Privacy and Utility in Dynamic Systems: Verification and Enforcement

by

Andrew Wintenberg

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan
2024

Doctoral Committee:

Professor Stéphane Lafortune, Co-Chair
Associate Professor Necmiye Ozay, Co-Chair
Assistant Professor Jean-Baptiste Jeannin
Associate Professor Vijay Subramanian
Assistant Professor Xinyu Wang

Andrew Wintenberg

awintenb@umich.edu

ORCID iD: 0000-0001-7522-5309

ACKNOWLEDGMENTS

First, I must express my gratitude for my advisors Necmiye Ozay and Stéphane Lafortune, whose guidance, expertise, and support defined not only the shape of my research in this dissertation but also the shape of academic interests and future scholarly pursuits. While Necmiye's instantaneous insight into problems that had puzzled me for days was sometimes intimidating, I have learned so much about problem-solving from her. Likewise, Stéphane's astonishing depth of knowledge and patience have always inspired me to be a better researcher. I aspire to emulate both of them in my future academic endeavors. I am also immensely grateful for my other committee members, Vijay Subramanian, Jean-Baptiste Jeannin, and Xinyu Wang, who with their diverse backgrounds provided much insightful feedback, greatly improving the quality of this work. I would also like to thank my committee for their availability and flexibility in these matters. My collaborators have also played an essential role in shaping this dissertation. Many thanks to Madeline Blischke, Sahar Mohajerani, Anne-Kathrin Schmuck, Ana Maria Mainhardt, Borzoo Bonakdarpour, Ana da Costa Oliveira, Tzu-Han Hsu, and Jiří Balun. I must also thank my current and former colleagues with whom I have had countless discussions about the many diverse topics subjects that they research. They have significantly broadened my knowledge while at the same time helped me focus my own ideas. A special thanks to Kwesi and Rômulo who taught me many of the tools and skills I continue to use in my research today. Thanks to Stéphanes's UMDES group including Shoma, Yuqing, and Ritsuka and Ozay group including Zexiang, Antoine, Glen, Liren, Mohamad, Daphna, Sunho, Zhe, Xingze, and Ruya among others. My sincerest appreciation to the staff and faculty of the EECS and Robotics departments, I have benefited greatly from their dedication to creating an educational environment that accommodates everyone. I would also like to acknowledge Remus Nicaoara and Jochen Denzler for fostering my love of math as an undergraduate student. I must also acknowledge the sources of funding which has enabled the research presented in this dissertation including the National Science Foundation (NSF), Cisco Research, and the Rackham Graduate School.

On a personal note, I would like to thank all of my family and friends whose support in all aspects of my life from emotional to academic was invaluable. To my mother and father, Kim and Alan, who provided a loving, patient environment to escape from stress whenever I needed it, and likewise a stern push to accomplish my goals when I needed that. And to my sister, Molly, whose perseverance through her own academic trials will always inspire me. To the friends I have made at Michigan, especially Max and Marion, our chats about everything from math and rocketry to pizza and street-racing were endless sources of entertainment. And to my friends abroad. Ana, our

friendship helped me through the end of my studies at Michigan and reignited my excitement for the future. A special thanks to my friends from home in Knoxville, Tanner, Michael, and Jon. Our late nights of games always gave me something to look forward to at the end of the week. And our discussions never ceased to yield valuable insights into my research. I am eternally grateful for our lifelong friendship. I could not have asked for a better set of friends. Lastly, I would like to thank everyone else that has helped me through this doctoral journey. I have been truly fortunate to receive so much support along the way.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

LIST OF APPENDICES

ABSTRACT

The development of cyber-physical systems, which integrate physical processes across cyber-networks, has had a profound impact on our society. Many of these systems, ranging from location-based services on our phones to devices on the Internet of Things in our homes, rely on the communication of sensitive information which may be vulnerable to eavesdropping. The physical harm that can result from leaking this information, like your current location or the occupancy of your home, is well documented. This has lead to strict privacy and security requirements which are often difficult to implement in practice. Motivated by the success of formal methods in developing provable guarantees for cyber-physical systems with strict safety requirements, in this dissertation we focus on the following problems: (1) How can we verify that a system maintains privacy? and (2) How can we enforce privacy upon a system while maintaining its original purpose or utility?

Verification is used to analyze vulnerabilities in existing systems and provide feedback to guide the design of new systems. Existing approaches to verification are limited by their ability to express complex privacy requirements and by their high computational burden. We propose a general framework expressing privacy as the formal information flow property of opacity. Over discrete state models (automata), we show how this framework unifies many of the existing notions of opacity studied in the area of discrete event systems. Within this framework, we develop approaches to verification based upon elementary automata constructions that exhibit competitive performance to existing techniques. Additionally, to overcome the inherent computational complexity of verification, we propose a relaxation of opacity that captures bounds on the ability of observers to reason about the system. We develop a corresponding verification approach based on an encoding to the Boolean satisfiability problem. These approaches are demonstrated on randomly generated systems alongside a novel server-load hiding problem.

Enforcement is used when a given system cannot be verified to maintain privacy, or when the design of systems by hand is impractical. We focus on the enforcement of privacy with obfuscation, altering communication on the network to shape the beliefs of observers. Over systems modeled by automata, obfuscation takes the form of edit functions which dynamically delete certain outputs and insert fictitious ones. We first show how to apply existing methods for designing such obfuscators within our general framework for opacity. To generalize the limited notion of utility guaranteed by these methods, we then propose a new approach to obfuscation which explicitly models the

information flow from the system to its authorized users as a distributed system. Finally, we integrate control into this approach over a variety of network architectures. These approaches use a blend of techniques from supervisory control for DES and reactive synthesis, and are demonstrated on a contact-tracing problem as well as a smart building access-control system.

CHAPTER 1

Introduction

1.1 Motivation

Privacy is an important concept discussed in areas as diverse as law, technology, and our everyday lives. It generally refers to the ability to control access to spaces both physical, like one's home, and abstract, like our information landscape. Although its origins are unclear, the need for privacy or a degree of separation from one's community is near universal, observed in both humans and non-human animals alike [72]. Regardless of whether hiding private information helps our society by promoting individuality or hurts it by facilitating illicit activity, it is necessary to understand and control how information is disclosed. Recent advancements in technology and the resulting reorganization of our society has rendered this problem much more difficult.

This is especially the case for cyber-physical systems (CPS), which integrate physical processes over cyber-networks. These systems, such as location-based services, building access-controls, and contact-tracing apps, have become ubiquitous in many areas of life. CPS collect information from their sensors, like user location data or home occupancy, which they communicate across networks to achieve their goals. This information is often sensitive and must be protected from unauthorized access and modification. Unfortunately, these networks, like the Internet or wireless communications, are vulnerable to eavesdropping and other cyber-attacks. Because of this, *information security* is of critical importance to the *correct* operation of CPS. A common model of information security is the CIA triad as depicted in Fig. 1.1, named for its three legs: confidentiality, integrity, and availability. The first leg of the triad is *confidentiality* which requires that unauthorized individuals do not have access to sensitive information. *Privacy* in context of the CPS literature is best understood as a form of confidentiality. While such a system can trivially be rendered confidential or private by removing communication of sensitive information, this information is often critical for system to achieve its goals or *utility*. This reflects the second leg of the triad, *availability*, which requires that authorized individuals do have access to information. The final leg of the triad is *integrity*, which requires that information remains accurate and unmodified by unauthorized individuals. The focus of this dissertation is the confidentiality and availability of

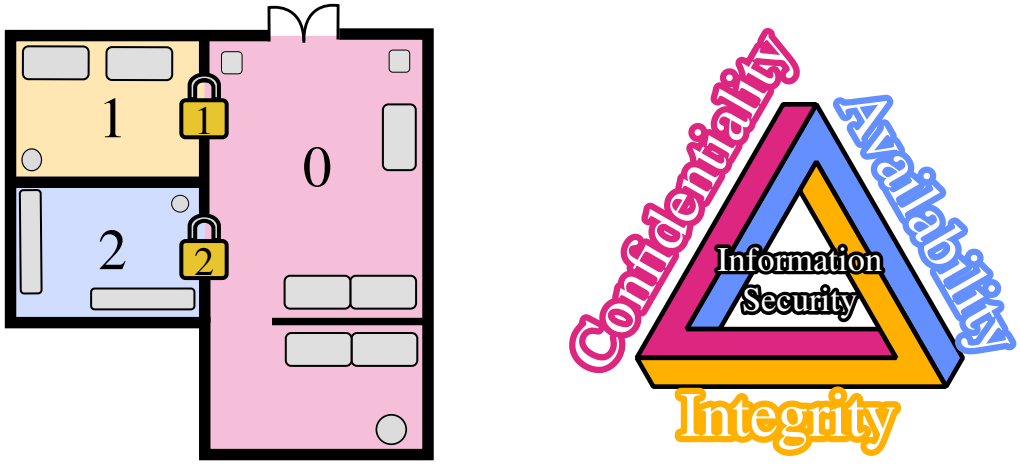

Figure 1.1: Knowledge of a building's layout like the one on the left can be used to track occupants' locations, violating their information security. Information security is commonly modeled by the CIA triad, visualized on the right.

CPS, understood as privacy and utility. The integrity of CPS is addressed by many existing works on intrusion detection [66] and resilience to attacks [59, 67].

