus, A. Sabatini, P. Maheshwari, and S. Sedigh Sarvestani, "Software-based monitoring and analysis of a USB host controller subject to electrostatic discharge," in *Proceedings of the CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, Jun. 2020.
    3. N. Jarus, A. Sabatini, P. Maheshwari, and S. Sedigh Sarvestani, "Detection, analysis, and prediction of the effects of electrostatic discharge on a USB host controller," submitted to *IEEE Transactions on Electromagnetic Compatibility.*
    4. N. Jarus, J. Schott, L. Klingbeil, and S. Sedigh Sarvestani, "Observation, analysis, modeling, and classification of USB host controller operation under electro-magnetic interference," submitted to *IEEE Transactions on Electromagnetic Compatibility.*

- Dependability modeling of smart grids:

    5. M. N. Albasrawi, N. Jarus, K. A. Joshi, and S. Sedigh Sarvestani, "Analysis of reliability and resilience for smart grids," in *Proceedings of the IEEE 38th Annual Computer Software and Applications Conference*, Jul. 2014.

- Survey on modeling of non-functional aspects of cyber-physical systems:

    6. N. Jarus, M. Woodard, K. Marashi, S. Sedigh Sarvestani, J. Lin, A. Faza, and P. Maheshwari, "Survey on modeling and design of cyber-physical systems," submitted to *ACM Transactions on Cyber-Physical Systems.*

- Creating a metamodeling approach based on abstract interpretation:

    7. N. Jarus, S. Sedigh Sarvestani, and A. R. Hurson, "Models, metamodels, and model transformation for cyber-physical systems," in *Proceedings of the Seventh International Green and Sustainable Computing Conference (IGSC)*, Nov. 2016.

    8. N. Jarus, S. Sedigh Sarvestani, and A. R. Hurson, "Facilitating model–based design and evaluation for sustainability," in *Proceedings of the Ninth International Green and Sustainable Computing Conference (IGSC)*, Oct. 2018.

- Formalizing model transformation using abstract interpretation of models:

    7. N. Jarus, S. Sedigh Sarvestani, and A. R. Hurson, "Models, metamodels, and model transformation for cyber-physical systems," in *Proceedings of the Seventh International Green and Sustainable Computing Conference (IGSC)*, Nov. 2016.

    9. N. Jarus, S. Sedigh Sarvestani, and A. R. Hurson, "Formalizing cyber–physical system model transformation via abstract interpretation," in *Proceedings of the IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, Jan. 2019.

- Creating sound generalization and refinement operators for models:

    10. N. Jarus, S. Sedigh Sarvestani, and A. R. Hurson, "Towards refinement and generalization of reliability models based on component states," in *Proceedings of the Resilience Week Symposium (RWS)*, Nov. 2019.

    11. N. Jarus, S. Sedigh Sarvestani, and A. R. Hurson, "Refinement and generalization of reliability models based on component states," in preparation.

1.4. OUTLINE

The first half of this document focuses on creating instrumentation and models to detect the effects of EMD on software operation. The second half is concerned with the metamodeling of complex hybrid systems.

My research contributions towards EMD detection and analysis are summarized in Section 2, along with relevant literature. Section 3 details the instrumentation methodology and experiment design. Statistical modeling and analysis of instrumentation data is discussed in Section 5. In Section 6, approaches to classifying peripheral operation are created and compared.

Section 7 provides an overview of my complex hybrid system metamodeling work and places my research in the context of related literature. It also contains a more thorough discussion of transformation, refinement, and generalization during the system modeling process. Sections 8, 9, and 10 detail my work on the identified research tasks. Section 8 describes how abstract interpretation can be applied to metamodeling and discusses the requirements we must meet in order to do so. Refinement and generalization of models is articulated in Section 9, which includes a case study on reliability models. Section 10 defines model transformation in terms of our abstract interpretation framework.

Finally, Section 11 summarizes this research and proposes future extensions.

The appendices contain detailed information on the various mathematical and modeling formalisms used in this work.

2. PART 1: INSTRUMENTATION AND ANALYSIS FOR EMD

This research investigates the manifestation of electromagnetic disturbances (EMD) in the operation of the software of system peripherals. We create a fine-grained software instrumentation approach to EMD monitoring, apply statistical techniques to validate that it is capable of detecting changes in operation when the system is exposed to EMD, and use classification approaches to differentiate normal and EMD-exposed operation.

Modern electronics are increasingly susceptible to such disturbances, which may take the form of silent failures, software crashes, system resets, or permanent hardware damage. Existing hardware-based instrumentation approaches are invasive and costly, and software-based instrumentation methods focus on high-level effects of EMD, such as screen glitches. Our work explores the gap between these extremes: we have created low-level software instrumentation that can detect even EMD which does not manifest as a user-visible failure. This approach may be applied to commercial hardware in field-test conditions to ascertain whether EMD mitigations are sufficient, assist with root cause analysis of EMD propagation and EMD-caused failures, or for continual monitoring and detection as part of an EMD resilience program.

We group both Electrostatic Discharge (ESD) and Electromagnetic Interference (EMI) into one term (EMD) because our instrumentation approach can be seamlessly applied to both. ESD and EMI differ in their physical manifestation: ESD is caused by a very fast equalization of charge potentials, whereas EMI is caused by coupling of an electromagnetic wave into a system. ESD potentials range from sub-1kV to 10kV, with equalization times in the nanosecond range. It may be introduced into a system via direct injection or by electric (E-field) or magnetic (H-field) coupling. EMI is characterized by high-frequency (tens of kHz to GHz) waves with field strengths of 10kV/m or more and manifests in a system via electromagnetic field coupling. Despite these differences, both effect similar faults in hardware (e.g., bit flips, transistor latch-up, bond wire destruction) and software (e.g., data corruption, system resets). Thus we group the two into EMD for our instrumentation and analysis purposes.

The work presented here reflects two instrumentation and analysis projects, the first researching ESD in a Samsung System on a Chip (SoC) and the second researching EMI in a RockChip SoC. For both systems, we instrument the USB host controller, which manages all communication across the USB bus and is thus particularly vulnerable to EMD. The ESD project offered promising

preliminary results which motivated the EMI project; the EMI project reinforced these results and expanded on the statistical and classification techniques applied. Across both projects, we make the following research contributions:

1. A method of using software instrumentation to observe and record the operation of system peripherals (Section 3) [6–9].
2. An approach for analyzing instrumentation data using statistical techniques to validate the instrumentation technique and characterize the change in system operation due to EMD (Section 5) [6–9].
3. An approach to classifying system operation to distinguish when the system is experiencing EMD (Section 6) [8, 9].

The remainder of this chapter discusses existing approaches to ESD and EMI instrumentation and system operation monitoring. In Section 2.1 we characterize the effects of ESD on a system and describe existing approaches to instrumentation and modeling of those effects. Section 2.2 likewise characterizes EMI and describes hardware and software instrumentation methodologies. In addition to EMI- and ESD-specific approaches, a variety of system monitoring and sensor anomaly detection techniques have been investigated; the relevant ones are summarized in Section 2.3. Section 2.4 places our work in the context of this body of literature.

2.1. ESD INSTRUMENTATION APPROACHES

ESD-induced failures can be broadly categorized as either *hard failures* or *soft failures* [17]. In this context, a hard failure permanently damages the system so that components must be replaced. Soft failures, on the other hand, can be recovered from; these failures are further characterized into three levels based on the visibility of the failure and the action needed to recover from it:

Level 1) The system automatically recovers with no user-visible faults or loss or corruption of data. Often this recovery is possible due to ESD-robust hardware and fault-tolerant control protocols.

Level 2) The system experiences a system-level manifestation, such as momentary screen or data corruption, but recovers without intervention.

Level 3) The system crashes or requires the user to perform an action, such as resetting the system or unplugging and re-plugging a device, to recover from a fault condition.

These failures are studied using a variety of hardware- and software-based techniques.

Numerous studies have investigated the relationship between ESD interference and level 2 and 3 soft failures. Hardware ESD fault injection with direct injection and field injection probes is described in [18–20]. These studies characterize integrated circuit (IC) immunity to ESD. The sensitivity threshold for each IC was determined by injecting ESD at increasing voltages and observing when errors occurred. In these studies, only user-visible errors, such as screen glitches or hardware resets, were investigated.

Izadi *et al.* [21] extend this fault injection process by mapping the ESD sensitivity of a single-board computer.The injection probes are attached to a 2-D scanner that sweeps them across the board. At each point on a grid over the CPU, ESD is injected and the level at which the device becomes vulnerable is recorded. The resulting map can be used to identify traces and components that are at risk for ESD damage. Mapping is carried out at various CPU loads and clock speeds; the authors determine that the system is most susceptible under heavy load and low clock speed.

Vora *et al.* [22] study user-visible soft failures in a microprocessor, a microcontroller, and an FPGA. In particular, they observed a relationship between CPU load and likelihood of display flicker on a microprocessor, indicating that ESD was coupling to the CPU chip rather than to the display itself. Furthermore, they observed that the likelihood of certain failures—process termination and display flicker—depend on the program executing at the time of the ESD event.

Investigating level 1 soft failures and understanding the root causes of higher-level soft failures requires the ability to observe a system's operation at a high level of detail. Vora *et al.* [22, 23], Feng *et al.* [24] use a custom microcontroller running code which monitors register values and system interrupts to study the effects of ESD on CPUs. While too invasive to use on a system performing additional tasks, this approach gives a very fine-grained view of observable soft failures. In particular, the authors observe numerous multiple bit errors in IO registers and frequent spurious interrupt triggers.

The effect of ESD on USB devices in particular has also been investigated. Maghlakelidze *et al.* [25] develop an automated testing system for studying soft failures in a USB interface on a single-board computer. The system is characterized by injecting ESD pulses of varied voltage and pulse width into specific IC pins. Soft failures are observed based on data transmission rate and error messages in kernel logs. Under positive voltage injections, most failures did not require user intervention; however, negative voltage injections produced numerous severe soft failures. Koch *et al.* [26] further test USB-related soft failures and determine that likelihood of failure is also dependent

on the state of the USB protocol, i.e., what type of packets are being transmitted at the time of the injection. Root cause analysis shows that many failures are caused by ESD coupling to the power domains in the USB controller rather than to data lines.

While some soft failures are not user-visible, they may still be observable by software monitoring of low-level system operation. Yuan *et al.* [27] continuously poll the status of a phase-lock loop (PLL) embedded in a microcontroller; if the PLL unlocks, it can be assumed that the system has experienced an ESD shock. While this approach provides an excellent measure of ESD events on the microcontroller, it cannot measure peripheral ESD events because most peripherals do not contain a separate PLL that can be monitored by the microcontroller.

Another case study of low-level system monitoring is carried out in [28] on a wireless router. A debugging serial port on the router logs every context switch performed by the processors, giving an approximate record of the execution path taken by processes running on the router. This data is collected into system function graphs of both reference operating function and ESD-exposed function. Several graph metrics are applied to these graphs; differences in metric values indicate that soft failures can be observed by this monitoring technique.

2.2. EMI INSTRUMENTATION APPROACHES

The effects of EMI can be categorized according to their mechanism, broadly *no effect*, *interference*, or *destruction*; by their duration, ranging from *observed only for the duration of an EMI event* to *permanent hardware damage*; and by their criticality [29]:

U *Unknown*: Unobserved or indeterminate due to other failures.

N *No effect*: System fulfills its mission without disturbance.

I *Interference*: Disturbance does not influence the system's function.

II *Degradation*: Disturbance impairs system capability or efficiency.

III *Loss of main function*: Disturbance prevents system from functioning.

Assessing the systemic effect of EMI on a sub-system incorporates these three perspectives plus knowledge of the sub-system's criticality in the function of the system.

Low-level investigation of EMI incorporates both modeling [30] and experimentation [31, 32]. These studies provide a detailed understanding of EMI-related component faults and suggest physical mitigation techniques [33]. However, their precision and experimental rigor (e.g., requirements for PCB design and instrumentation) make them infeasible for analyzing large-scale systems or consumer hardware.

System-level EMI analysis for electromagnetic compatibility purposes primarily focuses on EMI events with type II or III criticality. The effect of interference is determined based on reported alarms, errors recorded in system logs, loss of network traffic, user-visible "glitches" in displays or audio, or system crashes or resets [34, 35]. Such studies can characterize whole systems or focus on the vulnerability of specific sub-systems [36]. A variety of confounding factors must be considered when assessing system vulnerability to EMI, including EMI frequency and waveform, simultaneous exposure to multiple waveforms, waveform reflections, mechanical vibrations, and intermodulation effects [37, 38].

While shielding is an effective method for mitigating EMI effects [39], it is often not feasible or cost-effective, especially for systems that are frequently upgraded or used in environments where EMI exposure has not been characterized. EMI *resilience* takes a complementary approach wherein systems are continually monitored for interference and improved as required [2, 40].

As part of an EMI resilience program, existing system peripherals have been investigated for use as EMI sensors. Errors from USB and PS/2 devices and serial data communication rates have been found to be correlated with EMI exposure [41, 42]. Furthermore, analog sensors including temperature sensors for hard drives and processors and wifi and cellular received power indicators also show anomalies under EMI exposure [43]. In addition to these studies, software instrumentation for ESD detection has been investigated in [7, 8, 44].

2.3. SYSTEM MONITORING AND ANOMALY DETECTION APPROACHES

While not directly related to ESD events, software-based as well as combined hardware and software system monitoring approaches have been studied extensively. Watterson and Heffernan [45] outline research related to monitoring for runtime verification. System state is monitored by some combination of hardware and software; this information is then used to verify that the system is operating within specification. A software-specific study of fault monitoring is carried out in [46]. The authors present a taxonomy of runtime monitoring approaches and discuss various system requirements for different monitoring techniques.

Choudhuri and Givargis [47] develop a mixed hardware and software approach for logging non-deterministic behavior in embedded systems. They modify a compiler to emit code that logs messages to an attached storage system, reducing processing overhead on the low-power embedded hardware being monitored. Reinbacher *et al.* [48] create a tool that converts an embedded system software specification into both an executable and a configuration for a hardware monitor. The hardware monitor interfaces with the embedded CPU and its communication buses and verifies the operation of the system.

Delgado *et al.* [46] develop a software-based monitoring system for ATMs by instrumenting the drivers for each hardware component to measure state and performance. A runtime checker uses the resulting data to determine if the system is operating correctly. If not, recovery actions can be taken to restore system availability.

False data injection, where sensor values are spoofed to hide a physical fault, can be detected through clustering or statistical analysis [49, 50]. Detection of faults or attacks from instrumented software or processor performance counters is also feasible via a combination of classification and statistical correlation [51, 52].

2.4. SUMMARY

The distinction of our work is twofold. Our instrumentation captures much finer-grained events, offering a more detailed view of how system operation is affected by EMD. For EMI events, we gain visibility into U or I criticality as well as II or III criticality; for ESD, we make some level 1 soft failures visible. We achieve this with minimal impact to system operation, as the visibility and operation of EMD-induced failures can change based on the processes running on the system. Furthermore, we pioneer the use of categorical time series statistics and classification algorithms in characterizing and identifying EMI-related operation anomalies. We use these tools to validate our instrumentation and to identify anomalous sequences of events as a foundation for building real-time EMI detection software or for root-cause analysis of EMI-caused failures.

3. SOFTWARE INSTRUMENTATION APPROACH

The goal of our instrumentation approach is to precisely capture the operation of a system peripheral and to record that operation for analysis. EMD in an embedded system peripheral can lead to incorrect peripheral operation, here termed 'anomalous execution'. This includes unexpected changes to values in the peripheral's memory, failure to communicate, and peripheral resets. We hypothesize that these effects are visible to software running on the system's CPU and thus should be detectable by monitoring software. Furthermore, peripherals with connections to external devices, such as sensors, can be used to monitor the system for EMD.

This work can be applied to many computer peripherals, but we present it in the context of a USB Host Controller on an embedded system running Linux. We have chosen to instrument a USB host controller for our experiments, since USB is widely used and peripherals are readily available. The role of the USB host controller is to handle the physical communication between the host computer and the various USB devices connected to it, as shown in Figure 3.1. All control and data communications to or from attached USB devices go through the host controller. The host controller connects directly to the USB power and data lines, making it an excellent point to detect EMD entering the system through those points.

The host computer communicates with the host controller via memory-mapped registers. The host controller updates these registers as it performs various operations. It can also alert the host computer that an event has happened via interrupts. As these registers and interrupts are the only insight the host computer has into the internal operation of the host controller, our instrumentation must be based on these data.

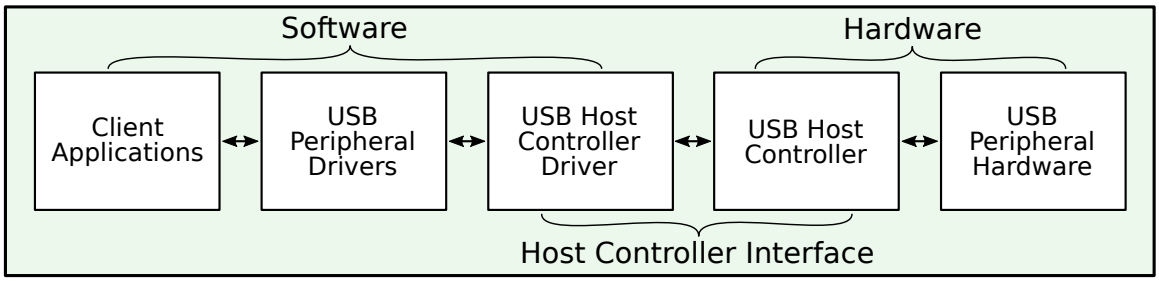


Figure 3.1. USB subsystem block diagram

Our work focuses on non-invasive monitoring of the effects of EMD using software that does not interfere with normal system operation. We primarily study changes in register values, as those values control the operation of the peripheral device. Each system state is represented by an n-tuple consisting of the values of the peripheral registers at a specific time. Some of the changes in system state will be part of normal operation. When EMD is induced, however, we anticipate observation of new (abnormal) states or unexpected transitions between normal states. These abnormal states and transitions can indicate that the system is experiencing EMD. Our analysis avoids state-space explosion by considering only states that are observed during system operation.

The USB host controller is a complex piece of hardware whose operation is quite opaque to the system CPU. We cannot inspect any of its internal registers or microcode execution process. The extent of our visibility into its operation is the control registers, which are exposed to the system. We monitor these register values as an approximation of the host controller's internal state. Recording snapshots of register values as the system performs USB operations gives a trace of host controller execution. The goal of this research is to use these traces to identify anomalous operation potentially caused by EMD.

While the host controller's registers are mapped in system memory, Linux's memory protection mechanisms prevent unprivileged programs from reading them. Thus, we must insert some software into the Linux kernel to allow us access to those memory addresses. This approach captures register values every time they are relevant to software executing on the CPU.

The remainder of this section details our process for creating instrumentation for two USB host controllers. We begin with an unsuccessful attempt, then a successful approach to instrumenting a host controller for ESD event detection. Following this, we replicate our approach on new hardware for EMI detection.

3.1. ESD INSTRUMENTATION

We instrumented the USB host controller of a Samsung SoC. This host controller conforms to the Open Host Controller Interface (OHCI) specification, which details the control registers and appropriate values. Further details on the hardware can be found in Section 4.1.1.

3.1.1. Initial Approach. Our first design focused on directly reading USB register values from their physical memory addresses. We adapted the Myregrw [53] software to better suit our needs as a softprobe for ESD. This software consists of a Linux system driver and a program that

communicates with it. The driver reads the values of requested physical memory addresses. The user-level program reads a configuration file specifying which addresses to request, repeatedly requests the data at those addresses, and stores that data to a file.

We configured Myregrw to record the control and status registers for the USB host interface.[1] We injected ESD into the host controller while Myregrw continuously sampled the registers. In theory, ESD-caused changes should appear in the recorded register values.

However, the sampling rate of this softprobe was not sufficient to observe ESD-induced errors. We empirically determined that the sampling rate of the software running on the system used in our experiments is, on average, 342Hz. We can assume in the worst case that the system executes one instruction per cycle and would reset a register value after one instruction. The system we ran these experiments on has a 400MHz clock (see Section 4.1.1 for details). We can calculate a pessimistic lower bound on the sampling rate by assuming the worst case scenario of a register value changing every clock cycle. In this situation, we would log on average $\frac{342}{400*10^6} * 100 = 0.000856\%$ of its values. As we have twenty-three registers to monitor, the effective sampling rate will be even lower. Considering this low probability and the lack of information recorded from our experiments, we devised a new measurement methodology with a higher sampling rate capable of recording additional register values.

A confounding issue with this approach is the competition for access to these values between the Myregrw driver and the USB host controller driver. By default, Linux drivers have execution priority over any user applications, meaning that it would be nearly impossible to read all of the register values after an error but before the USB host controller driver modifies the registers. Therefore, we developed a new methodology that, in addition to providing a faster sampling rate, ensures the register values are recorded before the USB host controller driver can modify them.

3.1.2. Improved Approach. We first enabled the debugging configuration already present in the USB host controller driver. We then modified the drivers for the USB host controller. The host controller driver consists of several functions that are called when certain events occur; for example, `ohci_irq` is called when an IRQ occurs for the host controller. We configured each function to first log its name and the values of the host controller registers to the system log. These modifications allow us to observe not only register state changes but also the order in which different driver functions are called. An example of such a log entry is shown in Figure 3.2.

[1] These are mapped in the physical address range `0x49000000`–`0x49000014`.

```
function: ohci_irq
HcControl: 0x83
HcCommandStatus: 0x4
HcInterruptStatus: 0x24
HcInterruptEnable: 0x8000005e
HcInterruptDisable: 0x8000005e
HcHCCA: 0x338b1000
HcPeriodCurrentED: 0x0
HcControlHeadED: 0x339b2000
HcControlCurrentED: 0x0
HcBulkHeadED: 0x339b2080
HcBulkCurrentED: 0x0
HcDoneHead: 0x0
HcFmInterval: 0xa7782edf
HcFmRemaining: 0x80002760
HcFmNumber: 0x921d
HcPeriodicStart: 0x2a2f
HcLSThreshold: 0x628
HcRhDescriptorA: 0x2001202
HcRhDescriptorB: 0x0
HcRhStatus: 0x8000
HcRhDescriptorA: 0x2001202
HcRhPortStatus[0]: 0x103
HcRhPortStatus[1]: 0x100
```

Figure 3.2. OHCI host controller register snapshot and state

This approach is minimally invasive as the driver modifications are minor and do not affect the logic of the driver itself. While this induces a constant overhead, in practice the overhead is small and can be reduced by using, e.g., a buffer to hold log entries and a separate program to write those entries to a file. The sampling rate is variable, but it exactly captures the CPU-visible operation of the USB host controller.

3.2. EMI INSTRUMENTATION

The instrumentation for the EMI experiments follows the paradigm laid out in Section 3.1.2. The logging format was modified to improve logging speed, and performance was measured. We apply it to a Rockchip SoC with a USB host controller which conforms to the Enhanced Host Controller Interface (EHCI) specification [54]. Our instrumentation is performed by making slight modifications to the EHCI host controller driver. This driver has a number of procedures which are run when the host controller causes an interrupt. We modify these procedures to record a snapshot of all the host controller register values before the driver modifies them. In this fashion, we can precisely capture the system-visible operation of the host controller without additional overhead from redundant checks.[2]

An example snapshot is shown in Figure 3.3. Notably, the EHCI specification has far fewer registers than the OHCI specification.

[2] 8.6% wall clock overhead; 37.9% CPU cycle overhead.

Register snapshot:

```
Jul 29 21:38:38 rockpro64 kernel:  [ 2789.449136] [EHCI DEBUG DUMP ehci_handshake] 0x10025, 0x8009,
0x37, 0x1abb, 0x0, 0xdfb5d000, 0xdfb5b000, 0x0, 0x1, 0x10025, 0x10025
```

Corresponding state:

Driver Function	`ehci_handshake`
Command	`0x10025`
Status	`0x8009`
Interrupt Enable	`0x37`
TX Fill Tuning	`0x0`
"Configured" Flag	`0x1`
USB Mode	`0x10025`
USB Mode Extended	`0x10025`

Figure 3.3. EHCI host controller register snapshot and corresponding state

4. EMD EXPERIMENT DESIGN

The instrumentation approach described in Section 3 is a means to differentiating "normal" and "abnormal" peripheral operation. As a basis for this characterization, we use our instrumentation to record peripheral operation both under standard operating conditions and under exposure to EMD. The data collected in these experiments forms the foundation for further analysis (Section 5) and classification (Section 6) of peripheral operation.

The issue of determining what constitutes "normal" operation is quite challenging, and depends on the environment in which a system is operating and the purpose for which it is being used. One approach would be to place the system in an anechoic chamber during the experiments which characterize "normal" operation. While this would precisely capture system operation under no interference, it may miss operation which is well within "normal" operation but which is caused by typical levels of interference in a given environment.

Another aspect of determining "normal" operation is choosing the operations a user will perform on a system. On the one hand, characterizing "normal" operation by including a wide variety of operations is essential to producing a dataset useful for detecting EMD in the field. On the other, characterizing "normal" operation by a specific sequence of operations is encouraged by good experimental design practices.

Thus, in this work, we instead characterize "baseline" system operation, defined by a specific sequence of operations performed on a system operating on a laboratory bench. In addition, instead of characterizing "abnormal" operation, we characterize "EMD-exposed" operation. The distinction between these two is that, if the field strength is low enough, EMD may not couple into the system, or it may not propagate to cause a software fault. It is not guaranteed that data recorded from an EMD-exposed experiment contains operational anomalies, though it is likely. EMD-exposed experiments are carried out with the system executing the same sequence of operations as the baseline experiments.

4.1. EXPERIMENT DESIGN

This section describes our experimental design for both ESD and EMI detection, including EMD generation, instrumented hardware, and experimental conditions.

4.1.1. ESD Experiment Design. The system used for tests was the FriendlyArm Mini2440 embedded development platform with a Samsung S3C2440 ARM926T processor [55]. Its USB host interface conforms to the Open Host Controller Interface specifications [56]. The system ran a modified Linux kernel based on the version 2.6.29 kernel downloaded from the FriendlyArm website [55]. We set up the system with our logging software and connected it to a PC to control it during the tests. During testing, a standard USB 2.0 flash drive was connected to the system's USB port. To ensure that the host controller is active during ESD injection, we copied a large file to or from the flash drive during tests.

To thoroughly characterize system operation, ESD interference was injected using electric (E) field and magnetic (H) field probes powered by a transmission line pulse (TLP) generator. For each probe, multiple tests were run with varying pulse voltages. In addition, different sizes of probes were used to adjust the intensity of the fields injected. The E-field probe does not have an orientation; we positioned it across the USB port or over the host controller IC. E-field interference was injected using an EZ-3 probe at voltages between 500 and 5500 volts with a pulse width between 0.1 and 0.25 seconds. Because the magnetic fields generated by the H-field probe are directional, we conducted tests with the probe in parallel with and perpendicular to the data and control lines. We used two probes, the HX-5 and the HX-1T2, injecting ESD betweeen 500 and 8000 volts with pulse widths between 0.1 and 0.6 seconds. The system was more resilient to H-field interference, allowing us to perform H-field tests with more intense ESD conditions than were possible with E-field tests.

4.1.2. EMI Experiment Design. For our experiments, we selected the well-documented and affordable Rock Pro 64 system which uses a Rockchip RK3399 System-on-Chip that has built-in USB 3.0 and 2.0 host controllers. We focus on the USB 2.0 host controller, which conforms to the EHCI specification [54]. This specification details both the physical power and communication requirements for the host controller and the contents of the host controller's registers. We installed a Linux operating system on this device running version `4.4.202-1237-rockchip-ayufan` of the Linux kernel. The kernel contains a driver — a software program — that configures and manages communication with the host controller.

When performing experiments, we need to generate traffic on the USB bus to stimulate host controller operation and, consequently, the creation of register snapshots by our instrumentation. To keep test conditions consistent, we connect only one USB flash drive to the system and copy the same

file to it in each test. This produces constant traffic to the USB host controller for the duration of one EMI injection. To speed up the rate at which experiments were performed, we developed a script which automated this task as well as the task of saving the log of snapshots after an experiment.

Two types of experiments were performed, termed *baseline* and *EMI-exposed*. For both tests, our instrumented driver was used and USB traffic was generated as described above. The baseline tests allow us to capture the typical operation of the host controller and are performed with the device, consisting of the computer and flash drive, sitting on a bench. Our assumption is that any interference arising from these conditions is typical of what the device should withstand under normal operation and thus not meaningful to distinguish from truly interference-free operation.

EMI-exposed tests are conducted with the device placed between the plates of the EMI generator described in [57]. This generator repeatedly produces a 30 MHz damped sinusoidal electric field wave which interferes with the device operation. Tests were carried out to characterize the limits of what the device can withstand without resetting the host computer or causing permanent harm to the device — conditions under which any software instrumentation will struggle to accurately capture system operation. Tests are carried out with the device placed both vertically and horizontally in the generator, as orientation can affect how the EMI enters the device.

4.2. EXPERIMENTAL DATA

This section describes the datasets gathered from our experiments and discusses details of processing the recorded data for analysis and classification.

4.2.1. Collected ESD Experiment Data. The log files generated by the instrumented driver consist of lines each having a timestamp, register name, and associated register value. We parse these lines into n-tuples containing snapshots of register values at the time of each function call. The sequence of n-tuples from each log constitutes an *execution trace*.

Many of these execution traces revisit the same states repeatedly. By identifying these repeated states and coalescing them, we can develop an *execution graph*. This graph is a directed graph where each node is a unique system state and an edge from node s to node t indicates that the system went from state s to state t in the corresponding execution trace. An execution trace then becomes a path through the execution graph.

Once execution graphs for each log have been created, we repeat the deduplication process to produce the unified execution graph of all runs. This allows us to identify similarities and differences among system execution traces.

The operation of the analysis code can be summarized as follows:

1. Parse the log files to create states based on the registers' values.
2. Deduplicate these execution traces to derive a per-run execution graph.
3. Deduplicate the execution graphs of different runs to derive a universal execution graph.
4. Using this execution graph and each execution trace, perform statistical analysis on the data to ascertain whether our instrumentation can capture differences in operation correlated with ESD.

4.2.1.1. Constructing execution graphs. The first stage of analysis parses register values from the log file for each run. After creating tuples for each of the states in that file, we deduplicate the sequence of states to create the nodes of the execution graph. We then derive the execution trace path through the execution graph from the state sequence. We also record the number of times each transition is taken.

4.2.1.2. Constructing the unified execution graph. The next analysis step combines the data from each log into a unified execution graph. The process is similar to that used to develop the execution graph for each log. Certain registers for the host controller contain memory addresses that change every time the driver is reloaded.[1] The values of these registers are not significant to our analysis; however, changes in the values are significant as they indicate changes in host controller execution. Therefore, we create a new globally unique state each time the value changes within a trace.

4.2.1.3. Graph analysis. We divide the data collected from test runs into two groups: baseline and ESD-exposed. Baseline traces are traces of the system operating normally; they provide us with the system's expected state machine ESD-exposed traces document how the system transitions into and out of unexpected operation due to ESD exposure.

After we create the graph of globally unique states, we analyze the baseline and ESD-exposed traces individually to observe how system operation differs among them. We subtract the set of states reached in baseline traces from the set of states reached in ESD-exposed traces to get a list of states only reached during ESD injection. These state sets can be used to show where and when the system transitioned into a state that can potentially be attributed to ESD. Similarly, we can determine which transitions between states are present only during ESD exposure.

[1] These registers are `HcPeriodCurrentED`, `HcBulkCurrentED`, `HcFmRemaining`, `HcHCCA`, `HcControlHeadED`, `HcControlCurrentED`, `HcBulkHeadED`, `HcFmNumber`, and `HcDoneHead`.

Table 4.1. Summary of data collected from experiments

Dataset	Experiments	Traces Collected
Baseline	24	24
Experiment 1	25	17
Experiment 2	42	30

4.2.2. Collected EMI Experiment Data. EMI-exposed experiments were carried out on two separate dates. Along with each log of register snapshots, we saved the transferred file as written to the USB drive. None of these files exhibit corruption, so further analysis of that data is not merited. In addition, we recorded the associated field strength and device orientation.

The data gathered from the experiments is summarized in Table 4.1. We gathered a total of 24 baseline sequences and a total of 47 EMI-exposed sequences from our experiments. Note that the number of sequences collected may be less than the number of experiments due to data loss caused by hardware crashes. Furthermore, sequences vary in length due to changes in system operation such as crashes, USB device disconnects, and data frame retransmissions.

We interpret these logs of register snapshots as sequences of host controller operation states. States and snapshots correspond one to one; each state is defined by the register values in its corresponding snapshots. We treat two states as equal when their register values are all equal, excluding the values of some registers which are set only by the host controller driver and thus do not reflect host controller operation.[2]

We define the state space to be $\mathcal{S} = \{s_1, \cdots, s_{19}\}$ where each s_i corresponds to a specific unique state observed in our data. See Table 4.2 for the complete listing of the state space.

[2]The registers we exclude are `frame_index`, `segment`, `frame_list`, and `async_next`.

Table 4.2. State space and register values from corresponding snapshot for EMI experiment

State	Registers						
	Command	Config'd Flag	Interrupt Enable	Status	TX Fill Tuning	Mode	Mode Extended
s_1	0x10005	0x1	0x37	0x0000	0x0	0x10005	0x10005
s_2	0x10005	0x1	0x37	0x0008	0x0	0x10005	0x10005
s_3	0x10025	0x1	0x37	0x8000	0x0	0x10025	0x10025
s_4	0x10025	0x1	0x37	0x8001	0x0	0x10025	0x10025
s_5	0x10025	0x1	0x37	0x8008	0x0	0x10025	0x10025
s_6	0x10025	0x1	0x37	0x8009	0x0	0x10025	0x10025
s_7	0x10025	0x1	0x37	0x8020	0x0	0x10025	0x10025
s_8	0x10025	0x1	0x37	0x8021	0x0	0x10025	0x10025
s_9	0x10025	0x1	0x37	0x8028	0x0	0x10025	0x10025
s_{10}	0x10025	0x1	0x37	0xA000	0x0	0x10025	0x10025
s_{11}	0x10025	0x1	0x37	0xA001	0x0	0x10025	0x10025
s_{12}	0x10025	0x1	0x37	0xA008	0x0	0x10025	0x10025
s_{13}	0x10025	0x1	0x37	0xA009	0x0	0x10025	0x10025
s_{14}	0x10025	0x1	0x37	0xA020	0x0	0x10025	0x10025
s_{15}	0x10025	0x1	0x37	0xA021	0x0	0x10025	0x10025
s_{16}	0x10065	0x1	0x37	0x8000	0x0	0x10025	0x10025
s_{17}	0x10065	0x1	0x37	0x8000	0x0	0x10065	0x10065
s_{18}	0x10065	0x1	0x37	0xA000	0x0	0x10025	0x10025
s_{19}	0x10065	0x1	0x37	0xA020	0x0	0x10025	0x10025

5. STATISTICAL ANALYSIS OF PERIPHERAL OPERATION

The experiments described in Section 4 produce a collection of "baseline" sequences, and a collection of "EMD-exposed" sequences of register snapshots. The goal of our statistical analysis is to answer two key questions:

1. Is our instrumentation capable of detecting changes in system operation correlated with exposure to EMD?

2. What are the characteristics of baseline and EMD-exposed operation?

Formally, the state space $\mathcal{S} = \{s_1, s_2, \cdots, s_m\}$ consists of distinct snapshots of register values s_i. The number of states and the values of the registers in each state are determined empirically from the experimental data. While the number of possible states (every combination of every possible value of every register) can be quite large, in practice most of them are unreachable. Thus, for the peripherals we have studied so far, we have not encountered issues with state space explosion. We can describe the j^{th} sequence $(x_1, x_2, \cdots, x_{n_j})$ as a series of n_j states taken from a state space $\mathcal{S}$. The set of these sequences $\mathcal{D}$ can be partitioned into two sets, the baseline data $\mathcal{D}_B = \{(x_t, 1 \leq t \leq n_j) \mid 1 \leq j \leq N_B\}$ and the EMD-exposed data $\mathcal{D}_E = \{(x_t, 1 \leq t \leq n_j) \mid N_B + 1 \leq j \leq N_B + N_E\}$.

The first modeling perspective, alluded to in the previous section, is to view these sequences of snapshots as paths taken through an execution graph. In this graph, nodes correspond to states s_i, and an edge exists between two states s_i and s_k if there exists a sequence in $\mathcal{D}$ which has as a sub-sequence (s_i, s_k). We can also speak of the "baseline execution graph" or the "EMD-exposed execution graph" where the transitions are determined only by the sequences in $\mathcal{D}_B$ or $\mathcal{D}_E$, respectively. Baseline and EMD-exposed operation can be differentiated by comparing the reachable nodes and allowed transitions in each graph.

The second modeling perspective interprets these sequences as categorical time series of observations taken from a stochastic process $\{X_t\} \in S$. The sequences are categorical because the states in $\mathcal{S}$ are discrete and unordered, meaning one cannot be ranked greater than or less than another. We assume that the data in $\mathcal{D}_B$ is generated by one process and the data in $\mathcal{D}_E$ by another. No further assumptions about independence or identicality of distribution are made. The data from each stochastic process can be analyzed to determine the difference between the two stochastic processes: one corresponding to baseline peripheral operation, the other to EMD-exposed operation.

Table 5.1. Probability distribution of register values: `HcInterruptEnable` and `HcInterruptDisable`

Value	Baseline Probability	ESD-Exposed Probability		
		`Enable`	`Disable`	Difference
`0x8000005E`	0.222 89	0.200 41	0.200 41	0
`0x8000001A`	0.013 50	0.096 77	0.096 66	0.000 11
`0x8000005A`	0.761 56	0.659 32	0.659 43	−0.000 11
`0x8000001E`	0.002 02	0.043 59	0.043 59	0

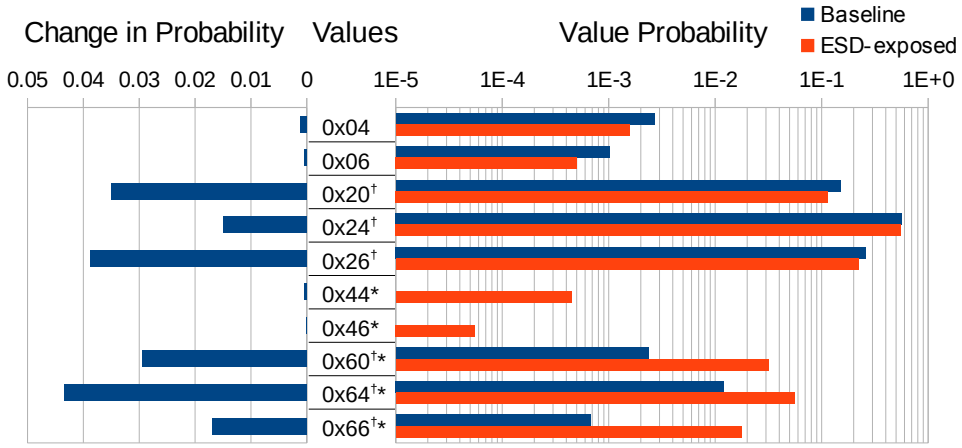
These perspectives are complementary: one focuses more on the nature of the software producing the traces, while the other focuses on the stochastic nature of the physical events which may alter the operation of the software. This section presents multiple techniques from both perspectives and applies them to the collected data to demonstrate that our instrumentation approach is effective.

5.1. ANALYSIS OF DATA FROM ESD INSTRUMENTATION

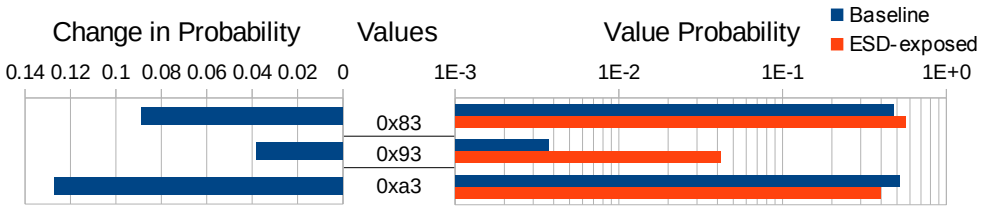
We investigate several aspects of our ESD dataset. First, we compare the probability distribution of several registers' values, revealing changes due to ESD. Second, the execution graphs of the data are compared and several indicators of anomalous execution are found. Finally, the effect of pulse voltage on anomalous execution is investigated.

5.1.1. Registers of Interest. Certain registers on the host controller can were observed to give indications of ESD. In particular, we consider the values of the registers for interrupt enabling and disabling (`HcInterruptEnable` and `HcInterruptDisable`), interrupt status (`HcInterruptStatus`), control (`HcControl`), and port status (`HcRhPortStatus0`). The host controller has multiple events and errors it can generate hardware interrupts for; the driver can enable and disable them depending on the current operation and check whether they have been triggered via the interrupt enable, disable, and status registers. The control register allows the driver to switch between various USB transfer modes and enable certain host controller features. The port status register reports whether a port is enabled, what device is connected to a port, device power configuration, etc.

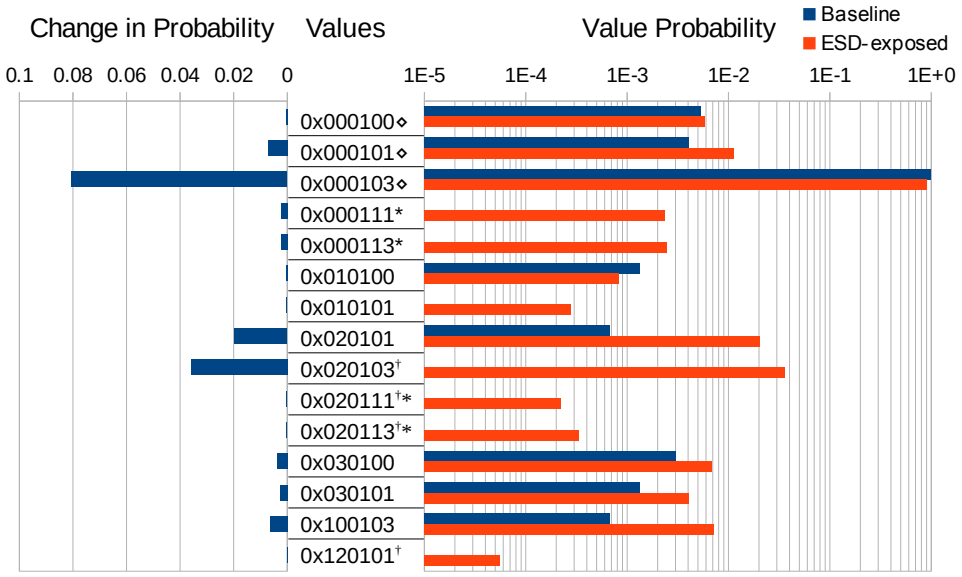
Per the OHCI specification [56], `HcInterruptEnable` and `HcInterruptDisable` should be duplicates of each other when read. However, as shown in Table 5.1, there are a few states in the ESD-exposed data where they are not duplicates. This may indicate ESD-induced bit flips or the system failing to properly update both registers when one is changed.



(a) `HcInterruptStatus`: † indicates frame counter overflow; * indicates status change



(b) `HcControl`



(c) `HcRhPortStatus0`: ◇ indicates device connected with no change in port status; † indicates port enabled/disabled; * indicates port reset

Figure 5.1. Probability distributions of register values

The `HcInterruptStatus` register values observed are shown in Figure 5.1a along with the probability of those values appearing in baseline and ESD-exposed traces and the absolute change in that probability due to ESD exposure. It shows a dramatic increase in values where the frame number counter overflowed (marked †) in the ESD-exposed traces, indicating that the system transmits many more frames during ESD exposure. In addition, values indicating the hub's status has changed (marked *) are also much more prevalent in ESD-exposed traces.

The `HcControl` register values provide a different perspective on the increase in the number of frames and hub status changes. Figure 5.1b shows a great increase in control frame processing (`0x93`) and a corresponding decrease in bulk data frame processing (`0xa3`). It is possible that ESD glitches are disrupting bus operation, requiring the host controller and device to send a greater number of status change frames. In addition, corruption in the bulk data frames would require retransmissions and therefore increase the number of new control and data frames (`0x83`).

The `HcRhPortStatus0` register contains status information about the port the USB drive was plugged into during testing. Figure 5.1c shows a marked decrease in states where the port status remains unchanged (marked ⋄ in Figure 5.1c) and an increase in states indicating the port has been enabled or disabled (†). As well, port resets (*) were only observed in ESD-exposed traces. The prevalence of resets and toggling whether the port is enabled hint that the host controller is experiencing unexpected errors and attempting to recover by resetting the port's status. The presence of a port reset where the driver or host controller would not usually issue one is a particularly strong indicator of ESD exposure.

5.1.2. Execution Graphs. Manual examination of execution graphs showed several potential effects of ESD on the system state. Figure 5.2 shows an execution graph of sample baseline and ESD-exposed execution traces. The set of nodes and solid arcs on the left of the figure is the execution graph of the baseline traces. The right of the figure consists of the additional states and transitions present in the sample ESD-exposed traces. Green states and solid edges indicate states and transitions present in the baseline traces. Red states and dashed edges indicate states and transitions appearing only in ESD-exposed traces. In addition to observing states in ESD-exposed traces that are not present in baseline traces, we observed four kinds of anomalous transitions:

① Transitions from baseline to non-baseline states,

② Transitions between non-baseline states,

③ Transitions between baseline states that are not observed in baseline traces, and

④ Transitions from non-baseline to baseline states.

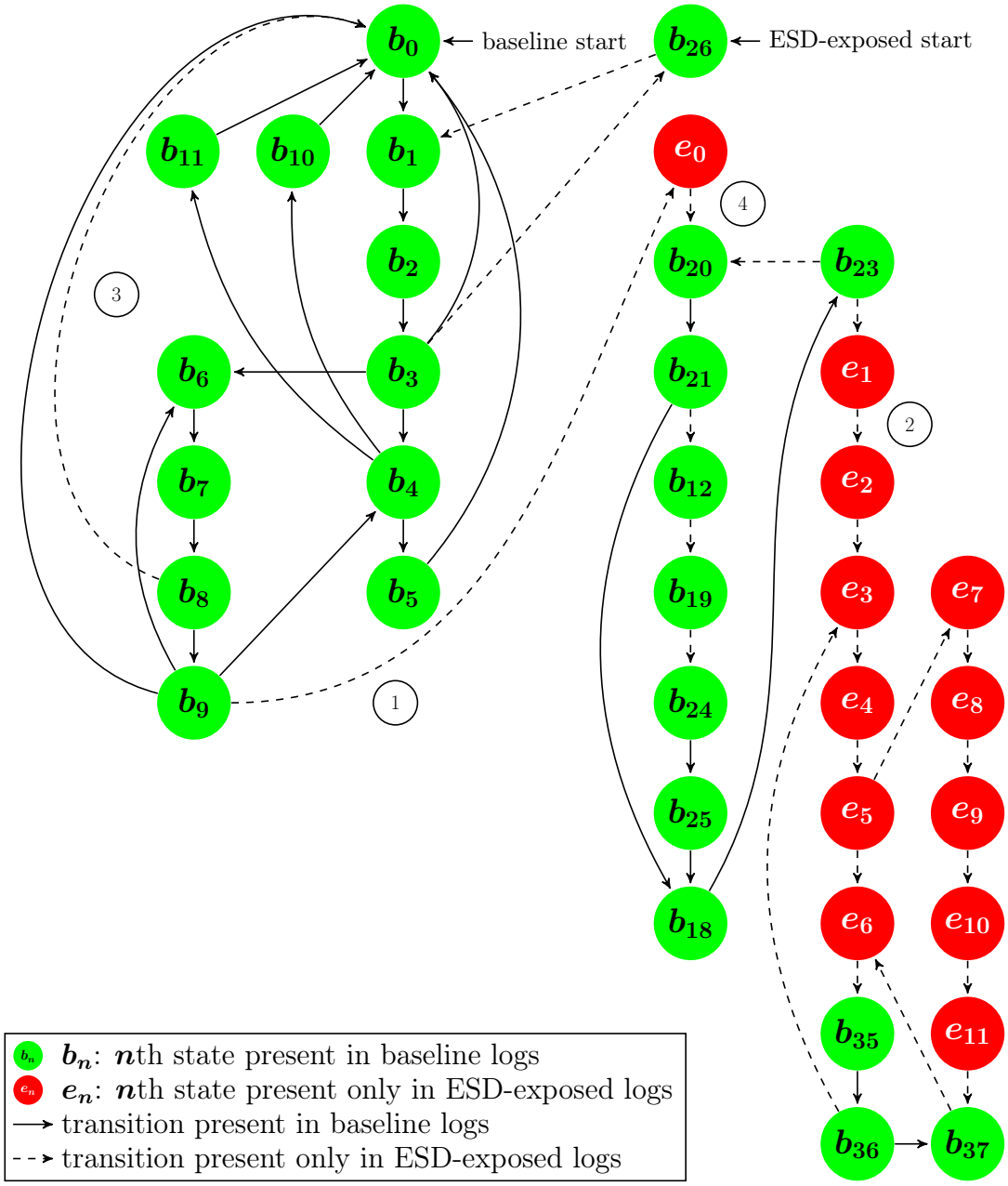


Figure 5.2. Execution graph of one baseline trace and one ESD-exposed execution trace

Consider how we should expect the system to behave under normal conditions and under ESD exposure. Normally, it should have a small number of common code paths and some edge case handling. Under ESD exposure, we should see a number of anomalous states caused by various

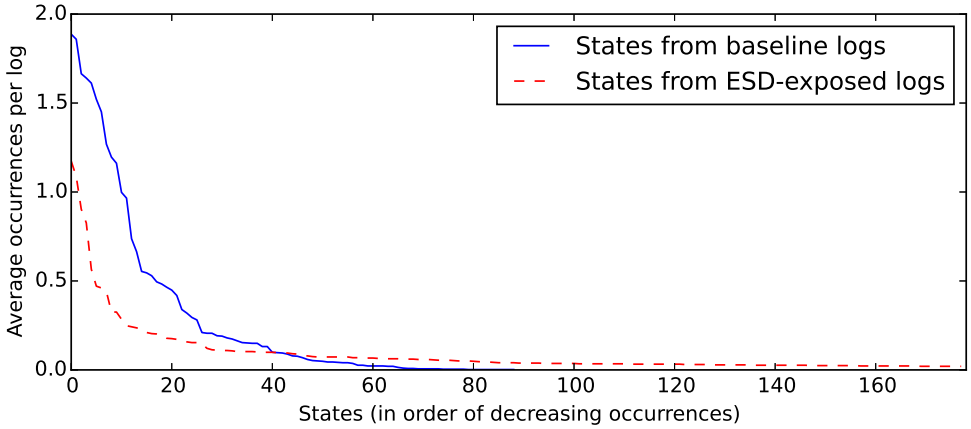


Figure 5.3. Average state occurrences per log

register bits being flipped as well as control flow anomalies. Figure 5.3 shows the average number of occurrences of states per baseline and ESD-exposed traces. The baseline traces show a few states that are very common and a small tail of less common states. There are far more unique states in ESD-exposed traces, and they are far less likely to occur. (We have omitted half of the ESD-exposed state tail to make the interesting portion of the graph more legible.) This graph provides quick verification of our methodology; we can see that the data we have collected reflects expected system operation.

5.1.3. TLP pulse voltage. Figure 5.4 compares the TLP pulse voltage with the percentage of transitions to or from states not in the baseline traces. The lack of a clear relationship between observed ESD coupling and pulse voltage indicates that there are confounding factors between ESD exposure and system operation. These factors may include field type and orientation, injection location, pulse frequency, and the operation being performed by the host controller at the time of injection. In addition, the ESD injection may cause the system to crash almost instantaneously, in which case the resulting state trace will have relatively few states caused by ESD. More work is needed to assess the effect each of these factors has on system operation.

5.2. ANALYSIS OF DATA FROM EMI EXPERIMENTS

Our experimental process produces sequences of states; we analyze these sequences to determine whether the sequences from baseline experiments differ from those produced by EMI-exposed experiments. We can model these data as categorical discrete time series: sequences

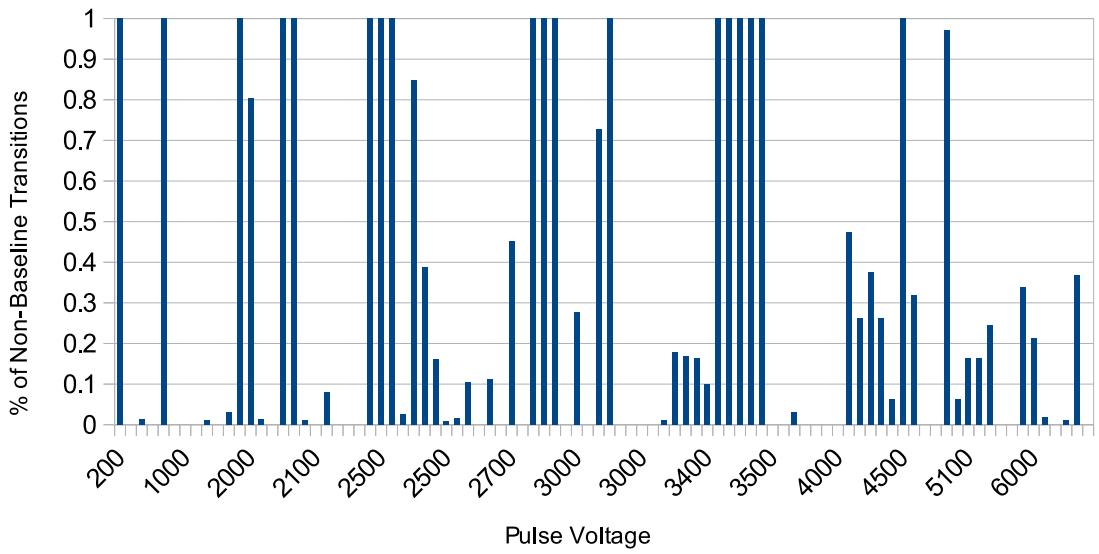


Figure 5.4. Relationship between pulse voltage and ESD-caused transitions

of observations of states from $\mathcal{S}$ over time. These states are discrete, rather than continuous, and unordered, meaning one cannot be ranked less than another. The events where we take our observations are the interrupts from the host controller; the times between events are stochastically distributed. While the space of possible register values is quite large, most of the possible states do not arise in practice: our data contain 19 distinct states. Further analysis is required to determine whether all registers are critical to determining the presence of EMI or if it is possible to reduce the state space further.

Formally, the j^{th} time series contains n_j observations $(x_1, \cdots, x_{n_j})$ from a stochastic process $\{X_t\} \in \mathcal{S}$. The number of observations n_j can be modeled as an observation from a random variable N determining the sampling time of each sequence. We have two datasets: baseline data $\mathcal{D}_B = \{(x_t, 1 \leq t \leq n_j) \mid 1 \leq j \leq 24\}$ and EMI-exposed data $\mathcal{D}_E = \{(x_t, 1 \leq t \leq n_j) \mid 25 \leq j \leq 72\}$. Our assumption is that the sequences in $\mathcal{D}_B$ are generated from one stochastic process and the sequences in $\mathcal{D}_E$ are generated from another. We make no further assumptions about independence or identicality of distribution.

In summary, this section demonstrates that our instrumentation is capable of detecting changes in device operation when exposed to EMI. To do so, we characterize the time series in $\mathcal{D}_B$ and $\mathcal{D}_E$ using categorical time series equivalents of variance and autocorrelation. Significant differences in these measures indicate that our instrumentation is effective.

5.2.1. Variance & Dispersion. One question we might ask of a dataset is, "how much variation does it have?" For real-valued data, the variation in a dataset is quantified by *variance*; however, this measure is derived from the mean, which is not defined for categorical data. Instead, we measure *dispersion*: a widely disperse distribution is close to a *uniform distribution*, whereas a minimally disperse distribution is close to a *one-point distribution*. The more disperse a distribution is, the more uncertain we are about the outcome of observing the random variable.

Several dispersion measures have been proposed; we choose two measures which are suitable for our particular datasets. One measure is the *Gini index* [58, §6.2.1], defined as

$$\nu_G = \frac{d+1}{d}\left(1 - \sum_{j\in\mathcal{S}} \pi_j^2\right), \tag{5.1}$$

where $d+1$ is the number of categories (in our case, 19), $\mathcal{S}$ is the set of categories (in our case, the set of host controller states as defined in Section 4.2.2), and π_j is the probability of observing the jth category. The value of the Gini index falls in the range $[0; 1]$, with 0 indicating a one-point, minimally disperse distribution and 1 indicating a uniform, maximally disperse distribution.

The other measure we have chosen is the *extropy* [59] of a distribution, defined as

$$\nu_{Ex} = \frac{-1}{d\ln\left(\frac{d+1}{d}\right)} \sum_{j\in\mathcal{S}} (1-\pi_j)\ln(1-\pi_j). \tag{5.2}$$

Extropy shares the same range and behavior as the Gini index: higher extropy means a more disperse distribution. Extropy is the complementary dual of information theory's *entropy*; we select it over entropy due to domain restrictions of their corresponding signed serial dependence measures as explained in Section 5.2.2.2.

Dispersion measures for each time series are aggregated using box plots in Figure 5.5. Both measures qualitatively agree: the baseline time series are less disperse than all but a few of the EMI-exposed time series. In the Experiment 2 dataset, where most interference was near the limit of what the system could withstand, all time series are more disperse than baseline time series.

5.2.2. Autocorrelation & Serial Dependence. In addition to variance, we can characterise time series data by how repetitive it is. Conceptually, this is determined by how similar the time series is to a copy of itself shifted back by one or more time steps (called *lag*). For real-valued time series, this measure is called *autocorrelation*; however, it depends on expected values, which are not defined for categorical data.

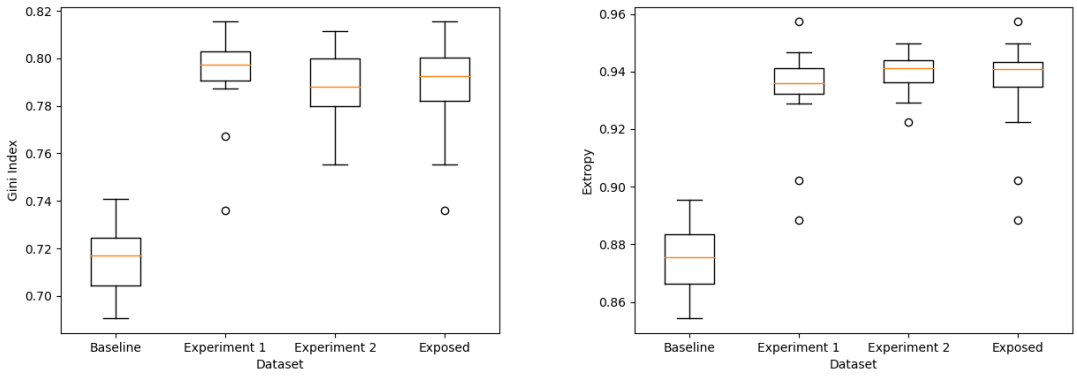


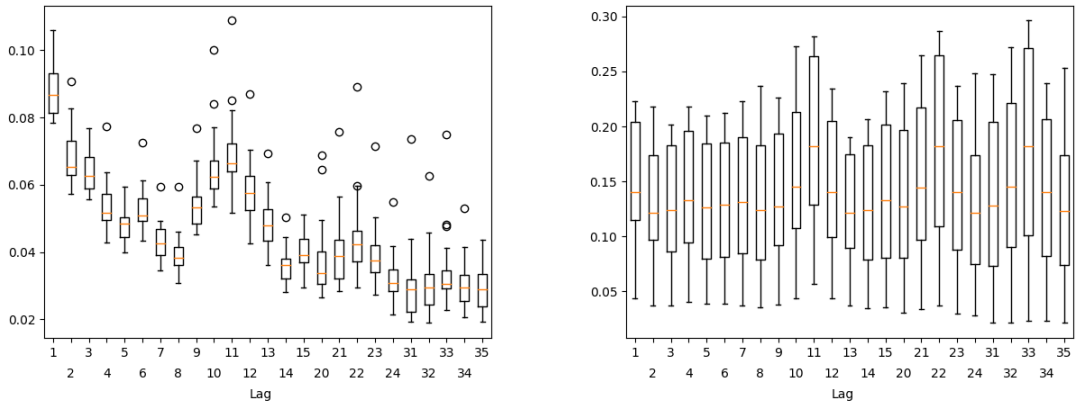
Figure 5.5. Gini and Extropy dispersion measures for each dataset as well as for all EMI-exposed time series ("Exposed")

The corresponding categorical time series property is known as *serial dependence*. Serial dependence comes in two variants: signed and unsigned. These measure various properties of the *conditional lagged bivariate probabilities* $p_{i|j}(k) = P(X_t = i \mid X_{t-k} = j)$. Here, i and j are categories in $\mathcal{S}$ and k is the lag. $p_{i|j}(k)$ is the probability of observing i at time t given that j was observed at time $t - k$. Further information on computing $p_{i|j}(k)$ can be found in [58, § 6.3].

- *Serial Independence* occurs when observing j at $t - k$ gives no insight into what might be observed at t. Equivalently, $p_{i|j}(k) = \pi_i$ for any $i, j \in \mathcal{S}$.
- *Unsigned Serial Dependence* occurs when observing j at $t - k$ gives perfect insight into what is observed at t. Equivalently, for any j, we can find an i where $p_{i|j}(k) = 1$.
- *Positive Serial Dependence* occurs when observing i at $t - k$ means we observe i again at t. Equivalently, for any i, $p_{i|i}(k) = 1$.
- *Negative Serial Dependence* occurs when observing i at $t - k$ means we will not observe i at t. Equivalently, for any i, $p_{i|i}(k) = 0$.

5.2.2.1. Unsigned serial dependence. As a measure of unsigned serial dependence, we use *Cramer's* v, defined by

$$v(k) = \sqrt{\frac{1}{d} \sum_{i,j \in \mathcal{S}} \frac{(p_{ij}(k) - \pi_i \pi_j)^2}{\pi_i \pi_j}}, \tag{5.3}$$



(a) Baseline data

(b) EMI-exposed data

Figure 5.6. Cramer's v measures for lags 1–15, 20–24, and 31–35

where k is the lag and $p_{ij}(k) = P(X_t = i, X_{t-k} = j)$ is the *lagged bivariate probability*; that is, the probability of observing i at time t and observing j at time $t-k$. This relates to the conditional lagged bivariate probability mentioned above: $p_{i|j}(k) = \frac{p_{ij}(k)}{\pi_j}$. $v(k)$ has range $[0; 1]$, with 0 corresponding to serial independence and 1 corresponding to unsigned serial dependence. The larger $v(k)$ of a time series is, the more accurately we can predict future values of that time series.

A convenient property of Cramer's v is that it follows a χ^2 distribution when it is estimated by sampling an independent and identically distributed (i.i.d.) process. A process where every observation is independent and identically distributed is the epitome of serial independence! Thus, we can use a χ^2 significance test to determine whether an estimate for v for a particular time series is statistically significant. The exact test performed is

$$v(k) > \sqrt{\frac{1}{Td}\chi^2_{d^2;1-\alpha}} \tag{5.4}$$

where T is the number of observations in the time series and $\chi^2_{d^2;1-\alpha}$ is the inverse χ^2 CDF with d^2 degrees of freedom evaluated at $1-\alpha$. When this condition holds, we reject the null hypothesis that our data is not serially dependent. Since we are testing unsigned serial dependence, this is a one-tailed test.

We compute Cramer's v for our four datasets at lags 1–15, 20–24, and 31–35. Initially, we used lags 1–15, which revealed a peak at lag 11. Thus, we extended the analysis to cover lags around 22 and 33, which revealed peaks at lags which are multiples of 11 as well. Figure 5.6 shows box plots for the baseline and combined EMI-exposed datasets. They show slightly more serial dependence in

the EMI-exposed data than what is observed in the baseline data. The baseline data has a small peak at lag 11, while the EMI-exposed data has more distinct peaks at lags 11, 22, and 33. Unfortunately, serial dependence at all lags across both datasets fails to achieve statistical significance, though 27% of traces in the EMI-exposed data achieve significance at lags 11, 22, and 33.

5.2.2.2. Signed serial dependence. We select two measures of signed serial dependence that relate to our chosen dispersion measures. The first measure, Cohen's κ [58, §6.3] is defined by

$$\kappa(k) = \frac{\sum_{j\in\mathcal{S}} \left(p_{jj}(k) - \pi_j^2\right)}{1 - \sum_{j\in\mathcal{S}} \pi_j^2}. \tag{5.5}$$

Values of $\kappa(k)$ fall in the range $\left[-\frac{\sum_{j\in\mathcal{S}} \pi_j^2}{1-\sum_{j\in\mathcal{S}} \pi_j^2}; 1\right]$, with 0 corresponding to serial independence, positive values corresponding to positive serial dependence, and negative values corresponding to negative serial dependence. Note that the denominator of the lower bound for $\kappa(k)$ scales negatively with the Gini index of the marginal distribution $\boldsymbol{\pi}$ of the dataset. Thus, $\kappa(k)$ is undefined when $\boldsymbol{\pi}$ is a one-point distribution. In the i.i.d. case, that is, when a sequence is serially independent, Cohen's κ follows a normal distribution.

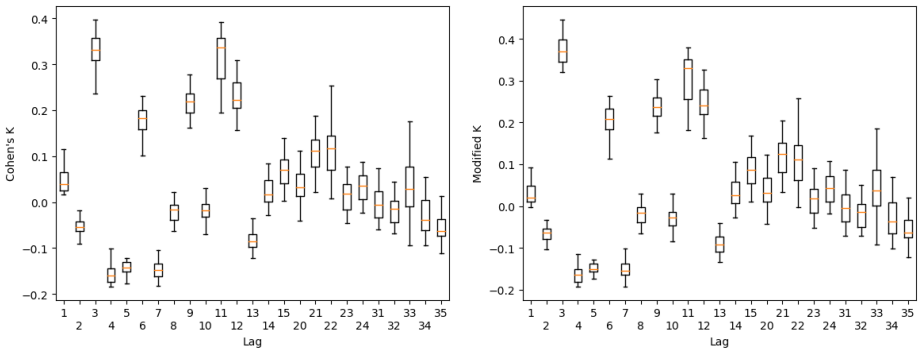
The second measure of serial dependence we selected is a modified version of $\kappa(k)$. This measure, $\kappa^\star(k)$ [60] is defined by

$$\kappa^\star(k) = \sum_{j\in\mathcal{S}} \frac{p_{jj}(k) - \pi_j^2}{1 - \pi_j}. \tag{5.6}$$

Values of $\kappa^\star(k)$ fall in the range $\left[-\sum_{j\in\mathcal{S}} \frac{\pi_j^2}{1-\pi_j}; 1\right]$, with positive values corresponding to positive serial dependence and vice versa. $\kappa^\star(k)$ is undefined in the case that $\boldsymbol{\pi}$ is a one-point distribution. More disperse datasets also have a wider range of $\kappa^\star(k)$ values. In the i.i.d. case, as with Cohen's κ, $\kappa^\star$ follows a normal distribution.

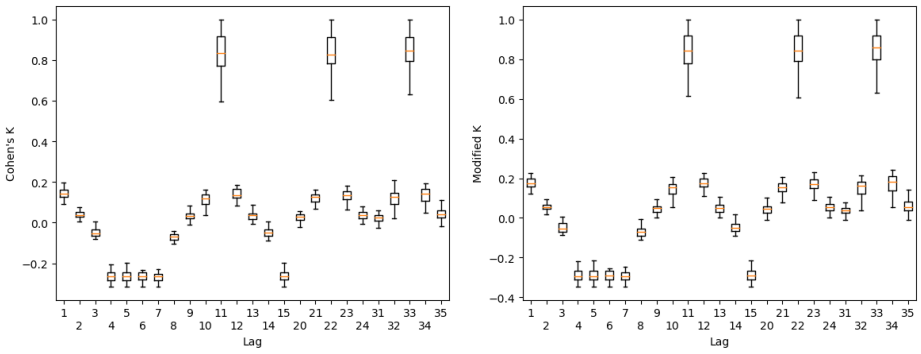
These measures, κ and $\kappa^\star$, relate to our dispersion measures ν_G and ν_{Ex} respectively by appearing as terms in measures of their bias [60]. The measure of serial dependence κ^* which is associated with entropy is undefined in the case that $\pi_j = 0$ for any state j, making it an undesirable statistic for our datasets. Hence, we choose extropy and $\kappa^\star$ over entropy and κ^* as measures of variance and serial dependence.

We compute Cohen's κ and $\kappa^\star$ for lags 1–15 and additionally 20–24 and 31–35 to confirm recurring serial dependence at lags that are multiples of 11. Box plots and critical values for the baseline and EMI-exposed datasets are shown in Figures 5.7a, 5.7b, 5.7c, and 5.7d. Our discussion focuses on dependence exhibited by both measures.



(a) Cohen's κ for baseline data

(c) Modified $\kappa^\star$ for baseline data



(b) Cohen's κ for EMI-exposed data

(d) Modified $\kappa^\star$ for EMI-exposed data

Figure 5.7. Signed serial dependence measures for lags 1–15, 20–24, and 31–35

The baseline data measures (figures 5.7a and 5.7c) reveal some significant positive serial dependence at small lags, notably lags 3, 6, 9, 11, and 12. Small, but still significant, negative serial dependence is observed in the Cohen's κ values at lags 4, 5, and 7; however, many of the corresponding values for $\kappa^\star$ are not significant. Above lag 12, serial dependence rapidly vanishes in the baseline dataset aside from mild positive dependence lag 21. In the EMI-exposed data, the only significant positive serial dependence is at lags 11, 22, and 33; in addition, some sequences have positive serial dependence at lag 34. Both Cohen's κ and $\kappa^\star$ show significant negative serial dependence at lags 4, 5, and 7. In addition, significant negative serial dependence is observed at lag 15.

These results are summarized in Table 5.2. The entries marked '?' deserve additional commentary: at lag 6, most EMI-exposed sequences have negative serial dependence, but one or two have positive serial dependence. This may indicate that the EMI-exposed data is multimodal, with some sequences exhibiting one mode of operation and others another, or it may indicate that certain sequences do not exhibit interference. From the entries marked '+', we can see that the chosen serial

Table 5.2. Percent of sequences where serial dependence is significant ($\alpha = 0.05$) at a given lag. Entries marked '*', '+', or '?' have at least 50% of sequences showing significant dependence; entries marked '+' exhibit the same result for both measures; entries marked '?' have at least one sequences showing significant dependence of the opposite sign as well.

	Cohen's κ				Modified $\kappa^\star$			
	Baseline		EMI-exposed		Baseline		EMI-exposed	
Lag	% +ve	% −ve	% +ve	% −ve	% +ve	% −ve	% +ve	% −ve
1	33.3	0.0	95.6*	0.0	0.0	0.0	42.2	0.0
2	0.0	41.7	2.2	0.0	0.0	0.0	0.0	0.0
3	100.0+	0.0	13.3	4.4	100.0+	0.0	6.7	2.2
4	0.0	100.0+	0.0	100.0+	0.0	83.3+	0.0	97.8+
5	0.0	100.0+	0.0	100.0+	0.0	79.2+	0.0	93.3+
6	100.0+	0.0	4.4	91.1?	91.7+	0.0	2.2	82.2?
7	0.0	100.0+	0.0	100.0+	0.0	66.7+	0.0	97.8+
8	0.0	12.5	0.0	8.9	0.0	0.0	0.0	2.2
9	100.0+	0.0	13.3	0.0	95.8+	0.0	6.7	0.0
10	0.0	4.2	73.3*	0.0	0.0	0.0	33.3	0.0
11	100.0+	0.0	100.0+	0.0	100.0+	0.0	100.0+	0.0
12	100.0+	0.0	97.8*	0.0	100.0+	0.0	48.9	0.0
13	0.0	87.5*	2.2	0.0	0.0	0.0	0.0	0.0
14	20.8	0.0	6.7	2.2	0.0	0.0	0.0	2.2
15	62.5*	0.0	0.0	93.3+	16.7	0.0	0.0	82.2+
20	37.5	0.0	6.7	0.0	0.0	0.0	0.0	0.0
21	83.3+	0.0	82.2*	0.0	50.0+	0.0	35.6	0.0
22	83.3*	0.0	100.0+	0.0	37.5	0.0	100.0+	0.0
23	20.8	0.0	95.6*	0.0	0.0	0.0	46.7	0.0
24	33.3	0.0	4.4	0.0	0.0	0.0	0.0	0.0
31	12.5	4.2	2.2	2.2	4.2	0.0	0.0	0.0
32	0.0	16.7	77.8*	2.2	0.0	0.0	37.8	0.0
33	41.7	8.3	100.0+	0.0	20.8	0.0	97.8+	0.0
34	8.3	33.3	91.1+	0.0	0.0	0.0	51.1+	0.0
35	0.0	54.2	13.3	0.0	0.0	0.0	0.0	0.0

dependence measures agree more frequently on the baseline dataset than on the EMI-exposed dataset. The modified $\kappa^\star$ measure universally reports serial dependence less or equally frequently compared to Cohen's κ, as shown by the '*' entries.

At a minimum, we can say that there are discernible differences between the baseline and EMI-exposed datasets. Since the magnitudes of the serial dependences do not decrease monotonically, we can conclude the underlying stochastic processes do not have the Markov property. More analysis of the underlying data is necessary before a complete explanation of these differences can be offered. We hypothesize that during "normal" operation, the host controller's operation is characterized by 3 state and 11 state loops, which appear as significant serial dependence at lags which are multiples of 3

and 11. When exposed to EMI, however, the 11-step processes become dominant; perhaps this is due to built-in fault-recovery methods or due to some unforeseen interaction between the experimental setup and the host controller's operation. The negative serial dependences are less dramatic and are likely a side effect of the 3 and 11 state loops.

6. CLASSIFICATION OF PERIPHERAL OPERATION

Having determined that the injection of EMD causes observable differences in the traces collected by our instrumentation, we study whether these differences can be used as indicators of EMD. The ultimate goals of this investigation are threefold:

1. Identifying sequences of states from execution traces for further root-cause analysis.
2. Detecting anomalous operation in real time on devices in the field.
3. Recovering from interference with minimal interruptions to system operation.

In this section, we present work that lays a foundation for addressing these questions. Specifically, we study whether the differences in operation correlated with EMD can be used to infer whether EMD was present during all or part of a trace. This is a classification problem with two labels, "baseline" and "EMD-exposed".

Our approach should meet two criteria:

1. Sufficient granularity to identify anomalous sub-sequences
2. Ability to produce results from a stream of data, not just finite-length sequences

For the ESD dataset, we create a bespoke classification approach which can be applied to whole sequences or sub-sequences thereof. We train and test it on full sequences from the ESD experimental data. The promising results from this preliminary work leads us to pursue a more rigorous classification approach for the EMI dataset.

With the EMI data, we study classifiers which produce a stream of classification results, as well as sliding-window classifiers; classifiers from both categories meet the second criteria above. We propose a method to synthesize sequences of states which contain both known-baseline and known-EMI-exposed subsequences. This allows us to train and validate the classifiers on data that is more representative of what a real-time EMI monitor would produce. Our classifier evaluation process is more detailed, encompassing multiple metrics and assessing statistical significance.

6.1. DETECTING ESD EVENTS

We present a preliminary classifier for the host controller state data. While the approach here is limited to offline training and classification, it has been designed with the ability to be easily converted to real-time operation. As such, the classification process is computationally inexpensive in order to reduce overhead on an embedded system.

The classifier takes an execution trace as input and produces a label, either *normal* or *ESD-exposed*, and a confidence level associated with the label. This label is computed based on the similarity between states and transitions in the input trace and the training data. The classifier has a concept of *confidence* that describes the likelihood that the classifier label is correct. Confidence is quantified as a value between 0 and 1, with 0 representing no ability to discern a label and 1 representing complete confidence in the result.

6.1.1. Training. In order to deduce when the system is exposed to ESD based on our data, we must consider three classes of states: states appearing only in baseline traces, states only appearing in ESD-exposed traces, and states appearing in both. Our underlying assumption is that states only in baseline traces indicate normal operation and states only in ESD-exposed traces indicate ESD exposure. States appearing in both provide less information about whether the system is operating as intended or experiencing ESD interference.

We use baseline traces to provide an approximation of normal operation and ESD-exposed traces to approximate ESD-exposed operation. This approach assumes that ESD-exposed traces are exposed to ESD throughout the duration of the trace. We also suppose that any departure from normal operation can be attributed to ESD. This assumption holds for our training and evaluation data, although the classifier could be expanded to differentiate among various kinds of interference if this assumption were to not hold.

The value of confidence is calculated by assigning each system state a weight. We are able to combine label and confidence computations by representing weights as numbers in the range $[-1, 1]$ where positive weights correspond to a label of 'ESD-exposed' and a negative weight corresponds to a label of 'normal'. The confidence in the label is the absolute value of the weight.

Assignment of weights must compensate for the fact that the training data will contain more ESD-exposed traces than baseline traces, since we performed more ESD-exposed tests. Thus, we assign the confidence for each state's label proportionately, based on the probability of that state appearing in baseline and ESD-exposed traces. The normalized weight, w_i, for a unique state i is

defined as:

$$w_i = \begin{cases} \dfrac{-\dfrac{b_i}{N_B} + \dfrac{e_i}{N_E}}{\left|\dfrac{b_i}{N_B}\right| + \left|\dfrac{e_i}{N_E}\right|} & \text{if } b_i > 0 \text{ or } e_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where N_B and N_E are the total number of states in all baseline and ESD-exposed execution traces, respectively, and b_i and e_i are the number of times state i appears in baseline and ESD-exposed execution traces. The case where $b_i = e_i = 0$ occurs when assigning a weight to a state not present in the training data. In that case, the state provides no information about whether or not the system is experiencing ESD.