Privacy in CPS presents a number of unique challenges. The first challenge results from the presence of dynamics describing how the system evolves over time. Aspects of these dynamics may be public knowledge or even learned by motivated observers. This knowledge can refine beliefs about the system's behavior beyond a sequence of observations alone. For example, the floor plan of a building restricts the mobility of its occupants, constraining the set of behaviors that can be realized. This information in conjunction with observations of the building may enable the tracking of its occupants' locations. The second challenge is the serious harm posed by violations of privacy. As by definition, CPS interact with the physical world, leaking information like personal location can have physical consequences. While some aspects of privacy in CPS are well-studied, there remains a need of more tools that address these challenges, providing provable guarantees of privacy while accounting for dynamics. Techniques developed in the area of formal methods, originating from the computer science community, verify that software meets its specifications and even design it [22]. The control systems community has adapted these techniques to CPS with broad success in application to systems with specifications like safety and liveness [1]. More recent works in this direction have proposed approaches to verification and enforcement of privacy and utility specifications for CPS. However, application of these approaches in practice is limited by (1) computational tractability and (2) the ability to express specifications. This dissertation presents a formal methodology addressing these limitations on the verification and enforcement of privacy and utility in cyber-physical systems.

1.2 Related Works

We now provide a brief overview of existing literature on privacy in CPS. We first discuss how privacy requirements are formally specified and verified, followed by a presentation of several approaches to privacy enforcement. The methodology of this dissertation is derived largely from the area of discrete event systems (DES), which is the focus of this discussion of related works. DES are event-driven systems whose behaviors are characterized by discrete transitions over time. Whereas analyzing the complex behavior of CPS concretely is often intractable, DES can provide high-level abstractions of CPS which are more computationally manageable.

1.2.1 Specification and Verification of Privacy and Utility

Privacy can be modeled by a variety of formal information flow and security properties. Many of these properties were originally studied in the computer science community and concerned the separation of high security clearance and low security clearance users. Such properties include, for example, non-interference [41], anonymity [79], and non-inference or non-deducibility [65]. Of particular interest in this dissertation is the property of *opacity* which captures the notion of plausible deniability: *a secret remains private if it can be plausibly denied.* Formally, the property holds if an observer cannot deduce sensitive information about a system's behavior from their observations and knowledge of the dynamics. Opacity (sometimes called opaqueness) was first introduced by Mazaré and Hughes [45, 64]. Opacity has been widely adopted by the DES community to specify privacy over transition systems [11] and later finite automata [3, 47, 83]. These works typically assume that the observer has knowledge of a system's dynamics but only partial observation of its behavior. Many distinct notions of opacity have been proposed corresponding to different types of secrets to be hidden. Language-based opacity (LBO) [57] describes secrets as sequences of the system's events, i.e., the secret behavior is specified by a sublanguage of the system. Alternatively, state-based notions describe secrets in terms of visits to so-called *secret states*. For example, current-state opacity (CSO) [83] and initial-state opacity (ISO) [84] require hiding whether the system currently or initially inhabits a secret state, respectively.

A common trend of this literature is the proposal of a new notion of opacity along with a corresponding verification technique tailored to it. The disconnected nature of these approaches makes it difficult to apply improvements on one verification technique to another. The first step towards a more unified theory was provided by Wu et al. [101], who showed that CSO, ISO, and other notions of opacity can be transformed into LBO. We continue this research direction by developing a general framework for specifying both state and language-based notions of opacity with automata, along with a uniform approach to verification. In particular, our proposed framework is applied to the more complex notions of K-step opacity and infinite-step opacity. Roughly, K-step

opacity requires that visits to secret states during the last K steps of a system are hidden, while infinite step opacity requires that visits to secret states are never revealed [86]. Verification of K-step opacity is difficult as one must consider *smoothing*, i.e. how observers use current observations to refine past beliefs. Many approaches to more efficiently verify K-step opacity have been developed, such as the one we propose, the two-way observer [108], and more recent techniques [6, 7]. While these approaches reduce the computational load of verification, they struggle with scaling to more complex systems. This is in part explained by the fundamental computational complexity of the opacity verification problem, which is PSPACE-complete [15]. This limitation can only be overcome by solving a different problem, such as the relaxed problem presented in this dissertation that is co-NP-complete.

1.2.2 Enforcement

When a system cannot be verified to preserve privacy, using the aforementioned opacity verification techniques for example, we can consider the problem of enforcing privacy upon it. Using opacity to model privacy, the earliest proposed enforcement mechanism over DES was supervisory control [28]. Supervisory control enforces opacity by restricting behaviors of the system which would reveal sensitive information. In doing so, it is important that supervisors do not restrict the system unnecessarily, reducing its utility. This is typically achieved by computing maximally-permissive solutions. While control provides a robust enforcement mechanism, it may be unfeasible to restrict certain aspects of a system's behaviors. For example, human components of a system like occupants moving around a building often cannot and should not be controlled. In this case, the observations of the system must be modified rather than its behavior. While a variety of existing mechanisms such as encryption use this approach, we consider privacy enforcement by modifying the outputs of the system to hide information while mimicking the original system. We generally refer to this approach as *obfuscation*.

The word obfuscation has been used to refer to a variety of concepts across different communities. In a sense, the meaning of the word is itself is obfuscated. For example, data obfuscation or masking alters sensitive data to prevent unauthorized users from accessing sensitive information for the purpose of anonymity or even hiding malicious data [5]. Code obfuscation modifies the source and implementation of software programs to be difficult to understand or reverse engineering in order to protect against piracy [25]. In common usage, obfuscation often refers deliberately confusing speech or coded messages such as doublespeak. The term has even taken on political connotations, referring to small acts of rebellion against the current age of surveillance as described in the book "Obfuscation" [10]. The common theme among these concepts is the propagation of ambiguous or misleading information in order to hide sensitive information. Obfuscation for opacity enforcement in DES can be realized with a variety of mechanisms including dynamically

masking the information gathered from the system's sensors [110], delaying observations of the system [31], and event-based cryptography [56]. In this dissertation, we consider obfuscation with edit functions that selectively delete and insert fictitious events to the stream of outputs from a system [100]. Privacy preserving edit functions can be synthesized using tools from supervisory control both in the case where the edit function is not publicly known [100, 103, 104] and when it is publicly known [48]. We consider the case where the edit function is not publicly known and adapt these tools to our general framework in order to enforce more complex notions like K-step opacity. An alternative approach to enforcing K-step opacity with edit functions was later presented in [60] which extracts edit functions as solutions to a two-player game constructed specifically for K-step opacity.

As with supervisory control to enforce privacy, one must be careful to maintain utility when obfuscating a system's outputs. For example, privacy can be trivially enforced by deleting all of a system's outputs; however, such a system would not be of any use to an outside observer. Previous work such as [103] model these requirements with *utility constraints*. For each state the actual system inhabits, the constraints describe states that an observer of obfuscated outputs should believe the system inhabits. In this way, utility constraints can express the requirement that an observer can infer information about the current state of the system. However, systems may have conflicting privacy and utility requirements, requiring that secret information is shared with one observer but not with an eavesdropper. As existing methods consider a single type of observer with uniform knowledge of the system model and obfuscation, there is no way to hide information from some but reveal it to others. To address this limitation of obfuscation, we propose a new framework which distinguishes observers by providing them with partial information about the obfuscator. Borrowing terms from cryptography, we can think of this information as a "key" mapping obfuscated observations to inferences about the system. After designing the obfuscator, these keys can be securely shared with recipients using a variety of methods. So although the system's outputs are obfuscated preventing general observers from deducing secrets, the system achieves utility as intended recipients can infer a specified amount of information about the behavior of the system. An example network implementing this approach is depicted in Fig. 1.2. A similar approach with an intended recipient was also proposed in [61].

1.2.3 Alternative Privacy Formulations

While we discuss privacy modeled by opacity over systems modeled by automata in this dissertation, many other notions of privacy have been formulated over a variety of models. For example, it has been shown that many observational properties can be expressed as hyperproperties specified by formal temporal logics like HyperLTL [113]. This result is critical to the development of the enforcement mechanisms we discuss later. Within DES, Petri nets provide a more expressive

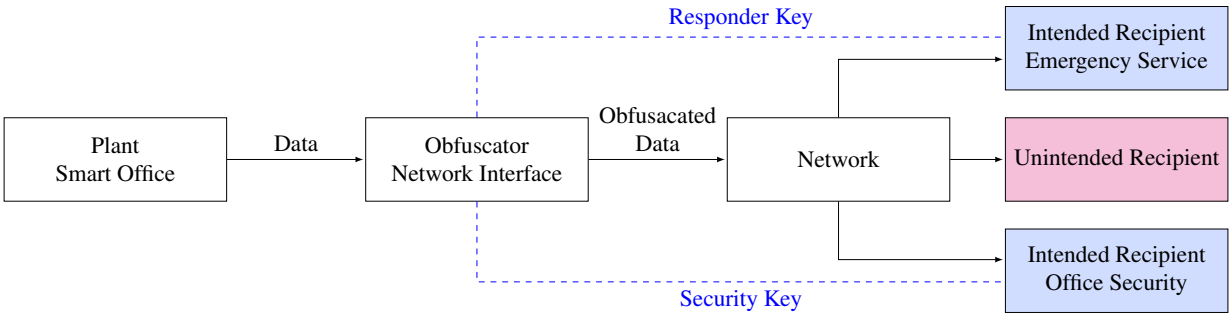

Figure 1.2: An example network architecture using obfuscation to enforce privacy on a smart office building. Sensitive information output by the system is obfuscated before being broadcast on an open network. Aspects of the original information can be recovered by emergency services and an office security provider using their keys to partially invert obfuscation. Unintended recipients without this key are and unaware of obfuscation believe they are directly observing outputs from the system that do not reveal any sensitive information.

modeling framework than finite automata, while still maintaining a compact representation. Several works have proposed opacity verification techniques for bounded Petri nets [11, 91]. We note that the most common notions of opacity over DES (current-state, initial-state, and K-step) concern the timing of visits to secret states. This prevalence of these notions reflects the time-sensitive nature of privacy in practice, i.e., *when did a secret occur? and when does an observer learn of it?* This motivates the use of timed models in studying opacity. Indeed, the discrete steps in K-step opacity are often used as a surrogate for the passing of time in reality. Unfortunately, while there is some work on the verification of opacity for timed automata [2], the general problem is undecidable [14].