6.1.2. Classification. We use these weights to build a classifier for the traces we have collected. The classifier operates on the entirety of an execution trace after it has been recorded.

This classification approach could be easily adapted to a look at a fixed 'window' of states for real-time classification. An ESD event would then appear as an increasingly positive confidence that the system is ESD-exposed. Recovery from such an event would appear as a decrease in confidence that the system is ESD-exposed and an increase in confidence that it is operating normally.

The weights can be used to classify a trace as follows: An execution trace, T, is a sequence of system states. The trace's classification, C_T, is computed by

$$C_T = \frac{1}{|T|} \sum_{i \in T} w_i \tag{6.2}$$

If C_T is positive, we classify the trace as having experienced ESD, and if it is negative or zero, we classify it as having not experienced ESD.

Classifier support can be measured by the percentage of states for which weights are nonzero. This measures the proportion of the input trace considered when choosing a label. We define accuracy as the percentage of correct classifications.

We tested this procedure over 22 base traces and 113 ESD-exposed traces. The classifier had 68.5% average accuracy and 84.6% average support with k-fold cross-validation where $k = 10$.

6.1.3. System State Transitions. We hypothesize that the operation of the host controller is not Markovian; that is, knowing how the system reached the current state may allow us to predict the states to which it can transition with higher accuracy. For example, a frame counter overflow may occur during either data frame or control frame transmission. However, the states leading up to and following an overflow while transmitting a data frame will be different from those surrounding

Table 6.1. Average classifier performance for various n state trajectories

n	Accuracy	Accuracy with δ	Support	Table Entries	Best δ
1	68.5%	88.5%	84.6%	428	0.17
2	85.4%	88.5%	62.5%	1256	0.275
3	84.6%	86.9%	39.7%	2231	0.138
4	86.2%	86.9%	22.7%	3324	0.115
5	87.3%	87.3%	14.2%	4348	0.125
6	87.3%	88.9%	8.3%	5258	0.377

an overflow while transmitting a control frame. In addition, ESD exposure may cause the system to abnormally transition between two normal states. This effect of ESD is not captured in the above classifier design.

We can approximately capture this more complex operation model by classifying system transitions instead of system states. Additionally, we can extend the concept of a transition, which is a 2-tuple of states, to an 'n state trajectory', an n-tuple of states in the order they were executed. In graph theoretic terms, an n state trajectory is a path of length $n-1$ in the execution graph. Weight assignment, classification, and performance metrics are defined analogously to their counterparts in the state-based classification approach.

One disadvantage of adding context in this fashion is the increasing likelihood of encountering an n state trajectory in the validation data that was not present in the training data. Thus, we must choose a trade off between increasing accuracy and decreasing support. Another disadvantage is the increased storage overhead for the weight table, which must be able to fit into memory on an embedded system.

Table 6.1 shows classifier performance for different n state trajectories, again averaged over $k = 10$ folds. The $n = 1$ data is the same as that presented in the previous section.

6.1.4. Delta. During development, we determined that the classifier performed poorly in part due to underestimating the weights of states indicating ESD exposure. That is, certain states which appeared in both baseline and ESD-exposed traces were under-represented in ESD-exposed traces, causing the classifier to underestimate their significance. We introduced an empirically-determined bias parameter, δ, as a means of adjusting the weight distribution. Values for δ are selected from the interval $[0, 1]$.

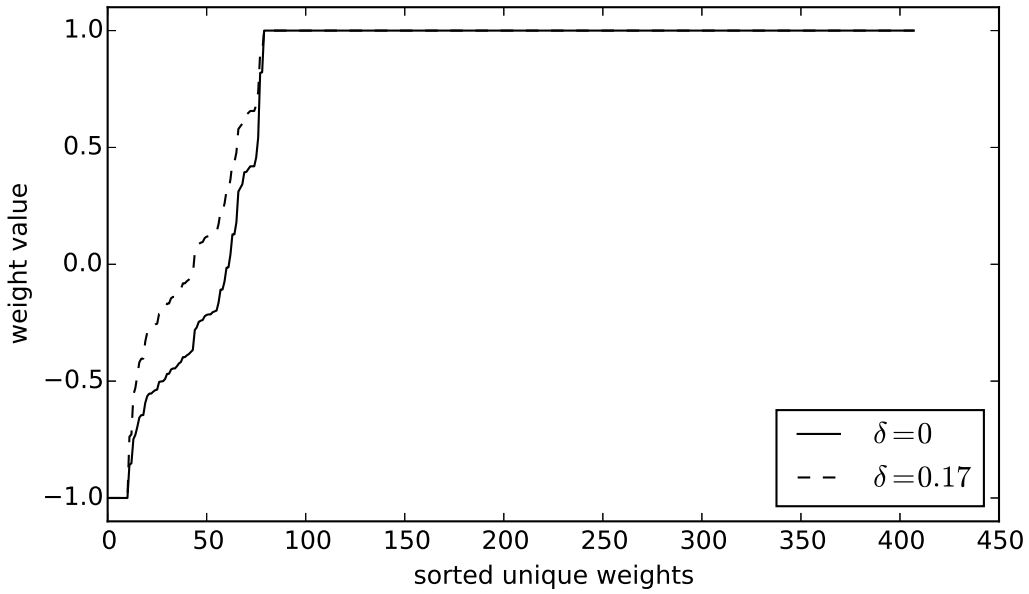


Figure 6.1. Weights with and without δ

Individual state or n state trajectory weights are computed as follows:

$$w_i = \begin{cases} \dfrac{-\left(1+\frac{1}{N_B}\right) b_i\delta + \left(1+\frac{1}{N_E}\right) e_i\delta}{\left|\left(1+\frac{1}{N_B}\right) b_i\delta\right| + \left|\left(1+\frac{1}{N_E}\right) e_i\delta\right|} & \text{if } b_i > 0 \\ & \text{or } e_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6.3}$$

Figure 6.1 shows the change in the distribution of weights without δ and with the optimal $\delta = 0.17$.

Running $k = 10$-fold cross-validation for several n state trajectories and choosing the optimal value of δ, the classifier performs as shown in Table 6.1. As we expect, δ has no effect on support, but it does have an effect on accuracy. It allows us to reduce the amount of context the classifier needs to perform well (smaller n), allowing us to have a more precise classifier without losing accuracy.

To demonstrate that δ does not cause the model to overfit, Figure 6.2 shows the effect of δ on the accuracy of the classifier for each fold. The accuracy shown is the accuracy of the classifier on the data it was trained on. It is essential that we choose a value of δ based only on information gained from the training data lest we bias the classifier towards the validation data. The vertical dashed line shows the value of δ that maximizes accuracy across all folds. If δ were causing the

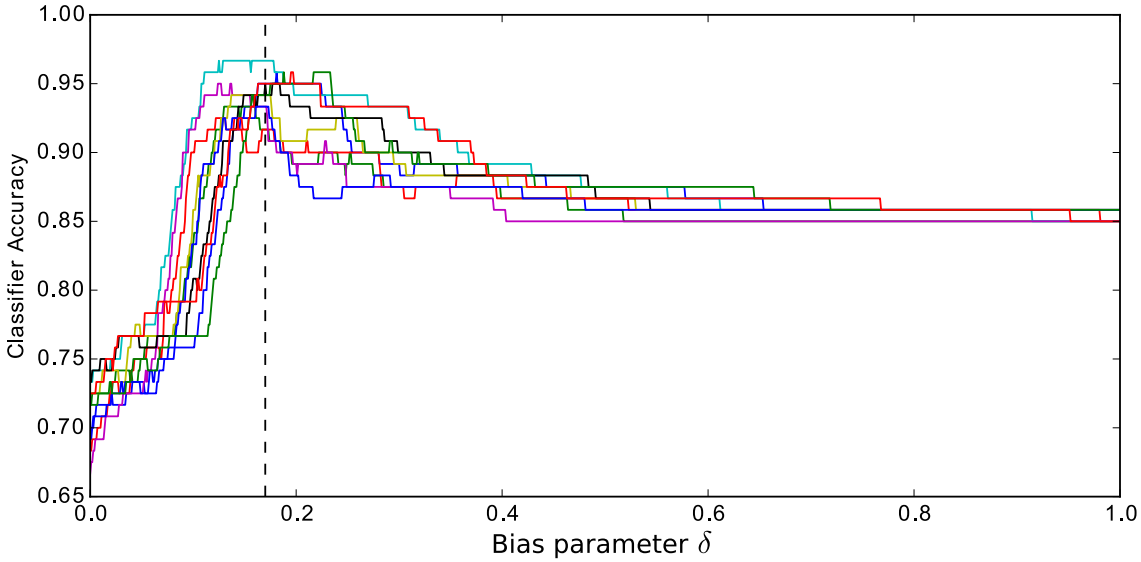


Figure 6.2. Effect of δ on accuracy

classifier to overfit to the training data, we would expect to see each fold having a wildly different optimal δ. However, in this plot, each fold's accuracy peaks near the same δ value, indicating that the model does not overfit.

6.2. DETECTING EMI EVENTS

We present our methodology and results for five classification techniques. These classifiers are applied to several types of events derived from our EMI datasets. In addition, we devise a method for synthesizing sequences which contain both known-baseline and known-EMI-exposed states for training purposes.

6.2.1. Classification Events. When developing classifiers for observed system operation, we can choose to classify sequences of register snapshots, or we can classify sequences of *events* derived from these snapshots. One example of such a sequence of derived events is pairs of snapshots $(x_{i-1}, x_i), 1 < i \leq n$ produced from a window of n snapshots. Such a derived event allows classifiers which otherwise ignore the order of states in a window to capture some temporal dependences in the data. We refer to such a pair of states as "lag 1" events; "lag 0" events refer to the sequence of snapshots x_i. This approach can be extended with further "lookback" — a "lag 2" event is the pair (x_{i-2}, x_i). In addition, we can classify over multiple lags: "lag 0 and 1" data contains an event for every observed state s_i as well as an event for every observed state pair (x_{i-1}, x_i).

Since the space of possible events grows exponentially with the number of snapshots used to derive an event, we do not consider triplets (e.g., (x_{i-2}, x_{i-1}, x_i)) of snapshots in this work.

6.2.2. Classification Techniques. The first set of techniques we evaluated produce a stream of results: a Hidden Markov Model (HMM) and a Recurrent Neural Network (RNN) Long Short Term Memory (LSTM) classifier. In our HMM classifier, the hidden states correspond to our classification labels: "Baseline" and "EMI-exposed" and the observed events to events from a trace. When trained, the HMM learns the expected marginal distribution of the classification events in each category. Classifying a sequence of events produces the most likely sequence of classification labels — hidden states — given the observed events. The LSTM approach incorporates a memory of previous events which is considered when classifying a new event. While it seems a promising fit for our problem, we lack sufficient training data to achieve passable classification performance.

The second set of techniques, the sliding window classifiers, all classify on the distribution of events from a window. This choice keeps the input vectors to the classifier small — "lag 0" events require a 19 element vector — and independent of window size, as well as eliminates any overfitting from classifiers associating labels with a particular event's absolute position in the window.

We evaluated four sliding window classification techniques: an Artificial Neural Network (ANN), a Support Vector Machine (SVM), a Random Forest Classifier (RFC), and a Gradient Boosted Classifier (GBC). The ANN classifier features two neuron layers: an input layer the size of the event space and a single neuron sigmoidal output layer. This learns a direct mapping from the event marginal distribution to a value between 0 and 1, which is then compared against a cutoff to determine the classification label applied. The SVM classifier partitions the space of possible event marginal distributions and assigns categories to regions of this space. Random forest classifiers consist of a "forest" of decision trees; the majority vote of the trees determines the classification label applied to an input vector. Internal nodes in these trees pick a particular element of the input vector and compare it to a cutoff to determine which branch of the tree to continue the decision process along; leaf nodes are classification labels. Gradient boosted classifiers are a variation on RFCs wherein trees are not trained independently; instead, each tree is selected to minimize the error of the previous tree.

6.2.3. Training and Evaluation Approach. Training and evaluating a classifier intended to detect when a system is experiencing interference requires a dataset of sequences with known-baseline and known-interfered-with subsequences. Recall that our dataset contains either wholly-baseline or wholly-interfered-with sequences, but none that exhibit both modes of operation in one

sequence. We devised an approach to synthesize sequences that are suitable for training and evaluating the classifiers from this dataset. Sequences from the baseline dataset and from the EMI-exposed dataset were alternated in either `BEBE` or `BEEB` patterns, with `B` referring to a baseline sequence and `E` an EMI-exposed sequence. These synthesized sequences were then sliced into windows, with the desired label (baseline or EMI-exposed) for the window determined by the label of the majority of its states. Synthesized sequences could have unbalanced class representation or have balanced class representation by oversampling short baseline or EMI-exposed sequences until all subsequences used to synthesize a sequence contained the same number of snapshots. It is critical that balanced sequences be used only for training and not for classifier evaluation. For each synthetic dataset, 150 sequences were selected at random and 150 windows selected from each sequence for training and evaluation.

In addition to classifier-specific hyperparameters, we have several dataset-level hyperparameters:

- Synthesis pattern: `BEBE` or `BEEB`
- Balanced or unbalanced class representation
- Classification events, or "lags"
- Window length, for sliding-window classifiers

Two scores were selected to evaluate classifier performance: the standard F1 score and Matthews Correlation Coefficient (MCC). The MCC score is balanced: it takes into account true and false positives and negatives, and produces informative scores even when class representations are imbalanced, as they are in our dataset. All classification results are averaged across 5-fold cross-validation.

6.2.4. Results. Classifier-specific hyperparameter selection is performed using 5-fold cross-validation; the ideal hyperparameter settings for each classifier are as follows:

ANN Trained for 100 epochs, learning rate of 0.001, Adam optimization with binary cross entropy loss.

SVM Linear kernel, cost of 10,000.

RFC 100 trees, max tree depth of 12.

GBC 100 trees, max tree depth of 12, learning rate of 0.1.

Table 6.2. Events generated from dataset at each lag

Lag	Events
0	x_i
0,1	$x_i, (x_{i-1}, x_i)$
1	(x_{i-1}, x_i)
2	(x_{i-2}, x_i)
3	(x_{i-3}, x_i)
4	(x_{i-4}, x_i)

We examine the variation in classifier performance due to our dataset-specific hyperparameters. Classifiers were trained for Lag 0; Lags 0,1; Lag 1; Lag 2; Lag 3; and Lag 4. Table 6.2 reviews the events used by the classifier at these lags.

The effect of window size is shown in Figure 6.3a. Note that for Lag 0,1, the window size used is double what is shown on the axis label. Empirically, doubling events requires double the window size to achieve equivalent performance to single-lag classifiers; this is an inherent disadvantage to multiple-lag classifiers. The reduced performance of the Lag 0 classifiers compared to all others indicates that the sequential relationship of state observations plays some role in accurately identifying IEMI events. Optimal window size for each classifier is determined by a one-sided Welch's t-test with $\alpha = 0.05$. For the ANN and SVM classifiers, the best window size is 80 states ($p_{\text{ANN}} < 0.01, p_{\text{SVM}} = 0.01$); the RFC and GBC classifiers achieve best results with a window size of 60 states ($p_{\text{RFC}} < 0.01, p_{\text{GBC}} < 0.01$). While peaks in the graph may appear somewhat later, the increase in performance is not statistically significant. Selecting for a smaller window size reduces classifier memory requirements and produces results more amenable to manual analysis.

Performance across different data synthesis approaches is shown in Figure 6.3b. Again, the performance increase from including sequence information (Lags > 0) can be seen, except for in the HMM. HMM performance, on the other hand, seems to be driven mostly by the number of observable states; fewer observable states leads to better results.

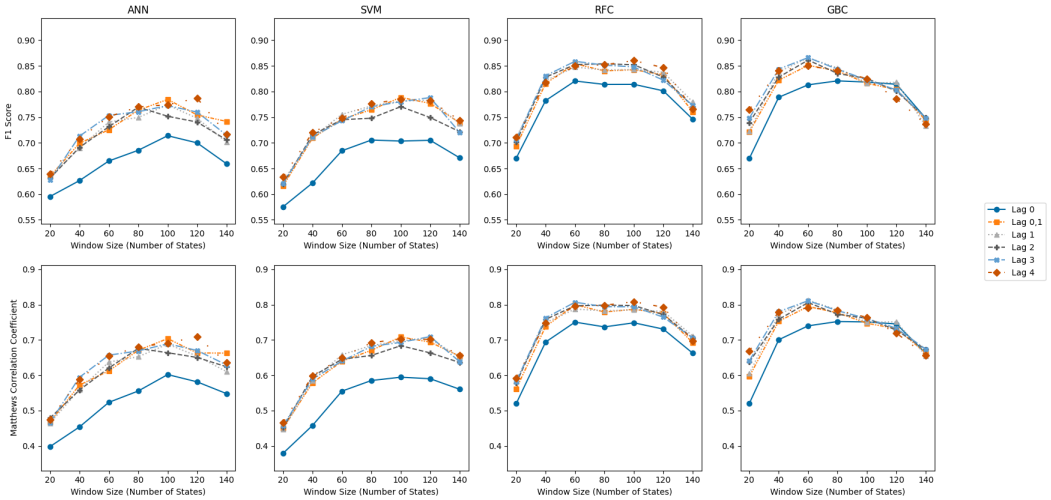
Balancing the class representation in the training dataset does not lead to an improvement in performance as determined by a two-tailed Welch's t-test for inequality, $p > 0.25$. However, we do observe a difference between `BEBE` and `BEEB` data ($p < 0.01$) for all but the HMM classifier. From this, we can conclude that the classifiers struggle the most to classify windows containing states from both `B` and `E` datasets. The increase in performance on the `BEEB` datasets is due to those datasets having only ⅔ of the mixed windows of the `BEBE` datasets. The HMM is an exception to this trend, with equivalent ($p = 0.99$) performance on either dataset.

Table 6.3. Best classifier configurations and performance. Best values in each column are marked with *.

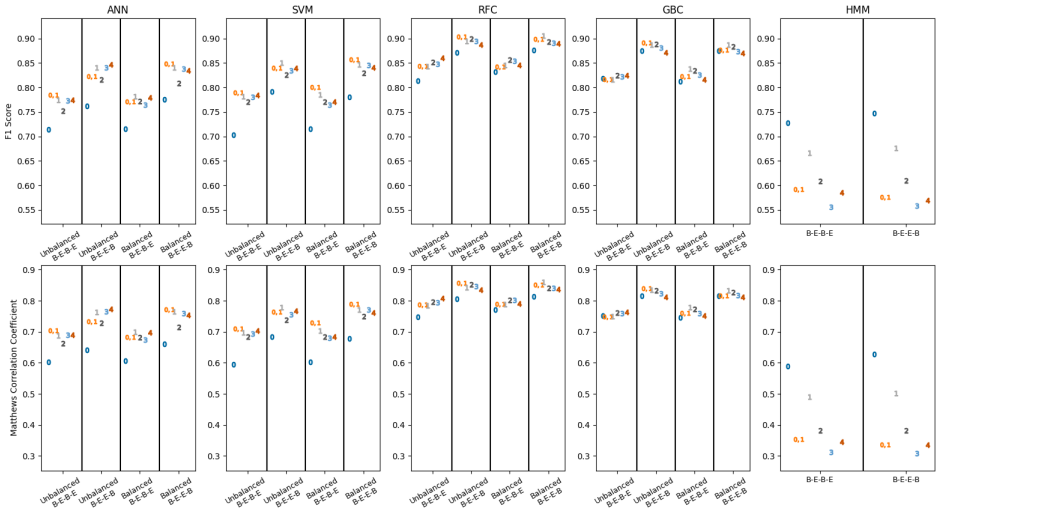
	Metrics				Hyperparameters	
Classifier	F1	MCC	Accuracy	Recall	Window Size	Lag
ANN	0.77	0.68	0.87	0.83	80	4
SVM	0.78	0.69	0.87	0.85	80	4
RFC	0.86	0.81*	0.92*	0.87	60	3
GBC	0.87*	0.81*	0.92*	0.92*	60	3
HMM	0.73	0.59	0.82	0.73	N/A	1

The MCC and F1 scores agree across all results, indicating good classifier performance on identifying both baseline and EMI-exposed states.

Best classification results on the unbalanced `BEBE` dataset are shown in Table 6.3. Of these, the gradient boosted classifier performed the best on this dataset.



(a) Classifier performance by window size on the unbalanced **BEBE** synthetic dataset



(b) Classifier performance by synthesized dataset with a window size of 100 (200 for Lag 0,1)

Figure 6.3. Comparison of classifier performance

7. PART 2: METAMODELING FOR COMPLEX HYBRID SYSTEMS

The goal of this research is to create a basis for sound metamodeling of complex hybrid systems. To this end, we focus on two metamodeling tasks: refinement/generalization and transformation of models.

We use abstract interpretation as a basis for defining operations on models and proving that those operations give meaningful results — that is, results preserve, to the greatest extent possible, the faithfulness of the inputs. Abstract interpretation is a theory of sound approximation of semantics originating in the field of program analysis. We apply it here because all of the metamodeling tasks in question constitute approximation of semantics. Our research provides a common basis for describing metamodeling operations and enables research on one operation to make use of results from studies of other operations.

Abstract interpretation of models relies on three key components: i) "entailment" between systems and properties and between systems and models, ii) modeling formalisms and system properties being complete lattices, and iii) abstraction and concretization operators defined between modeling formalisms and system properties. We use system *properties* to explicitly represent information about a system; often, these properties are derived from models of the system. The "entailment" relationship between systems and models and between systems and properties allows us to establish that, if a model is faithful to a system, then the properties of the model are also faithful to the system and vice versa. Thus we can show that model faithfulness is preserved by our metamodeling operations. Lattices provide a tool for describing approximate knowledge of a system: models and properties are ordered by specificity, giving a way to describe, for instance,

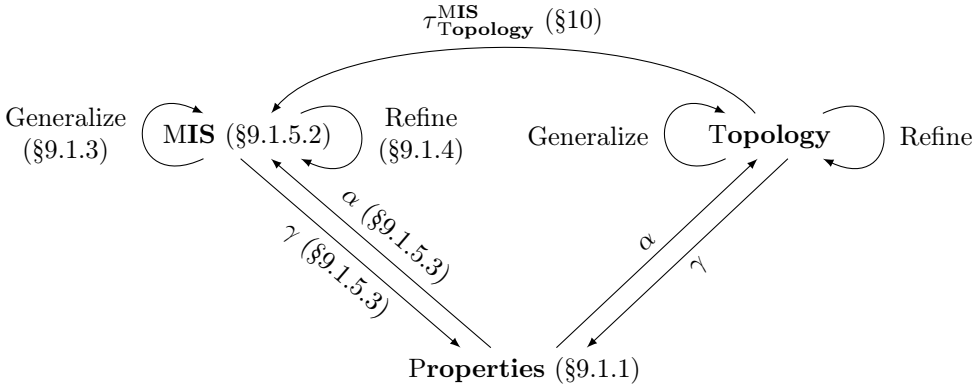


Figure 7.1. Relationships among MIS and Topology modeling domains and Properties domains

models that are consistent with, but more general than, a given model. Finally, the abstraction and concretization operators form a mapping between models and properties that accounts for the inability of models to represent every last detail of a system. Figure 7.1 shows two modeling formalisms and the relationships our metamodeling approach provides among them.

We rigorously formalize these three ideas and demonstrate their application to two common modeling operations: *transformation* and *generalization and refinement*. Our contributions are as follows:

1. Creating a theory of abstract interpretation for models of complex hybrid systems (Section 8) [11, 12].

2. Creating verifiably sound refinement and generalization operations for certain model formalisms (Section 9) [14].

3. Creating an approach to verifiably sound model transformation (Section 10) [11, 13].

The remainder of this section provides a more thorough discussion of system modeling and metamodeling. In Section 7.1, we summarize a typical design process, explaining why models are created and changed in the course of this process. Furthermore, we introduce metamodeling concepts that relate one model to another or a model to the system it represents. In addition, we describe related literature on metamodeling (Section 7.2), refinement and generalization (Section 7.3), and model transformation(Section 7.4), providing a context for our work. Section 7.5 restates the distinction of our research.

Parts of the discussion in this section are in the context of cyber-physical systems (CPSs), rather than the broader context of complex hybrid systems. While our work is not restricted to CPSs, we emphasize these systems because the tight coupling between the physical system and cyber control portions of CPSs makes such systems particularly interesting and challenging from modeling and metamodeling perspectives.

7.1. OVERVIEW OF MODELING AND METAMODELING

Models are created to enable questions to be answered from several perspectives [61]. Broadly speaking, modelers can work from a *descriptive* perspective, creating models of existing systems or physical phenomena in order to understand them, or from a *prescriptive* perspective, creating

models that serve as a specification for a system that is to be built.[1] Most systems are modeled to answer questions such as, "Can this system serve the demand of all its users?", "What parts of this system will fail in the next 20 years?", or "Is it possible for this system to be harmful to its users?". Modelers can also ask more obscure questions: "How easy is it to re-configure this system?", "What is the least it can cost to maintain this system per year?", and many others. The models required to answer these questions incorporate knowledge about the system at varying levels of granularity: a system reliability model may only need to know what components are in the system, while a safety model may depend on specific implementation details of components. These models also incorporate assumptions about the system made both by the modeler when creating the model and by the limitations of the chosen modeling formalism. Furthermore, different modeling formalisms require different evaluation approaches. Physics-based models are often based on differential equations and evaluated through numerical simulation, dependability models incorporate probability distributions and draw conclusions through statistical significance testing, and economic cost models are formulated as optimization problems. Identifying the overlapping knowledge represented by different models of a system thus becomes a truly daunting task.

In order to meaningfully relate models that represent a system, we must first understand how models and systems relate. A system is a physical entity; a model is some description of a system. We denote a model that is a "good" description of a system as *faithful* to that system [62, § 2.2]. Faithfulness is not an absolute property of a model; it is defined relative to "relevant" properties of a system and its models. An absolutely faithful model would be analogous to a map of a forest that is the same size as the forest. Such a map would be unwieldy to use and would need to be constantly re-drawn as plant growth, animal movement, and erosion change the forest being mapped. The process of determining a model's faithfulness is known as *model validation*: checking that a model's predictions correspond with the reality defined by the system [63]. Our central goal will be to define model manipulations that preserve some notion of faithfulness. In other words, modelers should be able to trust these operations to produce models that represent, as much as is possible, the same system the input models represent.

When thinking about model manipulations, it is helpful to start with an understanding of how models evolve. A system representation often starts as a few high-level models that provide a general idea of how the system meets its goals. For example, a power grid representation may start with a topology model that captures generator capacity, approximate load demand, and a

[1] Lee and Sirjani [61] term these as *scientific* and *engineering* perspectives, respectively, but we find these names to be too prescriptive when assigning roles to modelers; an engineer often uses both perspectives when studying an existing system or validating a new one.

general layout of the main distribution lines. In addition, modelers may build an associated reliability model that allows them to verify the system's susceptibility to blackouts. Simulating the topology model allows modelers to infer constraints on line capacity and transformer locations. Likewise, the reliability model can help determine whether additional redundancy is necessary or if the power network needs to be improved to help localize faults.

Once these high-level models demonstrate adequate fidelity, more detail can be added to further specify the system.

Consider a reliability model that starts out relatively crude; additional detail is added over time, *refining* the model. Conversely, use of a model may reveal that it is unnecessarily detailed, motivating *generalization*. Removal of detail is typically denoted as "abstraction" in the literature; we use "generalization" instead to avoid confusion with "abstract interpretation", which we will discuss later. As many modeling choices can be made when creating a model, many refinements for a given model are possible, some of which may contradict each other. Likewise, generalization can undo refinements made to a model, or simplify it to make its simulation feasible. Ideally, such simplifications are done without producing results that would be contradicted by the more detailed model.[2]

Now, consider refining the reliability model begins to reflect the operation of circuit breakers and other protection devices. At this point, the reliability model jumps from a purely probabilistic model to one that reflects discrete-time control operation to some extent. This change in semantics defines a split between two *model formalisms*: one "probabilistic reliability" and the other "probabilistic reliability and discrete time". Another model manipulation comes into play here: we want to *transform* the probabilistic model into one that can also reflect the operation of protection devices. Our goal is to define such transformations in a way that preserves model faithfulness: the transformed model should not contradict the initial model, and both should reflect the system being modeled.

These concepts of *transformation*, *refinement*, and *generalization* are also of interest from a scientific perspective. The process of defining a modeling formalism and refinements and generalizations for it consists of understanding that formalism's perspective of a system, encoding that perspective into a language, and determining what operations are possible in that language. Doing so can lead to a better understanding of the limitations of a given formalism and the assumptions it places on system operation. Model transformation translates a formalism's perspective (and its

[2]What do we mean by "contradicted by" here? The short answer is "it depends on the modeling formalism"; for the long answer, you'll have to keep reading!

operations) into the language of a different model formalism. By creating transformations between formalisms we can, for example, determine what effect changing the topology of a system has on that system's reliability.

7.2. METAMODELING APPROACHES

The task of *metamodeling* is to understand models as entities in themselves rather than strictly a means of analyzing the operation of a system. Informally, metamodeling is "modeling of modeling" rather than "modeling of systems"; to this end, some approaches go so far as to describe metamodels with the same formalisms used to describe models, such as modeling UML diagrams in UML. As with system modeling, there are numerous complementary approaches to metamodeling that vary based on the models they are intended to metamodel, the intended uses of the metamodeling approach, and performance and scalability requirements. We provide an overview of several metamodeling approaches that can be applied to complex hybrid system modeling.

Metamodeling has been extensively studied in the context of software engineering [64]. Most metamodeling approaches in software engineering lack support for the common modeling formalisms used to describe complex hybrid systems. However, some efforts have been made to incorporate continuous-time physical dynamics and non-functional attributes into software metamodeling approaches to allow for metamodeling of CPS control software. One example is CHESS, which links state-based reliability modeling and UML architecture modeling to allow for semi-automated reliability analysis of computer systems [65, 66]. A UML architecture model of system software and hardware components is annotated with failure rates; based on the UML diagram, CHESS can construct a reliability model of the system.

Several modeling formalisms, including Petri nets, Markov chains, fault trees, and bond graphs, can be represented as graphs. Thus, multiple metamodeling approaches represent each model as a graph accompanied by semantics for evaluating that graph. One such approach is AToM^3 [67, 68], which incorporates a variety of formalisms for representing system operation, including differential equations, timed automata, and Petri nets. Individual models are represented graphs, as are metamodels of those models — typically using an Entity-Relationship style notation to describe the edges and vertices of a model as well as permissible connections between them. Metamodeling with graphs allows for model transformations to be represented as graph rewriting rules. This facilitates, for example, conversion of a non-deterministic finite automaton (NFA) to its deterministic counterpart. The source model is searched for patterns specified by rewrite rules; upon a match,

the matched part of the graph is rewritten in the result model. While this approach is powerful, graph pattern matching is a nondeterministic polynomial (NP) problem[3] and it can be infeasible to compute efficiently.

Another graph-based metamodeling approach is the OsMoSys system [69, 70], which is capable of metamodeling many probabilistic graph-based models, including Petri nets, fault trees, and queueing networks. Models using these formalisms can be created and connected through the use of "bridge formalisms" that convert data from one model's semantics to that of another. This allows, for example, the throughput of a queuing network to be determined by a Petri net model [71]. The SIMTHESys project [72, 73] extends this graph-based approach to hybrid systems via the hybrid Petri net and piecewise deterministic Markov process formalisms [74, 75]. Continuous-time dynamics, such as room temperature or traffic speed, can be incorporated into discrete-event probabilistic models with these formalisms.

Heterogeneous model co-simulation techniques, where models of several sub-systems are simultaneously simulated, require identification of some overlaps between different modeling formalisms. The Möbius project [76, 77] takes a state-variable metamodeling approach, which can be applied to models such as Petri nets and stochastic process algebras. Each modeling formalism is described in terms of state variables and actions that update those state variables; Möbius uses these to develop an evaluation strategy that ensures all models are simulated correctly [78]. However, their state-variable and action metamodel is not intended to capture continuous-time physics dynamics.

An alternate approach to heterogeneous co-simulation is taken by Ptolemy [79], which uses "directors" to evaluate submodels [80, 81]. For example, a system consisting of a discrete event controller for a generator may consist of a model with a discrete event director [79, § 1.9]. This model contains a sub-model of the gas-powered generator that uses a continuous-time director to evaluate continuous mechanical and electrical dynamics. The discrete event director evaluates each of the model components in its model; it hands evaluation of the continuous-time submodel over to the continuous-time director. Composing directors is not always possible, depending on the semantics they implement. Models may need special elements for signals to be usable to the containing model, such as applying a discrete-time signal sampler to a continuous-time signal.

A third approach to co-simulation — and more broadly, co-modeling — is taken by the INTO-CPS project [82]. This approach is based on development of a language, CyPhyCircus, which can represent specifications for both discrete- and continuous-time behavior. Models made using familiar tools, such as VDM-RT for control software [83] or Modelica for continuous-time physics [84],

[3]Graph pattern matching requires solving the *subgraph isomorphism* problem, which is NP-complete.

can be mapped into CyPhyCircus specifications. Simulation software can use these specifications to evaluate models [85] or the specifications can be formally verified using an interactive theorem prover [86, 87]. This approach, which is more challenging than Ptolemy's, allows verification of the soundness of mappings between models.

Bhave *et al.* [88] develop a general approach to multi-domain modeling based on architectural views. In this system, a "base architecture" is created which captures all components in the system as well as the physical and communication links between them. Each model of the system has an associated "architectural view" representing the portion of the system relevant to that model; for example, a network queueing model would use a view of only the networked components in the base architecture. Each part of the model is mapped to a component in the view via a many-to-one relationship; each component in the view is mapped to a corresponding part of the base architecture. Restrictions on these relationships can be used to ensure all views are consistent with each other [89]. Furthermore, these views can be used to relate information among models to enable safety verification of the whole system [90, 91].

7.3. REFINEMENT AND GENERALIZATION OF MODELS

Model refinement, broadly speaking, refers to the process of adding more detail to a model, leading to a more precise specification of a system's operation. The exact process of increasing the precision of a model depends on its formalism; thus, refinements are usually developed for each formalism individually. Much of the research on refinement focuses on specification languages, though there is some research into refining performance and dependability models as well. In this section, we survey model refinement approaches in general; in addition, we review refinements of Markov chains to provide context for Section 9.1.

A common application of specification languages is in the field of software engineering, where formal specifications are a key component of several engineering methodologies. Refinement of specifications for software programs has been studied extensively; see [92, 93] for an introduction and [94] for a recent survey of the literature. The essence of program specification and refinement is augmenting a programming language with a specification language. Specifications describe "what" a program should do; as a specification is refined, it begins to also describe "how" a program meets that specification. For example, $\sqrt{x} * \sqrt{x} = x$ is a specification for a square root function; this specification can be refined further until the programmer arrives at an implementation of various root-finding techniques. Thus, programs are specifications that are also executable. To derive programs from

non-executable specifications, a refinement relation is defined and various refinements of specifications are developed. This allows one to start with a high-level specification of a program's behavior and derive, through repeated refinement, an executable program whose specification refines the initial specification. Specifications typically describe the function of a program, but non-functional properties can also be used to guide program refinement [95].

A noteworthy bridge between program refinement and system model refinement is hybrid Event-B, a modeling methodology that views a model at a lower abstraction level of a system as a refinement of one at a higher abstraction level [96, 97]. This methodology is supported by the tool Rodin, which automates part of the modeling process and enables formal verification of control software for hybrid systems. Modeling begins at a high level, describing the controller interface, events that can occur, and invariants that hold for the system. The system is then refined by adding more detail: decomposing events and interfaces into sub-events or sub-interfaces and adding invariants. This requires the modeler to show that the refinement still conforms to the higher-level specification; the proof obligations generated by this refinement can be proven with help from Rodin. This approach has been used to model safety-critical operating systems [98], airplane landing gear [99], pacemakers [100], aircraft fuel supply [101], and numerous other systems.

Research on refinement of Markov chains has taken two forms. The first focuses on Interval Markov Chains (IMCs) and their extension, Constraint Markov Chains (CMCs) [102, 103]. In these formalisms, transition probabilities are not given exactly, but are bounded within an interval or given by algebraic constraints, respectively. As each IMC or CMC corresponds to a collection of Markov chains that satisfy the requirements given, it is possible to define refinement directly in terms of these formalisms, rather than using a separate "system constraints" formalism, as we do. Each system specification can be written as an IMC or CMC and then refined into a complete system model via refinement and conjunction operations.

The second approach uses counterexample generation to validate Markov chain abstractions used in model checking [104]. Starting with a coarse approximation of the original Markov chain, model checking is performed until a counterexample is found. This counterexample is checked against the original specification; if the counterexample does not hold, the approximate system is refined so the counterexample no longer holds. This process repeats until a genuine counterexample is found (one that holds for the original specification) or the model checking algorithm cannot find

a counterexample. A related work [105] bounds the uncertainty introduced by this approach to state-space reduction by separately modeling the uncertainty present in the model and the uncertainty added through abstraction.

A particularly helpful way to think of model refinement is in terms of refinement *operations* that can be performed on a given model. These operations are the smallest changes to the model that can be made while preserving some properties of the original model. One example of this is given by Mitsch *et al.* [106], who define refinement operations for differential dynamic logic, a logic used for writing specifications of hybrid systems. Refinement is based on state reachability: one program refines another if its reachable states are a subset of the other's. The authors develop structural refinement operations that remove redundancies in specifications and behavioral refinement operations that can introduce additional control paths or broaden system behavior. Some of these operations require the modeler to prove that they have indeed refined the original system, but the effort required to write such proofs is significantly less than the effort required to re-verify the entire system specification. Basile *et al.* [107] take an alternate approach to state reachability refinement wherein contract automata models are refined based on predicates that specify unreachable states. This approach is useful when modeling several interacting controllers: each controller can be modeled independently and their resulting automata composed together. The predicates that refine this system specify further assumptions about how the controllers interact. Refinements in contract automata models can be carried out on corresponding stochastic activity net models as well, allowing for partial refinement of hybrid system models.

Another approach by Ishigooka *et al.* [108] focuses on model generalization, rather than model refinement. This is still applicable to refinement, though, since if model a is a generalization of model b, it is also the case that b is a refinement of a.[4] The authors discuss the issue of generalizing models to reduce the effort required to verify models and the resources required to simulate them. They use the Bond Graph modeling formalism, which captures the flow of energy through a system, to model physical system operation. Such models consist of energy storage and transformation components, such as springs, masses, and levers, as well as links ("bonds") between them. Their generalization efforts focus both on generalizing the equations describing each component — for instance, approximating a non-linear spring as a linear spring — and on replacing a group of related components with fewer components that approximate the more complex operation of the original group. This process is semi-automated, with generalizations being applied from a database and the results checked for unacceptable errors by both design and verification engineers. However, no formal

[4] Mathematically, we say that generalization is *dual* to refinement.

description of the system properties to be preserved by generalization is offered, nor is a means by which the generalization process itself can be verified. Thus, the modelers of the system must independently verify whether a model produced by this process is truly a generalization of the source model.