While in this dissertation we consider "binary" notions of opacity, i.e., secret behavior is either revealed or it is not, quantifying the degree of privacy may be an important in practice. For example, there is a substantial body of work on stochastic notions of opacity, quantifying the probability that a secret is revealed [87] and how uncertain an observer is that they observed a secret [9]. Beyond discrete event system models, the base notion of opacity has been investigated over discrete-time linear systems [77]. Towards the goal of adapting opacity to general continuous state-space, the notion of approximate opacity was developed for nonlinear control systems [111]. Approximate opacity can be verified via abstraction to discrete systems with an appropriate simulation relation [114]. Control barrier functions have also been proposed as an efficient method for enforcing approximate opacity.

Beyond opacity, many notions of privacy over CPS have been investigated. Alongside encryption, which we will discuss later, the privacy of digital and analog communications has long been a focus of channel coding theory. An early example of this is the study of *wire-tap channels*, where a wiretapper (eavesdropper) observes communications on a network through an additional channel that has been degraded with noise. Encoding schemes or codes have been proposed which guarantee

information-theoretic privacy even when the eavesdropper is aware of this scheme [55, 106]. In contrast to the deniability described by opacity, techniques from information theory guarantee that no information about a given secret is revealed. These information theoretic techniques quickly evolved to solve more complex problems like the private and secure sharing of secrets [75] as well as the multi-party computation of functions [70, 107]. For example, the concept of differential privacy [29] was originally formulated for databases, requiring that aggregate information shared by the database does not reveal sensitive information about individuals. Differential privacy is often implemented for numerical data with the addition of statistical noise to observations of the system. Alternatively the "exponential mechanism" based on random sampling has been applied to enforce differential privacy for discrete systems like Markov chains [19]. In either case, the outputs produced are guaranteed to be *close* to the original data in some sense. As such, these mechanisms for differential privacy are conceptually similar to obfuscation, modifying the outputs of the system to ensure privacy while maintaining some utility. Importantly, differential privacy assumes the implementation of the enforcement mechanism is publicly known. Obfuscation is also similar to *steganography*, the practice of encoding messages within other types of data, such as images. Unlike differential privacy, steganography and the obfuscation we consider here are forms of *information hiding*, relying on the enforcement mechanism being unknown to hide the mere existence of private communications. The application of steganography to CPS is sometimes referred to as *network steganography* [62]. An information-theoretic discussion of information hiding including applications to watermarking, steganography, and data embedding can be found in this overview [73].

1.2.4 Encryption

One of the most common mechanisms for privacy enforcement is *encryption*. Encryption encodes information with unintelligible streams of data that can only be decoded with the use of a private key. The use of encryption is obvious to observers; it is often possible to determine the exact technique used from the encrypted data alone. The key feature of these techniques are their strong theoretical guarantees of privacy against eavesdroppers, even when the technique is known, so long as the key remains secret. While this type of privacy guarantee is more than sufficient for many systems, the conspicuous nature of encryption puts systems at risk of other forms of attack. Knowledge of the existence of privacy enforcement mechanisms may be enough to motivate more invasive and destructive attempts to access the system's sensitive information. The goal of our proposed obfuscation mechanism is to mimic behavior expected by eavesdroppers in order to avoid detection. In this way, obfuscation provides privacy guarantees which are orthogonal to encryption. Techniques like obfuscation that rely upon secrecy of the technique itself are historically subject to criticism as *security through obscurity*. But more recently, some of these techniques have gained

support as instances of *cyber deception*. Deception techniques which "mislead, confuse, hide critical assets from, or expose covertly tainted assets to the adversary", have been identified as an important part of the cyber resiliency engineering framework presented by the United States National Institute of Standards and Technology's [81].

There are also some settings in which typical encryption is difficult to implement. For example, the computational burden of many such techniques is too high for implementation on low-power devices, such as those commonly found on the Internet of Things. Additionally, the existing infrastructure of some systems may be brittle, failing in unexpected ways if the form of communications are altered. In the setting of static messages, this is the motivation of format-preserving encryption [8] which guarantees, for example, that location coordinates still look like coordinates encryption. Finally, while we typically discuss communication of digital data over cyber-networks, our framework for obfuscation permits the modeling of different types of networks where encryption cannot be implemented. For example, communication may refer to the observation of physical phenomena such as power usage within a smart grid. In this case, obfuscation can be implemented by controlling the power usage of loads or buffering power with a battery. The dynamics governing these processes are readily integrated within our formulation of obfuscation in a dynamic system.

1.3 Summary of Contributions

We now summarize the main contributions and organization of this dissertation. The relevant publications corresponding to the material of each chapter are indicated.

Chapter 2: Preliminaries

This chapter establishes the notation used by this dissertation and reviews the concepts necessary to understand it. In particular, it presents an overview of formal languages and automata, along with the common automata operations used in DES. In addition, an introduction to computational complexity theory and reactive synthesis is provided.

Chapter 3: Specification and Verification of General Notions of Opacity [97]

In this chapter, we present a general framework for specifying notions of opacity to capture the many complex privacy requirements on systems encountered in practice. The subsequent chapters on both verification and enforcement discuss opacity within this framework. We show how many existing state-based notions of opacity in DES can be expressed uniformly as a specific language inclusion after a certain transformation. In particular, we focus on K-step opacity, unifying the two existing notions in the literature and proposing two new notions as well.

Additionally, three approaches to verification which check this language inclusion are developed. These approaches are *complete*, always correctly identifying if a system is opaque given enough runtime. We evaluate these approaches in comparison with existing ones by developing theoretical

bounds on their runtimes and measuring their runtimes experimentally over randomly generated systems.

Chapter 4: Opacity Against Observers with a Bounded-Memory [98]

To overcome the inherent complexity (PSPACE-complete) of the opacity verification problem, we propose a meaningful relaxation of opacity in this chapter. This relaxation, called k-bounded memory opacity, explicitly models the deduction process of an observer as an automaton so that constraints on the observer's memory correspond to bounds on the state space of this automaton. We present reductions to the Boolean satisfiability problem (SAT) and from the k-bounded nonuniversality problem, showing that the problem of verifying this relaxed notion of opacity does not hold is NP-complete. Furthermore, the reduction to SATprovides an efficient method for verification which we evaluate over randomly generated examples. Finally, we extend this method using MAX-SAT to solve an optimal sensor placement problem in the context of a server load-hiding problem.

Chapter 5: Enforcement of Opacity with Obfuscation [95]

This chapter discusses what can be done when a system cannot be verified as opaque. We define the general problem of opacity enforcement with a specific focus on obfuscation, which enforces opacity by altering the observations made of the system. Realizing obfuscation with edit functions, we generalize the existing notions of private and public safety for current-state opacity to the framework of Chapter 3. This generalization permits the use of existing tools for designing edit functions to enforce K-step opacity. We demonstrate this approach on a contact-tracing system.

Chapter 6: Enforcement of Opacity and Utility with Distributed Synthesis [96]

This chapter addresses the limitations of existing obfuscation methods, like those employed in Chapter 5, to enforce utility. We propose a stronger notion of utility by explicitly modeling the intended recipient of the system's outputs as a process in a distributed system as depicted in Fig. 1.2. We then present a technique for the implementation of the obfuscator and intended recipient processes using distributed reactive synthesis. This implementation is secretly communicated to the recipient to act as a key to interpret obfuscated information, introducing information asymmetry necessary to enforce privacy while maintaining utility. The expanded capabilities of this approach is demonstrated on the contact-tracing system from Chapter 5.

Chapter 7: Integrating Control and Obfuscation [99]

The previous two chapters considered the enforcement of opacity with obfuscation alone, which is applicable when altering the underlying behaviors of the system is infeasible. In practice, privacy may require both the control of these underlying behaviors alongside obfuscation. In this chapter, we extend the solution approach of Chapter 6 to more general network architectures which integrate obfuscation and control. We explain this approach by solving three representative problems over such architectures representing (1) designing a local feedback controller, (2) designing a remote

feedback controller, (3) securing an existing remote feedback controller. These explanations feature solutions to these problems over a smart building access-control system.

Chapter 8: Conclusion

In this chapter, we summarize the contributions of this dissertation and present possible directions for future work.

CHAPTER 2

Preliminaries

This chapter presents the notation used by this dissertation as well as a review of the concepts necessary to understand it. In particular the topics of formal languages and automata, logic and complexity theory, and reactive synthesis are discussed. The set of real numbers is denoted by $\mathbb{R}$ and the set of natural numbers by $\mathbb{N} = \{0, 1, 2, \cdots\}$. The natural numbers and positive natural numbers bound by $n \in \mathbb{N}$ are denoted by $[k] = \{0, 1, \cdots, k\}$ and $[k]^+ = \{1, \cdots, k\}$, respectively. The set of Boolean values is denoted by $\mathbb{B} = \{0, 1\}$ with 0 denoting *false* and 1 denoting *true*. Given a function $f : X \to Y$ and a subset $Z \subseteq Y$, we denote the *preimage* by $f^{-1}(Z) = \{x \in X \mid f(x) \in Z\}$.

2.1 Formal Languages

Behaviors of a DES can be described as a formal language over the system's events. This section provides the basic definitions and constructions of languages and their representation as automata as used in this dissertation. A detailed discussion of these topics can be found in an introductory text on DES [13].

Languages and Finite Automata

Given a finite alphabet of symbols called *events* Σ, the sets of *finite*, *nonempty*, and *infinite sequences* over Σ are denoted by Σ^*, Σ^+, Σ^ω, respectively. Such sequences, called *strings*, represent the order that events occurred in a behavior of the system. Given a string $s \in \Sigma^*$ with length $n = |s|$, we write $s = s_0, \cdots, s_{n-1}$ where $s_i \in \Sigma$. By convention the set Σ^* contains the empty string ϵ which represents the behavior in which no events occurred. A $*$-*language* is a subset $L \subseteq \Sigma^*$ while an ω-*language* is a subset $M \subseteq \Sigma^\omega$. When clear from contex, both are referred to simply as languages. The set of finite *prefixes* of a finite or infinite string s is denoted by $\overline{s} \subseteq \Sigma^*$. Likewise, the set of finite prefixes of all strings in a language L is denoted by $\overline{L}, \overline{M} \subseteq \Sigma^*$. Given a finite string s and a finite or infinite string s', their *concatenation* is denoted by ss' or for emphasis $s \cdot s'$. A similar notation is used for languages. The set of all finite concatenations of a language $L \subseteq \Sigma^*$,

called the *Kleene-star*, is denoted by $L^* = \{\epsilon\} \cup L \cup L^2 \cup \cdots \subseteq \Sigma^*$. Correspondingly, the set of infinite concatenations of a language $L \subseteq \Sigma^*$ containing a nonempty string is denoted by $L^\omega \subseteq \Sigma^\omega$.

Automata provide a finite representation of a class of languages that are well-suite for computation. A finite *automaton* is a tuple $G = (Q, \Sigma, \delta, Q_0, Q_m)$ with a finite set of states Q, a finite set of events Σ, a transition relation $\delta \subseteq Q \times \Sigma \times Q$, initial states $Q_0 \subseteq Q$, and *marked* states $Q_m \subseteq Q$. A *subautomaton* of G is an automaton whose states and transitions are a subset of those in G. The *size* of an automaton G refers to its number of states $|Q|$. A run of G over a string $s = s_0 s_1 \cdots s_{n-1} \in \Sigma^*$ is a sequence of states $q_0, q_1, \cdots, q_n \in Q$ such that $q_0 \in Q_0$ and for all $j\,[n]$, $(q_j, s_j, q_{j+1}) \in \delta$. The *language generated by* G is the set $\mathcal{L}(G) \subseteq \Sigma^*$ containing the strings for which there exists a corresponding run over G. Given any set of states $Q' \subseteq Q$ the language of G marked by Q' is the set $\mathcal{L}_{Q'}(G) \subseteq \mathcal{L}(G)$ containing strings with a run ending in Q'. In particular, the *language marked by* G is the set $\mathcal{L}_m(G) = \mathcal{L}_{Q_m}(G)$. A language $L \subseteq \Sigma^*$ is said to be *regular* if it is marked by some finite automaton. We say that G is *deterministic* if its strings each correspond to a unique run, i.e., if both $|Q_0| \leq 1$ and for all $q \in Q$ and $s \in \Sigma$, there is at most one $q' \in Q$ such that $(q, s, q') \in \delta$. Deterministic finite automaton is abbreviated as DFA and nondeterministic finite automaton (including deterministic ones) as NFA. An example of how automata are depicted in this dissertation can be seen in Fig. 2.1.

We make similar definitions of automata over infinite strings by considering different marking or acceptance conditions. A *Büchi automaton* is a finite automaton $B = (Q, \Sigma, \delta, Q_0, Q_m)$ with the interpretation that an infinite run is accepted if it visits Q_m an infinite number of times. The *ω-language accepted by* B is the set $\mathcal{L}^\omega(B) \subseteq \Sigma^\omega$ of infinite strings with an accepting run over B. We say an ω-language $M \subseteq \Sigma^\omega$ is *ω-regular* if it is accepted by a Büchi automaton. When the context is clear, we drop the word *infinite* and prefix ω when referring to infinite strings and ω-languages for convenience.

Automata Constructions

Now we present a number of standard automata constructions corresponding to transformations and compositions of the systems behaviors as languages. In DES, it is common to model observation of a language over the events Σ with a subset of *observable events* $\Sigma_O \subseteq \Sigma$. In this case, observation is described by the *natural projection* $\mathrm{P} : \Sigma^* \to \Sigma_O^*$ which simply erases the occurrence of unobservable events. Formally this map is defined recursively by $\mathrm{P}(\epsilon) = \epsilon$ and for all $s \in \Sigma^*$ and $\sigma \in \Sigma$

$$\mathrm{P}(s\sigma) = \begin{cases} \mathrm{P}(s)\sigma & \sigma \in \Sigma_O, \\ \mathrm{P}(s) & \text{otherwise.} \end{cases} \tag{2.1}$$

Given an automaton G over events Σ, we can construct the *projected automaton* $\mathrm{P}(G)$ over observable events Σ_O which marks observations of behaviors in G through the projection

$$\mathcal{L}(\mathrm{P}(G)) = \mathrm{P}(\mathcal{L}(G)), \quad \mathcal{L}_m(\mathrm{P}(G)) = \mathrm{P}(\mathcal{L}_m(G))\,. \tag{2.2}$$

This is done by replacing events σ labeling transitions in G by their image $\mathrm{P}(\sigma)$ in $\mathrm{P}(G)$, removing ϵ transitions as necessary (for details see [13]). Note, $\mathrm{P}(G)$ may be nondeterministic even if G is deterministic. Because, they are often required in computation, there is a need to construct deterministic automaton that are language-equivalent to a given nondeterministic automaton. Given an NFA G, the *powerset construction* is a DFA $\det(G) = \left(2^Q, \Sigma, \delta_{\det}, \{Q_0\}, Q_{m,\det}\right)$ where

$$\delta_{\det} = \{(\mathbf{q}, \sigma, \mathbf{q}') \mid \forall q' \in \mathbf{q}'.\ \exists q \in \mathbf{q}.\ (q, \sigma, q') \in \delta\}\,, Q_{m,\det} \quad = \{\mathbf{q} \mid \mathbf{q} \cap Q_m \neq \emptyset\}\,. \tag{2.3}$$

This construction guarantees that

$$\mathcal{L}(\det(G)) = \mathcal{L}(G), \quad \mathcal{L}_m(\det(G)) = \mathcal{L}_m(G)\,. \tag{2.4}$$

The powerset construction is also commonly referred to as the observer construction in DES when combined with projection of observable events, or more generally as determinization. As an example of the need for deterministic automata, given a DFA G, the *complement automaton* $\mathrm{comp}(G)$ is constructed by adding transitions to a sink state labeled by events absent in the original. Formally, $\mathrm{comp}(G) = (Q \cup \{q_{\text{sink}}\}, \Sigma, \delta^c, Q_0, Q \setminus Q_m \cup \{q_{\text{sink}}\})$ where

$$\delta^c = \delta \cup \{(q, \sigma, q_{\text{sink}}) \mid \forall q' \in Q.\ (q, \sigma, q') \notin \delta\}\,. \tag{2.5}$$

This construction guarantees that

$$\mathcal{L}(\mathrm{comp}(G)) = \Sigma^*, \quad \mathcal{L}_m(\mathrm{comp}(G)) = \Sigma^* \setminus \mathcal{L}_m(G)\,. \tag{2.6}$$

In addition to projeciton, another operation which can produce an NFA from a DFA is the *reverse automaton*. The reverse of a finite string $s = s_0 \cdots s_n$ is denoted by $\mathrm{rev}(s) = s_n \cdots s_0$. We use the same notation for the reverse of a language. Given an NFA G, its reverse is an NFA $\mathrm{rev}(G) = (Q, \Sigma, \delta_{\text{rev}}, Q_m, Q_0)$, where the marked and initial states have been swapped and transitions reversed, i.e.,

$$\delta_{\text{rev}} = \{(q', \sigma, q) \mid (q, \sigma, q') \in \delta\}\,. \tag{2.7}$$

This construction is useful for transforming properties about the termination of strings to ones at initialization, or vice versa.