7.4. MODEL TRANSFORMATION APPROACHES

The field of model transformation contains numerous approaches for solving a variety of related, but distinct, problems. We will focus on transformations between disparate modeling formalisms at the same level of abstraction relative to the system they model — what Mens and Van Gorp [109] would call *horizontal, endogenous, semantical* model transformations. Examples of such transformations include mapping a power grid architecture model to a set of differential equations describing its operation or to a model of its reliability. A related, but distinct, concern is transforming several models of system components into a model of the whole system; we refer to this as *model composition*.

One can think of a model transformation as a *mapping* from a source model (or models) to a target model. In addition to studying these mappings, we study the properties these mappings may have [109]. One such property is *bidirectionality*, meaning that a mapping from a source to a target also produces a reverse mapping from the target to the source. These may be seen as analogous to the mathematical notion of *invertible functions*; however, invertible transformations are only a subset of bidirectional transformations. Applying a bidirectional transformation followed by its reverse transformation need not produce the original source model. Bidirectionality is desirable because it can reduce the effort required to develop reverse transformations or even allow them to be automatically generated. However, producing meaningful reverse transformations can be challenging; for instance, if we think of compiling a C program as a transformation, there are potentially infinite C programs (commonly referred to as "disassemblies") which produce the same machine code.[5]

Another useful property of model transformations is *automation*: being able to operate without user input or direction. From a practical perspective, automated transformations are easier to use and thus more useful; however, sufficiently complex transformations may be difficult to automate. Related is the ability to propagate changes in a source model to an already-existing target

[5]For instance, the x86 assembly `add EAX, ECX` could be the result of compiling `total = sub_total + tax;`, `index += increment;`, or even `running_sum(count);` if the function `running_sum` is inlined.

model, possibly without having to re-compute the entire transformation. Avoiding re-computation is especially useful in the case where the transformation is not entirely automatic. Keeping models synchronized as they evolve is essential to maintaining a consistent set of system models.

Finally, it is desirable for model transformations to be *verifiably sound*. Given a source model that corresponds to a system, a verifiably sound model transformation produces a target model that can also be shown to correspond to that system. In other words, a sound model transformation preserves some properties of the system specified by the source model. Continuing with the C programming language example, a *verified* C compiler is guaranteed to produce an assembly program that replicates a C program's behavior, as specified by the C standard's abstract machine.[6]

Model transformation has been studied extensively in the context of software engineering, primarily from two perspectives: generating source code from software diagrams (such as UML diagrams), and generating performance models from software models [109]. Much of this work focuses on transforming functionality models to implementations or performance models [110], but there has been some focus on transforming functional models to dependability models as well [111–113]. None of these transformations are specifically designed to be bidirectional, although there are some practical implementations of deriving UML diagrams from source code [114]. Some of this research has focused on using types and type safety to build and verify these transformations [115]. Unfortunately, models from software engineering do not provide good means for describing the continuous-time dynamics of physical systems, so any techniques will have to be adapted to be applicable to CPSs [116, 117].

Existing literature includes several different approaches to metamodel-based model transformation for complex hybrid systems. UML class diagrams are used to metamodel Simulink and Architecture Analysis and Design Language (AADL) models in [118, 119]. Simulink models partially describe the physics of a system as well as controller operation. AADL models describe the architecture of a system and can be used for model checking and other model validation purposes. This process requires some annotation of the Simulink source model to add information required for the AADL model. After the transformation, the AADL model can be refined with additional execution and timing details for verification.

A Formalism Transformation Graph is developed in [5, 62] and used as a basis for model transformation in AToM^3 [68]. This graph encodes the transitions defined among a number of formalisms, including bond graphs, differential algebraic equations (DAEs), Petri nets, and cellular

[6]Most C compilers are not verified to be correct, and sometimes produce incorrect assembly. The C abstract machine also does not always provide the guarantees that C programmers expect, leading to compilers taking optimizations that seem wrong but are legally "correct". Choosing which properties a transformation must preserve is an ongoing engineering challenge.

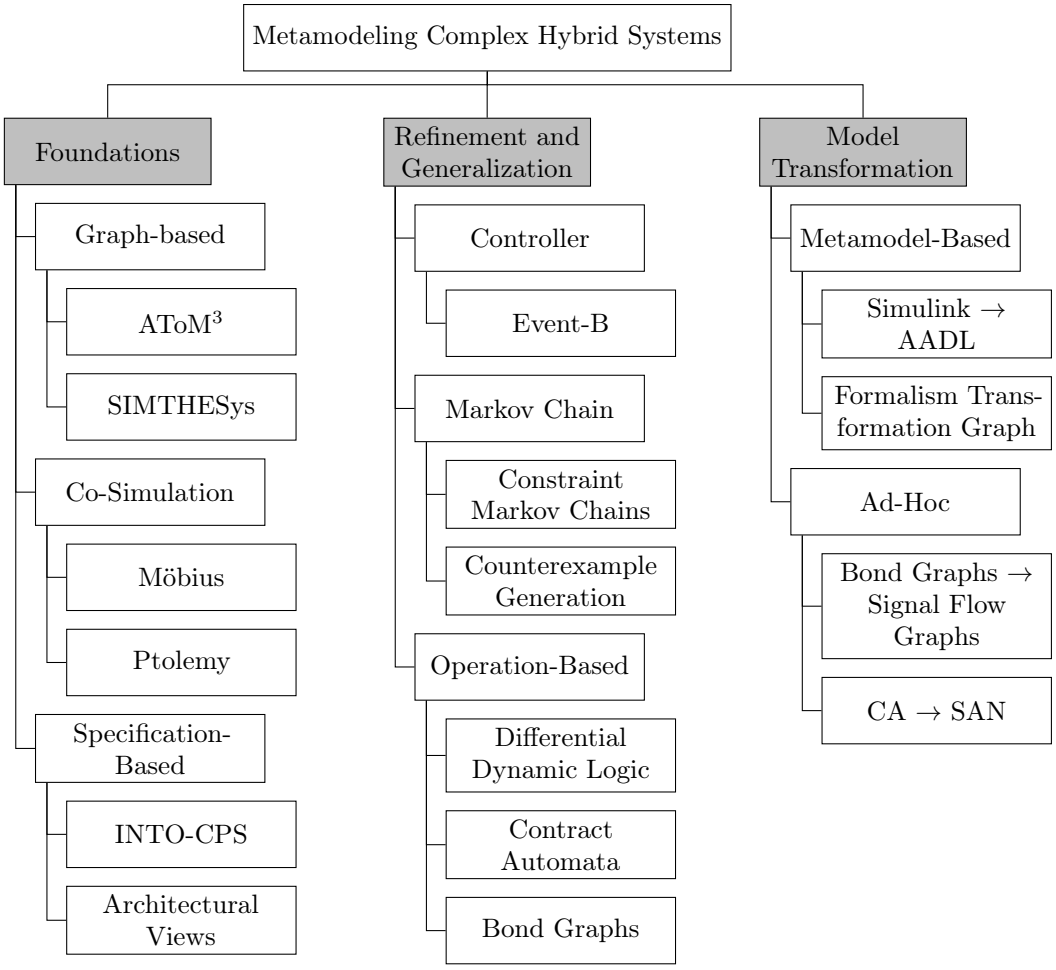


Figure 7.2. Overview of related literature

automata. Transformations can be bidirectional, but most are homomorphic relationships where some information from the source model is not preserved in the target model. The most significant instance of this information loss is in the transformation of continuous-time models, such as bond graphs and DAEs, to discrete-event modeling formalisms. However, this can be an advantage for producing simulation models: a continuous-time physics-based model can be transformed into a discrete time model, then merged with discrete-time control models to produce a discrete-time simulation of the entire system. A distinct advantage of the formalism transformation graph is that introducing a new formalism requires one to define only a single transformation to the next "closest" formalism. Transformations to other formalisms are then performed by a series of transformations, each one step through the graph from the source formalism to the target formalism.

Numerous examples of ad-hoc model transformations for complex hybrid systems can be found in the literature. Ishigooka *et al.* [108] develop a transformation from bond graphs, which encode continuous-time power flow through a physical system, into discrete-time signal flow diagrams. These signal flow diagrams can be combined with other discrete-time control elements and easily simulated. The transformation is automatic, but not designed to be bidirectional or verifiable. Basile *et al.* [107] take a different approach, starting with contract automata (CA) models of controllers, then transforming them to stochastic activity network (SAN) models, which allow for representation of continuous-time thermodynamics. They show that this automatic process produces models that are bisimilar to the input models, meaning that verified properties of CA models also hold for their SAN equivalents.

7.5. SUMMARY

The modeling of complex hybrid systems is a multifaceted challenge, as reflecting different aspects of system operation may necessitate the use of several distinct models. Maintaining consistency among the various models of a system, enabling modelers to add information about a system to a model, and developing new models from existing ones are can be accomplished with metamodeling. While metamodeling encompasses a variety of operations on models, we focus on three particular aspects: metamodeling foundations, model refinement, and model transformation. Related work on these topics is summarized in Figure 7.2.

Our work examines the interaction between model transformation, model refinement, and sound metamodeling techniques — ones which produce faithful models. We also identify desirable properties of metamodeling approaches motivated by a desire to develop multi-purpose metamodeling approach, rather than focusing exclusively on, e.g., co-simulation. In this aspect, our work complements that of the INTO-CPS project, making explicit some of the foundational choices they have made. As model refinement is central to our work, we depend on the existing literature for refinement operations, but extend those approaches to other formalisms used in modeling of complex hybrid systems. Finally, we aim to provide a more formal basis for transformations between models, ensuring soundness and bidirectionality. This approach has the unique benefit of specifying system properties that are preserved through a particular transformation.

8. ABSTRACT INTERPRETATION OF MODELS

In this section, we propose a theory of Abstract Interpretation for system modeling. This theory provides a foundation for reasoning about the semantics of systems and developing approximations of those semantics. We use it to relate the semantics of a system to the semantics of models of that system. A more traditional development of abstract interpretation in the context of program execution can be found in Appendix C.

8.1. SOUNDNESS AND COMPLETENESS

Two properties of approximate semantics are relevant to our work. First, an approximate semantics is *sound* if it always produces results that contain the exact result. For example, if we approximate whether $-5 + 2$ is positive, negative, or zero, the result "any option" is sound, as is "negative or zero" or just "negative", but the approximation 'positive or zero" is not sound. In other words, sound approximations at worst overapproximate the correct result. Second, an approximate semantics is *complete* if it always produces the exact result. Thus, in the previous example, only the answer "negative" is acceptable for a complete semantics. Completeness is a rather strong requirement, so complete semantics are uncommon.

8.2. SEMANTICS OF PROGRAMS AND SYSTEMS

In these definitions, we talk of "exact results"; by what process are these results produced? The equation $-5 + 2$ is a starting state; it is reduced to -3 by a computation following the exact semantics of arithmetic on the integers. Likewise, our approximations of the sign of the result are computed following some approximate semantics. Finally, there is some underlying logic and equality that allow us to make statements about these results.

More precisely, we might understand this as a program p that takes a pair of inputs and computes their sum, i.e., $(x, y) \mapsto x + y$. In the context of p, its concrete semantics reduce $(-5, 2)$ to -3; we write this as $p \vdash (5, -2) \rightsquigarrow -3$. Likewise, a sound approximate semantics of p might give $p \vdash (\textsc{negative}, \textsc{positive}) \rhd \textsc{negative}$. Abstract interpretation provides a means of relating these states and semantics to show their soundness.

This notion of states and transitions also applies to physical systems. We view model semantics as abstractions of system semantics. A system transitions from one physical state to another; this may entail a change in a model's state. For some system $\mathcal{S}$, concretely we have

$\mathcal{S} \vdash s_1 \rightsquigarrow s_2$ and abstractly $\mathcal{S} \vdash m_1 \rhd m_2$. These transitions take a variety of forms: for instance, an air conditioner turns on, increasing load on a power grid; scale builds up in a pipe, reducing flow in a water distribution network; a circuit breaker fails shut, reducing the ability of a control system to contain a future fault. Because the model semantics $\rhd$ is an abstraction of the system's behavior $\rightsquigarrow$, not every state transition will lead to a model transition. This allows models to only capture certain behaviors of a system.

The remaining component of our theory thus far is the context in which these transitions take place: p or $\mathcal{S}$, respectively. For programs, p is some syntactic representation of that program and $\vdash$ is some notion of logical entailment that establishes the relationship between p's syntax and the various interpretations of p. Unfortunately, physical systems lack a well-defined syntax; however, this is not an insurmountable issue. The purpose of $\vdash$ is to establish a relationship between a system $\mathcal{S}$ and possible behaviors of that system and of models of that system. This relationship is known in the modeling world as "faithfulness": the extent to which behaviors of a model reflect the behaviors of a system. Thus, when we write $\mathcal{S} \vdash m_1 \rhd m_2$, we mean that the given model transition is faithful to the physical system's behavior.

8.3. SPECIFYING SYSTEM SEMANTICS

Another challenge to applying abstract interpretation to system modeling is the issue of representing system semantics. There is no generally accepted means to exactly specify a cyber-physical system's semantics, though several logics have been proposed for specifying various properties. Fortunately, an exact representation of semantics is not necessary for this approach; we can achieve many of our goals by working with the abstract semantics of models and the less-abstract semantics of partially specified system properties.

Whether modelers are developing a design of a new system or describing an existing system, it is quite unlikely for them to fully know every detail of that system. Thus, we build a domain of system properties with a notion of "specificity" which allows properties to be refined as the modeling and validation process continues. Lattices (see Appendix A) offer a useful formalism for describing such a domain.

We define a complete $\mathbf{Properties}$ lattice ordered by specificity: for $p_1, p_2 \in \mathbf{Properties}, p_1 \sqsubseteq p_2$ means that the constraints in p_1 and p_2 are not contradictory and that p_1 places the same or more constraints on a system than p_2 does. For example, p_2 could constrain the reliability of a component to fall in the range $(0, 1]$, whereas p_1 could require that component to have a reliability of 0.95.

The *meet* (denoted as $\sqcap$) of two elements of P**roperties** places the constraints of both elements on a system; the *join* (denoted as $\sqcup$) implies satisfaction of the constraints of either element. Suppose p_1 requires a component's reliability to fall in $[0.8, 1.0]$ and p_2 constrains it within $[0.75, 0.9]$. Then $p_1 \sqcap p_2$ will require it to be in $[0.8, 0.9]$ and $p_1 \sqcup p_2$ within $[0.75, 1.0]$. $\bigsqcap$ and $\bigsqcup$ extend this concept to subsets of P**roperties**.

For certain $p_1, p_2 \in$ P**roperties** are contradictory, $p_1 \sqcap p_2$ will result in a constraint that is impossible to satisfy. If p_1 requires a component to have a reliability in $[0.5, 0.7]$ and p_2 requires it in $[0.9, 1]$, then it is impossible for any component to meet both constraints. In this paper, we require that every element of P**roperties** to be satisfiable except for $\bot$, the "impossible" constraint. Therefore, for this example, $p_1 \sqcap p_2 = \bot$. Note that $\forall p \in$ P**roperties**, $\bot \sqsubseteq p$.

To summarize, each element of the P**roperties** lattice describes one or more systems. In the general case, p describes a set of systems, all of which meet the constraints in p. If every constraint in $p \in$ P**roperties** has exactly one possible choice, p will describe a single system.

8.4. SPECIFYING MODEL SEMANTICS

We now consider how modeling formalisms can be represented in a way that is compatible with P**roperties**. As a given element of the P**roperties** lattice may not define a single system, we must account for the possibility that multiple models describe the given properties. For example, if $p \in$ P**roperties** does not constrain the reliability of a component to a single value, several reliability models may plausibly be abstracted from p. Therefore, in the same way that the P**roperties** lattice is defined, we also define the domain of each modeling formalism to account for the nature of potentially imprecise system specifications. This allows us to abstract a "most general" model from p.

To ensure that this approach is broadly applicable, we can define this domain using structure external to the modeling formalism itself. Thus we do not have to require, say, that a reliability model formalism be able to express the concept of a component having a range of possible reliabilities. We use a *powerset lattice* to provide this extra structure. For a given model formalism, the set **Model** contains all possible models expressible in that formalism. The powerset lattice $\mathcal{P}$(**Model**) then forms a lattice ordered by specificity: for $M_1, M_2 \subseteq$ **Model**, $M_1 \subseteq M_2$ indicates that M_1 contains fewer possible models describing a system, and thus places more constraints on the system, than M_2 does. Likewise, $M_1 \cap M_2$ produces a set of models that fit the constraints associated with M_1 and

with M_2; $M_1 \cup M_2$ produces a set of models where constraints from either may hold. Singleton sets (i.e., sets of the form $\{m\}$, $m \in \mathbf{Model}$) correspond to fully-specified models, and $\emptyset = \bot$ corresponds to an "impossible" system—one with contradictory modeling requirements.

However, this construction does have a tradeoff: P**roperties** must also be a powerset lattice. Otherwise, it is not possible for P**roperties** to fully capture the constraints given by a set of models. This may lead to "property explosion" or "model explosion" (in the sense of "state space explosion" faced by some model formalisms).

An alternate solution that avoids these issues and places fewer constraints on P**roperties** is to imbue a model formalism itself with some specificity ordering. Section 9 discusses this approach and develops it for a reliability modeling formalism.

Either way, we will assume we have a set of models of a given formalism, **Model**, a lattice of these models ordered by specificity, M**odel**, and an inclusion function $\mathsf{lift} : \mathbf{Model} \to \mathrm{M}\mathbf{odel}$ that connects the two. In the case where M**odel** $= (\mathcal{P}(\mathbf{Model}), \subseteq)$, we have $\mathsf{lift}(m) = \{m\}$. For the lattice M**odel**, we will use the rounded operators $(\subseteq, \bigcap, \bigcup)$ to prevent confusion with the square operators of the lattice P**roperties**, and of lattices in the abstract.

8.5. RELATING MODELS AND PROPERTIES

Any system $\mathcal{S} \in \mathcal{S}\mathbf{ys}$ is described by a number of elements of P**roperties**. To formalize this notion, we use a *correctness relation* to relate a system to properties (and later, models) that describe it. We suppose a relation $R_{\mathrm{P}} : \mathcal{S}\mathbf{ys} \to \mathrm{P}\mathbf{roperties}$ where $\mathcal{S}\, R_{\mathrm{P}}\, p$ if and only if p describes the system $\mathcal{S}$. We must assume the existence of R_{P}, since the properties of the system being designed are determined by the designer. However, abstract interpretation allows us to induce correctness relationships between systems and models based on R_{P}—in other words, abstract interpretation enables sound transformations between system properties and system models.

Definition 8.1. A *correctness relation* $R_{\mathrm{L}} : \mathcal{S}\mathbf{ys} \to \mathrm{L}$ relates systems to elements of a lattice L. Two attributes hold for R_{L}:

(i) If $\mathcal{S}\, R_{\mathrm{L}}\, l_1$ and $l_1 \sqsubseteq l_2$, then $\mathcal{S}\, R_{\mathrm{L}}\, l_2$.

(ii) If $\forall l \in \mathbf{L}' \subseteq \mathrm{L}, \mathcal{S}\, R_{\mathrm{L}}\, l$, then $\mathcal{S}\, R_{\mathrm{L}} \bigsqcap \mathbf{L}'$.

$$\begin{array}{ccccc} \mathcal{S} & \vdash & s_1 & \rightsquigarrow & s_2 \\ & & \| & & \| \\ & & R_{\mathrm{M}} & \Rightarrow & R_{\mathrm{M}} \\ & & \| & & \| \\ \mathcal{S} & \vdash & m_1 & \rhd & m_2 \end{array}$$

Figure 8.1. Soundly connecting models, properties, and systems

In terms of P**roperties** and its correctness relation R_{P}, Property (i) states that we can relax correct constraints without violating their correctness. The reverse does not hold, otherwise, the inconsistent constraint $\perp$ would describe every system. The formalization of relaxation of constraints as described by Property (i) allows us to generalize constraints and therefore plays a crucial role in modeling abstraction.

Property (ii) requires that for any set of constraints $\mathbf{L}'$ there exist a "best" constraint that correctly describes any system described by every constraint in $\mathbf{L}'$. We can apply this property to the constraints derived from several models to narrow down our description of a given system's properties. In this sense, it allows us to derive a specific result from a number of more general results. Note that the converse of (ii) follows from (i), so (ii) could also be written as a biconditional.

Given these correctness relationships, we can connect models and properties soundly, as shown in Figure 8.1. Suppose that the system $\mathcal{S}$ transitions ($\rightsquigarrow$) from state s_1 to s_2. If our model soundly abstracts (R_{M}) the initial system state s_1 with model state m_1, and the model of $\mathcal{S}$ transitions ($\rhd$) from state m_1 to state m_2, then we know that m_2 soundly abstracts (R_{M}) the final state s_2. This conclusion holds under the assumption that the model is faithful to the system ($\mathcal{S} \vdash$).

8.6. ABSTRACTION AND CONCRETIZATION

Given a correctness relation R_{P} for P**roperties**, we desire to define a mapping between P**roperties** and a modeling formalism M**odel** that induces a correctness relation $R_{\mathrm{M}} : \mathcal{S}\mathbf{ys} \rightarrow \mathrm{M}\mathbf{odel}$. Furthermore, this mapping must allow for the modeling domain to abstract system constraints. For instance, a topology model should be able to discard constraints on component reliability.

The formalism of choice for this task is a *Galois connection*:

Definition 8.2. A *Galois Connection* $(\mathrm{P}, \alpha, \gamma, \mathrm{M})$ between two complete lattices P and M consists of a pair of monotone functions $\alpha : \mathrm{P} \to \mathrm{M}$ and $\gamma : \mathrm{M} \to \mathrm{P}$ for which the following relationships hold:

$$(\gamma \circ \alpha)(p) \sqsupseteq p \tag{8.1}$$

$$(\alpha \circ \gamma)(m) \sqsubseteq m \tag{8.2}$$

We refer to P as the *concrete domain*, M as the *abstract domain*, α as the *abstraction operator*, and γ as the *concretization operator*.

In terms of models and properties, α abstracts a model, m, from a set of constraints on a system, p, and γ derives, or *concretizes*, system constraints from a model of that system. Relationship (8.1) states that abstracting the model m from constraints p, then concretizing constraints from that model, results in constraints that are at most more general than those of p. In other words, abstraction may relax constraints irrelevant to the model formalism, but it cannot produce a model that implies constraints that contradict p. Relationship (8.2) requires that P**roperties** be able to completely capture the constraints imposed by each model formalism, meaning that if constraints are concretized from a model, m, of a system, any other model abstracted from these constraints will be as least as specific as the original model, m. Concretization may introduce additional constraints, but in practice, the $\sqsubseteq$ of (8.2) will often be strict equality in practice.

We describe the interactions between these lattices and the underlying set of models via the diagram in Figure 8.2.

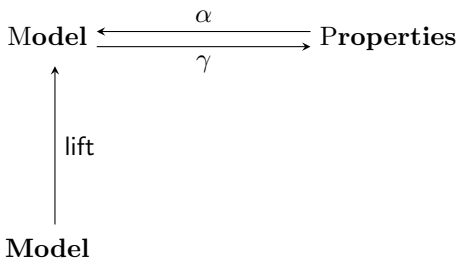


Figure 8.2. Model and system properties interaction diagram

Next, we show that each Galois connection induces a correctness relation R_{M} on the abstract domain.

Theorem 8.1. *Given a Galois connection* $(\mathrm{P}, \alpha, \gamma, \mathrm{M})$ *and a correctness relation* $R_{\mathrm{P}} : \mathcal{S} \to \mathrm{P}$*, the relation* $R_{\mathrm{M}} : \mathcal{S} \to \mathrm{M}$ *defined by* $s \, R_{\mathrm{M}} \, m \iff s \, R_{\mathrm{P}} \, \gamma(m)$ *is a correctness relation.*

Proof. We must show that properties ((i)) and ((ii)) from Definition 8.1 hold for R_{M}. Take $s \in \mathcal{S}$ and $m_1, m_2 \in \mathrm{M}$.

$$
\begin{aligned}
& s \, R_{\mathrm{M}} \, m_1 \wedge m_1 \sqsubseteq m_2 \\
\iff & s \, R_{\mathrm{P}} \, \gamma(m_1) \wedge m_1 \sqsubseteq m_2 && \text{(Defn. of } R_{\mathrm{M}}\text{)} \\
\iff & s \, R_{\mathrm{P}} \, \gamma(m_1) \wedge \gamma(m_1) \sqsubseteq \gamma(m_2) && (\gamma \text{ monotone}) \\
\implies & s \, R_{\mathrm{P}} \, \gamma(m_2) && \text{(Prop. ((i)) for } R_{\mathrm{P}}\text{)} \\
\iff & s \, R_{\mathrm{M}} \, m_2 && \text{(Defn. of } R_{\mathrm{M}}\text{)}
\end{aligned}
$$

The proof of ((ii)) uses the fact that γ is completely multiplicative, that is, $\bigsqcap\{\gamma(m) \mid m \in \mathbf{M}'\} = \gamma\left(\bigsqcap \mathbf{M}'\right)$. Take $s \in \mathcal{S}$ and $\mathbf{M}' \subseteq \mathrm{M}$.

$$
\begin{aligned}
& \forall m \in \mathbf{M}' s \, R_{\mathrm{M}} \, m \\
\iff & \forall m \in \mathbf{M}', s \, R_{\mathrm{P}} \, \gamma(m) && \text{(Defn. of } R_{\mathrm{M}}\text{)} \\
\implies & s \, R_{\mathrm{P}} \bigsqcap \{\gamma(m) \mid m \in \mathbf{M}'\} && \text{(Prop. ((ii)) for } R_{\mathrm{P}}\text{)} \\
\iff & s \, R_{\mathrm{P}} \, \gamma\left(\bigsqcap \mathbf{M}'\right) && \text{(Multiplicativity of } \gamma\text{)} \\
\iff & s \, R_{\mathrm{M}} \bigsqcap \mathbf{M}' && \text{(Defn. of } R_{\mathrm{M}}\text{)}
\end{aligned}
$$

□

Put in terms of models and system properties, if we define a Galois connection between P**roperties** and the lattice for a given modeling formalism M**odel**, then every correct collection of system constraints abstracts to a correct model and every correct model concretizes to a correct collection of system constraints. Therefore, this is a provably sound definition of the nature of model abstraction.

9. REFINEMENT & GENERALIZATION

The concepts of model *refinement* and *generalization* necessarily operate relative to specific system properties. Depending on the exact model, we can change numbers in equations or constraints, insert or remove edges in graphs, add or remove equations, and much more; determining which of these operations are refinements and which are generalizations depends wholly on the semantics of the model formalism. Therefore, when creating model transformations, we will explicitly describe these properties and how they relate to the model formalism in question.

For example, consider a model which captures the components in a system along with their reliabilities. Roughly speaking, if the model m_r is a refinement of a model m_g, the constraints imposed on the system by m_r should imply the constraints imposed by m_g. For example, if m_r requires a component c to have reliability ≥ 0.9, m_g can require that c have reliability ≥ 0.7—this constraint is strictly weaker than the constraint of m_r. However, m_g could not require c to have reliability ≥ 0.99. In other words, a system meeting the requirements of m_r would provide equal or better reliability than a system meeting m_g's requirements alone. If m_r refines m_g, then m_g generalizes m_r, so we can use the same implication relationship to describe both refinement and generalization. We formally abstract system properties and implication to analyze the soundness of our definitions of generalization and refinement.

9.1. MARKOV IMBEDDABLE STRUCTURE MODELS

We propose a method for generalization and refinement of Markov Imbeddable Structure (MIS) reliability models, where the system-level state of a system is determined based on the state of its individual components. The initial state is one where every component is functional, the terminal state is one where enough components have failed to cause system failure, and intermediate states correspond to the system remaining functional despite the failure of some of its components. These models describe a system composed of n components as a Markov chain, encoding each component's reliability and the effect of its failure on other components. The reliability of the system is then the probability that the system remains functional after taking n steps through the Markov chain. Our work focuses on MIS models where the states of the Markov chain are defined by component status (e.g., "component 3 failed" or "only component 2 functional") and where the component status described by a state remains the same regardless of which component's failure is being considered.

This encompasses the vast majority of MIS models, especially as used in practice; however, it does not encompass certain unusual MIS models, such as models of consecutive-k-of-n systems.[1] These we will address in the future.

9.1.1. Properties of MIS Reliability Models. The MIS models we consider in this work place three broad constraints on a system: what components are in the system, how reliable each component is, and which components depend on others to remain functional. The properties domain P**rop** defines these as a lattice, allowing us to relate these properties to MIS models.

As we will need some way to identify components, let $\mathbf{Comps} \triangleq \{c_1, c_2, \dots\}$ be the set of all possible component names.

Each element $p \in \text{P}\mathbf{rop}$ is a triplet $p = (\mathbf{C}, \mathsf{R}, \mathbf{D})$ where

- $\mathbf{C} \subseteq \mathbf{Comps}$ is the finite set of names of components in the system (e.g., $\{c_1, c_2, c_3\}$);
- $\mathsf{R} : \mathbf{C} \to [0, 1]$ is a function that specifies a lower bound for the reliability of each component: if the reliability of c is p, then $\mathsf{R}(c) \leq p$; and
- $\mathbf{D} \subseteq \mathbf{Deps}$ is the finite set of component dependencies, as described in the next section.

For example, a system consisting of two 90% reliable power lines in parallel where the failure of one causes the other to become overloaded and thus fail as well would be described by the properties $(\mathbf{C} = \{c_1, c_2\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = 0.9, \mathbf{D} = \{\langle c_1 \rightsquigarrow c_2, \mathcal{S}\rangle, \langle c_2 \rightsquigarrow c_1, \mathcal{S}\rangle\})$.

Component dependencies (elements of **Deps**) are represented by the relation $\langle \_ \rightsquigarrow \_ \rangle : \mathcal{P}(\mathbf{C}) \to \mathcal{P}(\mathbf{C} \cup \{\mathcal{S}\})$.[2] The statement $\langle \cdots_1 \rightsquigarrow \cdots_2 \rangle$ means "the failure of the components in the set $\cdots_1$ immediately leads to the failure of the components in $\cdots_2$". Should $\mathcal{S}$ appear in $\cdots_2$, the system also fails as a result of the components of $\cdots_1$ failing. The components on the left side ($\cdots_1$) are referred to as *causes* and the components on the right ($\cdots_2$) as *effects*.

These dependencies correspond to state transitions. Suppose we have a system with components $\mathbf{C} = \{c_1, c_2, c_3\}$. We can represent the state of the components as three-bit strings: (111) corresponds to the system state where all components are functional, (101) corresponds to the state where c_2 has failed, etc. A dependency $\langle c_1 \rightsquigarrow \emptyset \rangle$ corresponds to a transition from (111) to (011) when c_1 fails—the failure of c_1 does not influence the functionality of other components in the system. Likewise, a dependency $\langle c_1, c_2 \rightsquigarrow c_3, \mathcal{S} \rangle$ corresponds to transitions from (101) to (000) when c_1 fails and from (011) to (000) when c_2 fails; furthermore, in state (000) the system is considered failed. Sec. 9.1.5 formalizes this correspondence.

[1] In short, the transition probability matrices for consecutive-k-of-n systems are not upper triangular; for more detail, see [120, pp. 344–345].

[2] $\mathcal{P}(\mathbf{S})$ denotes the set of subsets ("powerset") of the set $\mathbf{S}$.

As there are a number of ways to write dependencies, we place some constraints on them to ensure the constraints on the system are consistent with how components fail and fully cover all cases of system behavior. These constraints are split into *equivalences* and *well-formedness (WF) properties*.

9.1.1.1. Equivalences. The first equivalence rule states that if a component appears on both sides of $\rightsquigarrow$, we can remove it from the right side. The failure of any component trivially causes that component to fail; this rule states that we need not write this fact explicitly:[3]

$$\langle c \cdots_1 \rightsquigarrow c \cdots_2 \rangle \equiv \langle c \cdots_1 \rightsquigarrow \cdots_2 \rangle. \tag{Tautology}$$

The remaining two equivalences are between sets of dependencies, rather than between two individual dependencies. If we have two dependencies with the same cause but different effects, we can produce one dependency that represents both by taking the union of their effects:

$$\left\{ \begin{array}{l} \langle \cdots_1 \rightsquigarrow \cdots_2 \rangle \\ \langle \cdots_1 \rightsquigarrow \cdots_3 \rangle \end{array} \right\} \equiv \left\{ \langle \cdots_1 \rightsquigarrow \cdots_2 \cdots_3 \rangle \right\}. \tag{Union}$$

Finally, a dependency with no causes cannot occur:

$$\{ \langle \emptyset \rightsquigarrow \cdots \rangle \} \equiv \emptyset. \tag{Inaction}$$

9.1.1.2. Well-formedness properties. The WF properties describe a system-level view of dependencies: what dependencies need to be present in $\mathbf{D}$ to make a consistent set of system constraints. First, every component must have a dependency where it is the sole cause of failure (although the effect may be the empty set). These correspond to transitions from the initial $\overline{1 \cdots 1}$ state:

$$\forall c \in \mathbf{C}, \exists \langle c \rightsquigarrow \cdots \rangle \in \mathbf{D}. \tag{Initiality}$$

In addition, at least one sequence of failures must lead to the system failing (otherwise, the system's reliability would be 1 and there would be nothing to model):

$$\exists \langle \cdots_1 \rightsquigarrow \mathcal{S} \cdots_2 \rangle \in \mathbf{D}. \tag{Termination}$$

[3] A note on notation: $c \cdots_1$ refers to a set containing the component c and the components of the set $\cdots_1$.

Finally, components cannot recover as a result of the failure of other components. Thus, if components $\cdots_1$ cause components $\cdots_2$ to fail, any other dependency where $\cdots_1$ have failed must also have $\cdots_2$ failed.

$$\begin{aligned} &\forall\langle \cdots_1 \rightsquigarrow \cdots_2\rangle \in \mathbf{D}, \\ &\forall\langle \cdots_1 \cdots_3 \rightsquigarrow \cdots_4\rangle \in \mathbf{D}, \\ &\cdots_2 \subseteq \cdots_3 \cup \cdots_4 . \end{aligned} \qquad \text{(Monotonicity)}$$

For instance, if we have $\langle c_1 \rightsquigarrow c_2\rangle$, Monotonicity would permit the dependencies $\langle c_1, c_3 \rightsquigarrow c_2\rangle$ and $\langle c_1, c_2 \rightsquigarrow c_3\rangle$ but forbid $\langle c_1, c_3 \rightsquigarrow \emptyset\rangle$, as c_2 must always fail when c_1 fails.

9.1.2. Examples. Before addressing generalization and refinement of properties, we demonstrate a few examples of how dependencies are used to specify system behavior. First, consider the dependencies in the earlier parallel-component example: $\mathbf{D} = \{\langle c_1 \rightsquigarrow c_2, \mathcal{S}\rangle, \langle c_2 \rightsquigarrow c_1, \mathcal{S}\rangle\}$. In this system, the failure of component c_1 leads to the failure of c_2 and system failure, and vice versa for c_2. This system has two states, (11) and (00); the failure of either component causes a transition from the first to the second.

By contrast, a parallel-component system where the two components are independent would be specified by $\mathbf{D} = \{\langle c_1 \rightsquigarrow \emptyset\rangle, \langle c_2 \rightsquigarrow \emptyset\rangle, \langle c_1, c_2 \rightsquigarrow \mathcal{S}\rangle\}$. This system has all four possible states and all valid transitions between states.

A system with two components in series produces a more interesting "failed" state. These components are independent, as one failing does not cause the other to fail, but both need to be functional for the system to function: $\mathbf{D} = \{\langle c_1 \rightsquigarrow \mathcal{S}\rangle, \langle c_2 \rightsquigarrow \mathcal{S}\rangle\}$. This system also has two states: the initial state (11) and the failed superstate $\boxed{01}\boxed{10}$.[4] Once the system has failed, we are no longer interested in its behavior; thus, for this system, we consider (00) unreachable.

9.1.3. Generalization of MIS Properties. Now that we have described the elements of P**rop**, we can describe how to generalize them. The goal of generalizing an element of P**rop** is to produce an element of P**rop** that relaxes the constraints of the first element but does not contradict it. Thus, generalizations are sound by definition; one can always soundly generalize a model. Understanding how constraints can be generalized allows us to order P**rop** by generalization.

9.1.3.1. One-step generalizations of dependencies. For a given reliability model, one way to generalize dependencies is to lower the constraint on a component's reliability: a more reliable component can always be substituted for a less reliable one. We can relax the reliability of a

[4] MIS modeling requires a single "failed" system (super)state; we leave unification of functional states into superstates for future work.

component, c, to a lower constraint $r < \mathsf{R}(c)$ by

$$\begin{aligned}&\mathsf{relax\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_,\_] : \mathbf{C} \to [0,1] \to \mathbf{Prop}\\&\mathsf{relax\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[c,r] \triangleq (\mathbf{C},\mathsf{R}',\mathbf{D})\end{aligned} \tag{9.1}$$

where

$$\mathsf{R}'(c') \triangleq \begin{cases} r & \text{if } c = c' \\ \mathsf{R}(c') & \text{otherwise.} \end{cases} \tag{9.1a}$$

The other means of generalizing system constraints is to generalize component dependencies. We begin by considering the smallest actions we can take that generalize system dependencies while maintaining the WF properties. There are two possible operations: merging two components and adding a new dependency $\langle \cdots \rightsquigarrow c \rangle$ among existing components. Both of these operations take one element of P**rop** and infer another.

Two distinct components c_1 and c_2 can be merged into a single component c_m (where the name c_m does not already appear in $\mathbf{C} \setminus \{c_1, c_2\}$) by replacing every instance of c_1 and c_2 with c_m:

$$\begin{aligned}&\mathsf{merge}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_,\_ \to \_] : \mathbf{C} \to \mathbf{C} \to \mathbf{Comps} \to \mathbf{Prop}\\&\mathsf{merge}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[c_1, c_2 \to c_m] \triangleq (\mathbf{C}',\mathsf{R}',\mathbf{D}')\end{aligned} \tag{9.2}$$

where

$$\mathbf{C}' \triangleq \{c_m\} \cup \mathbf{C} \setminus \{c_1, c_2\} \tag{9.2a}$$

$$\mathsf{R}'(c) \triangleq \begin{cases} \min(\mathsf{R}(c_1), \mathsf{R}(c_2)) & \text{if } c = c_m, \\ \mathsf{R}(c) & \text{otherwise.} \end{cases} \tag{9.2b}$$

$$\mathbf{D}' \triangleq \{\langle m(\mathbf{c}) \rightsquigarrow m(\mathbf{e}) \rangle \mid \langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle \in \mathbf{D}\} \tag{9.2c}$$

$$m(\mathbf{c}) \triangleq \begin{cases} \{c_m\} \cup \mathbf{c} \setminus \{c_1, c_2\} & \text{if } c_1 \in \mathbf{c} \vee c_2 \in \mathbf{c}, \\ \mathbf{c} & \text{otherwise.} \end{cases} \tag{9.2d}$$

When defining a generalization, we should ensure that it only relaxes constraints. Thus, when choosing the reliability bound $\mathsf{R}'(c_m)$ of the merged component, we must pick the least restrictive choice $\min(\mathsf{R}(c_1), \mathsf{R}(c_2))$. Effectively, this choice performs two generalizations: first, we relax the tighter of the reliability bounds of c_1 and c_2 by setting $\mathsf{R}(c_1) = \mathsf{R}(c_2)$, then we merge c_1 and c_2 into one component.