Given two automata $G_1 = (Q_1, \Sigma_1, \delta_1, Q_{0,1}, Q_{m,1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, Q_{0,2}, Q_{m,2})$, the *parallel composition* is the automaton $G_1 \mid\mid G_2 = \big(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{\mid\mid}, Q_{0,1} \times Q_{0,2}, Q_{m,1} \times Q_{m,2}\big)$, where

$$\begin{aligned}\delta_{\mid\mid} = \{((q_1, q_2), \sigma, (q_1', q_2')) \mid \forall j \in \{0,1\}.\ \big(\sigma \notin \Sigma_j \ \wedge\ q_j = q_j'\big) \vee \\ \big(\sigma \in \Sigma_j \ \wedge\ (q_j, \sigma, q_j') \in \delta_j\big)\}\,.\end{aligned} \tag{2.8}$$

This construction represents the parallel evolution of the systems' behaviors. Defining the natural projections $\mathrm{P}_j : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_j^*$ for $j \in \{1, 2\}$, the languages of the composition are given by

$$\mathcal{L}(G_1 \mid\mid G_2) = \mathrm{P}_1^{-1}(\mathcal{L}(G_1)) \cap \mathrm{P}_2^{-1}(\mathcal{L}(G_2)) \quad \mathcal{L}_m(G_1 \mid\mid G_2) = \mathrm{P}_1^{-1}(\mathcal{L}_m(G_1)) \cap \mathrm{P}_2^{-1}(\mathcal{L}_m(G_2)) \tag{2.9}$$

In the case where $\Sigma_1 = \Sigma_2$, we refer to this construction as the *product automaton* and write $G_1 \times G_2$. In terms of their languages,

$$\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2) \quad \mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2) \tag{2.10}$$

Both parallel composition and the product can be extended commutatively a finite collection of automata by induction.

Finite Transducers

More general than projection, observation of a system can be modeled as a relation between the behaviors of the system and their observations. Given input events Σ_I representing the system and output events Σ_O representing observations, a relation is a set $R \subseteq \Sigma_I^* \times \Sigma_O^*$. We can view such a relation as a nondeterministic mapping, i.e., given an input string $s_I \in \Sigma_I^*$, the corresponding set of outputs are $R(s_I) = \{s_O \in \Sigma_O^* \mid (s_I, s_O) \in R\}$. Similarly, we denote the set outputs over an input language $L \subseteq \Sigma_I^*$ by $R(L)$. The composition of two relations $R \subseteq \Sigma_I^* \times \Sigma_O^*$ and $R' = \subseteq \Sigma_U^* \times \Sigma_I^*$ as mappings is denoted by $R \circ R' = \{(s, s'') \mid (s, s') \in R' \ \wedge\ (s', s'') \in R''\}$. We will consider relations generated by the evolution of a dynamic system, in particular, modeled by an automaton. A *finite transducer* is an automaton $T = (Q_T, \Sigma, \delta_T, Q_{0,T}, Q_{m,T})$ over an alphabet $\Sigma = (\Sigma_I \cup \{\epsilon\}) \times (\Sigma_O \cup \{\epsilon\})$. By convention, we assume that all transducer states are marked which corresponds to observations generated sequentially rather than in blocks. Strings s over Σ can be decomposed into an input component $s_I \in \Sigma_I$ and output component $s_O \in \Sigma_O$. In this way, the transducer defines the relation $\mathcal{R}(T) \subseteq \Sigma_I^* \times \Sigma_O^*$ which relates all such inputs s_I to corresponding outputs s_O. Relations which can be represented by a finite transducer, we call *regular*. Such relations are also called rational in some sources [80]. Similar to the product construction for automata, a transducer T can be composed with an automaton G over a compatible event set to

Figure 2.1: An example automaton G (left) and transducer T (right). Marked states are denoted by double outlines. So the marked language of G is $\mathcal{L}_m(G) = \{\sigma\}$. In the transducer, the notation $i|o$ represents an input i resulting in output o. So the marked relation of T is $\mathcal{R}(T) = \{(i, o)\}$.

construct an automaton $T \otimes G$ such that $\mathcal{L}_m(T \otimes G) = R(L)$ where $R = \mathcal{R}(T)$ and $L = \mathcal{L}_m(G)$. Furthermore, the transducer T may be composed with another transducer T' with a compatible output event set to construct a transducer $T \otimes T'$ with $\mathcal{R}(T \otimes T') = \mathcal{R}(T) \circ \mathcal{R}(T')$. An example of how transducers are depicted in this dissertation can be seen in Fig. 2.1. Transducers and their constructions are used in Chapter 5 to represent edit functions.

2.2 Complexity Theory

In order to characterize the theoretical performance or *complexity* of algorithms solving the problems proposed in this dissertation, we now briefly review computational complexity theory. For more information, the reader may consult a standard reference such as [39].

Complexity Theory

A *decision problem* relates the inputs defining an instance of the problem to a corresponding yes or no answer as output. We are interested in the asymptotic complexity of algorithms solving the problem in terms of the resources they consume, e.g., time and space, in a given model of computation, i.e., Turing machines. Formally, the complexity of an algorithm represented as a function of the input size $f : \mathbb{N} \to \mathbb{N}$ is asymptotically bounded by the function $g : \mathbb{N} \to \mathbb{N}$ if for large x it holds that $f(x) \leq Mg(x)$ for some fixed $M \in \mathbb{N}$. In shorthand, this is written as $f(x) = \mathcal{O}(g(x))$. Complexity theory classifies decisions problems in terms of the complexity required by any algorithm to solve the problem. For example, the classes P and PSPACE denote problems that can be solved by a deterministic machine in polynomial time and space, respectively, i.e., the complexity $f(x)$ is $\mathcal{O}(p(x))$ for some polynomial p. Likewise, NP denotes problems solved by nondeterministic machines in polynomial time, i.e., problems for which there exist *certificates* proving the correctness of *yes* answers in polynomial time. Similarly, co-NP denotes the complement of NP, i.e., problems for which there exist certificates proving the correctness of *no* answers in polynomial time. Higher in the complexity hierarchy, the class n-EXP denotes problems which can be solved in $\mathcal{O}\left(\underbrace{(((2^2)^{\cdots})^2}_{n \text{ times}})^{p(x)}\right)$ time for some polynomial p. The class 1-EXPis often

called exponential time and denoted by EXP. While it is known that NP, coNP $\subseteq$ PSPACE $\subseteq$ EXP, it is unknown if these inclusions are strict. A problem in a complexity class $\mathbb{C}$ is said to be $\mathbb{C}$-*complete* if it is as hard as any other problem in $\mathbb{C}$, i.e., there is a polynomial time algorithm reducing any problem in $\mathbb{C}$ to it.

Boolean Satisfiability

The Boolean Satisfiability Problem (SAT), which asks if a given Boolean formula can be satisfied, is the standard example of an NP-complete problem. Formally, Boolean formulas over a set of variables V are specified by the following grammar

$$\varphi ::= v \mid \neg\varphi \mid \varphi \ \vee \ \varphi \,, \tag{2.11}$$

where $v \in V$ denoting a Boolean variable, $\neg$ denoting *not*, and $\vee$ denoting *or* have their standard interpretations. An assignment $A \subseteq V$, represented by the set of variables assigned to be true, satisfying the formula φ is denoted by $A \models \varphi$. We see that such a satisfying assignment is a certificate for SAT, proving its membership in NP. Conversely, the complement problem asking if a formula is unsatisfiable is by definition coNP-complete.

A common approach to analyzing decision problems is to reduce them to SAT or one of its extensions. For example, MaxSAT is an NP-complete optimization problem extending SAT with a notion of hard and soft constraints. It requires maximizing the sum of weights assigned to satisfied clauses of the formula. Another extension of SAT considers inputs given by formulas with quantification over Boolean variables. Quantified Boolean Formulas (QBF) are specified by the following grammar

$$\psi ::= \exists v.\psi \mid \forall v.\psi \mid \varphi \,, \tag{2.12}$$

where φ is a Boolean formula over V. Note QBFs are equally expressive as unquantified Boolean formulas. Indeed, existential and universal quantifiers over Boolean variables may be replaced by disjunction and conjunction, respectively. However, representation as QBF may be exponentially more compact. As a consequence of this reduction in input size, the problem of QBF satisfiability, simply denoted as QBF, is the standard example of a PSPACE-complete problem. While all of these problems (SAT, MaxSAT, and QBF) cannot be solved in polynomial time (unless P = NP), modern solvers are able to solve many instances in practice by taking advantage of advanced heuristics.

2.3 Reactive Synthesis

Given an existing system, we may be interested in verifying if it meets some requirements. This is the problem of verification or model checking [4]. When the system does not meet some

requirements, we may consider designing additional components changing or controlling its behavior. In this case we refer to the original system as the *plant*, and the additional component as a *controller*. The problem of *controller synthesis* is to design a controller which in composition with the plant satisfies the requirements. In DES, it is common to discuss controllers at the supervisory layer of a system, i.e., a *supervisor* dictates which behaviors in the plant are enabled or disabled. Alternatively in computer science, it is common to discuss the design of a system which reacts to its environment, e.g., human input, satisfying some requirements, a problem known as *reactive synthesis*. Viewing the plant as the environment and the controller as the system to design, the problems of supervisory control and reactive synthesis are closely related [30]. As such, we may leverage many techniques from reactive synthesis to enforce requirements on DES. This section presents the framework of reactive synthesis and distributed synthesis along with a few key results.