The dependencies that merge generates are the result of applying the following rules until a fixed point is reached (i.e., no more dependencies match the left-hand side):

$$\langle c_1 \cdots_1 \rightsquigarrow \cdots_2 \rangle \mapsto \langle c_m \cdots_1 \rightsquigarrow \cdots_2 \rangle$$
$$\langle c_2 \cdots_1 \rightsquigarrow \cdots_2 \rangle \mapsto \langle c_m \cdots_1 \rightsquigarrow \cdots_2 \rangle$$
$$\langle \cdots_1 \rightsquigarrow c_1 \cdots_2 \rangle \mapsto \langle \cdots_1 \rightsquigarrow c_m \cdots_2 \rangle$$
$$\langle \cdots_1 \rightsquigarrow c_2 \cdots_2 \rangle \mapsto \langle \cdots_1 \rightsquigarrow c_m \cdots_2 \rangle$$

The other possible generalization is adding a dependency among existing components. This may seem counterintuitive; however, it is a stronger claim to say that a component is independent of another—the fewer dependencies a system has, the more reliable it is. Adding a dependency from a nonempty set of components $\mathbf{c}$ to a component $e \notin \mathbf{c}$ means that whenever the components in $\mathbf{c}$ cause a failure, e is amongst the effects. As all the components in $\mathbf{c}$ and e are in $\mathbf{C}$ already, we need only modify the dependencies:

$$\begin{aligned} &\mathsf{add\_dep}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_ \rightsquigarrow \_] : \mathcal{P}(\mathbf{C}) \to \mathbf{C} \to \mathbf{Prop} \\ &\mathsf{add\_dep}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\mathbf{c} \rightsquigarrow e] \triangleq (\mathbf{C}, \mathsf{R}, \mathbf{D}') \end{aligned} \tag{9.3}$$

where

$$\mathbf{D}' \triangleq \{a(\langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle) \mid \langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle \in \mathbf{D}\} \tag{9.3a}$$
$$\cup \{\langle \mathbf{c} \rightsquigarrow \mathbf{u} \cup \{e\} \rangle\}$$

$$a(\langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle) \triangleq \begin{cases} \langle \mathbf{c}' \setminus \{e\} \rightsquigarrow \mathbf{e}' \cup \{e\} \rangle & \text{if } \mathbf{c} \subseteq \mathbf{c}', \\ \langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle & \text{otherwise.} \end{cases} \tag{9.3b}$$

$$\mathbf{u} \triangleq \bigcup \{\mathbf{e}' \mid \langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle \in \mathbf{D} \text{ where } \mathbf{c}' \subseteq \mathbf{c}\} \tag{9.3c}$$

Again, we can view the dependencies in $\mathbf{D}'$ after adding the dependency $\langle \cdots_1 \rightsquigarrow e \rangle$ as the result of rewriting matching dependencies in $\mathbf{D}$:

$$\langle e \cdots_1 \cdots_2 \rightsquigarrow \cdots_3 \rangle \mapsto \langle \cdots_1 \cdots_2 \rightsquigarrow e \cdots_3 \rangle$$
$$\langle \cdots_1 \cdots_2 \rightsquigarrow \cdots_3 \rangle \mapsto \langle \cdots_1 \cdots_2 \rightsquigarrow e \cdots_3 \rangle$$

and adding the dependency $\langle \cdots_1 \rightsquigarrow e \cdots_2 \rangle$ where the components $\cdots_2$ are the effects of failures of components in $\cdots_1$ as required by Monotonicity.

For an example of the effect of generalization operations on a system, consider a system with three independent components:

$$\begin{aligned} p = (\mathbf{C} = \{c_1, c_2, c_3\}, \mathsf{R}(\_) = 0.9, \mathbf{D} = \{ \\ \langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle, \\ \langle c_1, c_2, c_3 \rightsquigarrow \mathcal{S} \rangle \\ \}) \end{aligned}$$

Introducing a dependency $\langle c_1, c_2 \rightsquigarrow c_3 \rangle$ results in the following system:

$$\begin{aligned} p' &= \mathsf{add\_dep}_p[c_1, c_2 \rightsquigarrow c_3] \\ &= (\mathbf{C}' = \{c_1, c_2, c_3\}, \mathsf{R}'(\_) = 0.9, \mathbf{D}' = \{ \\ &\quad \langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle, \\ &\quad \left. \begin{array}{l} \langle c_1, c_2 \rightsquigarrow c_3 \rangle^{\dagger}, \\ \langle c_1, c_2 \rightsquigarrow c_3, \mathcal{S} \rangle^{\ddagger} \end{array} \right\} \equiv \langle c_1, c_2 \rightsquigarrow c_3, \mathcal{S} \rangle \\ &\quad \}) \end{aligned}$$

Of note: the dependency marked † is the new dependency added by $\mathsf{add\_dep}$ and the dependency marked ‡ is the result of the first substitution rule in (9.3b). Both rules reduce to one via the Union property.

Continuing the example, merging c_2 and c_3 into c_4 gives us the system

$$\begin{aligned}
p'' &= \mathsf{merge}_{p'}[c_2, c_3 \to c_4] \\
&= (\mathbf{C}'' = \{c_1, c_4\}, \mathsf{R}''(_) = 0.9, \mathbf{D}'' = \{ \\
&\quad \langle c_1 \rightsquigarrow \emptyset \rangle, \\
&\quad \langle c_4 \rightsquigarrow \emptyset \rangle, \\
&\quad \langle c_1, c_4 \rightsquigarrow c_4, \mathcal{S} \rangle \equiv \langle c_1, c_4 \rightsquigarrow \mathcal{S} \rangle \\
&\}\,)
\end{aligned}$$

where both components are independent and the failure of both leads to system failure.

9.1.3.2. Multi-step generalization of dependencies. The example of the previous section illustrates the process by which successive generalization steps are applied to system properties. To describe this more formally, let $\mathbf{G}$ be the set of all generalization operations and $\mathbf{G}^*$ be the set of finite sequences of elements of $\mathbf{G}$. We define the act of applying a sequence of generalizations to an element of properties, $[\![\_]\!](\_) : \mathbf{G}^* \to \mathbf{Prop} \to \mathbf{Prop}$, by

$$[\![g]\!](p) \triangleq \begin{cases} p & \text{if } g = () \\ [\![gs]\!](g'_p) & \text{if } g = (g', gs). \end{cases} \tag{9.4}$$

9.1.3.3. Generalization as a partial order. With the ability to apply a sequence of generalizations, we now turn to the task of ordering elements of $\mathbf{Prop}$. First, we prove some monotonicity properties of any $q \in \mathbf{Prop}$ generalized from some $p \in \mathbf{Prop}$.

Theorem 9.1. *For all $p = (\mathbf{C}, \mathsf{R}, \mathbf{D}) \in \mathbf{Prop}$ and for all $g \in \mathbf{G}$ where $(\mathbf{C}', \mathsf{R}', \mathbf{D}') = [\![g]\!](p)$,*

(i) $|\mathbf{C}'| \leq |\mathbf{C}|$,

(ii) $\forall c \in \mathbf{C} \cap \mathbf{C}', \mathsf{R}'(c) \leq \mathsf{R}(c)$, *and*

(iii) if $\mathbf{C} = \mathbf{C}'$, $\forall \langle \mathbf{c} \rightsquigarrow \mathbf{e}' \rangle \in \mathbf{D}'$, *if* $\langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle \in \mathbf{D}$, $\mathbf{e} \subseteq \mathbf{e}'$.

Proof. Proceed by case analysis on g.

Case $g = \mathsf{relax\_rel}[c, r]$:

(i) $\mathbf{C}' = \mathbf{C}$

(ii) For $c' \in \mathbf{C}$, $\begin{cases} \mathsf{R}'(c') < \mathsf{R}(c') & \text{if } c = c' \\ \mathsf{R}'(c') = \mathsf{R}(c') & \text{otherwise} \end{cases}$

(iii) $\mathbf{D}' = \mathbf{D}$

Case $g = \mathsf{merge}[c_1, c_2 \to c_m]$:

(i) $|\mathbf{C}'| = |\mathbf{C}| - 1 \leq |\mathbf{C}|$

(ii) For $c \in \mathbf{C}$, $\begin{cases} \mathsf{R}'(c) \leq \mathsf{R}(c) & \text{if } c = c_m \\ \mathsf{R}'(c) = \mathsf{R}(c) & \text{otherwise} \end{cases}$

(iii) $\mathbf{C} \neq \mathbf{C}'$

Case $g = \mathsf{add\_dep}[\mathbf{c} \rightsquigarrow e]$:

(i) $\mathbf{C}' = \mathbf{C}$

(ii) $\mathsf{R}' = \mathsf{R}$

(iii) Consider $\langle \mathbf{c}'' \rightsquigarrow \mathbf{e}'' \rangle \in \mathbf{D}'$.

If $\mathbf{c} = \mathbf{c}''$, then (by Union)

$$\langle \mathbf{c}'' \rightsquigarrow \mathbf{e}'' \rangle \equiv \left\{ \begin{array}{c} a(\langle \mathbf{c}'' \rightsquigarrow \mathbf{e_1} \rangle) \\ a(\langle \mathbf{c}'' \cup \{e\} \rightsquigarrow \mathbf{e_2} \rangle) \\ \langle \mathbf{c}'' \rightsquigarrow \mathbf{u} \cup \{e\} \rangle \end{array} \right\}$$

so $\mathbf{e}'' = \mathbf{e_1} \cup \mathbf{e_2} \cup \mathbf{u} \cup \{e\}$, where $\mathbf{u}$ is defined as in Equation 9.3c. If $\langle \mathbf{c}'' \rightsquigarrow \mathbf{e_1} \rangle \in \mathbf{D}$, then $\mathbf{e_1} \subseteq \mathbf{e}''$.

Otherwise,

$$\langle \mathbf{c}'' \rightsquigarrow \mathbf{e}'' \rangle \equiv \left\{ \begin{array}{c} a(\langle \mathbf{c}'' \rightsquigarrow \mathbf{e_1} \rangle) \\ a(\langle \mathbf{c}'' \cup \{e\} \rightsquigarrow \mathbf{e_2} \rangle) \end{array} \right\}$$

and $\mathbf{e_1} \subseteq \mathbf{e}'' = \mathbf{e_1} \cup \mathbf{e_2}$ if $\langle \mathbf{c}'' \rightsquigarrow \mathbf{e_1} \rangle \in \mathbf{D}$. □

To form a partial order on P**rop** using these generalization operations, we say that if p_g generalizes p_r, there exists some sequence of generalizations that witnesses that fact:

Definition 9.1. $p_g \in$ P**rop** *generalizes* $p_r \in$ P**rop**, written $p_r \sqsubseteq p_g$, if $\exists g \in \mathbf{G}^*, [\![g]\!](p_r) = p_g$.

Theorem 9.2. $\sqsubseteq$ *forms a partial order on* P**rop**.

Proof. *Reflexivity*: $\forall p \in \mathbf{Prop}, [\![()]\!](p) = p \implies \forall p \in \mathbf{Prop}, p \sqsubseteq p$.

Antisymmetry: Take $p, q \in \mathbf{Prop}$ such that $p \sqsubseteq q$ and $q \sqsubseteq p$. Then there exist $g_p, g_q \in \mathbf{G}^*$ such that $[\![g_p]\!](p) = q$ and $[\![g_q]\!](q) = p$. Proceed by case analysis on g_p.

If $g_p = ()$, then $q = [\![g_p]\!](p) = [\![()]\!](p) = p$.

Otherwise $g_p = (g, gs)$; we desire to show, using Theorem 9.1, that g cannot be "undone" by any generalization and thus $q \not\sqsubseteq p$. Let $(\mathbf{C}_p, \mathsf{R}_p, \mathbf{D}_p) = p$, $(\mathbf{C}', \mathsf{R}', \mathbf{D}') = [\![g]\!](p)$, and $(\mathbf{C}_q, \mathsf{R}_q, \mathbf{D}_q) = q$.

If $g = \mathsf{merge}_p[c_1, c_2 \to c_m]$, then $|\mathbf{C}_q| < |\mathbf{C}_p|$.

If $g = \mathsf{relax\_rel}_p[c, r]$, then $\mathsf{R}_q(c) \leq \mathsf{R}'(c) < \mathsf{R}_p(c)$ or $c \notin \mathbf{C}_q$.

If $g = \mathsf{add\_dep}_p[\mathbf{c} \rightsquigarrow e]$, then either $\langle \mathbf{c} \rightsquigarrow \mathbf{e}_p \rangle \notin \mathbf{D}_p$ or $\mathbf{e}_p \subset \mathbf{e}_q$.

Thus, by Theorem 9.1, $q \not\sqsubseteq p$.

Transitivity: Take $p, q, r \in \mathbf{Prop}$ such that $p \sqsubseteq q$ and $q \sqsubseteq r$. Then there exist $g_p, g_q \in \mathbf{G}^*$ such that $[\![g_p]\!](p) = q$ and $[\![g_q]\!](q) = r$. The composition of these generalizations is equal to the concatenation of their sequences: $[\![g_q]\!]([\![g_p]\!](p)) = [\![g_p; g_q]\!](p)$. Furthermore, the concatenation of two finite sequences is a finite sequence, so $g_p; g_q \in \mathbf{G}^*$. As $[\![g_p; g_q]\!](p) = r$, $p \sqsubseteq r$. □

9.1.4. Refinement of MIS Properties. In addition to generalization of constraints, we are interested in refining them: adding new constraints or increasing the strictness of existing ones. Refinements are dual to generalizations, so for each generalization we expect a corresponding refinement. Since refinements introduce additional information, showing their soundness requires some external justification. While we cannot automatically justify an arbitrary refinement, we can propagate a justified refinement in one model to other models of the same system; Section 10 details this process.

9.1.4.1. One-step refinements. Corresponding to $\mathsf{relax\_rel}$ we have $\mathsf{tighten\_rel}$ which raises the bound on the reliability of component c to a higher constraint $r > \mathsf{R}(c)$:

$$\begin{aligned} &\mathsf{tighten\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_, \_] : \mathbf{C} \to [0,1] \to \mathbf{Prop} \\ &\mathsf{tighten\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[c, r] \triangleq (\mathbf{C}, \mathsf{R}', \mathbf{D}) \end{aligned} \tag{9.5}$$

where

$$\mathsf{R}'(c') \triangleq \begin{cases} r & \text{if } c = c' \\ \mathsf{R}(c') & \text{otherwise} \end{cases} \tag{9.5a}$$

To undo a merge, we split one component, c_m, into two, c_1 and c_2 (where $c_1, c_2 \notin \mathbf{C} \setminus \{c\}$). When splitting two components, we make each fully dependent on the other, as that is the most general set of constraints we can generate. In other words, the result of $\mathsf{split}_p[c_m \to c_1, c_2]$ is the maximal element of the set $\{q \in \mathbf{Prop} \mid p = \mathsf{merge}_q[c_1, c_2 \to c_m]\}$.

$$\mathsf{split}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_ \to \_, \_] : \mathbf{C} \to \mathbf{Comps} \to \mathbf{Comps} \to \mathbf{Prop}$$

$$\mathsf{split}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[c_m \to c_1, c_2] \triangleq (\mathbf{C}', \mathsf{R}', \mathbf{D}') \tag{9.6}$$

where

$$\mathbf{C}' \triangleq \{c_1, c_2\} \cup \mathbf{C} \setminus \{c_m\} \tag{9.6a}$$

$$\mathsf{R}'(c) \triangleq \begin{cases} \mathsf{R}(c_m) & \text{if } c = c_1 \vee c = c_2, \\ \mathsf{R}(c) & \text{otherwise.} \end{cases} \tag{9.6b}$$

$$\mathbf{D}' \triangleq \bigcup \{s(\langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle) \mid \langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle \in \mathbf{D}\} \tag{9.6c}$$

$$s(\langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle) \triangleq \begin{cases} \left\{ \begin{array}{l} \langle \{c_1, c_2\} \cup \mathbf{c}' \rightsquigarrow \mathbf{e} \rangle \\ \langle \{c_1\} \cup \mathbf{c}' \rightsquigarrow \mathbf{e} \cup \{c_2\} \rangle \\ \langle \{c_2\} \cup \mathbf{c}' \rightsquigarrow \mathbf{e} \cup \{c_1\} \rangle \end{array} \right\} & \text{if } c_m \in \mathbf{c} \\ \left\{ \begin{array}{l} \langle \mathbf{c} \rightsquigarrow \mathbf{e}' \cup \{c_1, c_2\} \rangle \\ \langle \mathbf{c} \rightsquigarrow \mathbf{e}' \cup \{c_1\} \rangle \\ \langle \mathbf{c} \rightsquigarrow \mathbf{e}' \cup \{c_2\} \rangle \end{array} \right\} & \text{if } c_m \in \mathbf{e} \\ \{\langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle\} & \text{otherwise.} \end{cases} \tag{9.6d}$$

$$\mathbf{c}' \triangleq \mathbf{c} \setminus \{c_m\} \tag{9.6e}$$

$$\mathbf{e}' \triangleq \mathbf{e} \setminus \{c_m\} \tag{9.6f}$$

The resulting dependencies can be understood as the result of applying the following rewrite rules to dependencies in $\mathbf{D}$:

$$\langle \cdots_1 c_m \rightsquigarrow \cdots_2 \rangle \mapsto \begin{cases} \langle \cdots_1 c_1 c_2 \rightsquigarrow \cdots_2 \rangle \\ \langle \cdots_1 c_1 \rightsquigarrow \cdots_2 c_2 \rangle \\ \langle \cdots_1 c_2 \rightsquigarrow \cdots_2 c_1 \rangle \end{cases}$$

$$\langle \cdots_1 \rightsquigarrow \cdots_2 c_m \rangle \mapsto \begin{cases} \langle \cdots_1 \rightsquigarrow \cdots_2 c_1 c_2 \rangle \\ \langle \cdots_1 \rightsquigarrow \cdots_2 c_1 \rangle \\ \langle \cdots_1 \rightsquigarrow \cdots_2 c_2 \rangle \end{cases}$$

Finally, remove_dep corresponds to undoing an add_dep operation. Adding a dependency $\langle \cdots_1 \rightsquigarrow e \rangle$ states that e depends on all of $\cdots_1$ and therefore every dependency containing $\cdots_1$ is rewritten to preserve Monotonicity. Removing a dependency $\langle \cdots_1 \rightsquigarrow e \rangle$ states that e is *independent* of all components in $\cdots_1$, so every dependency whose causes are contained in $\cdots_1$ is rewritten.

$$\begin{aligned} &\mathsf{remove\_dep}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_ \rightsquigarrow \_] : \mathcal{P}(\mathbf{C}) \to \mathbf{C} \to \mathbf{Prop} \\ &\mathsf{remove\_dep}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\mathbf{c} \rightsquigarrow e] \triangleq (\mathbf{C}, \mathsf{R}, \mathbf{D}') \end{aligned} \tag{9.7}$$

where

$$\mathbf{D}' \triangleq \{r(\langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle) \mid \langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle \in \mathbf{D}\} \tag{9.7a}$$

$$r(\langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle) \triangleq \begin{cases} \langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \setminus \{e\} \rangle & \text{if } \mathbf{c}' \subseteq \mathbf{c}, \\ \langle \mathbf{c}' \rightsquigarrow \mathbf{e}' \rangle & \text{otherwise.} \end{cases} \tag{9.7b}$$

The dependencies resulting from removing the dependency $\langle \cdots_1 \rightsquigarrow e \rangle$ follow from the rewrite rule:

$$\langle \cdots_2 \rightsquigarrow \cdots_3 e \rangle \mapsto \langle \cdots_2 \rightsquigarrow \cdots_3 \rangle \text{ if } \cdots_2 \subseteq \cdots_1$$

9.1.4.2. Multi-step refinements. As with generalizations, let $\mathbf{R}$ be the set of all refinement operations and $\mathbf{R}^*$ be the set of all sequences of refinements. We abuse notation slightly to define application of a sequence of refinements using the same notation: for $rs \in \mathbf{R}^*$, $[\![rs]\!](p)$ is the result of applying that sequence of refinements to some system properties p.

9.1.4.3. Refinement as the dual of generalization. Each generalization operation and its corresponding refinement are not necessarily inverses, as most generalization operations map several elements of P**rop** to the same more general system (i.e., they are not injective). Thus, we do not have that $\forall g \in \mathbf{G}$, if $q = [\![g]\!](p)$ then $\exists r \in \mathbf{R}, p = [\![r]\!](q)$. However, we can show the opposite: if $q = [\![r]\!](p)$, then p *covers* q: there is no r such that $q \sqsubset r \sqsubset p$.

Furthermore, the refinement operations form a dual order to the order defined by generalization:

Theorem 9.3. *$\forall p_r, p_g \in$ P**rop**, $p_r \sqsubseteq p_g$ if and only if $\exists rs \in \mathbf{R}^*, p_r = [\![rs]\!](p_g)$.*

As such, p_r *refines* p_g if $p_g \sqsupseteq p_r$, or, equivalently, $p_r \sqsubseteq p_g$.

9.1.5. Connecting MIS Models With Their Properties. In this section we make explicit the Galois connection between the properties of MIS models and the models themselves.

9.1.5.1. The properties lattice. To be able to use a Galois connection to relate our notions of generalization and refinement to MIS models, we must define P**rop** as a lattice. As such, we need to define top and bottom elements of P**rop**, least upper bounds (or *joins*), and greatest lower bounds (*meets*).

The top element of P**rop** is the one-element system with unconstrained component reliability:

$$\top \triangleq (\{c\}, \mathsf{R}(c) = 0, \{\langle c \rightsquigarrow \mathcal{S} \rangle\}). \tag{9.8}$$

Any other one-element system constrains component reliability and thus can be generalized to $\top$ by relax_rel. Removing the one dependency results in a system that does not meet the WF properties, and no further dependencies can be added without adding another component. Finally, given $p \in$ P**rop**, we can show $p \sqsubseteq \top$ by repeatedly merging components in p until the result has one component, then relaxing that component's reliability bound, if necessary.

The bottom element of P**rop** is a special element which corresponds to an "overdetermined" system—one where the constraints are contradictory. We do not concern ourselves with its representation, but simply define it as the element $\bot \in$ P**rop** such that $\forall p, \bot \sqsubseteq p$.

The join of two elements $(\mathbf{C_1}, \mathsf{R}_1, \mathbf{D_1})$ and $(\mathbf{C_2}, \mathsf{R}_2, \mathbf{D_2})$ is equal, up to renaming of components, to $(\mathbf{C_1} \cap \mathbf{C_2}, \min\{\mathsf{R}_1, \mathsf{R}_2\}, \mathbf{D_1} \cup \mathbf{D_2})$. Joins can be computed by repeatedly applying merge to combine components, then applying add_dep to add dependencies as needed, then applying relax_rel

to reduce reliabilities if necessary. The meet, likewise, is equal up to renaming of components to $(\mathbf{C_1} \cup \mathbf{C_2}, \max\{\mathsf{R}_1, \mathsf{R}_2\}, \mathbf{D_1} \cap \mathbf{D_2})$. It can be shown that the meet can be generalized to either element by appropriate application of merge, add_dep, and relax_rel.

9.1.5.2. MIS models. Markov Imbeddable Structure models are one approach to deriving a system's reliability from the reliability of its components. These models consist of states and transitions between states caused by the failure of components. The reliability of the system is determined by computing the probability of the system not reaching the "failed" state after considering the effect of each component.

This paper considers MIS models where the states are defined by the components functional in that state; e.g., (1101) corresponds to the state of a 4-component system where components 1, 2, and 4 are functional and component 3 has failed. Components cannot repair themselves, so every transition is either from one state to that same state or from one state to a state with more failed components. The *failed* state is absorbing—once the system fails, we are no longer interested in its behavior.

These transitions are usually represented in the form of transition probability matrices (TPMs) T_i, one for each component. As the system always starts in the fully functional state, the initial state probability vector is $\Pi_0 \triangleq [1, 0, \dots]$. Another vector $u \triangleq [1, \dots, 0]$ defines which states are considered functional. The system reliability is given by the product of the initial state probabilities, the TPMs, and the u vector:

$$\mathsf{R}(\mathcal{S}) \triangleq \Pi_0^T * T_1 * T_2 * \cdots * T_n * u \tag{9.9}$$

As an example, consider the system with two components in series where $\mathsf{R}(c_1) = \mathsf{R}(c_2) = p = 1 - q$. The TPM for both components is given by

$$T_1 = T_2 = \begin{pmatrix} p & q \\ 0 & 1 \end{pmatrix}$$

and the resulting system reliability is

$$\mathsf{R}(\mathcal{S}) = \Pi_0^T * T_1 * T_2 * u = p^2$$

The Markov chain this system follows is illustrated in Fig 9.1 where the transitions are labeled by the component taking them and the probability of being taken.

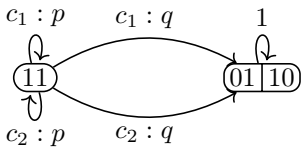


Figure 9.1. Two-Component series system Markov chain

9.1.5.3. Abstraction and concretization. To apply our formalization of refinement and generalization to MIS models, we need to connect our properties domain P**rop** to MIS models. We achieve this by an *abstraction* operator which converts system constraints to MIS models and a *concretization* operator which derives constraints from MIS models.

To abstract an MIS model from $(\mathbf{C}, \mathsf{R}, \mathbf{D}) \in \mathbf{Prop}$, for each $c_i \in \mathbf{C}$ let $p_i = 1 - q_i = \mathsf{R}(c_i)$ be its reliability and let T_i be its TPM. Let $n = |\mathbf{C}|$ be the number of components in the system. Then, begin with the initial fully-functional state $\overline{(1 \cdots 1)}$. For each dependency $\langle c_i \rightsquigarrow \mathbf{e} \rangle \in \mathbf{D}$, insert a transition from $\overline{(1 \cdots 1)}$ to $\overline{(1 \cdots 1)}$ with probability p_i in T_i and a transition from $\overline{(1 \cdots 1)}$ to the state where all components except c_i and those in $\mathbf{e}$ are functional with probability q_i in T_i. If $\mathcal{S} \in \mathbf{e}$, then mark that state as "failed". For each non-"failed" state added in the previous step, let $\mathbf{s}$ be the components functional in that state and let $\mathbf{f} = \mathbf{C} \setminus \mathbf{s}$ be the set of failed components. For each component $c_i \in \mathbf{s}$, select the dependency $\langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle \in \mathbf{D}$ where $c_i \in \mathbf{c}$ and $\mathbf{c}$ is the largest set such that $\mathbf{c} \subset \mathbf{f}$. Insert transitions from $\overline{(\mathbf{s})}$ to $\overline{(\mathbf{s})}$ with probability p_i and from $\overline{(\mathbf{s})}$ to $\overline{(\mathbf{s} \setminus \mathbf{e})}$ with probability q_i into T_i. For each component $c_i \in \mathbf{f}$, insert a transition from $\overline{(\mathbf{s})}$ to $\overline{(\mathbf{s})}$ with probability 1 into T_i. Repeat this step until there are no more non-failed states to consider.

Concretizing properties from an MIS model proceeds in an analogous fashion. For each T_i create a component c_i and set $\mathsf{R}(c_i) = p_i$. For each c_i, first let $\mathbf{s}'$ be the set of components functional after c_i fails from the initial $\overline{(1 \cdots 1)}$ state and add a dependency $\langle c_i \rightsquigarrow \mathbf{C} \setminus \mathbf{s}' \rangle$ to $\mathbf{D}$. Then consider all transitions in T_i from state $\overline{(\mathbf{s})}$ to state $\overline{(\mathbf{s}')}$ where $\mathbf{s}' \subset \mathbf{s}$. Let $\mathbf{f} \triangleq \mathbf{s} \setminus \mathbf{s}' \setminus \{c_i\}$ be the set of components that also fail as a result of the failure of c_i. Take $\langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle \in \mathbf{D}$ where $c_i \in \mathbf{c}$ and $\mathbf{c}$ is the largest set such that $\mathbf{c} \subset (\mathbf{C} \setminus \mathbf{s})$. If $\mathbf{e} \neq \mathbf{f}$, add a dependency $\langle \mathbf{C} \setminus \mathbf{s} \setminus \{c_i\} \rightsquigarrow \mathbf{f} \rangle$.

9.1.5.4. Example. As an example of the power of this approach, let us refine a 2-of-3 system from $\top$. Our starting system is

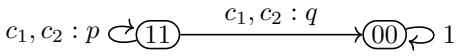
$$\top = (\{c_1\}, \mathsf{R}(c_1) = 0, \{\langle c_1 \rightsquigarrow \mathcal{S} \rangle\}).$$

If we refine c_1's reliability to p by $s_1 = \mathsf{tighten\_rel}_{\top}[c_1, p]$, the resulting system has reliability $\mathsf{R}(\mathcal{S}) = p$.

First, we create another component via $s_2 = \mathsf{split}_{s_1}[c_1 \to c_1, c_2]$, we get the following system:

$$\begin{aligned} s_2 = (\{c_1, c_2\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = p, \{ \\ \langle c_1 \rightsquigarrow c_2, \mathcal{S}\rangle, \langle c_2 \rightsquigarrow c_1, \mathcal{S}\rangle \\ \langle c_1, c_2 \rightsquigarrow \mathcal{S}\rangle \\ \}) \end{aligned}$$

which abstracts to the Markov chain:

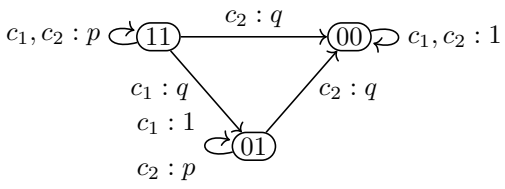


This gives $\mathsf{R}(\mathcal{S}) = p^2$ as we now take two steps through the Markov chain.

We can avoid adding excessive dependencies later by removing two, making c_1 independent: $s_3 = \mathsf{remove\_dep}_{s_2}[c_1 \rightsquigarrow c_2, \mathcal{S}]$.[5]

$$\begin{aligned} s_3 = (\{c_1, c_2\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = p, \{ \\ \langle c_1 \rightsquigarrow \emptyset\rangle, \langle c_2 \rightsquigarrow c_1, \mathcal{S}\rangle \\ \langle c_1, c_2 \rightsquigarrow \mathcal{S}\rangle \\ \}) \end{aligned}$$

Removing these dependencies adds a new state to the Markov chain:



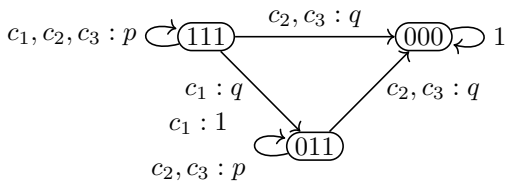
This gives $\mathsf{R}(\mathcal{S}) = p^2 + pq$—either both components remain functional, or c_1 fails and c_2 remains functional.

[5]This is equivalent to performing two $\mathsf{remove\_dep}$ operations, one for the dependency on c_2 and one for $\mathcal{S}$.

Next, we introduce c_3 by $s_4 = \mathsf{split}_{s_3}[c_2 \to c_2, c_3]$.

$$
\begin{aligned}
s_4 = (\{c_1, c_2, c_3\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = p, \{ \\
&\langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow c_1, c_3, \mathcal{S} \rangle, \langle c_3 \rightsquigarrow c_1, c_2, \mathcal{S} \rangle \\
&\langle c_1, c_2 \rightsquigarrow \mathcal{S} \rangle, \langle c_1, c_3 \rightsquigarrow c_2, \mathcal{S} \rangle, \langle c_2, c_3 \rightsquigarrow c_1, \mathcal{S} \rangle \\
\})
\end{aligned}
$$

The Markov chain is similar to the one abstracted from s_3, but c_3 adds its own transition probabilities.

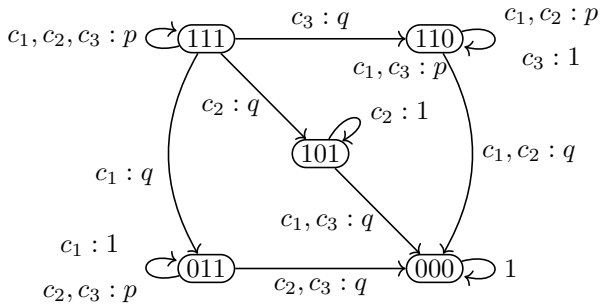


This gives $\mathsf{R}(\mathcal{S}) = p^3 + p^2 q$—either all components remain functional, or c_1 fails and c_2 and c_3 remain functional.

Finally, we arrive at the desired 2-of-3 system by removing unneeded dependencies: $s_5 = \mathsf{remove\_dep}_{s_4}[c_2 \rightsquigarrow c_1, c_3, \mathcal{S}]$ and $s_6 = \mathsf{remove\_dep}_{s_5}[c_3 \rightsquigarrow c_1, c_2, \mathcal{S}]$.

$$
\begin{aligned}
s_6 = (\{c_1, c_2, c_3\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = p, \{ \\
&\langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle \\
&\langle c_1, c_2 \rightsquigarrow \mathcal{S} \rangle, \langle c_1, c_3 \rightsquigarrow c_2, \mathcal{S} \rangle, \langle c_2, c_3 \rightsquigarrow c_1, \mathcal{S} \rangle \\
\})
\end{aligned}
$$

The abstracted Markov chain has two new states:

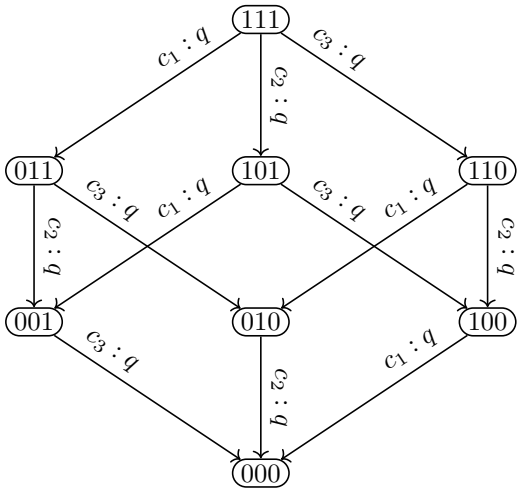


This gives $\mathsf{R}(\mathcal{S}) = p^3 + 3p^2 q$—either all components remain functional, or only one fails.

9.1.6. Superstates and Non-deterministic Choice. One problem MIS models face is state space explosion: in a naïve model where every component is independent of the others, adding a new component doubles the number of states in the model. A solution to this problem,

which does not require numerical approximation methods or otherwise reduce the accuracy of the computed result, is to allow one state to represent more than one configuration of "up" components. In the literature, these states are referred to as *superstates*; we have already seen examples of these in various failed states, such as the superstate $\boxed{01}\boxed{10}$ in Figure 9.1. Thus far, we have modeled the "system failed" superstate in an ad-hoc fashion; we now turn to the issue of modeling arbitrary superstates in P**rop**.

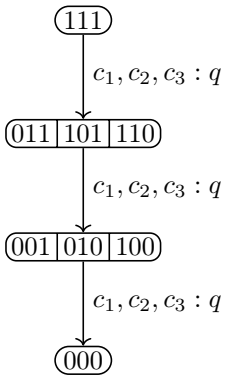
To motivate the following developments, we introduce an example of where superstates become useful. Suppose we have a system with three independent components in parallel. The system is functional as long as any single component remains functional. This reliability structure can be represented by the following Markov chain (for readability, we elide the self-loops for each state):



The corresponding element of P**rop** is:

$$
\begin{aligned}
s_{\text{state}} = (\{c_1, c_2, c_3\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = p, \{ \\
&\langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle \\
&\langle c_1, c_2 \rightsquigarrow \emptyset \rangle, \langle c_1, c_3 \rightsquigarrow \emptyset \rangle, \langle c_2, c_3 \rightsquigarrow \emptyset \rangle \\
&\langle c_1, c_2, c_3 \rightsquigarrow \mathcal{S} \rangle \\
\})
\end{aligned}
$$

This model has $\mathsf{R}(\mathcal{S}) = p^3 + 3p^2q + 3pq^2$. This is the worst-case situation of 3 components leading to 2^3 states. However, the identity of any failed component is irrelevant to the computation; knowledge of the number of failed components suffices. We can reduce the state space of this model by creating superstates $\boxed{011}\boxed{101}\boxed{110}$ and $\boxed{001}\boxed{010}\boxed{110}$, representing one and two failed components respectively:



In this model, we still have $\mathsf{R}(\mathcal{S}) = p^3 + 3p^2q + 3pq^2$, but significantly fewer states and correspondingly smaller matrices. It is also worth noting that using superstates does not require that all components have the same reliability. Components are still considered individually when computing reliability; however, when defining transitions between superstates, we "forget" the specific state of a superstate which characterizes the system. If the system is in a superstate, we merely know that it is in one of the states of that superstate.

This notion of "forgetting" maps cleanly onto the concept of *non-deterministic choice*, denoted here with the $\oplus$ operator. Given two sets $\mathbf{C_1}, \mathbf{C_2}$, the value of $\mathbf{C_1} \oplus \mathbf{C_2}$ is one of the two specified sets, but it is unknown which set is chosen. By extending the notation of **Deps** to allow non-deterministic set choices to appear in the cause of a dependency, we can represent the Markov chain:

$$
\begin{aligned}
s_{\text{superstate}} = (\{c_1, c_2, c_3\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = p, \{ \\
&\langle c_1 \oplus c_2 \oplus c_3 \rightsquigarrow \emptyset \rangle, \\
&\langle c_1, c_2 \oplus c_1, c_3 \oplus c_2, c_3 \rightsquigarrow \emptyset \rangle, \\
&\langle c_1, c_2, c_3 \rightsquigarrow \mathcal{S} \rangle \\
\})
\end{aligned}
$$

In the dependency $\langle c_1 \oplus c_2 \oplus c_3 \rightsquigarrow \emptyset \rangle$, we know only that one of c_1, c_2, or c_3 have failed, not which one. This allows us to capture the notion of "one component" having failed. With this example in hand, we can proceed to formally define and explore non-deterministic choice in P**rop**.