Modeling Distributed Systems

We consider systems composed of interconnected reactive processes. These processes generate outputs dynamically according to the inputs they have previously received. We model these inputs and outputs by assignments to a subset of Boolean variables V, e.g., for variables $V' \subseteq V$ the set of possible inputs is $2^{V'}$. A behavior of a process with variables $V_p \subseteq V$ evolving over time is described by an infinite string or *trace* $t \in \left(2^{V_p}\right)^\omega$. It is often convenient to discuss the restriction of a trace $t = t_0 t_1 \cdots$ to such a subset of variables $V' \subseteq V_p$ which we define as $t|_{V'} = (t_0 \cap V')(t_1 \cap V') \cdots \in \left(2^{V'}\right)^\omega$. We can make similar definitions for the restriction of finite traces and a set of traces. Similarly, it is convenient to discuss the lifting of a set of strings $M \subseteq \left(2^{V_p}\right)^\omega$ to a larger set of variables $V' \supseteq V_p$ which we define by $M|^{V'} = \{t \in \left(2^{V'}\right)^\omega \mid t|_{V_p} \in L\}$. We can make a similar definition for a set of finite traces. A deterministic implementation of a process is a *strategy* $s : \left(2^I\right)^+ \to 2^O$ mapping a history of inputs $i[0] \cdots i[n]$ to a single output $s(i[0] \cdots i[n])$. The set of traces associated with a strategy is defined by

$$\mathrm{Tr}(s) = \left\{s \in \left(2^{I \cup O}\right)^\omega \mid \forall n \in \mathbb{N}.\ s(s_0 \cdots s_n|_I) = s_n|_O\right\}. \tag{2.13}$$

Strategies can be represented with transducers that generate their set of traces or input-output relation. We say a strategy is *finite* when there exists such a transducer that is finite.

We now present a framework for distributed systems adapted from [38]. There, the environment is *unconstrained*, producing arbitrary sequences of outputs. As such, relations between the environment and the system must be encoded as a kind of *assume-guarantee* specification. As our focus is feedback control of the environment, we present a framework with constrained environments and make this encoding of constraints as specifications explicit.

The arrangement of processes and the interconnection of their inputs and outputs in a distributed

system is called an *architecture*. The environment is a special process, with the rest classified as either *white-box* with a fixed, known implementation or *black-box* otherwise. Formally, an *architecture* over a set of variables V is a tuple $A = (P, W, \mathbf{env}, E, O, H)$ with processes P, white-box processes $W \subseteq P$, environment process $\mathbf{env} \in P \setminus W$, and interconnections $E \subseteq P \times P$ forming a directed graph with nodes P and edges E. These edges are labeled by the set of *observable outputs* $O = \{O_e \subseteq V \mid e \in E\}$ communicated along the corresponding connection. Likewise the nodes are labeled by a set of *hidden outputs* $H = \{H_p \subseteq V \mid p \in P\}$ produced in the corresponding process but not communicated to others. For each process, we require that the observable outputs are disjoint from the hidden outputs. In addition, the set of both observable and hidden outputs should be mutually disjoint for all processes. For convenience we denote the set of outputs, both observable and hidden, for a process $p \in P$ by $O_p = \bigcup_{p' \in P} O_{(p,p')} \cup H_p$. Similarly we denote the set of inputs of a process $p \in P$ by $I_p = \bigcup_{p' \in P} O_{(p',p)}$. The set of all variables for process p is defined by $V_p = I_p \cup O_p \cup H_p$. When the context is clear, we may describe the interconnections using only the inputs I_p and outputs O_p of the processes $p \in P$.

In a given architecture A, white-box processes have a known *implementation* represented as a set of strategies $S_W = \{s_p : \left(2^{I_p}\right)^+ \to 2^{O_p} \mid p \in W\}$, whereas non-environment black-box processes will have an unknown implementation denoted by $S = \{s_p : \left(2^{I_p}\right)^+ \to 2^{O_p} \mid p \in B\}$. We say an implementation is finite if all of its strategies are finite. To model the non-determinism of the environment, we describe its behavior in relation to the other processes as a set of traces $M_{\text{env}} \subseteq \left(2^{V_{\text{env}}}\right)^\omega$. The implementations and environment interact with each other according to the architecture. The behavior of the overall or composed system is described by the set of traces consistent with each process's traces:

$$\mathrm{Tr}(A, S, S_W, M_{\text{env}}) = \bigcap_{\substack{p \in P \\ p \neq \text{env}}} \mathrm{Tr}(s_p)|^V \cap M_{\text{env}}|^V . \tag{2.14}$$

In this dissertation, we consider distributed systems without *delay*, i.e., output from one process is available to the next immediately. This is encoded in our definition of traces of a strategy in Equation (2.13). In general, feedback in systems without delay can result in inconsistent implementations, i.e., a process's input may depend on its outputs in a way that cannot be resolved statically, like in a ring oscillator. However, there is a simple transformation from this problem to the one with delay as discussed in [38, 76]. Solutions to the transformed problem represent exactly the consistent solutions to the problem with delay, i.e., those whose behavior can be expressed as a single monolithic implementation.

Distributed Synthesis

We consider the problem of implementing a distributed system to satisfy a given formal specification. We model these specifications by ω-regular languages $\varphi \subseteq \left(2^V\right)^\omega$ over the set of traces $\mathrm{Tr}(A, S, S_W, M_{\text{env}})$. The problem is then to find an implementation S of the black-box processes so that the composed system traces satisfy the specification.

Problem 2.1. *Given an architecture $A = (P, W, \mathbf{env}, E, O, H)$ over variables V with a white-box implementation S_W and environment traces given by the ω-regular language $M_{env} \subseteq \left(2^{V_{env}}\right)^\omega$, find an implementation S for A such that*

$$\mathrm{Tr}(A, S, S_W, M_{env}) \subseteq \varphi\,. \tag{2.15}$$

In general, this problem is known to be undecidable [38, 76]; however, there are architectures for which it is decidable. For example, the problem is decidable for *pipeline architectures* which consist of a directed, linear arrangement of processes [76]. More generally it was shown in [38] that the problem is decidable exactly for the architectures without so-called *information forks*. Essentially, an information fork occurs when two processes possess incomparable information about the environment's output history, i.e., both processes have knowledge that the other does not. In the absence of such forks, the processes may be ordered by the level of information they possess. An algorithm for distributed synthesis utilizing this fact is presented in Section 4 of [38], which we summarize with the following result.

Theorem 2.1 (Adapted from Theorem 4.12 [38]). *Let $A = (P, W, \mathbf{env}, E, O, H)$ be an architecture over V without information forks such that $I_{env} = \emptyset$ and $M_{env} = \left(2^{V_{env}}\right)^\omega$. Let S_W be a finite white-box implementation and $\varphi \subseteq \left(2^V\right)^\omega$ be an ω-regular language. Then the distributed synthesis problem for A, S_W, M_{env}, φ is decidable.*

Furthermore, if a solution implementation S exists, then the synthesis algorithm is guaranteed to find a finite solution. The complexity of the algorithm is related to the number of information levels of the black-box processes. If there are n information levels, the algorithm is n-exponential in the size of the automata representing the specification and implementation of the white-box properties. For example, in a pipeline architecture with 2 black-box processes and a specification represented with an automaton of size m, the complexity would be $O(2^{2^m})$. Distributed reactive synthesis is used in Chapter 6 and Chapter 7 to design edit functions and controls which enforce privacy and utility.

Specification Logics

Temporal logics provide a convenient way to represent formal specifications. Linear temporal logic (LTL) describes sets of traces over a given set of Boolean variables V (also called propositions). LTL is defined with the following syntax which extends propositional logic temporal operators:

$$\varphi ::= v \mid \neg\varphi \mid \varphi \ \vee \ \varphi \mid \ \mathrm{X}\,\varphi \mid \varphi \ \mathrm{U} \ \varphi\,, \tag{2.16}$$

where $v \in V$. Here, X and U denote the "next" and "until" operators. From these operators, we can define the always operator G, the eventually operator F, and the weak until operator W. For a precise definition of the semantics of LTL, the reader may consult [4]. There it is also shown that LTL formulas express exactly the ω-regular languages (after a small transformation). In Chapter 6 and Chapter 7 where we utilize tools from distributed reactive synthesis, it is convenient to specify utility properties with LTL, such as the system avoids unsafe states or eventually returns to desirable ones. In this setting, we may also use LTL to specify the secret behavior to be hidden in our definition of privacy. For example, if the utility requirement of the system is known, the secret behavior of the system might be specified as traces satisfying the utility requirement which visit some secret state.

While LTL is useful for specifying trace properties, requirements like information flow cannot be expressed directly as trace properties. These requirements concern relations between traces of a system, which are described by *hyperproperties* [24]. Hyperproperties are satisfied by systems, i.e., sets of traces and can be specified with temporal logics like HyperLTL [23]. HyperLTL is an extension of LTL with explicit trace quantifiers given by the following syntax:

$$\psi ::= \exists t.\psi \mid \forall t.\psi \mid \varphi\,, \tag{2.17}$$

where φ is an LTL formula over the variables $v[t]$ where $v \in V$ and t is a trace variable. The reader may consult [23] for the precise definition of HyperLTL semantics. The information flow of a distributed reactive system can be specified with HyperLTL [35]. We apply this result in Chapter 6 and Chapter 7 to solve the distributed reactive synthesis problem as a HyperLTL synthesis problem.