9.1.6.1. Non-deterministic choice of causes and effects. Non-deterministic choice plays a key role in abstraction and refinement for both software and systems [93]. In the process of deriving programs from specifications, it encodes the notion that one may pick arbitrarily among the programs which meet a particular specification; any aspect left unspecified by the specification is thus ambiguous. For example, given the specification "$f(x) = y$ such that $y * y = x$", we may implement $f(x)$ by either `sqrt(x)` or `-sqrt(x)`. Using the notation of non-deterministic choice, this specification can be represented by $f(x) =$ `sqrt(x)` $\oplus$ `-sqrt(x)`. When proving properties of $f(x)$, we use the *demonic choice principle*, which states that any statements true of a non-deterministic choice must hold regardless of which case of the choice is taken. Thus, we cannot argue that $f(x) \geq 0$, even though that property holds for one of the cases of the choice. Effectively, the demonic choice principle requires us to consider every case.

In this work, we will use non-deterministic choices to distinguish between sets, rather than specifications:

Definition 9.2. Given a set of sets $\mathcal{C} = \{\mathbf{C_1}, \mathbf{C_2}, \cdots\}$, the set $\bigoplus \mathcal{C}$ is one of the sets $\mathbf{C_1}$, $\mathbf{C_2}$, $\cdots$. The specific set is chosen arbitrarily. Denote $\bigoplus\{\mathbf{C_1}, \mathbf{C_2}\}$ by $\mathbf{C_1} \oplus \mathbf{C_2}$. $\bigoplus\{\mathbf{C_1}\} = \mathbf{C_1}$. $\bigoplus \emptyset$ has no value.

Note that non-deterministic choices can be "flattened"; that is,

$$\bigoplus \left\{ \mathbf{C_1}, \bigoplus \mathcal{C} \right\} = \bigoplus \{\mathbf{C_1}\} \cup \mathcal{C}.$$

It follows that $\oplus$ is commutative and associative by commutativity and associativity of $\cup$. Furthermore, $\mathbf{C} \oplus \mathbf{C} = \mathbf{C}$.

For convenience, we define a function to apply a set-valued function to each case of a non-deterministic choice:

$$\mathsf{map}_{\oplus}(f) \left(\bigoplus \mathcal{C} \right) = \bigoplus \{ f(\mathbf{C}) \mid \mathbf{C} \in \mathcal{C} \} \tag{9.10}$$

When introducing non-deterministic choice into the causes and effects of dependencies, we must be careful to observe the demonic choice principle. Namely, once we generalize two dependencies $\langle c_1 \rightsquigarrow \cdots \rangle$ and $\langle c_2 \rightsquigarrow \cdots \rangle$ into $\langle c_1 \oplus c_2 \rightsquigarrow \cdots \rangle$, erasing the distinction between the failure of c_1 and

c_2, we must ensure that all other dependencies involving c_1 and c_2 also "forget" which of the two has failed. This may entail the definition of additional superstates. An analogous situation occurs when generalizing $\langle \cdots \rightsquigarrow c_1 \rangle$ and $\langle \cdots \rightsquigarrow c_2 \rangle$ into $\langle \cdots \rightsquigarrow c_1 \oplus c_2 \rangle$.

As an example, consider what happens in the previous example if we create the superstate $\boxed{011|101|110}$ but leave the states $\boxed{001}$, $\boxed{010}$, and $\boxed{100}$ as-is. This would correspond to the following invalid properties:

$$\begin{aligned} s_{\text{superstate}} = (\{c_1, c_2, c_3\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = p, \{ \\ \langle c_1 \oplus c_2 \oplus c_3 \rightsquigarrow \emptyset \rangle, \\ \langle c_1, c_2 \rightsquigarrow \emptyset \rangle, \langle c_1, c_3 \rightsquigarrow \emptyset \rangle, \langle c_2, c_3 \rightsquigarrow \emptyset \rangle \\ \langle c_1, c_2, c_3 \rightsquigarrow \mathcal{S} \rangle \\ \}). \end{aligned}$$

We attempt to define three transitions out of $\boxed{011|101|110}$, one for each component. In the case that we consider the failure of c_1, it is unclear whether the dependency $\langle c_1, c_2 \rightsquigarrow \emptyset \rangle$ or $\langle c_1, c_3 \rightsquigarrow \emptyset \rangle$ applies, since we cannot determine whether it is c_2 or c_3 that has failed so far. The failure of c_1 in superstate $\boxed{011|101|110}$ cannot cause two transitions, so we must instead propagate forward the erasure of which component has failed and define a transition from $\boxed{011|101|110}$ to $\boxed{001|010}$ when c_1 fails. Continuing this reasoning, considering the failure of c_2, we would define a transition from $\boxed{011|101|110}$ to $\boxed{010|100}$. However, $\boxed{010}$ is already a member of the superstate $\boxed{001|010}$, so we must instead expand this superstate to $\boxed{001|010|101}$. Therefore we arrive at the result of the previous example, with a (super)state for zero, one, two, and three failed components.

As another example, consider a four-component system with the dependency $\langle c_1 \rightsquigarrow c_2 \oplus c_3 \rangle$, where it is ambiguous which component c_1 causes to fail. This corresponds to a transition from $\boxed{1111}$ to $\boxed{0011|0101}$ caused by the failure of c_1. An invalid dependency for this system would be $\langle c_1, c_2, c_4 \rightsquigarrow \emptyset \rangle$, as this would only cover one case of the $\boxed{0011|0101}$ superstate. Instead, we could write $\langle c_1, c_2, c_4 \oplus c_1, c_3, c_4 \rightsquigarrow \emptyset \rangle$, which would define a transition from $\boxed{0011|0101}$ to $\boxed{0010|0100}$ when c_4 fails.

9.1.6.2. Well-formedness properties with non-deterministic choice. We incorporate the requirements of the demonic choice principle into the well-formedness (WF) properties defined in Section 9.1.1.2. Since we have $\bigoplus\{\mathbf{C_1}\} = \mathbf{C_1}$, we can state all the WF properties in terms of non-deterministic choice over a set of causes or set of effects, generalizing the properties defined earlier.

For initiality, we allow the component to be part of a non-deterministic choice:

$$\forall c \in \mathbf{C}, \exists \langle \bigoplus \mathcal{C}' \rightsquigarrow \cdots \rangle \in \mathbf{D} \text{ where } \{c\} \in \mathcal{C}'. \tag{NDC-Initiality}$$

Termination comes with two requirements: not only must the system fail, but in any non-deterministic choice over failures, it must fail in every one:

$$\begin{aligned} &\exists \langle \cdots \rightsquigarrow \bigoplus \mathcal{E} \rangle \in \mathbf{D} \text{ where } \exists \mathbf{E} \in \mathcal{E}, \mathcal{S} \in \mathbf{E} \text{ and} \\ &\forall \langle \cdots \rightsquigarrow \bigoplus \mathcal{E} \rangle \in \mathbf{D} \text{ where } \exists \mathbf{E} \in \mathcal{E}, \mathcal{S} \in \mathbf{E}, \text{ then } \forall \mathbf{E} \in \mathcal{E}, \mathcal{S} \in \mathbf{E}. \end{aligned} \tag{NDC-Termination}$$

Monotonicity must hold for some particular cause and all resulting effects:

$$\begin{aligned} &\forall \langle \bigoplus \{\mathbf{C_1}, \mathbf{C_2}, \cdots\} \rightsquigarrow \bigoplus \{\mathbf{E_1}, \mathbf{E_2}, \cdots\} \rangle \in \mathbf{D} \\ &\forall \langle \bigoplus \{\mathbf{C_1} \cup \mathbf{C_3}, \mathbf{C_2} \cup \mathbf{C_4}, \cdots\} \rightsquigarrow \bigoplus \{\mathbf{E_3}, \mathbf{E_4}, \cdots\} \rangle \in \mathbf{D} \\ &\forall \mathbf{E} \in \{\mathbf{E_1}, \mathbf{E_2}, \cdots\}, \forall \mathbf{E}' \in \{\mathbf{E_3}, \mathbf{E_4}, \cdots\}, \\ &\exists \mathbf{C}' \in \{\mathbf{C_3}, \mathbf{C_4}, \cdots\}, \mathbf{E} \subseteq \mathbf{E}' \cup \mathbf{C}' \end{aligned} \tag{NDC-Monotonicity}$$

Finally, we introduce a new property to ensure that the "forgetfulness" or "erasure" of demonic choice is preserved. Before we can state this property, we introduce a notation, overloading the combinatorial "binomial choice" operator to work on sets. Given a set of sets $\mathcal{C} = \{\mathbf{C_1}, \mathbf{C_2}, \cdots, \mathbf{C_n}\}$ and an integer $0 \leq x \leq n$, define $\binom{\mathcal{C}}{x}$ to be the set of unions of combinations of $\mathbf{C_1}, \cdots, \mathbf{C_n}$. Thus, for example, $\binom{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}}{2} = \{\mathbf{C_1} \cup \mathbf{C_2}, \mathbf{C_1} \cup \mathbf{C_3}, \mathbf{C_2} \cup \mathbf{C_3}\}$. Also, $\binom{\mathcal{C}}{0} = \emptyset$ and $\binom{\mathcal{C}}{n} = \bigcup \mathcal{C}$.

$$\begin{aligned} &\forall \langle \mathbf{C_1} \oplus \cdots \oplus \mathbf{C_n} \rightsquigarrow \mathbf{E_1} \oplus \cdots \oplus \mathbf{E_m} \rangle \in \mathbf{D} \\ &\text{if } \exists \langle \bigoplus \mathcal{C}' \rightsquigarrow \cdots \rangle \in \mathbf{D} \text{ where } \mathbf{C_1} \subseteq \mathbf{C}'' \in \mathcal{C}', \\ &\text{then } \mathcal{C}' = \binom{\mathbf{C_1}, \cdots, \mathbf{C_n}}{x} \times \binom{\mathbf{E_1}, \cdots, \mathbf{E_m}}{y} \\ &\text{for some } 1 \leq x \leq n, 0 \leq y \leq m. \end{aligned} \tag{NDC-erasure}$$

This rule forbids, for instance, the existence of two dependencies $\langle c_1 \oplus c_2 \rightsquigarrow \cdots \rangle$ and $\langle c_1 \rightsquigarrow \cdots \rangle$ and likewise given the dependency $\langle c_1 \oplus c_2 \rightsquigarrow c_3 \oplus c_4 \rangle$, any dependency with c_1, c_3 in the causes must be of the form $\langle c_1, c_3 \oplus c_1, c_4 \oplus c_2, c_3 \oplus c_2, c_4 \rightsquigarrow \cdots \rangle$. The associativity and commutativity of $\oplus$ means this rule applies to any set of causes and effects in a non-deterministic choice.

9.1.6.3. Generalizations and refinements for non-deterministic choice. Introducing a non-deterministic choice constitutes a loss of information about the model, so it is therefore a generalization. Likewise, removing a non-deterministic choice is a refinement.

Introducing a non-deterministic choice implicitly adds dependencies as needed to meet the WF properties. Erasing the distinction between failure of a set of components C_1 and C_2 is done by replacing each instance of either with a non-deterministic choice, then adding sufficient other terms to meet the combinatorial requirements of NDC-erasure:

$$\begin{aligned}&\mathsf{unify}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_,\_] : \mathcal{P}(\mathbf{C}) \to \mathcal{P}(\mathbf{C}) \to \mathbf{Prop}\\ &\mathsf{unify}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\mathbf{C_1},\mathbf{C_2}] \triangleq (\mathbf{C},\mathsf{R},\mathbf{D}')\end{aligned} \tag{9.11}$$

where

$$\mathbf{D}' \triangleq \left\{\langle \bigoplus U(\mathcal{C}') \rightsquigarrow \bigoplus U(\mathcal{E}')\rangle \;\middle|\; \langle \bigoplus \mathcal{C}' \rightsquigarrow \bigoplus \mathcal{E}'\rangle \in \mathbf{D}\right\} \tag{9.11a}$$

$$U(\mathcal{C}) \triangleq \begin{cases} A(\mathcal{C}) \times \binom{N(\mathbf{C_1}) \cup N(\mathbf{C_2})}{x(\mathcal{C})} & \text{if } \mathbf{C_1} \subseteq \mathbf{C}' \in \mathcal{C} \text{ or } \mathbf{C_2} \subseteq \mathbf{C}' \in \mathcal{C} \\ \mathcal{C} & \text{otherwise.} \end{cases} \tag{9.11b}$$

$$A(\mathcal{C}) \triangleq \{\mathbf{C}' \setminus N(\mathbf{C_1}) \setminus N(\mathbf{C_2}) \mid \mathbf{C}' \in \mathcal{C}\} \tag{9.11c}$$

$$N(\mathbf{C_1}) \triangleq \mathcal{C}' \text{ where } \langle \bigoplus \mathcal{C}' \rightsquigarrow \cdots\rangle \in \mathbf{D} \text{ and } \mathbf{C_1} \in \mathcal{C}' \tag{9.11d}$$

$$x(\mathcal{C}) \triangleq \max\left\{x' \;\middle|\; \exists \mathbf{B} \in \binom{N(\mathbf{C_1}) \cup N(\mathbf{C_2})}{x'}, \mathbf{B} \subseteq \mathbf{C}' \in \mathbf{C}\right\} \tag{9.11e}$$

Refining by splitting a non-deterministic choice splits the cases into separate dependencies as needed, first for the causes and then for the effects:

$$\begin{aligned}&\mathsf{separate}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_ \oplus \_] : \mathcal{P}(\mathbf{C}) \to \mathcal{P}(\mathbf{C}) \to \mathbf{Prop}\\ &\mathsf{separate}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\mathbf{C_1} \oplus \mathbf{C_2}] \triangleq (\mathbf{C},\mathsf{R},\mathbf{D}'')\end{aligned} \tag{9.12}$$

where

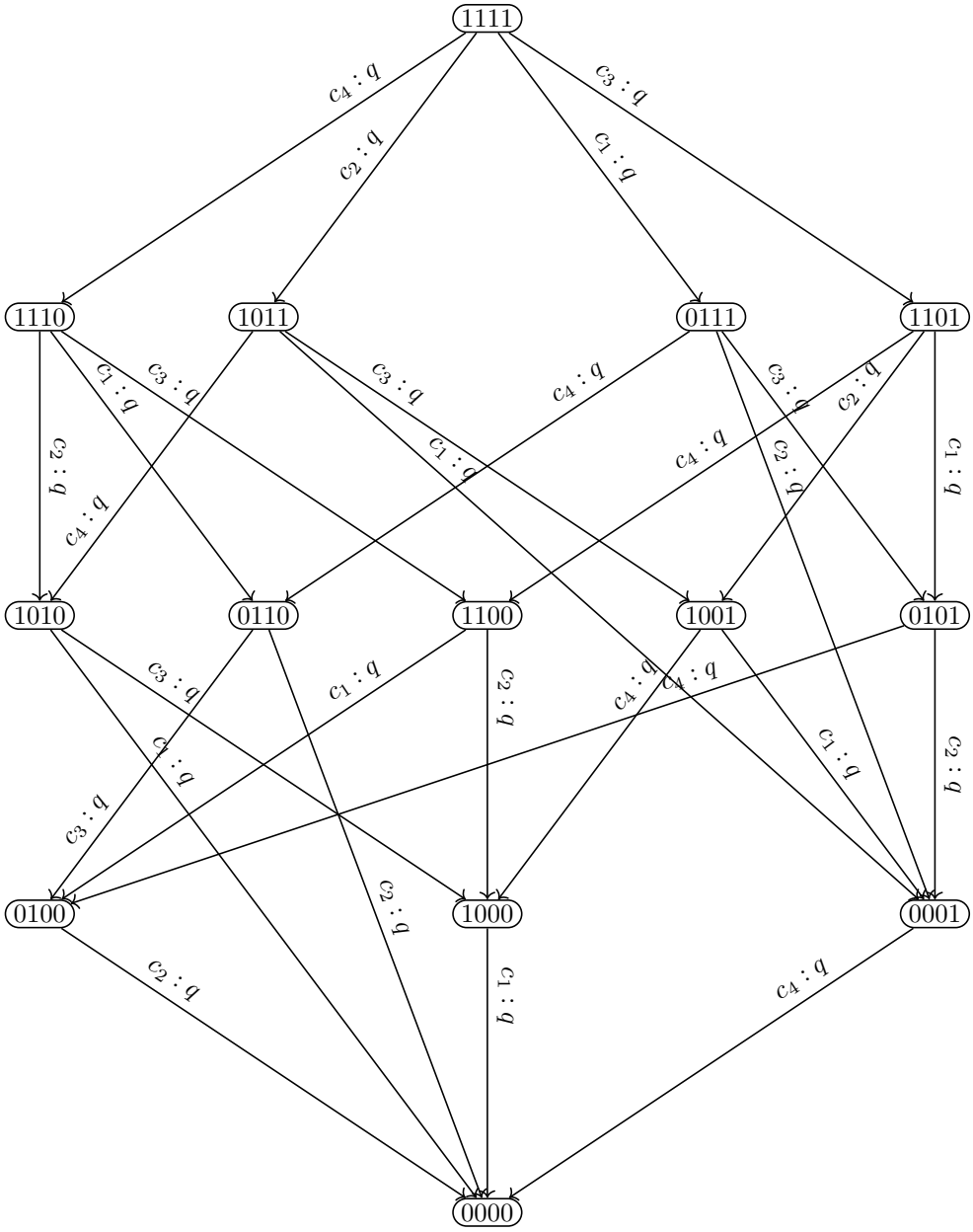
$$\mathbf{D}' \triangleq \left\{ SC\left(\langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle\right) \;\middle|\; \langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle \in \mathbf{D} \right\} \tag{9.12a}$$

$$SC\left(\langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle\right) \triangleq \begin{cases} \left\{ \begin{array}{l} \langle \bigoplus \{\mathbf{C}' \in \mathcal{C} \mid \mathbf{C_1} \subseteq \mathbf{C}' \text{ or } \mathbf{C_2} \nsubseteq \mathbf{C}'\} \rightsquigarrow \bigoplus \mathcal{E} \rangle \\ \langle \bigoplus \{\mathbf{C}' \in \mathcal{C} \mid \mathbf{C_2} \subseteq \mathbf{C}' \text{ or } \mathbf{C_1} \nsubseteq \mathbf{C}'\} \rightsquigarrow \bigoplus \mathcal{E} \rangle \end{array} \right\} & \begin{array}{l} \text{if } \mathbf{C_1} \subseteq \mathbf{C}' \in \mathcal{C} \\ \text{or } \mathbf{C_2} \subseteq \mathbf{C}' \in \mathcal{C} \end{array} \\ \langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle & \text{otherwise.} \end{cases} \tag{9.12b}$$

$$\mathbf{D}'' \triangleq \left\{ SE\left(\langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle\right) \;\middle|\; \langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle \in \mathbf{D}' \right\} \tag{9.12c}$$

$$SE\left(\langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle\right) \triangleq \begin{cases} \left\{ \begin{array}{l} \langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \{\mathbf{E}' \in \mathcal{E} \mid \mathbf{C_1} \subseteq \mathbf{E}' \text{ or } \mathbf{C_2} \nsubseteq \mathbf{E}'\} \rangle \\ \langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \{\mathbf{E}' \in \mathcal{E} \mid \mathbf{C_2} \subseteq \mathbf{E}' \text{ or } \mathbf{C_1} \nsubseteq \mathbf{E}'\} \rangle \end{array} \right\} & \begin{array}{l} \text{if } \mathbf{C_1} \subseteq \mathbf{E}' \in \mathcal{E} \\ \text{or } \mathbf{C_2} \subseteq \mathbf{E}' \in \mathcal{E} \end{array} \\ \langle \bigoplus \mathcal{C} \rightsquigarrow \bigoplus \mathcal{E} \rangle & \text{otherwise.} \end{cases} \tag{9.12d}$$

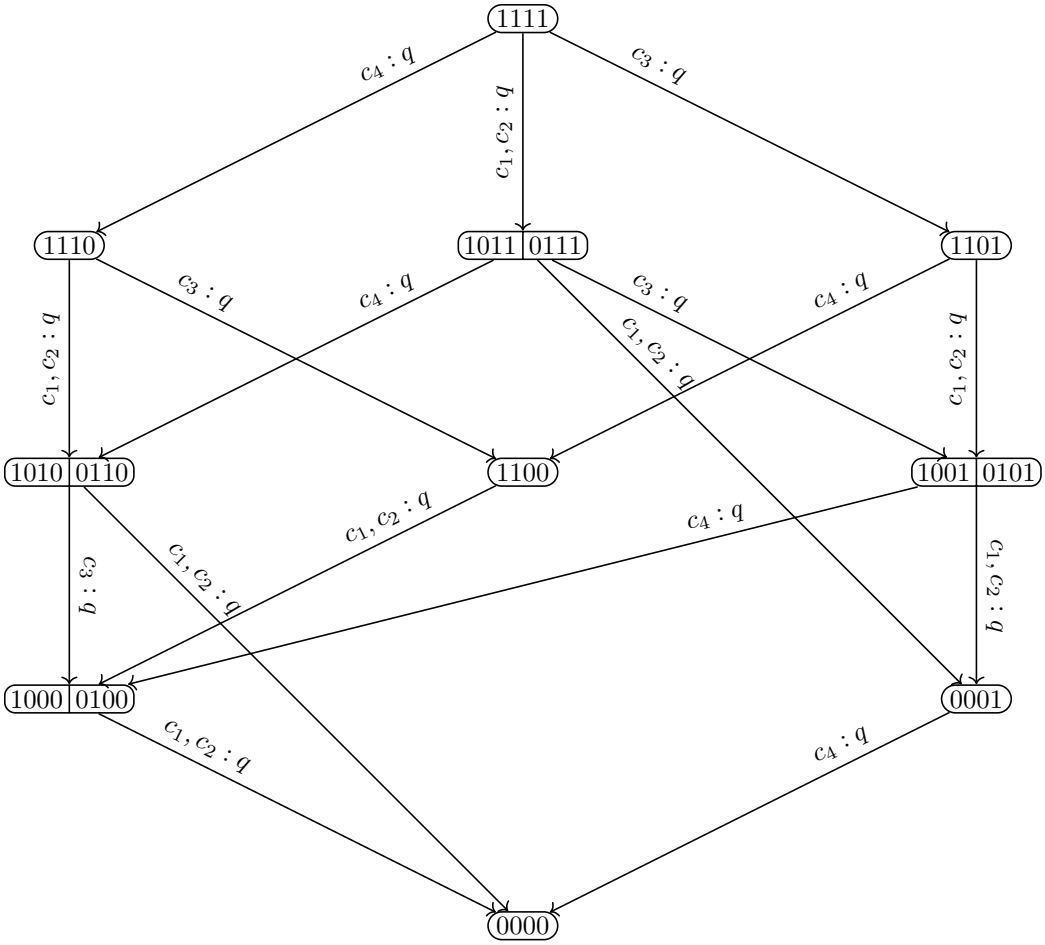
Consider the following example of generalization by introduction of non-deterministic choices. We begin with a system with all components in parallel and independent except that the failure of c_1 and c_2 causes the failure of c_3. The MIS model for this system contains the following Markov chain (again, with self-loops elided for readability):



With the corresponding abridged properties:

$$s_1 = (\{c_1, c_2, c_3, c_4\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = \mathsf{R}(c_4) = p, \{$$
$$\langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle, \langle c_4 \rightsquigarrow \emptyset \rangle,$$
$$\langle c_1, c_2 \rightsquigarrow c_3 \rangle, \langle c_1, c_2, c_3, c_4 \rightsquigarrow \mathcal{S} \rangle$$
$$\}).$$

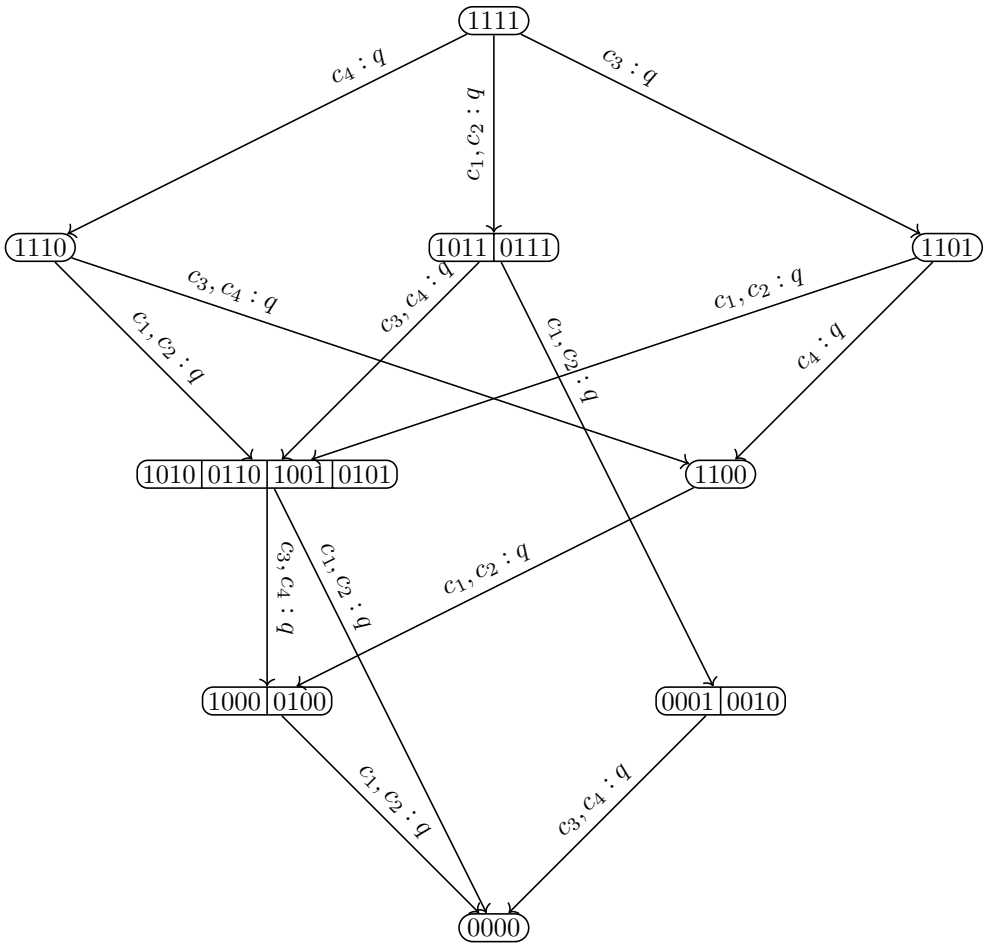
Suppose we generalize the system via $s_2 = \mathsf{unify}_{s_1}[c_1, c_2]$. Transitions caused by c_1 and c_2 consequently align:



With the corresponding abridged properties:

$$
\begin{aligned}
s_2 = (\{c_1, c_2, c_3, c_4\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = \mathsf{R}(c_4) = p, \{ \\
\langle c_1 \oplus c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle, \langle c_4 \rightsquigarrow \emptyset \rangle, \\
\langle c_1, c_2 \rightsquigarrow c_3 \rangle, \langle c_1, c_2, c_3, c_4 \rightsquigarrow \mathcal{S} \rangle \\
\}).
\end{aligned}
$$

We can see that states with components c_1 and c_2 are combined, and c_1 and c_2 share all transitions. However, the effect of unify is more complex than merge; consider a further generalization of $s_3 = \mathsf{unify}_{s_2}[c_1, c_3, c_1, c_4]$ followed by $s_5 = \mathsf{unify}_{s_4}[c_2, c_3, c_2, c_4]$:



With the corresponding abridged properties:

$$
\begin{aligned}
s_5 = (\{c_1, c_2, c_3, c_4\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = \mathsf{R}(c_4) = p, \{ \\
&\langle c_1 \oplus c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle, \langle c_4 \rightsquigarrow \emptyset \rangle, \\
&\langle c_1, c_2 \rightsquigarrow c_3 \oplus c_4 \rangle, \langle c_1, c_2, c_3 \oplus c_1, c_2, c_4 \rightsquigarrow \emptyset \rangle, \\
&\langle c_1, c_3 \oplus c_2, c_3 \oplus c_1, c_4 \oplus c_2, c_4 \rightsquigarrow \emptyset \rangle, \\
&\langle c_1, c_2, c_3, c_4 \rightsquigarrow \mathcal{S} \rangle \\
\}).
\end{aligned}
$$

Of note here: c_3 and c_4 stay independent (states (1101) and (1110), respectively) as long as both c_1 and c_2 are functional. However, once one of c_1 or c_2 fails, states containing a failed c_3 or c_4 are merged into superstates. Thus, the behavior of this system is beyond something expressible with merge alone. Finally, a note on refinement: $\mathsf{separate}_{s_5}[c_3 \oplus c_4]$ would result in both dependencies

$\langle c_1, c_2 \rightsquigarrow c_3 \rangle$ and $\langle c_1, c_2 \rightsquigarrow c_4 \rangle$, as the unify generalization erased the knowledge of which of c_3 or c_4 fails due to the failure of c_1 and c_2. The initial behavior can be refined from this model via calls to remove_dep.

10. MODEL TRANSFORMATION

When creating a mathematical representation of model transformation, we may initially conceive of it as a function that transforms models from the set $\mathbf{Model_1}$ to models in the set $\mathbf{Model_2}$ (Figure 10.1).

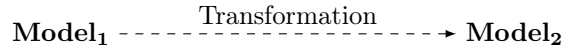


Figure 10.1. Initial model transformation concept

However, this approach poses some problems. First, it is not immediately clear how to define a transformation between some model types (for instance, transforming a system performance model to a system dependability model); in fact, this transformation may not be definable as a function in the mathematical sense. Second, if we have n modeling domains, in the worst case we will have to define $O(n^2)$ transformations since we cannot expect transformations to be transitive. In other words, transforming $\mathbf{Model_1} \rightarrow \mathbf{Model_2}$, then $\mathbf{Model_2} \rightarrow \mathbf{Model_3}$ may not be equivalent to directly transforming $\mathbf{Model_1} \rightarrow \mathbf{Model_3}$.

To address the second issue, we can insert an concrete domain, **Properties**, of collections of properties that describe systems. Properties can be concretized from models, and models can be abstracted from properties. We can then formalize transforming $\mathbf{Model_1}$ to $\mathbf{Model_2}$ as shown in Figure 10.2.

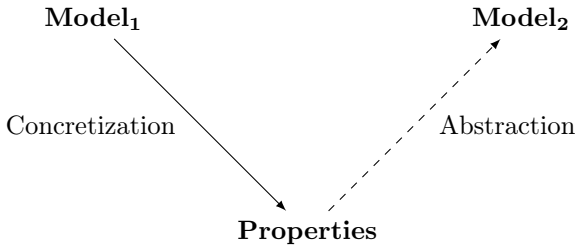


Figure 10.2. Transforming models through concretization and abstraction

Given a proper choice of properties domain, we can define an concretization function from model domains to the domain of properties. However, we are not guaranteed a corresponding abstraction function. For example, we would not expect a performance model to provide information about the reliability of components, so we cannot abstract a unique reliability model from a performance model.

However, we can cast this in terms of finding the most general model which holds for the given properties (see Fig. 10.3). Concretizing properties from a model can be viewed as deriving properties from just that model. Abstraction then produces all models, or a most general model, described by a collection of properties. If a set of models is produced, the user can then select the desired result model from this set of sound models.

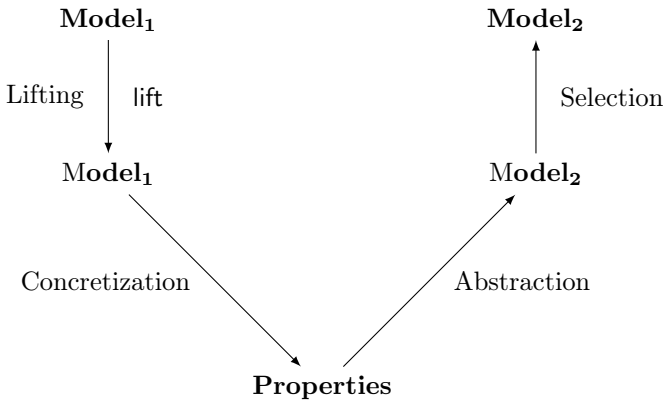


Figure 10.3. Transforming sets of models

10.1. SOUNDNESS

Given this formalization of system and model semantics, we can now formalize the problem of model transformation. Suppose we have a properties domain and two modeling formalisms with associated Galois connections to the properties domain $(\mathbf{Properties}, \alpha_{\mathbf{M_1}}, \gamma_{\mathbf{M_1}}, \mathbf{Model_1})$ and $(\mathbf{Properties}, \alpha_{\mathbf{M_2}}, \gamma_{\mathbf{M_2}}, \mathbf{Model_2})$. Furthermore, we have a correctness relation R_{P} which induces correctness relations $R_{\mathbf{M_1}}$ and $R_{\mathbf{M_2}}$.

Definition 10.1. A *model transformation* from $\mathbf{Model_1}$ to $\mathbf{Model_2}$ is a semantically sound mapping $\tau^{\mathbf{M_2}}_{\mathbf{M_1}} : \mathbf{Model_1} \to \mathbf{Model_2}$. That is, if $m_1 \in \mathbf{Model_1}$ is sound, then $\tau^{\mathbf{M_2}}_{\mathbf{M_1}}(m_1)$ is also sound.

We can define $\tau^{\mathbf{M_2}}_{\mathbf{M_1}}$ by first concretizing constraints from $m_1 \in \mathbf{Model_1}$, then abstracting an element of $\mathbf{Model_2}$ from it.

Theorem 10.1. *The mapping* $\tau_{\mathbf{M_1}}^{\mathbf{M_2}}(m_1) = (\alpha_{\mathbf{M_2}} \circ \gamma_{\mathbf{M_1}})(m_1)$ *is sound.*

Proof. Take $\mathcal{S} \in \mathcal{S}\mathbf{ys}$ and $m_1 \in \mathbf{Model_1}$.

$$
\begin{aligned}
& \mathcal{S}\, R_{\mathbf{M_1}}\, m_1 && \\
\iff & \mathcal{S}\, R_{\mathrm{P}}\, \gamma_{\mathbf{M_1}}(m_1) && \text{Defn. of } R_{\mathbf{M_1}} \\
\implies & \mathcal{S}\, R_{\mathrm{P}}\, (\gamma_{\mathbf{M_2}} \circ \alpha_{\mathbf{M_2}} \circ \gamma_{\mathbf{M_1}})(m_1) && \text{Eqn. (8.1), Prop. ((i))} \\
\iff & \mathcal{S}\, R_{\mathbf{M_2}}\, (\alpha_{\mathbf{M_2}} \circ \gamma_{\mathbf{M_1}})(m_1) && \text{Defn. of } R_{\mathbf{M_2}}
\end{aligned}
$$

□

To sum up the transformation process: begin with a model $m_1 \in \mathbf{Model_1}$. Concretize properties of the system from $\{m_1\}$, then apply $\tau_{\mathbf{M_1}}^{\mathbf{M_2}}$ to produce a set of models $M_2' \subseteq \mathbf{Model_2}$. Finally, select a model from M_2' by introducing information about the system not present in m_1.

Figure 10.4 illustrates the domains, mappings, and relationships present in this formalization of model transformation.

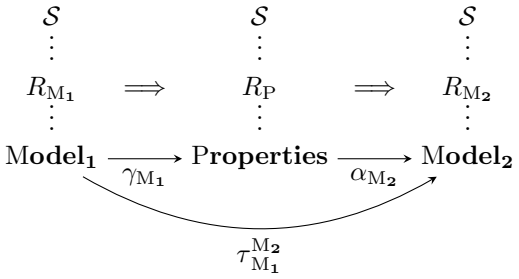


Figure 10.4. Sound model transformation

10.2. EXAMPLE

This example draws upon the MIS example in [10, § II.A] and [121, § 2]. For further detail on the MIS modeling formalism, see Section 9.1.5.2.

We consider a power grid with one generator (c_1), one load (c_2), and two power lines (c_3, c_4) connected in parallel, as shown in Figure 10.5. Defining in detail the meaning of this model is left for the future; for now, we will view it as only encoding the components of the system and their interconnections.

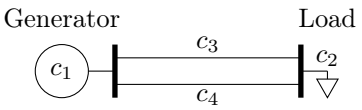


Figure 10.5. Two-line topology example

Next we describe the MIS model of this system presented in [10, § II.A]. Each line has reliability p_L; the generator and load are assumed to not fail. The system is considered to be functional as long as it is capable of transmitting power (at least one line is up).

The MIS model for the reliability of this system under these assumptions is given by four basic equations. The possible states are outlined in Table 10.1; we will generally think of each state as the set of functioning components in that state. (So $S_0 \simeq \{l_1, l_2\}$, $S_1 \simeq \{l_1\}$, etc.) The initial system state is given by Equation 10.1; this is generally of the form $[1, 0, \ldots, 0]$ in MIS reliability modeling. Equation 10.2 defines the states which are considered functional. The matrices in 10.3 and 10.4 describe how component failure causes the system state to change. Finally, equation 10.5 puts all these pieces together into an expression for system reliability.

$$\Pi_0 = [1, 0, 0, 0] \tag{10.1}$$

$$u = [1, 1, 1, 0] \tag{10.2}$$

$$P_{l_1} = \begin{bmatrix} p_L & 0 & q_L & 0 \\ 0 & p_L & 0 & q_L \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.3}$$

$$P_{l_2} = \begin{bmatrix} p_L & q_L & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & p_L & q_L \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.4}$$

$$R = \Pi_0^T * P_{l_1} * P_{l_2} * u = p_L^2 + 2p_L q_L \tag{10.5}$$

The Markov chain for this model is shown in Figure 10.6.

Table 10.1. State definition matrix

	Components	
States	l_1	l_2
S_0	1	1
S_1	1	0
S_2	0	1
S_3	0	0

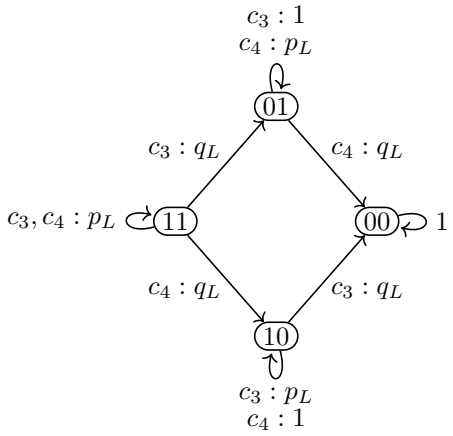


Figure 10.6. Markov chain representation of the example MIS model

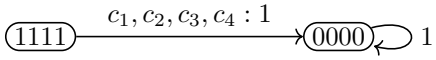
With this MIS model in mind, let us see how it can be derived, via transformation and refinement, from the topology model m_{top} of Figure 10.5. We begin by concretizing an element of P**rop** from this model: $\gamma_{top}(m_{top}) = p_{top}$. This is the greatest element in P**rop** described by m_{top}. Since our topology formalism only describes the elements of the system and their interconnections, the most P**rop** can deduce from it is that the system consists of four components. Thus, p_{top} is

$$p_{top} = (\{c_1, c_2, c_3, c_4\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = \mathsf{R}(c_4) = 0, \{$$
$$\langle c_1 \rightsquigarrow c_2, c_3, c_4, \mathcal{S}\rangle, \langle c_2 \rightsquigarrow c_1, c_3, c_4, \mathcal{S}\rangle,$$
$$\langle c_3 \rightsquigarrow c_1, c_2, c_4, \mathcal{S}\rangle, \langle c_4 \rightsquigarrow c_1, c_2, c_3, \mathcal{S}\rangle$$
$$\}).$$

In other words, all components are dependent on each other, their individual reliability is unconstrained, and failure of any component leads to a failure of the system. This collection of properties can be refined from $\top \in \mathbf{Prop}$ by repeatedly splitting components:

$$p_{top} = \mathsf{split}_{\top}[c_1 \to c_1, c_2, c_3, c_4]$$

The MIS model abstracted from this collection of properties is as shown:



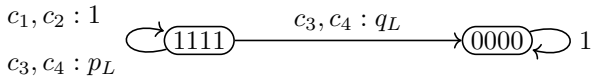
The reliability of this system is at least 0; since the component reliabilities are unconstrained, in the worst case, they immediately fail.