CHAPTER 3

Specification and Verification of General Notions of Opacity

3.1 Introduction

As stated in the introduction, the transmission of information across networks possesses an inherent risk of revealing private information to an outside observer called the *intruder*, potentially with malicious intent. Formal modeling of information flow properties has been proposed as a way to understand and manage these risks in networked dynamic systems. We consider privacy modeled by the property of opacity which captures the notion of "plausible deniability": opacity holds if an intruder cannot deduce sensitive information from their observations of a system's behavior. We will discuss the many notions of opacity developed for discrete event systems, including language-based opacity [57], current-state opacity [83], initial-state opacity [84], and the related notions of K-step and infinite step opacity [85, 86]. In addition to the type of private information, the capabilities of the intruder are also integral to notions of opacity. Although a variety of notions of opacity have been proposed, they may not directly capture the desired notion of privacy or security in a given networked system. One approach to analyzing specific notions of opacity is to transform them into existing notions where existing methods can be applied. While some transformations between the various forms of opacity over automata have been studied (for example between current-state, initial-state, language-based [101]), it is unclear if other notions like K-step opacity are comparable or how to handle new notions. The first contribution of this chapter is to develop a systematic approach for specifying and analyzing various notions of opacity. This is accomplished with a general definition of opacity extending the notion developed for transition systems [12]. We use this framework to model language-based opacity over automata and present several methods for verification thereof. Then we develop a general transformation between state-based and language-based notions of opacity. Using this, state-based notions of opacity can be described by constructing automata to specify secret behavior and verified using language-based methods. This approach is first demonstrated on the simple notions of current-state and initial-state opacity. The resulting verification methods resemble the existing standard approaches for verification of these forms of opacity.

The second contribution of this chapter is to apply the proposed framework and verification methods to the less well-understood notions of K-step and infinite step opacity. Whereas current-state opacity only considers an intruder's current state estimate, K-step and infinite step opacity may involve the intruder *smoothing* their estimates, i.e., improving estimates of the past with current information. While it may appear that these notions are incomparable [109], we provide a unified view of two prominent existing notions of K-step opacity along with two new ones that emerge using our framework. These notions are then transformed into language-based and hence current-state opacity. Furthermore, the resulting language-based verification methods offers considerable advantages over existing methods. We demonstrate this both formally and with numerical examples.

3.2 Problem Formulation

In this section we formalize the notion of opacity as a kind of *plausible deniability* in a system. We take a behavioral approach [94], describing systems in terms of abstract behaviors and observations . Formally, we denote the set of possible behaviors of the system as $\mathcal{B}$. For example, $\mathcal{B}$ may be the set of solutions to a differential equation modeling a continuous-time system or the language of an automaton modeling a discrete event system. Let $\mathcal{O}$ be the space of observations made of these behaviors. Then relation between behaviors and their corresponding observations is denoted by $\Theta \subseteq \mathcal{B} \times \mathcal{O}$. We identify the *system* with the tuple of behaviors and observation relation $\Delta = (\mathcal{B}, \Theta)$.

While many definitions of opacity assume that the observer knows the system exactly, more generally, we consider that the observer possesses an arbitrary model of the system called the *nominal system*. This nominal system also consists of behaviors $\hat{\mathcal{B}}$ with observations made in the same space $\mathcal{O}$ according to the relation $\hat{\Theta} \subseteq \hat{\mathcal{B}} \times \mathcal{O}$. We denote it as $\hat{\Delta} = (\hat{\mathcal{B}}, \hat{\Theta})$. If the observer is uncertain about the system, their model may be an overapproximation of the actual system, i.e., $\mathcal{B} \subseteq \hat{\mathcal{B}}$ and $\Theta \subseteq \hat{\Theta}$. Alternatively, the observer may possess incorrect beliefs about the system, for example if an enforcement mechanism is implemented without their knowledge. To emphasize the the difference with nominal system $\hat{\Delta}$, we refer to Δ as the *real system*.

We define opacity as the plausible deniability of certain behaviors to the observer. Rather than discuss the set of behaviors that must be denied, i.e., *secret behavior*, we use the complement of this set which contains behavior providing deniability, i.e., *nonsecret behavior*. As the observer reasons about observations with respect to the nominal system, we consider a specification of nonsecret behaviors φ_{NS} over the space of nominal behaviors $\hat{\mathcal{B}}$. In words, a system is opaque if after every real behavior, the observer believes a behavior in φ_{NS} could have occurred. Formally,

Definition 3.1. *We say that the system $\Delta = (\mathcal{B}, \Theta)$ is* opaque *for the nominal system $\hat{\Delta} = (\hat{\mathcal{B}}, \hat{\Theta})$*

and nonsecret behavior φ_{NS} if

$$\Theta(\mathcal{B}) \subseteq \hat{\Theta}(\hat{\mathcal{B}} \cap \varphi_{NS}) \,. \tag{3.1}$$

This definition corresponds to the notion of opacity presented in [12] where the behavior is given by runs of a transition system and the nominal system is equal to the real system. Here we say the system is opaque with respect to the secret information in the sense that the eavesdropper cannot *see through* the system in order to infer the secret. The observations not in $\hat{\Theta}(\hat{\mathcal{B}} \cap \varphi_{\text{NS}})$ and the behaviors that originate them are called *violating* for specification φ_{NS}. Specific notions of opacity correspond to different specifications of nonsecret behavior and nominal models possessed by the observer.

Joint & Separate Opacity

More complex notions of privacy can involve multiple, possibly overlapping specifications of nonsecret behaviors. Given a set of specifications $\{\varphi_{\text{NS},i}\}_{i\in I}$ indexed by the set I, we consider two forms of opacity over these specifications.

Definition 3.2. *We say that Δ is* jointly opaque *for $\hat{\Delta}$ and $\{\varphi_{NS,i}\}_{i\in I}$ if Δ is opaque for nominal $\hat{\Delta}$ and nonsecret*

$$\varphi_{NS} = \left(\bigcap_{i\in I} \varphi_{NS,i} \right) \,. \tag{3.2}$$

Joint opacity considers all secrets uniformly. It requires that an observation can be explained by a single behavior that is in every nonsecret specification.

Definition 3.3. *We say that Δ is* separately opaque *for $\hat{\Delta}$ and $\{\varphi_{NS,i}\}_{i\in I}$ if for all $i \in I$, Δ is opaque for nominal $\hat{\Delta}$ and nonsecret $\varphi_{NS,i}$. Separate opacity considers all specifications individually. It requires for each specification that an observation can be explained by behavior that is nonsecret in that specification, but possibly secret in another specification.*

Remark 3.1. *From these definitions, we can immediately deduce the following statements.*

1. *When $|I| = 1$, joint and separate opacity reduce to opacity as in Definition 3.1.*
2. *Joint opacity implies separate opacity.*
3. *For $I' \subseteq I$, joint (separate) opacity of $\{\varphi_{NS,i}\}_{i\in I}$ implies joint (separate) opacity of $\{\varphi_{NS,i}\}_{i\in I'}$, respectively.*

3.3 Opacity in DES and Verification Approaches

We now present how various notions of opacity developed for DES over automata can be expressed and verified in the framework. *Language-based opacity* (LBO) refers to the setting where behaviors of the system and nonsecret classes are described as languages represented by automata. More generally when these behaviors are also described in terms of the automata's states, we refer to this as *state-based opacity* (SBO). For example, many state-based notions involve visits to states designated as secret. It is known that some state-based notions of opacity like current-state opacity (CSO) and initial-state opacity (ISO) can be efficiently transformed into language-based opacity [101]. In this section, we begin by presenting several verification methods for LBO and then describe how SBO in general can be transformed into LBO, in particular reproducing the results for CSO and ISO.

Verification of Language-Based Opacity

We now consider the setting where the behavior $\mathcal{B}$ of the system is described by a regular language $L \subseteq \Sigma^*$ marked by an automaton G, i.e., $\mathcal{B} = L = \mathcal{L}_m(G)$. Without loss of generality, we assume that observations are given by a subset of observable events $\Sigma_{\text{obs}} \subseteq \Sigma$. We additionally assume the observation relation is given by a regular relation $\Theta \subseteq \Sigma^* \times \Sigma_{\text{obs}}^*$ generated by a transducer T. Likewise, we assume the nominal system consists of a regular language $\hat{L} = \mathcal{L}_m(\hat{G})$ and a regular relation $\hat{\Theta} = \mathcal{R}(\hat{T})$. Finally, the nonsecret class of behaviors is also described by a regular language $\varphi_{\text{NS}} \subseteq \Sigma^*$ marked by an automaton H_{NS}. We gather these descriptions into the following definition.

Definition 3.4. *Given L, Θ, $\hat{L}$, $\hat{\Theta}$, and φ_{NS} are all regular languages and relations, we refer to the opacity of $\Delta = (L, \Theta)$ to $\hat{\Delta} = (\hat{L}, \hat{\Theta})$ and φ_{NS} as* language-based opacity.

While the observations relations in general may be dynamic, e.g., dynamic masks [110], for simplicity we will only consider relations that are *static* or memoryless in this chapter. The proposed verification approaches are easily extended to dynamic observation relations described by transducers.

Definition 3.5. *A* static mask *over Σ is a relation $\Theta \subseteq \Sigma^* \times \Sigma_{obs}^*$ that satisfies*

1. $\Theta(\epsilon) = \{\epsilon\}$,
2. $\forall s \in \Sigma^*.\, \forall \sigma \in \Sigma.\, \Theta(s\sigma) = \Theta(s)\Theta(\sigma)$ *with* $\Theta(\sigma) \in \Sigma \cup \{\epsilon\}$.

Any relation $\Theta \subseteq \Sigma \times (\Sigma_{obs} \cup \{\epsilon\})$ can be uniquely extended inductively into a static mask.