At this point it is worth noting that, if our topology model were more detailed, we could derive a more detailed p_{top} from it. For instance, if the topology model contained line capacities, generator supply, load demand, and a definition of which loads are essential to serve, it would be possible to deduce more accurate component dependencies.

Though that information is not present in the topology model we are considering, we can introduce it to our reliability model via further refinements to p_{top}. We begin by refining the component reliabilities to match our above assumptions:

$$p_2 = \mathsf{tighten\_rel}_{p_{top}}[c_1, c_2, 1]$$
$$p_3 = \mathsf{tighten\_rel}_{p_3}[c_3, c_4, p_L]$$

This leads to a more representative MIS model:



This system's reliability is at least p_L^2: both lines must not fail for the system to be up.

What remains is to constrain the dependencies among components to match our assumptions about component failure: component failures are independent, and the system is functional as long as c_1, c_2, and either c_3 or c_4 are functional. First, we eliminate dependencies among components:

$$p_4 = \mathsf{remove\_dep}_{p_3}[c_2, c_3, c_4 \rightsquigarrow c_1]$$

$$p_5 = \mathsf{remove\_dep}_{p_4}[c_1, c_3, c_4 \rightsquigarrow c_2]$$

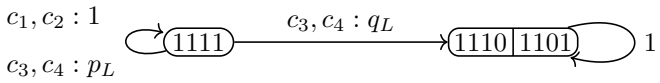
$$p_6 = \mathsf{remove\_dep}_{p_5}[c_1, c_2, c_4 \rightsquigarrow c_3]$$

$$p_7 = \mathsf{remove\_dep}_{p_6}[c_1, c_2, c_3 \rightsquigarrow c_4]$$

The properties generated from this, p_7, have a large set of **Deps**, but we can summarize them as follows:

$$p_7 \simeq (\{c_1, c_2, c_3, c_4\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = 1; \mathsf{R}(c_3) = \mathsf{R}(c_4) = p_L, \{$$
$$\langle c_1 \rightsquigarrow \mathcal{S}\rangle, \langle c_2 \rightsquigarrow \mathcal{S}\rangle, \langle c_3 \rightsquigarrow \mathcal{S}\rangle, \langle c_4 \rightsquigarrow \mathcal{S}\rangle$$
$$\})$$

The MIS model abstracted from p_7 has the same reliability as that of the model abstracted from p_3, since any component failure still leads to system failure. However, the Markov chain representation of the MIS model for p_7 is notably different:



Notably, the failed state is now a superstate indicating that the system fails as soon as c_3 or c_4 fail, but such failures do not lead to further component failures in the system. The (0000) state is now unreachable.

Finally, we make system failure independent of either c_3 or c_4 failing:

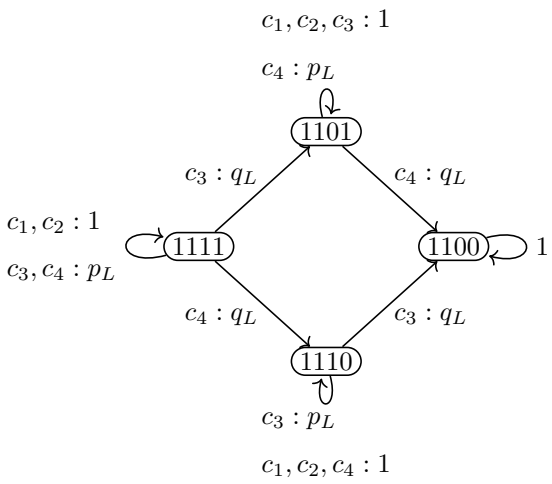
$$p_8 = \mathsf{remove\_dep}_{p_7}[c_3 \rightsquigarrow \mathcal{S}]$$

$$p_9 = \mathsf{remove\_dep}_{p_8}[c_4 \rightsquigarrow \mathcal{S}]$$

Again, we can summarize p_9 as follows:

$$p_9 \simeq (\{c_1, c_2, c_3, c_4\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = 1; \mathsf{R}(c_3) = \mathsf{R}(c_4) = p_L, \{$$
$$\langle c_1 \rightsquigarrow \mathcal{S}\rangle, \langle c_2 \rightsquigarrow \mathcal{S}\rangle, \langle c_3 \rightsquigarrow \emptyset\rangle, \langle c_4 \rightsquigarrow \emptyset\rangle, \langle c_3, c_4 \rightsquigarrow \mathcal{S}\rangle$$
$$\})$$

The effect of these refinements is to make (1110) and (1101) states where the system is functional and to make (1100) reachable and failed:



The system reliability given by the MIS model abstracted from p_9 is $p_L^2 + 2p_L q_L$, as desired. Note that this MIS model contains strictly more information than the one shown in Figure 10.6; it contains two extra components, but because they do not fail, they have no effect on the resulting reliability computation. If we desired, we could generalize this model slightly by $\mathsf{merge}_{p_7}[c_1, c_2, c_3 \to c_3]$, which obtains exactly the MIS model we set out to derive.

11. CONCLUSIONS AND FUTURE WORK

In this work, we set out to explore aspects of complex system modeling and metamodeling. We began with an exploration of the effects of electromagnetic disturbances on control system hardware. The goal of this project is to detect the effects of electromagnetic interference in system peripherals. We focus on a USB host controller, which is the device responsible for all USB communication to and from a system. The effects of two types of EMD are investigated: first electrostatic discharge, then electromagnetic interference. We capture sequences of snapshots of a host controller's control registers, both when the system is operating as intended and when it is being exposed to EMD. These sequences are processed into sequences of categorical time-series data for analysis. We divide the data into two datasets: baseline and EMD-exposed.

First, we analyzed various properties of these datasets. These datasets can be thought of as an execution graph, akin to a call graph as used in software analysis. Non-determinism in this execution graph is introduced by exposure to EMD; additional graph states (nodes) and transitions (arcs) can be found in the EMD-exposed dataset as compared to the baseline dataset. In addition, we investigated the statistical characteristics of our datasets. The variance, as measured by the Gini index and extropy, differed between the two datasets. Furthermore, the autocorrelation of the two datasets differed. At a minimum, these differences allow us to conclude that our instrumentation is capable of detecting some effects of EMD.

Second, we investigated the ability of classifiers to identify whether system operation was indicative of EMD exposure. As an initial test, we devised a classification method which produced promising results from the ESD datasets. We then extended this work significantly for the EMI datasets. We devised a method for synthesizing a dataset for training classifiers from our experimental data. Several classification approaches were evaluated: hidden Markov models, neural networks, support vector machines, random forest classifiers, and gradient boosted classifiers. The gradient boosted random forest classifier performed best on our sliding window classification task, reaching 92% accuracy, 92% recall, and 0.87 F1 score.

Having investigated system instrumentation and modeling, we then turned to metamodeling. This project seeks to capture relationships between models and the properties of a system they encode. We demonstrated a formalization of model and system semantics. Models abstract system semantics; therefore, we can derive constraints on a system from models of it. Conversely, given constraints on a system, we can abstract a set of models that are consistent with those constraints.

To formalize the soundness of this approach, we apply abstract interpretation, which defines a correctness relation between systems and constraints. If our abstraction and concretization mappings between a given modeling formalism and system constraints form a Galois connection between the two domains, we can show that these mappings and the correctness relation for system constraints induce a correctness relation between systems and the models of the modeling formalism.

A key aspect of these connections between models and properties is the ability to order the model and property domains by specificity. This order maps cleanly onto the notion of refinement and generalization. We demonstrated an approach to refinement and generalization of MIS reliability models. Key to this approach is a system constraints domain, which captures the operation of a system in an abstract, easily manipulated fashion. These constraints describe the components of the system, their reliability, and dependencies that describe how one set of component failures can trigger another. Given these constraints, we create generalization and refinement operators that allow us to relax or add constraints as needed. Thus, we can simplify a system for easier evaluation by generalizing it or we can iteratively develop one through repeated refinement. Finally, we link these constraints to MIS reliability models, enabling us to refine or generalize models of a common modeling formalism.

Through the lens of model abstraction and concretization, the process of model transformation becomes the process of concretizing system properties from one model, then abstracting a second model from these properties. We show that this process is sound; that is, if the initial model is correct, then the final model will also be correct. This formalization of model transformation demonstrates the utility of our metamodeling approach for proving a model manipulation sound.

The remainder of this chapter lays out ideas for bodacious projects which build on the totally awesome work presented in this dissertation.

11.1. FUNCTIONAL MODELING OF EMD EFFECTS

One particular challenge with bringing computational intelligence techniques to bear on a problem is collecting a large and detailed collection of data with which to create models. While our datasets suffice for confirmation of the effectiveness of our approach, additional system characterization is necessary for creating models which can be used for in-field monitoring. We can enhance the data we collect with additional indicators of EMD; for example, ESD and EMI generators contain a “trigger” output which can be correlated with the recorded register snapshots to indicate exactly when interference was induced in the system. Another datum which can be correlated with our

instrumentation's data is the power draw of the system under test: certain hardware-level effects of EMD cause changes in power draw, such as transistor latch-up, which is caused by excessive charge collecting on a transistor gate holding the transistor "on".

A multitude of tools for characterizing the stochastic processes which generate our datasets are available. In addition, models of stochastic processes may be fit to our data, allowing us to both confirm whether system operation conforms to a model of baseline operation and to generate synthetic datasets. The field of categorical process control charts may be applied to produce low-overhead real-time monitors for EMD.

The data collected from our instrumentation can also be interpreted in a graph theoretic fashion. A variety of graph-theoretic tools can thus be brought to bear on it. Various measures of graph centrality or other graph metrics can, for example, be used to characterise operation and validate instrumentation effectiveness. In addition, graph learning algorithms may be useful for our classification work.

A third perspective of our data is as traces of the execution of a particular process. This can be connected to machine learning techniques such as process discovery. In addition, logics such as linear temporal logic can be used to specify process operation and to automatically synthesize monitors necessary to ensure the operation meets the given specification.

Anomaly detection algorithms can also be applied to this dataset to identify sub-sequences of EMD-exposed operation that are of interest. In particular, algorithms developed for intrusion detection systems, which monitor the operation of network-connected systems to detect the presence of an attacker, are a promising fit for both the categorical nature of our data and our goal of real-time monitoring. Identified sub-sequences can also be analyzed further by hand to determine what meaning the peripheral's specification assigns to them. This may lead to better understanding of the path of EMD propagation through a system or to means of mitigating or correcting EMD effects in software.

Finally, this approach can be expanded to additional system peripherals, leading to a whole-system EMI instrumentation and detection method.

11.2. MODEL FAITHFULNESS

In our metamodeling work, we focus on the connections between models and the properties of a system that they encode. We create methods of soundly operating on models and properties so that these connections are preserved. However, this work does not make formal the connection between a system and its properties or models. This connection, which is represented as $\mathcal{S} \vdash$ in

Figure 8.1, ensures that models and properties reflect the physical system under consideration. The analogous software relationship is the statement that a program's text reflects both its concrete semantics and abstract semantics. For software, this is straightforward to show: one states how the syntax of a program is interpreted, concretely and abstractly. For systems, however, the situation is more complex, as the physical system is not a mathematical object in the way that program syntax is. Instead, we may envision the connection of systems to models and properties as a set of criteria to which the system must conform, within a given tolerance. Model refinements would add tests or tighten tolerances; model generalizations would relax the requirements. This connection in our work relates closely to the concept of model faithfulness.

Model faithfulness — the extent to which a model corresponds to a specific physical system — is a challenging topic for formal reasoning. Reasoning about models, software, and mathematics together is possible as all these topics are represented in language; incorporating physical objects is an altogether different challenge. However, this should not be reason for us to give up entirely!

Several approaches to reasoning about model faithfulness have been proposed in the literature. Validity frames [122, 123] are one such approach based on identifying a context in which a model is valid. For example, an Ohm's Law model of a resistor is valid (accurate within a certain percentage of actual operation) in a context defined by the temperature of the resistor, the voltage across and current flowing through it, and the electrical frequency. This validity frame connects a physical component, the resistor; experiment procedures and results, which establish the limits of the validity frame; and models which capture the properties of the component.

A second approach, based on architectural views is proposed in [89, 91]. These views map system components to elements of models; additional information can be added to these mappings to indicate assumptions about component operation. Critically, the authors show that a complete definition of assumptions about component operation is not required to perform model validation. Thus, these views can be applied to the model faithfulness problem without requiring a full specification of system operation.

11.3. REFINEMENT AND GENERALIZATION FOR PHYSICAL TOPOLOGY MODELS

Extending this metamodeling approach to more model formalisms increases the utility of this paradigm for system modelers. In particular, including a modeling formalism which expresses physical dynamics would enable changes to, e.g., a reliability model, to be soundly translated to the physical system. Selecting a modeling language for a physical system presents several issues.

To simplify the formalization process, we desire to select a modeling language more high-level than pure differential algebraic equations. In addition, the language we choose should be usable in complex hybrid system modeling and able to capture multiple physical domains, including electrical, mechanical, and hydraulic. Finally, we desire a language that is compact; formalizing all of Simulink or Modelica is a daunting task!

One possible language is that of bond graphs [124]. These models describe the flow of energy through a physical system and thus can represent components from a wide variety of domains. The language itself is simple: graphs consisting of energy storage, transmission, and dissipation elements connected through bonds that represent the flow of energy among these components. However, this language does present some challenges. Multiple representations of the energy flow through the same system are possible, so equivalent system representations will have to be accounted for in refinement and generalization operations. In addition, bond graphs inherently take a "whole system" perspective; creating sub-models and connecting them is not always simple [125].

Another language is that of linear graphs [126]. In these models, graph nodes are points where components, represented as directed edges, connect. Each edge has a "through" and "across" quantity; for electrical components, these are current and voltage, respectively. While this language eliminates some of the ambiguity present in bond graph models and provides clear points for connecting sub-models, it can require an unnatural choice of units in certain domains and is not as prevalent in hybrid system modeling.

An altogether different and much simpler approach would be to formalize a simple flow-graph representation where nodes represent components and edges the connections among them. For the case of a power grid, nodes could represent sources, loads, and lines, each with associated supply, capacity, or demand. Such a formalism closely follows the representation of power grids used as input for many power grid simulators. For evaluation, we could naïvely evaluate them as flow graphs, which would be relatively simple to formalize but not very accurate. On the other hand, these models could be simulated using commonly used engineering tools, which would be both more accurate and more practical, but possibly very difficult to prove sound.

APPENDIX A.
LATTICE THEORY

A lattice is a structure on top of a set of objects that imparts some partial ordering to those objects. In addition, lattices can guarantee the existence of certain objects and function properties. The order properties of lattices are commonly used to cut down the search space of certain algorithms. Order can also be used to formalize properties of functions; both applications are used in abstract interpretation. The discussion here is intended to provide a brief overview sufficient for description of our model transformation approach; readers are referred to [127] for greater detail.

1. PARTIALLY ORDERED SETS

Integral to the concept of a lattice is the concept of *ordering* on a set:

Definition A.1. A *partial ordering* on a set L is a relation $\sqsubseteq$: $L \times L \rightarrow \{\textbf{true}, \textbf{false}\}$ that is

1. *Reflexive*: $l \sqsubseteq l, \forall l \in L$
2. *Transitive*: If $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_3$, then $l_1 \sqsubseteq l_3$
3. *Anti-symmetric*: If $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_1$, then $l_1 = l_2$

Definition A.2. A *partially ordered set* or *poset* $(L, \sqsubseteq)$ is a set with a partial order on it.

Sometimes we will write $\sqsubseteq_L$ when working with multiple partially ordered sets.

Partial ordering does not require that $\forall l_1, l_2 \in L$, $l_1 \sqsubseteq l_2$ or $l_2 \sqsubseteq l_1$. Such a requirement would yield a *total* rather than partial ordering.

For example, the order relation $\sqsubseteq$ can be defined on a Cartesian coordinate system (R^2) as:

$$(x_1, y_1) \sqsubseteq (x_2, y_2) \iff x_1 \leq x_2 \text{ and } y_1 \leq y_2$$

With this definition, $(0,0) \sqsubseteq (1,2)$ and $(0,0) \sqsubseteq (2,1)$, but $(1,2)$ and $(2,1)$ are not comparable using $\sqsubseteq$, $1 \leq 2$ but $2 \not\leq 1$.

Definition A.3 (Upper Bound)**.** A set $Y \subseteq L$ has $l \in L$ as an *upper bound* if $y \sqsubseteq l, \forall y \in Y$. l is a *least upper bound* if, for any upper bound l', $l \sqsubseteq l'$. We denote the least upper bound by $\bigsqcup Y$ (sometimes called the *meet* of Y). $\bigsqcup\{l_1, l_2\}$ can also be written as $l_1 \sqcup l_2$.

Definition A.4 (Lower Bound)**.** l is a lower bound of Y if $l \sqsubseteq y, \forall y \in Y$. The *greatest lower bound* is defined analogously to the least upper bound. It is denoted $\bigsqcap Y$ and sometimes called the *join* of Y.

Continuing our example in R^2, $(0,0)$ can be identified as a lower bound of $\{(1,2),(2,1)\}$. In addition, $\bigsqcup\{(1,2),(2,1)\} = (2,2)$ and $\bigsqcup\{(0,0),(0,1),(1,0)\} = (1,1)$.

2. LATTICES

Lattices are partially ordered sets that guarantee the existence of certain elements. This existence is essential to building well-defined mechanisms that depend on partial orders.

Definition A.5. A *lattice* L is a partially ordered set where $l_1 \sqcup l_2$ and $l_1 \sqcap l_2$ are defined for all $l_1, l_2 \in L$.

Definition A.6. A *complete lattice* L is a partially ordered set where $\bigsqcup Y$ and $\bigsqcap Y$ are defined for all $Y \subseteq L$.

A consequence of A.6 is that every complete lattice has a least element, written $\bot$, and a greatest element, written $\top$. Proof: $\bot = \bigsqcap L$ and $\top = \bigsqcup L$ are defined for all complete lattices.

Note that every finite lattice is complete.

3. FUNCTIONS ON POSETS

We will often want to relate one poset to another. As with many other mathematical structures, we can define homomorphisms and isomorphisms between posets:

Definition A.7. A function $f : P \to Q$ between posets $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ is *monotone* (or *order-preserving*) if

$$p_1 \sqsubseteq_1 p_2 \Rightarrow f(p_1) \sqsubseteq_2 f(p_2), \forall p_1, p_2 \in P$$

As an example, let $f : \mathrm{R}^2 \to \mathrm{R}^2$ be defined as $f(x, y) = (x + 1, y)$. f is monotone, as:

$$(x_1, y_1) \sqsubseteq (x_2, y_2) \iff x_1 \leq x_2 \iff x_1 + 1 \leq x_2 + 1 \iff f(x_1, y_1) \sqsubseteq f(x_2, y_2)$$

Definition A.8. A function $f : P \to Q$ between posets is a *order embedding* (or a *lattice homomorphism*) if

$$p_1 \sqsubseteq_1 p_2 \iff f(p_1) \sqsubseteq_2 f(p_2), \forall p_1, p_2 \in P$$

Theorem A.1. *An order-embedding is an injective (1:1) function.*

Definition A.9. A function $f : P \to Q$ is an *order isomporphism* if f is a surjective (onto) order embedding. That is,

$$p_1 \sqsubseteq_1 p_2 \iff f(p_1) \sqsubseteq_2 f(p_2), \forall p_1, p_2 \in P$$

and

$$\exists p \in P \text{ such that } q = f(p), \forall q \in Q$$

Theorem A.2. *If $f : P_1 \to P_2$ is an order embedding between P_1 and P_2, then f is an order isomporphism between P_1 and $f(P_1)$.*[1] *Note that the order operation for the poset $f(P_1)$ is the order operation for P_2 restricted to values in $f(P_1)$.*

In addition to order-preserving maps, we can also have join- and meet-preserving maps between posets:

Definition A.10. A function $f : P_1 \to P_2$ between posets is *additive* (or a *join morphism*) if

$$f(p_1 \sqcup_1 p_2) = f(p_1) \sqcup_2 f(p_2), \forall p_1, p_2 \in P_1$$

f is *completely additive* if, for any $Y \subseteq P_1$ where $\bigsqcup_1 Y$ is defined,

$$f(\bigsqcup\nolimits_1 Y) = \bigsqcup\nolimits_2 \{f(y) : y \in Y\}$$

Definition A.11. A function $f : P_1 \to P_2$ between posets is *multiplicative* (or a *meet morphism*) if

$$f(p_1 \sqcap_1 p_2) = f(p_1) \sqcap_2 f(p_2), \forall p_1, p_2 \in P_1$$

f is *completely multiplicative* if, for any $Y \subseteq P_1$ where $\bigsqcap_1 Y$ is defined,

$$f(\bigsqcap\nolimits_1 Y) = \bigsqcap\nolimits_2 \{f(y) : y \in Y\}$$

4. POWERSETS

A common complete lattice is the set of all subsets of a set S, called the powerset and denoted $\mathcal{P}(S)$. $(\mathcal{P}(S), \subseteq)$ forms a complete lattice with $\bigsqcup = \bigcup$ and $\bigsqcap = \bigcap$. $\top = S$ and $\bot = \emptyset$ since every element of $\mathcal{P}(S)$ is a subset of S and $\emptyset$ is a subset of every set.

For example, if $S = \{1, 2, 3\}$,

$$\mathcal{P}(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

and the resulting lattice can be drawn like so[2]:

[1] $f(P_1)$ denotes the image of f in P_2, i.e. $\{f(p) : p \in P_1\}$.

[2] This figure is known as a *Hasse diagram*; a line from y down to x indicates that $x < y$ and there is no z such that $x < z < y$.

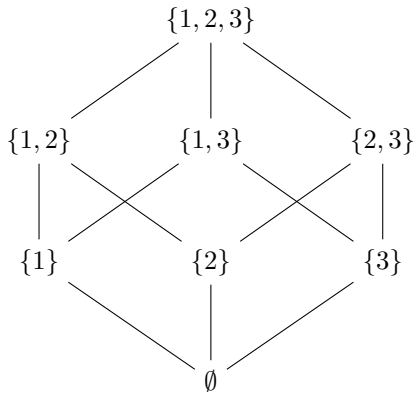


Figure A.1. The lattice for $(\mathcal{P}(S), \subseteq)$

Now, powersets are a pretty intuitive way to think about posets, but you may be worried that there are posets that look nothing like powersets. In fact, this is (almost) not the case!

Thinking in terms of powersets gives an intuitive mental picture of lattices, but will this intuition always hold? First, observe that any $M \subseteq \mathcal{P}(S)$ also forms a poset $(M, \subseteq)$. We will call posets ordered by inclusion ($\subseteq$) *inclusion posets*.

Theorem A.3. *Every poset is order-isomorphic to an inclusion poset.*

Proof. Let $(P, \sqsubseteq)$ be a poset. For each $p \in P$, let $\pi_p = \{x \in P : x \sqsubseteq p\}$.

Let $\Pi = \{\pi_p : p \in P\}$. Then $\Pi \subseteq \mathcal{P}(P)$ and thus $(\Pi, \subseteq)$ is an inclusion poset.

Define $f : P \to \Pi$ by $f(p) = \pi_p$. By our definition of π_p, $p_1 \sqsubseteq p_2 \iff f(p_1) \subseteq f(p_2)$. Thus, f is an order embedding. Furthermore, if $f(p_1) = f(p_2)$, then $p_1 \sqsubseteq p_2$ and $p_2 \sqsubseteq p_1$, so by anti-symmetry, f is injective.

Therefore, f is an order isomorphism from P to Π. □

As a consequence of this theorem, even if a poset or lattice is not an inclusion poset, we can still think of it as one.

APPENDIX B.
GALOIS CONNECTIONS

A Galois connection between two posets is a relationship that is somewhat weaker than an order-preserving isomorphism. They are commonly used to abstract information from one poset to another with better-understood structure. Galois connections are a generalization of the connection Evariste Galois discovered between field extensions and permutation groups.

Definition B.1. A *Galois connection* (P, α, γ, Q) between two posets $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$, is a pair of monotone functions $\alpha : P \to Q$ and $\gamma : Q \to P$ such that

$$p \sqsubseteq_P (\gamma \circ \alpha)(p), \forall p \in P$$

$$(\alpha \circ \gamma)(q) \sqsubseteq_Q q, \forall q \in Q$$

α is sometimes called the *abstraction operator* and γ the *concretization operator*.

Definition B.1 is sometimes written $\lambda p.p \sqsubseteq \gamma \circ \alpha$ and $\alpha \circ \gamma \sqsubseteq \lambda q.q$.

Note that, for any order isomorphism $f : P \mapsto Q$, (P, f, f^{-1}, Q) is a Galois connection. However, α and γ are not generally inverses nor order isomorphisms.

The study of Galois connections is exactly the study of category theory's *adjunctions*:

Definition B.2. An *adjunction* (P, α, γ, Q) between two posets $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$, is a pair of functions $\alpha : P \to Q$ and $\gamma : Q \to P$ such that

$$\alpha(p) \sqsubseteq_Q q \iff p \sqsubseteq_P \gamma(q)$$

α is sometimes called the *lower* or *left adjunct* and γ the *upper* or *right adjunct*. This clever naming is due to the fact that α appears on the left hand side of the order relationship and γ on the right hand side in the definition.

Theorem B.1. *(P, α, γ, Q) is a Galois connection if it is an adjunction.*

So, definitions B.1 and B.2 are two ways of looking at the same relationship.

From this point on, we are going to discuss Galois connections between complete lattices. Most of these properties can be applied to general posets, but having well-defined $\bigsqcap$ and $\bigsqcup$ operators makes the notation far cleaner.

Theorem B.2. *α uniquely determines γ by*

$$\gamma(q) = \bigsqcup \{p : \alpha(p) \sqsubseteq q\}$$

and likewise, γ uniquely determines α by

$$\alpha(p) = \bigsqcap\{q : p \sqsubseteq \gamma(q)\}$$

Proof.

$$\begin{aligned} \gamma(q) &= \bigsqcup\{p : p \sqsubseteq \gamma(q)\} && \text{definition of } \gamma \\ &= \bigsqcup\{p : \alpha(p) \sqsubseteq q\} && \gamma \text{ is an adjoint (Defn. B.2)} \end{aligned}$$

Thus, α determines γ.

If γ_1 and γ_2 both form Galois connections with α, then both γ_1 and γ_2 are defined to be

$$\bigsqcup\{p : \alpha(p) \sqsubseteq q\}$$

and thus $\gamma_1 = \gamma_2$.

The proof for α being uniquely determined from γ is analogous. □

Note that this theorem does not say that there exists a unique Galois connection between two lattices; we may have two connections $(P, \alpha_1, \gamma_1, Q)$ and $(P, \alpha_2, \gamma_2, Q)$.

Finally, we can repeatedly abstract and then concretize without losing further detail:

Theorem B.3. *$\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \alpha$.*

Proof. We'll use the alternate notation for Definition B.1:

$$\begin{aligned} & \lambda p.p \sqsubseteq \gamma \circ \alpha && \text{Defn. B.1} \\ \Rightarrow \quad & \alpha \sqsubseteq \alpha \circ \gamma \circ \alpha \\ & \alpha \circ \gamma \sqsubseteq \lambda m.m && \text{Defn. B.1} \\ \Rightarrow \quad & \alpha \circ \gamma \circ \alpha \sqsubseteq \alpha \\ \Rightarrow \quad & \alpha \circ \gamma \circ \alpha = \alpha && \text{Anti-symmetry} \end{aligned}$$

The proof for $\gamma \circ \alpha \circ \gamma = \gamma$ is analogous. □

APPENDIX C.
ABSTRACT INTERPRETATION

Abstract interpretation formalizes abstraction of system semantics. These abstractions are guaranteed to be sound, i.e., the results of the abstraction are consistent with the results of the concrete system. Our work will apply this to models and properties of models, but this section investigates abstract interpretation as it is usually presented.

Abstract interpretation was originally developed as a techinque to unify program analysis techniques. In a general sense, programs nondeterministically transform inputs to outputs. More formally, any program p transforms an initial value $v_1 \in V_1$ (the inputs to the program) into a final value $v_2 \in V_2$ (the outputs and effects of the program). We write $p \vdash v_1 \rightsquigarrow v_2$ if it is possible for p to transform v_1 into v_2. $\rightsquigarrow$ is almost never a function in this case; nondeterminism can allow a program to transform one value into several final values.

1. INTERPRETATION, CONCRETELY

To facilitate analyisis, we will cast this in terms of deterministic transformation. We will choose *properties* of values such that programs deterministically transform them. If a program p transforms a property $l_1 \in L_1$ into a property $l_2 \in L_2$, we write $p \vdash l_1 \rhd l_2$. Since $\rhd$ is deterministic, i.e., for any l_1, there is only one l_2 the program may transform it to, we can define a *transformation function* $f_p(l_1) = l_2$. We say that f_p *interprets* p in terms of our chosen properties.

We connect properties to values via *correctness relations* $R_i : V_i \to L_i$. If a value v_i is described by a property l_i, we write $v_i \, R_i \, l_i$. These correctness relations capture the *semantics* of p with respect to the chosen properties.

The soundness of our interpretation $\rhd$ follows from these correctness relations. $\rhd$ is sound if, given $p \vdash v_1 \rightsquigarrow v_2$, $p \vdash l_1 \rhd l_2$, and $v_1 \, R \, l_1$, then $v_2 \, R \, l_2$ as well. In other words, if a program takes an input v_1 with properties l_1, transforms the value to v_2, and the interpretation $\rhd$ transforms l_1 to l_2, then v_2 must have properties l_2.

This relationship between value transformation, property transformation, and correctness is expressed in the diagram in Figure C.1.

$$\begin{array}{ccccc} p & \vdash & v_1 & \rightsquigarrow & v_2 \\ & & \| & & \| \\ & & R_1 & \Rightarrow & R_2 \\ & & \| & & \| \\ p & \vdash & l_1 & \rhd & l_2 \end{array}$$

Figure C.1. Relationship between transformation and a correctness relation

As an example, let us analyze `plusOne` : N → N defined by `plusOne`$(n) = n + 1$. Since this program is deterministic, we could let our properties domains also be N and $\triangleright$ = `plusOne`. The correctness relationships are both equality (=). This interpretation is known as the *concrete interpretation* of `plusOne`.

While this analysis is trivially correct, it is not very informative. Let's interpret `plusOne` in terms of parity. The properties domains become P = {EVEN, ODD} and the correctness relation is the function $parityOf$: N → P which assigns each natural number its associated parity (e.g., $parityOf(2)$ = EVEN). A correct interpretation $f_{\texttt{plusOne}}$: P → P obeys the property

$$f_{\texttt{plusOne}}(parityOf(n)) = parityOf(\texttt{plusOne}(n)), \forall n \in \mathrm{N}$$

and a case analysis on $parityOf(n)$ shows that $f_{\texttt{plusOne}}$ is the function that flips parity.

While both of these interpretations are sound and useful in their own way, there is no clear connection between them. In particular, we state that $parityOf$ is the correctness relation for interpretation in terms of P, but we must take this claim as an axiom. For this example, the correctness of $parityOf$ may seem obvious, but analyses of more complex programs and properties will not be so simple. What we need is a means of taking an "obviously correct" correctness relation (such as the trivial relation =) and extending this correctness to relations on more abstract property domains. To accomplish this task, we modify our formalism slightly so that we can apply a Galois connection.

2. LATTICES OF PROPERTIES

To take advantage of the soundness guarantees offered by a Galois connection, our properties domain must be a lattice. Since we can choose the abstract domain as we desire, we can require that $(L_i, \sqsubseteq)$ is a complete lattice. If necessary, we can construct L from an underlying set of properties D_i by $L_i = \mathcal{P}(D_i)$. Continuing our example, we would use the property domains $\mathcal{P}$(N) and $\mathcal{P}$(P), respectively.

We can constrain our correctness relations $R_i : V_i \to L_i$ based on two properties of L_i:

(i) If $v\, R_i\, l_1$ and $l_1 \sqsubseteq l_2$, then $v\, R_i\, l_2$.

(ii) If $v\, R_i\, l, \forall l \in L' \subseteq L_i$, then $v\, R_i \bigsqcap L'$.

(i) tells us that the smaller a property is, the more precisely (thus, better) it describes a value. (ii) tells us that, amongst a set of properties describing a value, there uniquely exists a property that best describes that value.

At this point, we can define *representation functions* $\beta_i : V_i \to L_i$ that maps values to the best property that describes them. If $\beta_1(v_1) \sqsubseteq l_1$, $p \vdash v_1 \rightsquigarrow v_2$, $p \vdash l_1 \rhd l_2$, then $\beta_2(v_2) \sqsubseteq l_2$. ($\beta_i$ can be defined in terms of R_i or vice-versa.) This relationship is described by the diagram in Figure C.2.

$$\begin{array}{ccccc} p & \vdash & v_1 & \rightsquigarrow & v_2 \\ & & \beta_1 \downarrow & \Rightarrow & \beta_2 \downarrow \\ & & \mathrel{\mid\sqcap} & & \mathrel{\mid\sqcap} \\ p & \vdash & l_1 & \rhd & l_2 \end{array}$$

Figure C.2. Relationship between program transformation and representation functions

For our concrete interpretation example, $\beta_{\mathrm{N}} : \mathrm{N} \to \mathcal{P}(\mathrm{n})$ is defined by $\beta_{\mathrm{N}}(n) = \{n\}$. The correctness relations change from equality to the element relation: $n \, R \, N \iff n \in N$.

The representation function for the parity analysis likewise maps numbers to the set containing the parity of that number: $\beta_{\mathrm{P}}(n) = \{parityOf(n)\}$. Any parity interpretation $f_{\texttt{plusOne}}$ follows the property

$$f_{\texttt{plusOne}}(\beta_{\mathrm{P}}(n)) \supseteq \beta_{\mathrm{P}}(\texttt{plusOne}(n)), \forall n \in \mathrm{N}$$

which admits two possible interpretations: the function that flips parity and the function that always returns P.

3. INTERPRETATION, ABSTRACTLY

We will use a Galois connection to extend a correctness relation from one properties domain to another. Let (L, α, γ, M) be a Galois connection between two lattices, and let $R : V \to L$ be the correctness relation for properties L. Our goal is to construct a correctness relation $S : V \to M$ based on R, α, and γ.

Define S by $v \, S \, m \iff v \, R \, \gamma(m)$. We must show that it obeys properties (i) and (ii) of correctness relations.

Proof of (i). Given $v\,S\,m_1$ and $m_1 \sqsubseteq m_2$,

$v\,R\,\gamma(m_1)$ and $\gamma(m_1) \sqsubseteq \gamma(m_2)$	Defn. of S; monotonicity of γ
$v\,R\,\gamma(m_2)$	Prop. (i) for R
$v\,S\,m_2$	

□

Proof of (ii). Given $v\,S\,m, \forall m \in M' \subseteq M$,

$v\,R\,\gamma(m), \forall m \in M'$	Defn. of S
$v\,R\,\bigsqcap\{\gamma(m) : m \in M'\}$	Prop. (ii) for R
$v\,R\,\gamma(\bigsqcap M')$	γ is completely multiplicative
$v\,S\,\bigsqcap M'$	

□

Therefore S is a correctness relation.

We can also show that $\beta_S = \alpha \circ \beta_R$ is the representation function for S. All these functions relate via the diagram in Figure C.3.

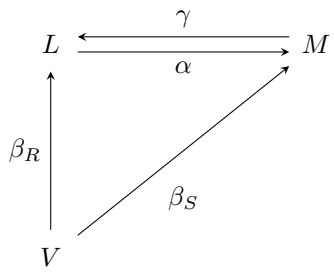


Figure C.3. Using a Galois connection to construct a representation function

To show that β_{P} is a representation function, we will show that it is generated by a Galois connection $(\mathcal{P}(\mathrm{N}), \alpha, \gamma, \mathcal{P}(\mathrm{P}))$.

Define $\alpha(N) = \{parityOf(n) : n \in N\}$ and $\gamma(P) = \{n \in \mathrm{N} : parityOf(n) \in P\}$. Then $(\alpha \circ \beta_{\mathrm{N}})(n) = \alpha(\{n\}) = \beta_{\mathrm{P}}(n)$, demonstrating that β_{P} is correct if β_{N} is.

REFERENCES

[1] Antsaklis, P. J., 'A brief introduction to the theory and applications of hybrid systems,' Proceedings of the IEEE, July 2000, **88**(7), pp. 879–887, ISSN 1558-2256, doi:10.1109/JPROC.2000.871299, conference Name: Proceedings of the IEEE.

[2] Giri, D. V., Hoad, R., and Sabath, F., *High-power electromagnetic effects on electronic systems*, Artech House, Boston London, 2020, ISBN 978-1-63081-588-2.

[3] Sun, M., Mohan, S., Sha, L., and Gunter, C., 'Addressing safety and security contradictions in cyber-physical systems,' 2009.

[4] Box, G. E. P. and Draper, N. R., *Empirical model-building and response surfaces*, Wiley series in probability and mathematical statistics. Applied probability and statistics, Wiley, New York, 1987, ISBN 978-0-471-81033-9.

[5] Vangheluwe, H. L. M., 'DEVS as a common denominator for multi-formalism hybrid systems modelling,' in 'CACSD. Conference Proceedings. IEEE International Symposium on Computer-Aided Control System Design (Cat. No.00TH8537),' September 2000 pp. 129–134, doi:10.1109/CACSD.2000.900199.

[6] Sabatini, A., Jarus, N., Maheshwari, P., and Sedigh, S., 'Software instrumentation for failure analysis of USB host controllers,' in '2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC),' May 2013 pp. 1109–1114, doi:10.1109/I2MTC.2013.6555586.

[7] Jarus, N., Sabatini, A., Maheshwari, P., and Sarvestani, S. S., 'Software-based monitoring and analysis of a USB host controller subject to electrostatic discharge,' in '2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST),' June 2020 pp. 1–7, doi:10.1109/RTEST49666.2020.9140117.

[8] Jarus, N., Sabatini, A., Maheshwari, P., and Sedigh Sarvestani, S., 'Detection, analysis, and prediction of the effects of electrostatic discharge on a USB host controller,' IEEE Transactions on Electromagnetic Compatibility, 2020, submitted.

[9] Jarus, N., Schott, J., Klingbeil, L., and Sedigh Sarvestani, S., 'Observation, analysis, modeling, and classification of USB host controller operation under electro-magnetic interference,' IEEE Transactions on Electromagnetic Compatibility, 2021, in preparation.

[10] Albasrawi, M. N., Jarus, N., Joshi, K. A., and Sarvestani, S. S., 'Analysis of reliability and resilience for smart grids,' in 'Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual,' IEEE, 2014 pp. 529–534, doi:10.1109/COMPSAC.2014.75.

[11] Jarus, N., Sarvestani, S. S., and Hurson, A. R., 'Models, metamodels, and model transformation for cyber-physical systems,' in 'Seventh International Green and Sustainable Computing Conference (IGSC),' IEEE, 2016 pp. 1–8.

[12] Jarus, N., Sarvestani, S. S., and Hurson, A. R., 'Facilitating model–based design and evaluation for sustainability,' in '2018 Ninth International Green and Sustainable Computing Conference (IGSC),' October 2018 pp. 1–2, doi:10.1109/IGCC.2018.8752119.

[13] Jarus, N., Sedigh Sarvestani, S., and Hurson, A. R., 'Formalizing cyber–physical system model transformation via abstract interpretation,' in '2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE),' January 2019 pp. 107–114, doi:10.1109/HASE.2019.00025.

[14] Jarus, N., Sarvestani, S. S., and Hurson, A. R., 'Towards refinement and generalization of reliability models based on component states,' in '2019 Resilience Week (RWS),' volume 1, November 2019 pp. 153–159, doi:10.1109/RWS47064.2019.8971824.

[15] Cousot, P. and Cousot, R., 'Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints,' in 'Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages,' ACM, 1977 pp. 238–252.

[16] Jarus, N., Woodard, M., Marashi, K., Sedigh Sarvestani, S., Lin, J., Faza, A. Z., and Maheshwari, P., 'Survey on modeling and design of cyber-physical systems,' ACM Trans. Cyber-Phys. Syst., 2020, under review.

[17] *White Paper 3, System Level ESD, Part II: Implementation of Effective ESD Robust Designs*, Industry Council on ESD Target Levels, March 2019.

[18] Maheshwari, P., Li, T., Lee, J., Seol, B., Sedigh, S., and Pommerenke, D., 'Software-based analysis of the effects of electrostatic discharge on embedded systems,' in 'IEEE Computer Software and Applications Conference (COMPSAC),' ISSN 0730-3157, July 2011 pp. 436 – 441, doi:10.1109/COMPSAC.2011.64.

[19] Kim, K. H. and Kim, Y., 'Systematic analysis methodology for mobile phone's electrostatic discharge soft failures,' IEEE Transactions on Electromagnetic Compatibility, Aug 2011, **53**(3), pp. 611–618, ISSN 0018-9375, doi:10.1109/TEMC.2011.2143719.

[20] Schwingshackl, T., Orr, B., Willemen, J., Simburger, W., Gossner, H., Bosch, W., and Pommerenke, D., 'Powered system-level conductive TLP probing method for ESD/EMI hard fail and soft fail threshold evaluation,' in '2013 35th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD),' ISSN 0739-5159, Sept 2013 pp. 1–8.

[21] Izadi, O. H., Hosseinbeig, A., Pommerenke, D., Shumiya, H., Maeshima, J., and Araki, K., 'Systematic analysis of ESD-induced soft-failures as a function of operating conditions,' in '2018 IEEE International Symposium on Electromagnetic Compatibility and 2018 IEEE Asia-Pacific Symposium on Electromagnetic Compatibility (EMC/APEMC),' May 2018 pp. 286–291, doi:10.1109/ISEMC.2018.8393784.

[22] Vora, S., Jiang, R., Vasudevan, S., and Rosenbaum, E., 'Application level investigation of system-level ESD-induced soft failures,' in '2016 38th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD),' September 2016 pp. 1–10, doi:10.1109/EOSESD.2016.7592565.

[23] Vora, S., Jiang, R., Vijayaraj, P. M., Feng, K., Xiu, Y., Vasudevan, S., and Rosenbaum, E., 'Hardware and software combined detection of system-level ESD-induced soft failures,' in '2018 40th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD),' September 2018 pp. 1–10, doi:10.23919/EOS/ESD.2018.8509783.

[24] Feng, K., Vora, S., Jiang, R., Rosenbaum, E., and Vasudevan, S., 'Guilty as charged: Computational reliability threats posed by electrostatic discharge-induced soft errors,' in '2019 Design, Automation Test in Europe Conference Exhibition (DATE),' March 2019 pp. 156–161, doi:10.23919/DATE.2019.8715149.

[25] Maghlakelidze, G., Wei, P., Huang, W., Gossner, H., and Pommerenke, D., 'Pin specific ESD soft failure characterization using a fully automated set-up,' in '2018 40th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD),' September 2018 pp. 1–9, doi: 10.23919/EOS/ESD.2018.8509693.

[26] Koch, S., Orr, B. J., Gossner, H., Gieser, H. A., and Maurer, L., 'Identification of soft failure mechanisms triggered by ESD stress on a powered USB 3.0 interface,' IEEE Transactions on Electromagnetic Compatibility, February 2019, **61**(1), pp. 20–28, ISSN 0018-9375, doi: 10.1109/TEMC.2018.2811645.

[27] Yuan, S.-Y., Wu, Y.-L., Perdriau, R., and Liao, S.-S., 'Detection of electromagnetic interference in microcontrollers using the instability of an embedded phase-lock loop,' IEEE Transactions on Electromagnetic Compatibility, April 2013, **55**(2), pp. 299–306, ISSN 0018-9375, doi: 10.1109/TEMC.2012.2218285.

[28] Liu, X., Izadi, O. H., Maghlakelidze, G., Pommerenke, M., and Pommerenke, D., 'A preliminary study of ESD effects on the process calls tree of a wireless router,' in '2018 IEEE Symposium on Electromagnetic Compatibility, Signal Integrity and Power Integrity (EMC, SI & PI),' July 2018 pp. 408–413, doi:10.1109/EMCSI.2018.8495346.

[29] Sabath, F., 'Classification of electromagnetic effects at system level,' in F. Sabath, D. Giri, F. Rachidi, and A. Kaelin, editors, 'Ultra-Wideband, Short Pulse Electromagnetics 9,' pp. 325–333, Springer, New York, NY, ISBN 978-0-387-77845-7, 2010, doi:10.1007/978-0-387-77845-7_38.

[30] Liang, T., Spadacini, G., Grassi, F., and Pignari, S. A., 'Worst-case scenarios of radiated-susceptibility effects in a multiport system subject to intentional electromagnetic interference,' IEEE Access, 2019, **7**, pp. 76500–76512, ISSN 2169-3536, doi:10.1109/ACCESS.2019.2921117.

[31] Guillette, D. S., Clarke, T. J., and Christodoulou, C., 'Intentional electromagnetic irradiation of a microcontroller,' in '2019 International Conference on Electromagnetics in Advanced Applications (ICEAA),' September 2019 pp. 1214–1218, doi:10.1109/ICEAA.2019.8879257.

[32] Burghardt, F. and Garbe, H., 'Effects of conducted interference on a microcontroller based on IEC 62132-4 and IEC 62215-3,' in '2020 International Symposium on Electromagnetic Compatibility — EMC EUROPE,' September 2020 pp. 1–5, doi:10.1109/EMCEUROPE48519.2020.9245782, iSSN: 2325-0364.

[33] Bai, J., Shi, Y., and Zhao, G., 'Research on electromagnetic interference of vehicle GPS navigation equipment,' in '2017 IEEE 5th International Symposium on Electromagnetic Compatibility (EMC-Beijing),' October 2017 pp. 1–5, doi:10.1109/EMC-B.2017.8260366.

[34] Camp, M., Schmitz, J., and Jung, M., 'Vulnerability and coupling behaviour of a TETRA communication system to electromagnetic fields,' in '2015 IEEE International Symposium on Electromagnetic Compatibility (EMC),' August 2015 pp. 344–349, doi:10.1109/ISEMC.2015.7256184.

[35] Lanzrath, M., Adami, C., Joerres, B., Lubkowski, G., Joester, M., Suhrke, M., and Pusch, T., 'HPEM vulnerability of smart grid substations coupling paths into typical SCADA devices,' in '2017 International Symposium on Electromagnetic Compatibility — EMC EUROPE,' September 2017 pp. 1–6, doi:10.1109/EMCEurope.2017.8094632.

[36] Mao, C., Canavero, F. G., Cui, Z., and Sun, D., 'System-level vulnerability assessment for EME: From fault tree analysis to bayesian networks—part II: Illustration to microcontroller system,' IEEE Transactions on Electromagnetic Compatibility, February 2016, **58**(1), pp. 188–196, ISSN 0018-9375, doi:10.1109/TEMC.2015.2502591.

[37] Parfenov, Y. V., Titov, B. A., Zdoukhov, L. N., and Radasky, W. A., 'About the assessment of electronic device immunity to high power electromagnetic pulses,' in '2015 7th Asia-Pacific Conference on Environmental Electromagnetics (CEEM),' November 2015 pp. 428–431, doi:10.1109/CEEM.2015.7368616.

[38] Armstrong, K. and Radasky, W. A., 'Extending the normal immunity tests to help prove functional safety,' in '2018 IEEE International Symposium on Electromagnetic Compatibility and 2018 IEEE Asia-Pacific Symposium on Electromagnetic Compatibility (EMC/APEMC),' May 2018 pp. 221–226, doi:10.1109/ISEMC.2018.8393770.

[39] Radasky, W. A., 'The role of electromagnetic shielding in dealing with the threat of Intentional Electromagnetic Interference (IEMI),' in '2015 International Conference on Electromagnetics in Advanced Applications (ICEAA),' September 2015 pp. 1145–1148, doi:10.1109/ICEAA.2015.7297298.

[40] Giri, D. V., Hoad, R., and Sabath, F., 'Implications of high-power electromagnetic (HPEM) environments on electronics,' IEEE Electromagnetic Compatibility Magazine, 2020, **9**(2), pp. 37–44, ISSN 2162-2272, doi:10.1109/MEMC.2020.9133238.

[41] Kasmi, C., Lopes-Esteves, J., Picard, N., Renard, M., Beillard, B., Martinod, E., Andrieu, J., and Lalande, M., 'Event logs generated by an operating system running on a COTS computer during IEMI exposure,' IEEE Transactions on Electromagnetic Compatibility, December 2014, **56**(6), pp. 1723–1726, ISSN 1558-187X, doi:10.1109/TEMC.2014.2357060.

[42] Lopes-Esteves, J., Cottais, E., and Kasmi, C., 'Software instrumentation of an unmanned aerial vehicle for HPEM effects detection,' in '2018 2nd URSI Atlantic Radio Science Meeting (AT-RASC),' May 2018 pp. 1–4, doi:10.23919/URSI-AT-RASC.2018.8471395.

[43] Kasmi, C., Lopes-Esteves, J., and Renard, M., 'Autonomous electromagnetic attacks detection considering a COTS computer as a multi-sensor system,' in '2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS),' August 2014 pp. 1–4, doi:10.1109/URSIGASS.2014.6929512.

[44] Liu, X., Maghlakelidze, G., Zhou, J., Izadi, O. H., Shen, L., Pommerenke, M., Ge, S. S., and Pommerenke, D., 'Detection of ESD-induced soft failures by analyzing Linux kernel function calls,' IEEE Transactions on Device and Materials Reliability, March 2020, **20**(1), pp. 128–135, ISSN 1558-2574, doi:10.1109/TDMR.2020.2965205.

[45] Watterson, C. and Heffernan, D., 'Runtime verification and monitoring of embedded systems,' Software, IET, 2007, **1**(5), pp. 172–179, ISSN 1751-8806, doi:10.1049/iet-sen:20060076.

[46] Delgado, N., Gates, A., and Roach, S., 'A taxonomy and catalog of runtime software-fault monitoring tools,' IEEE Transactions on Software Engineering, 2004, **30**(12), pp. 859–872, ISSN 0098-5589, doi:10.1109/TSE.2004.91.

[47] Choudhuri, S. and Givargis, T., 'FlashBox: a system for logging non-deterministic events in deployed embedded systems.' in S. Y. Shin and S. Ossowski, editors, 'Proceedings of the 2009 ACM symposium on Applied Computing (SAC),' ACM, ISBN 978-1-60558-166-8, 2009 pp. 1676–1682.

[48] Reinbacher, T., Horauer, M., and Steininger, A., 'A runtime verification unit for microcontrollers,' in 'System, Software, SoC and Silicon Debug Conference (S4D),' ISSN 2114-3684, September 2012 p. 1 – 6.

[49] Kitagawa, Y., Ishigooka, T., and Azumi, T., 'Anomaly prediction based on machine learning for memory-constrained devices,' IEICE Transactions on Information and Systems, September 2019, **E102.D**(9), pp. 1797–1807, ISSN 0916-8532, 1745-1361, doi:10.1587/transinf.2018EDP7339.

[50] Li, Y., Xue, W., Wu, T., Wang, H., Zhou, B., Aziz, S., and He, Y., 'Intrusion detection of cyber physical energy system based on multivariate ensemble classification,' Energy, December 2020, p. 119505, ISSN 0360-5442, doi:10.1016/j.energy.2020.119505.

[51] Zoppi, T., Ceccarelli, A., and Bondavalli, A., 'MADneSs: a Multi-layer Anomaly Detection Framework for Complex Dynamic Systems,' IEEE Transactions on Dependable and Secure Computing, 2019, pp. 1–1, ISSN 1545-5971, doi:10.1109/TDSC.2019.2908366.

[52] Alam, M., Bhattacharya, S., and Mukhopadhyay, D., 'Victims can be saviors: A machine learning-based detection for micro-architectural side-channel attacks,' ACM Journal on Emerging Technologies in Computing Systems, January 2021, **17**(2), pp. 14:1–14:31, ISSN 1550-4832, doi:10.1145/3439189.

[53] 'edwinrong/myregrw,' `https://github.com/edwinrong/myregrw`, 2009, accessed: 2020-01-20.

[54] Intel Corporation, 'Enhanced host controller interface specification for universal serial bus,' 2001.

[55] 'Downloads — FriendlyArm,' `http://dl.friendlyarm.com/mini2440`, 2012, accessed: 2020-01-20.

[56] *OpenHCI: Open host controller interface specification for USB*, Compaq, Microsoft, and National Semiconductor, October 2000, release: 1.0a.

[57] Izadi, O. H., Frazier, R. K., Altunyurt, N., Sedigh Sarvestani, S., Pommerenke, D., and Hwang, C., 'A new tunable damped sine-like waveform generator for IEMI applications,' in '2020 IEEE International Symposium on Electromagnetic Compatibility Signal/Power Integrity (EMCSI),' July 2020 pp. 282–286, doi:10.1109/EMCSI38923.2020.9191504.

[58] Weiß, C. H., *An introduction to discrete-valued time series*, John Wiley & Sons, Hoboken, NJ, 2017, ISBN 978-1-119-09698-6 978-1-119-09699-3.

[59] Lad, F., Sanfilippo, G., and Agrò, G., 'Extropy: Complementary dual of Entropy,' Statistical Science, February 2015, **30**(1), pp. 40–58, ISSN 0883-4237, 2168-8745, doi:10.1214/14-STS430.

[60] Weiß, C. H., 'Measures of dispersion and serial dependence in categorical time series,' Econometrics, June 2019, **7**(2), p. 17, doi:10.3390/econometrics7020017.

[61] Lee, E. A. and Sirjani, M., 'What good are models?' in K. Bae and P. C. Ölveczky, editors, 'Formal Aspects of Component Software,' Lecture Notes in Computer Science, Springer International Publishing, ISBN 978-3-030-02146-7, 2018 pp. 3–31.

[62] Zeigler, B. P., Muzy, A., and Kofman, E., *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations*, Academic Press, 2nd edition, 2000, ISBN 978-0-12-778455-1.

[63] Manna, Z. and Pnueli, A., 'On the faithfulness of formal models,' in G. Goos, J. Hartmanis, and A. Tarlecki, editors, 'Mathematical Foundations of Computer Science 1991,' volume 520, pp. 28–42, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-54345-9 978-3-540-47579-8, 1991, doi:10.1007/3-540-54345-7_46, series Title: Lecture Notes in Computer Science.

[64] Fatehah, M., Mezhuyev, V., and Al-Emran, M., 'A systematic review of metamodelling in software engineering,' in M. Al-Emran, K. Shaalan, and A. E. Hassanien, editors, 'Recent Advances in Intelligent Systems and Smart Applications,' Studies in Systems, Decision and Control, pp. 3–27, Springer International Publishing, Cham, ISBN 978-3-030-47411-9, 2021, doi:10.1007/978-3-030-47411-9_1.

[65] Montecchi, L., Lollini, P., and Bondavalli, A., 'A reusable modular toolchain for automated dependability evaluation,' in 'Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools,' ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013 pp. 298–303.

[66] Bonfiglio, V., Montecchi, L., Rossi, F., Lollini, P., Pataricza, A., and Bondavalli, A., 'Executable models to Support Automated Software FMEA,' IEEE, ISBN 978-1-4799-8111-3 978-1-4799-8110-6, January 2015 pp. 189–196, doi:10.1109/HASE.2015.36.

[67] De Lara, J. and Vangheluwe, H., 'AToMˆ3: A tool for multi-formalism and meta-modelling,' in 'FASE,' volume 2, Springer, 2002 pp. 174–188.

[68] Vangheluwe, H., De Lara, J., and Mosterman, P. J., 'An introduction to multi-paradigm modelling and simulation,' in 'Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems), Lisboa, Portugal,' 2002 pp. 9–20.

[69] Vittorini, V., Iacono, M., Mazzocca, N., and Franceschinis, G., 'The OsMoSys approach to multi-formalism modeling of systems,' Software and Systems Modeling, November 2003, **3**(1), pp. 68–81, ISSN 1619-1366, 1619-1374, doi:10.1007/s10270-003-0039-5.

[70] Franceschinis, G., Gribaudo, M., Iacono, M., Mazzocca, N., and Vittorini, V., 'Towards an object based multi-formalism multi-solution modeling approach,' Proceedings of the Second Workshop on Modelling of Objects, Components and Agents Aarhus (MOCA02), Denmark, 2002, **26**(27), pp. 47–65.

[71] Marco, G., Mazzocca, N., Francesco, M., and Vittorini, V., 'Multisolution of complex performability models in the OsMoSys/DrawNET framework,' in 'Second International Conference on the Quantitative Evaluation of Systems (QEST'05),' September 2005 pp. 85–94, doi:10.1109/QEST.2005.22.

[72] Iacono, M., Gribaudo, M., and Barbierato, E., 'Exploiting multiformalism models for testing and performance evaluation in SIMTHESys,' ACM, ISBN 978-1-936968-09-1, 2011 doi:10.4108/icst.valuetools.2011.245727.

[73] Iacono, M. and Gribaudo, M., 'Element based semantics in multi formalism performance models,' in '2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems,' August 2010 pp. 413–416, doi:10.1109/MASCOTS.2010.54, iSSN: 2375-0227.

[74] Barbierato, E., Gribaudo, M., and Iacono, M., 'Modeling hybrid systems in SIMTHESys,' Electronic Notes in Theoretical Computer Science, October 2016, **327**, pp. 5–25, ISSN 15710661, doi:10.1016/j.entcs.2016.09.021.

[75] Barbierato, E., Gribaudo, M., and Iacono, M., 'Simulating hybrid systems within SIMTHESys multi-formalism models,' in D. Fiems, M. Paolieri, and A. N. Platis, editors, 'Computer Performance Engineering,' Lecture Notes in Computer Science, Springer International Publishing, Cham, ISBN 978-3-319-46433-6, 2016 pp. 189–203, doi:10.1007/978-3-319-46433-6_13.

[76] Clark, G., Courtney, T., Daly, D., Deavours, D., Derisavi, S., Doyle, J., Sanders, W., and Webster, P., 'The Möbius modeling tool,' in '9th International Workshop on Petri Nets and Performance Models,' 2001 pp. 241–250, doi:10.1109/PNPM.2001.953373.

[77] Gaonkar, S., Keefe, K., Lamprecht, R., Rozier, E., Kemper, P., and Sanders, W. H., 'Performance and dependability modeling with Möbius,' SIGMETRICS Performance Evaluation Review, March 2009, **36**(4), pp. 16–21, ISSN 0163-5999, doi:10.1145/1530873.1530878.

[78] Deavours, D. and Sanders, W., 'Mobius: framework and atomic models,' in 'Proceedings 9th International Workshop on Petri Nets and Performance Models,' IEEE Comput. Soc, Aachen, Germany, ISBN 978-0-7695-1248-8, 2001 pp. 251–260, doi:10.1109/PNPM.2001.953374.

[79] Ptolemaeus, C., editor, *System design, modeling, and simulation: using Ptolemy II*, UC Berkeley EECS Dept, Berkeley, Calif, 1. ed., version 1.02 edition, 2014, ISBN 978-1-304-42106-7 978-1-304-42106-7.

[80] Goderis, A., Brooks, C., Altintas, I., Lee, E. A., and Goble, C., 'Heterogeneous composition of models of computation,' Future Generation Computer Systems, May 2009, **25**(5), pp. 552–560, ISSN 0167-739X, doi:10.1016/j.future.2008.06.014.

[81] Lee, E. A. and Zheng, H., 'Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems,' in 'Proceedings of the 7th ACM & IEEE international conference on Embedded software,' EMSOFT '07, Association for Computing Machinery, New York, NY, USA, ISBN 978-1-59593-825-1, September 2007 pp. 114–123, doi:10.1145/1289927.1289949.

[82] Larsen, P. G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., and Sadovykh, A., 'Integrated tool chain for model-based design of cyber-physical systems: The INTO-CPS project,' in '2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data),' IEEE, Vienna, Austria, ISBN 978-1-5090-1154-4, April 2016 pp. 1–6, doi:10.1109/CPSData.2016.7496424.

[83] Hasanagić, M., Tran-Jørgensen, P. W. V., Lausdahl, K., and Larsen, P. G., 'Formalising and validating the interface description in the FMI standard,' in J. Fitzgerald, C. Heitmeyer, S. Gnesi, and A. Philippou, editors, 'FM 2016: Formal Methods,' Lecture Notes in Computer Science, Springer International Publishing, Cham, ISBN 978-3-319-48989-6, 2016 pp. 344–351, doi:10.1007/978-3-319-48989-6_21.

[84] Foster, S., Thiele, B., Cavalcanti, A., and Woodcock, J., 'Towards a UTP semantics for Modelica,' in J. P. Bowen and H. Zhu, editors, 'Unifying Theories of Programming,' volume 10134, pp. 44–64, Springer International Publishing, Cham, ISBN 978-3-319-52227-2 978-3-319-52228-9, 2017, doi:10.1007/978-3-319-52228-9_3.

[85] Thule, C., Gomes, C., Deantoni, J., Larsen, P. G., Brauer, J., and Vangheluwe, H., 'Towards the verification of hybrid co-simulation algorithms,' in M. Mazzara, I. Ober, and G. Salaün, editors, 'Software Technologies: Applications and Foundations,' volume 11176, pp. 5–20, Springer International Publishing, Cham, ISBN 978-3-030-04770-2 978-3-030-04771-9, 2018, doi:10.1007/978-3-030-04771-9_1.

[86] Thule, C., 'Verifying the co-simulation orchestration engine for INTO-CPS,' in 'Proc. Of 21st International Symposium on Formal Methods,' CEUR-WS, Limassol, Cyprus, 2016 p. 6.

[87] Zeyda, F., Ouy, J., Foster, S., and Cavalcanti, A., 'Formalising cosimulation models,' in A. Cerone and M. Roveri, editors, 'Software Engineering and Formal Methods,' volume 10729, pp. 453–468, Springer International Publishing, Cham, ISBN 978-3-319-74780-4 978-3-319-74781-1, 2018, doi:10.1007/978-3-319-74781-1_31.

[88] Bhave, A., Krogh, B., Garlan, D., and Schmerl, B., 'Multi-domain modeling of CPS using architectural views,' in 'Proceedings of the First Analytic Virtual Integration of Cyber-Physical Systems Workshop,' IEEE Computer Society, San Diego, CA, USA, 2010 pp. 51–58.

[89] Bhave, A., Krogh, B. H., Garlan, D., and Schmerl, B., 'View consistency in architectures for cyber-physical systems,' in '2011 IEEE/ACM Second International Conference on Cyber-Physical Systems,' April 2011 pp. 151–160, doi:10.1109/ICCPS.2011.17.

[90] Rajhans, A., Bhave, A., Loos, S., Krogh, B. H., Platzer, A., and Garlan, D., 'Using parameters in architectural views to support heterogeneous design and verification,' in '2011 50th IEEE Conference on Decision and Control and European Control Conference,' December 2011 pp. 2705–2710, doi:10.1109/CDC.2011.6161408.

[91] Rajhans, A., Bhave, A., Ruchkin, I., Krogh, B. H., Garlan, D., Platzer, A., and Schmerl, B., 'Supporting heterogeneity in cyber-physical systems architectures,' IEEE Transactions on Automatic Control, December 2014, **59**(12), pp. 3178–3193, ISSN 1558-2523, doi:10.1109/TAC.2014.2351672.

[92] Morgan, C., *Programming from Specifications*, Prentice Hall international series in computer science, Prentice Hall, 1990, ISBN 0-13-726225-6.

[93] McIver, A. and Morgan, C., *Abstraction, Refinement, and Proof for Probabilistic Systems*, Springer monographs in computer science, Springer Science + Business Media Inc., 2005, ISBN 0-387-40115-6.

[94] Gulwani, S., Polozov, O., and Singh, R., 'Program synthesis,' Foundations and Trends in Programming Languages, July 2017, **4**(1-2), pp. 1–119, ISSN 2325-1107, 2325-1131, doi:10.1561/2500000010.

[95] Röttger, S. and Zschaler, S., 'Tool support for refinement of non-functional specifications,' Software & Systems Modeling, June 2007, **6**(2), pp. 185–204, ISSN 1619-1366, 1619-1374, doi:10.1007/s10270-006-0024-x.

[96] Abrial, J.-R., Su, W., and Zhu, H., 'Formalizing hybrid systems with Event-B,' in J. Derrick, J. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, editors, 'Abstract State Machines, Alloy, B, VDM, and Z,' Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, ISBN 978-3-642-30885-7, 2012 pp. 178–193, doi:10.1007/978-3-642-30885-7_13.

[97] Dupont, G., Aït-Ameur, Y., Pantel, M., and Singh, N. K., 'Formally verified architecture patterns of hybrid systems using proof and refinement with Event-B,' in A. Raschke, D. Méry, and F. Houdek, editors, 'Rigorous State-Based Methods,' Lecture Notes in Computer Science, Springer International Publishing, Cham, ISBN 978-3-030-48077-6, 2020 pp. 169–185, doi:10.1007/978-3-030-48077-6_12.

[98] Zhao, Y., Sanán, D., Zhang, F., and Liu, Y., 'Formal specification and analysis of partitioning operating systems by integrating ontology and refinement,' IEEE Transactions on Industrial Informatics, August 2016, **12**(4), pp. 1321–1331, ISSN 1941-0050, doi:10.1109/TII.2016.2569414.

[99] André, P., Attiogbé, C., and Lanoix, A., 'A tool-assisted method for the systematic construction of critical embedded systems using Event-B,' Computer Science and Information Systems, 2020, **17**(1), pp. 315–338, ISSN 1820-0214, 2406-1018, doi:10.2298/CSIS190501042A.

[100] Méry, D. and Singh, N. K., 'Formal specification of medical systems by proof-based refinement,' ACM Trans. Embed. Comput. Syst., January 2013, **12**(1), pp. 15:1–15:25, ISSN 1539-9087, doi:10.1145/2406336.2406351.

[101] Banach, R., 'Formal refinement and partitioning of a fuel pump system for small aircraft in hybrid Event-B,' in '2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE),' July 2016 pp. 65–72, doi:10.1109/TASE.2016.16.

[102] Caillaud, B., Delahaye, B., Larsen, K. G., Legay, A., Pedersen, M. L., and Wąsowski, A., 'Compositional design methodology with constraint Markov chains,' in '7th International Conference on the Quantitative Evaluation of Systems,' September 2010 pp. 123–132, doi:10.1109/QEST.2010.23.

[103] Delahaye, B., Larsen, K. G., Legay, A., Pedersen, M. L., and Wąsowski, A., 'Consistency and refinement for interval Markov chains,' Journal of Logic and Algebraic Programming, April 2012, **81**(3), pp. 209–226, ISSN 15678326, doi:10.1016/j.jlap.2011.10.003.

[104] Chadha, R. and Viswanathan, M., 'A counterexample-guided abstraction-refinement framework for Markov decision processes,' ACM Trans. on Computational Logic, November 2010, **12**(1), pp. 1:1–1:49, ISSN 1529-3785, doi:10.1145/1838552.1838553.

[105] Kattenbelt, M., Kwiatkowska, M., Norman, G., and Parker, D., 'A game-based abstraction-refinement framework for Markov decision processes,' Formal Methods in System Design, September 2010, **36**(3), pp. 246–280, ISSN 0925-9856, 1572-8102, doi:10.1007/s10703-010-0097-6.

[106] Mitsch, S., Quesel, J.-D., and Platzer, A., 'Refactoring, refinement, and reasoning,' in C. Jones, P. Pihlajasaari, and J. Sun, editors, 'FM 2014: Formal Methods,' Lecture Notes in Computer Science, Springer International Publishing, Cham, ISBN 978-3-319-06410-9, 2014 pp. 481–496, doi:10.1007/978-3-319-06410-9_33.

[107] Basile, D., Di Giandomenico, F., and Gnesi, S., 'A refinement approach to analyse critical cyber-physical systems,' in A. Cerone and M. Roveri, editors, 'Software Engineering and Formal Methods,' Lecture Notes in Computer Science, Springer International Publishing, ISBN 978-3-319-74781-1, 2018 pp. 267–283.

[108] Ishigooka, T., Saissi, H., Piper, T., Winter, S., and Suri, N., 'Safety verification utilizing model-based development for safety critical cyber-physical systems,' Journal of Information Processing, 2017, **25**, pp. 797–810, ISSN 1882-6652, doi:10.2197/ipsjjip.25.797.

[109] Mens, T. and Van Gorp, P., 'A Taxonomy of Model Transformation,' Electronic Notes in Theoretical Computer Science, March 2006, **152**, pp. 125–142, ISSN 15710661, doi:10.1016/j.entcs.2005.10.021.

[110] Gerber, A., Lawley, M., Raymond, K., Steel, J., and Wood, A., 'Transformation: The missing link of MDA,' in 'Graph Transformation,' pp. 90–105, Springer, 2002.

[111] Ameller, D., Franch, X., and Cabot, J., 'Dealing with Non-Functional Requirements in Model-Driven Development,' in 'Requirements Engineering Conference (RE), 2010 18th IEEE International,' September 2010 pp. 189–198, doi:10.1109/RE.2010.32.

[112] Röttger, S. and Zschaler, S., 'Model-Driven Development for Non-functional Properties: Refinement Through Model Transformation,' in T. Baar, A. Strohmeier, A. Moreira, and S. J. Mellor, editors, '«UML» 2004 — The Unified Modeling Language. Modeling Languages and Applications,' Number 3273 in Lecture Notes in Computer Science, pp. 275–289, Springer Berlin Heidelberg, ISBN 978-3-540-23307-7 978-3-540-30187-5, 2004.

[113] Rodrigues, G. N., Rosenblum, D. S., and Uchitel, S., 'Reliability Prediction in Model-Driven Development,' in L. Briand and C. Williams, editors, 'Model Driven Engineering Languages and Systems,' Number 3713 in Lecture Notes in Computer Science, pp. 339–354, Springer Berlin Heidelberg, ISBN 978-3-540-29010-0 978-3-540-32057-9, 2005.

[114] Kent, S. and Smith, R., 'The Bidirectional Mapping Problem,' Electronic Notes in Theoretical Computer Science, June 2003, **82**(7), pp. 151–165, ISSN 1571-0661, doi:10.1016/S1571-0661(04)80753-9.

[115] Berg, H. and Møller Pedersen, B., 'Type-Safe Symmetric Composition of Metamodels Using Templates,' in Ø. Haugen, R. Reed, and R. Gotzhein, editors, 'System Analysis and Modeling: Theory and Practice,' Number 7744 in Lecture Notes in Computer Science, pp. 160–178, Springer Berlin Heidelberg, ISBN 978-3-642-36756-4 978-3-642-36757-1, 2013.

[116] Feng, S. and Zhang, L., 'Model Transformation for Cyber Physical Systems,' in H. Y. Jeong, M. S. Obaidat, N. Y. Yen, and J. J. J. H. Park, editors, 'Advances in Computer Science and its Applications,' Number 279 in Lecture Notes in Electrical Engineering, pp. 83–87, Springer Berlin Heidelberg, ISBN 978-3-642-41673-6 978-3-642-41674-3, 2014.

[117] Lichen, L., 'Model Integration and Model Transformation Approach for Multi-Paradigm Cyber Physical System Development,' in H. Selvaraj, D. Zydek, and G. Chmaj, editors, 'Progress in Systems Engineering,' Number 330 in Advances in Intelligent Systems and Computing, pp. 629–635, Springer International Publishing, ISBN 978-3-319-08421-3 978-3-319-08422-0, 2015.

[118] Passarini, R., Buss Becker, L., and Farines, J.-M., 'The assisted transformation of models: Supporting cyber-physical systems design by extracting architectural aspects and operating modes from Simulink functional models,' in '2013 III Brazilian Symposium on Computing Systems Engineering (SBESC),' December 2013 pp. 47–52, doi:10.1109/SBESC.2013.25.

[119] Passarini, R. F., Farines, J.-M., Fernandes, J. M., and Becker, L. B., 'Cyber-physical systems design: transition from functional to architectural models,' Design Automation for Embedded Systems, December 2015, **19**(4), pp. 345–366, ISSN 1572-8080, doi:10.1007/s10617-015-9164-y.

[120] Kuo, W. and Zuo, M. J., *Optimal reliability modeling: principles and applications*, John Wiley & Sons, 2003.

[121] Jarus, N. and Sarvestani, S. S., 'Reliability modeling with the MIS technique,' 2015.

[122] Acker, B. V., Oakes, B. J., Moradi, M., Demeulenaere, P., and Denil, J., 'Validity frame concept as effort-cutting technique within the verification and validation of complex cyber-physical systems,' 2020, p. 10.

[123] Van Mierlo, S., Oakes, B. J., Van Acker, B., Eslampanah, R., Denil, J., and Vangheluwe, H., 'Exploring validity frames in practice,' in O. Babur, J. Denil, and B. Vogel-Heuser, editors, 'Systems Modelling and Management,' Communications in Computer and Information Science, Springer International Publishing, Cham, ISBN 978-3-030-58167-1, 2020 pp. 131–148, doi: 10.1007/978-3-030-58167-1_10.

[124] Karnopp, D., Margolis, D. L., and Rosenberg, R. C., *System dynamics: modeling and simulation of mechatronic systems*, Wiley, New York, 3rd ed edition, 2000, ISBN 978-0-471-33301-2.

[125] Bliudze, S., Furic, S., Sifakis, J., and Viel, A., 'Rigorous design of cyber-physical systems,' Software & Systems Modeling, June 2019, **18**(3), pp. 1613–1636, ISSN 1619-1374, doi:10.1007/s10270-017-0642-5.

[126] Trent, H. M., 'Isomorphisms between oriented linear graphs and lumped physical systems,' The Journal of the Acoustical Society of America, June 1955, **27**(3), p. 500, ISSN 0001-4966, doi:10.1121/1.1907949, publisher: Acoustical Society of AmericaASA.

[127] Davey, B. A. and Priestley, H. A., *Introduction to Lattices and Order*, Cambridge university press, 2002.

VITA

Natasha Amelia Jarus (on left, Figure V.1) received her Bachelor of Science in Computer Science, along with a minor in Mathematics, from the Missouri University of Science and Technology in December 2013. During her undergraduate career, she participated in undergraduate research with Dr. Sahra Sedigh Sarvestani on a project which she continued into her Ph.D program. She continued her education at the Missouri University of Science and Technology, receiving her Doctor of Philosophy in Computer Engineering in December 2021.

She worked as a Graduate Research Assistant from 2014 to 2021; during this time, she was a recipient of two Graduate Assistantships in Areas of National Need fellowships from the U.S. Department of Education. In addition, she worked as a Graduate Instructor from 2015 to 2018 for the Computer Engineering, Computer Science, and Mathematics departments. She created curricula for two courses: a special topics course on the Haskell programming language for junior and senior undergraduate students and the Data Structures Laboratory course for freshmen undergraduate students, which is now a required course for all students pursuing a bachelor's degree in Computer Science. She also co-authored the textbook for the Data Structures Laboratory course, *Tools for Programmers*. Furthermore, she led multiple workshops for prospective college students in cooperation

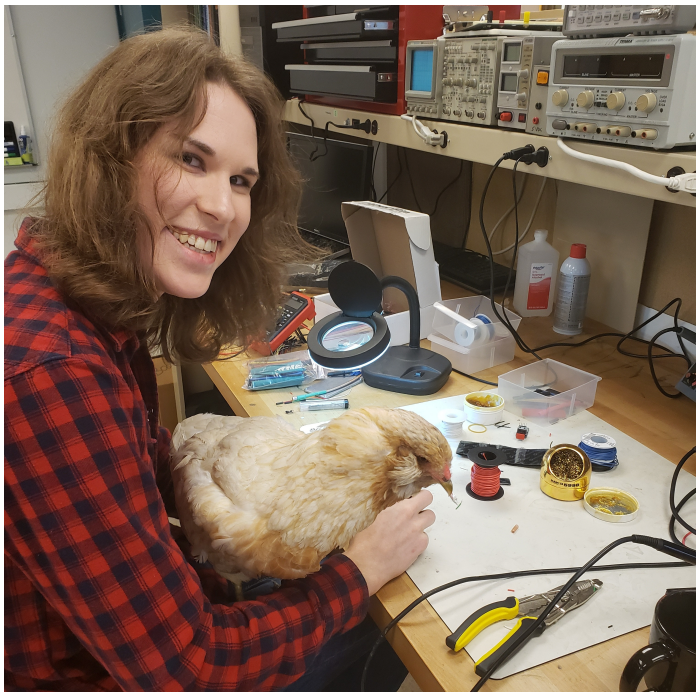


Figure V.1. Hengineerin'

with the Society for Women Engineers, Expanding Your Horizons, and the Kaleidoscope Discovery Center. During the 2020–2021 academic year, she served as the Electrical and Computer Engineering department representative to the Missouri S&T Council of Graduate Students.

In 2021, she became a full-time employee of ngrok at their Seattle, Washington office, developing software to enable programmers to introspect network traffic and to provide turn-key network service features such as authentication, load balancing, and link redundancy.

She was a member of the Institute of Electrical and Electronic Engineers, the IEEE Eta Kappa Nu honors society, the Association for Computing Machinery, the American Mathematical Society, the Association for Women in Mathematics, the Missouri S&T Intelligent Systems Center, and the Missouri S&T Center for Electromagnetic Compatibility.