We note that the natural projection $\mathrm{P} : \Sigma^* \to \Sigma_{\text{obs}}$ defines a static mask via its graph $\Theta = \{(s, \mathrm{P}(s)) \mid s \in \Sigma^*\}$. If we assume that the observer knows the system dynamics, i.e., the nominal model is equal to the system model, and that observation is given by the natural projection, then Definition 3.4 corresponds to the notions of strong opacity in [57] and language-based opacity in [101].

As we assume that everything is a regular language, LBO is equivalent to a regular language containment, comparing observed behavior in the system to the nominal model. To show this formally we observe that given the language L is marked by the automaton G, we can construct an automaton $\Theta(G)$ which marks the observations $\Theta(L)$. This is done similar to the construction for the projected automaton, replacing each transition labeled by an event σ in G by potentially multiple transitions labeled events in $\Theta(\sigma)$, contracting ϵ-transitions as needed. Likewise, we construct $\Theta(\hat{G})$ to mark observations of the nominal system. Then we have that LBO is equivalent to

$$\Theta(L) \subseteq \hat{\Theta}(\hat{L} \cap \varphi_{\text{NS}}) \;\Leftrightarrow\; \Theta(\mathcal{L}_m(G)) \subseteq \hat{\Theta}(\mathcal{L}_m(\hat{G} \times H_{\text{NS}})) \;\Leftrightarrow\; \mathcal{L}_m(\Theta(G)) \subseteq \mathcal{L}_m(\hat{\Theta}(\hat{G} \times H_{\text{NS}}))\,. \tag{3.3}$$

By expressing LBO as the well-understood problem of regular language containment, we can leverage existing techniques to verify LBO. It is well-known that verifying opacity, or equivalently checking the containment of languages represented by NFA, is PSPACE-complete [16, 89]. While this means any verification algorithm requires exponential time in general, unless $\mathsf{P} = \mathsf{NP}$, the specific numerical bound for different algorithms may vary significantly. In practice, we can greatly improve runtime performance by develop algorithms taking advantage of the structure present in a particular notion of opacity. Here, we present and later evaluate three such approaches for verifying opacity, specifically checking the language containment in Equation (3.3). As input, these methods take automata $G = (Q, \Sigma, \delta, Q_0, Q_m)$ and $\hat{G} = \left(\hat{Q}, \Sigma, \hat{\delta}, \hat{Q}_0, \hat{Q}_m\right)$, static masks Θ and $\hat{\Theta}$, and an automaton $H_{\text{NS}} = (Q_{\text{NS}}, \Sigma, \delta_{\text{NS}}, Q_{\text{NS},0}, Q_{\text{NS},m})$.

Approach 3.1 (Forward Comparison). *A standard approach for verifying language containment utilizes the following equivalence:*

$$L_1 \subseteq L_2 \;\Leftrightarrow\; L_1 \cap \mathrm{comp}(L_2) = \emptyset\,. \tag{3.4}$$

We construct the forward comparison automaton $G_F = \Theta(G) \times \mathrm{comp}(\det(\hat{\Theta}(\hat{G} \times H_{NS})))$ *such that*

$$\mathcal{L}_m(G_F) = \Theta(L) \cap \mathrm{comp}(\hat{\Theta}(\hat{L} \cap \varphi_{NS}))\,. \tag{3.5}$$

Note we employ the powerset construction as $\hat{\Theta}(\hat{G} \times H_{NS})$ is an NFA in general, and the complement construction requires a DFA. Hence, LBO holds if $\mathcal{L}_m(G_F)$ is empty, i.e., G_F contains no reachable, marked states.

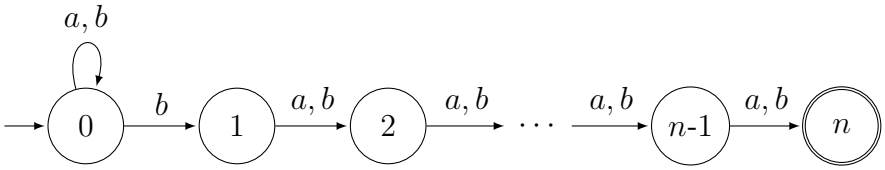

Figure 3.1: An automaton G_n with $n+1$ states for which the powerset construction $\det(G_n)$ has 2^n+1 states while the powerset construction for the reverse $\det(\mathrm{rev}(G_n))$ has only $n+1$ states.

Approach 3.2 (Reverse Comparison). *Instead checking the containment directly, we can equivalently check the containment of the reversed languages. Similar to the forward comparison method, we construct the* reverse comparison automaton $G_R = \mathrm{rev}(\Theta(G)) \times \mathrm{comp}(\det(\mathrm{rev}(\hat{\Theta}(\hat{G} \times H_{NS}))))$ *such that*

$$\mathrm{rev}(\mathcal{L}_m(G_R)) = \mathrm{rev}(\Theta(L)) \cap \mathrm{comp}(\mathrm{rev}(\hat{\Theta}(\hat{L} \cap \varphi_{NS}))) \,. \tag{3.6}$$

We then verify opacity by ensuring G_R contains no reachable, marked state. For some forms of LBO, reverse comparison significantly outperforms forward comparison. This is consistent with the fact that the powerset construction for some automata is exponentially larger the powerset construction for their reverse. For example, consider the automaton depicted in Fig. 3.1.

We can simplify verification by making additional assumptions. First, we will assume that the actual and nominal models are the same, i.e. $G = \hat{G}$ and $\Theta = \hat{\Theta}$. Second, we assume that H_{NS} is *complete*, i.e., at every state there is a transition for each event so $\mathcal{L}(H_{\mathrm{NS}}) = \Sigma^*$. This may be done without loss of generality by adding at most one sink state to H_{NS}. In this case, the automaton $G \times H_{\mathrm{NS}}$ encodes both the system behavior and nominal behavior in the nonsecret class with a different set of marked states

$$\mathcal{L}_{Q_m \times Q_{\mathrm{NS}}}(G \times H_{\mathrm{NS}}) = L, \quad \mathcal{L}_{Q_m \times Q_{\mathrm{NS},m}}(G \times H_{\mathrm{NS}}) = L \cap \varphi_{\mathrm{NS}} \,. \tag{3.7}$$

We can use this fact to show the following result

Theorem 3.1. *Given an automaton G, a static mask Θ, and a complete automaton H_{NS}, construct the* secret observer automaton $G_{SO} = \det(\Theta(G \times H_{NS}))$ *with marked states given by $Q_{m,SO} = \mathbf{Q}_1 \setminus \mathbf{Q}_2$ where*

$$\mathbf{Q}_1 = \{\mathbf{q} \in 2^{Q \times Q_{H_{NS}}} \mid \mathbf{q} \cap (Q_m \times Q_{H_{NS}}) \neq \emptyset\}, \quad \mathbf{Q}_2 = \{\mathbf{q} \in 2^{Q \times Q_{H_{NS}}} \mid \mathbf{q} \cap (Q_m \times Q_{m,H_{NS}}) \neq \emptyset\} \,. \tag{3.8}$$

Then LBO holds, i.e., $\Theta(\mathcal{L}_m(G)) \subseteq \Theta(\mathcal{L}_m(G \times H_{NS}))$, if and only if G_{SO} contains no reachable, marked states.

Proof. Note by the definition of the powerset construction, $\mathbf{Q}_1$ are the marked states of $\det(\Theta(G \times H_{\text{NS}}))$ if we consider $Q_m \times Q_{H_{\text{NS}}}$ marked in $G \times H_{\text{NS}}$. Likewise, $\mathbf{Q}_2$ would be the marked states if we consider $Q_m \times Q_{m,H_{\text{NS}}}$ marked $G \times H_{\text{NS}}$. Hence by Equation (3.7), we have

$$\mathcal{L}_{\mathbf{Q}_1}(G_{SO}) = \mathcal{L}_m(G), \quad \mathcal{L}_{\mathbf{Q}_2}(G_{SO}) = \mathcal{L}_m(G \times H_{\text{NS}})\,. \tag{3.9}$$

Finally as G_{SO} is deterministic, $\mathcal{L}_m(G_{SO}) = \mathcal{L}_{\mathbf{Q}_1}(G_{SO}) \setminus \mathcal{L}_{\mathbf{Q}_2}(G_{SO})$ which proves the claim. □

Here states in $\mathbf{Q}_1$ correspond to observations of behavior that can occur in the system while states in $\mathbf{Q}_2$ correspond to observations consistent with nonsecret behavior. This result motivates the following approach.

Approach 3.3 (Secret Observer). *We construct the secret observer $G_{SO} = \det(\Theta(G \times H_{NS}))$ with marked states as in Theorem 3.1. Then LBO holds if G_{SO} contains no reachable marked states.*

In each of these approaches, we verify opacity by constructing a verification automaton (G_F, G_R, or G_{SO}) and checking that there are no reachable, marked states. As these output automata are the largest constructed during these approaches, we quantify their complexity in terms of the number of states in the automata. Note, these verification approaches can be applied by incrementally constructing the verification automata and possibly terminating early when a violating state is found.

As in our proposed secret observer approach, most existing works in the literature also assume that the eavesdropper knows the exact model of the system, i.e., $G = \hat{G}$ and $\Theta = \hat{\Theta}$. In order to facilitate comparison, we will maintain this assumption in the remainder of this chapter. In this case we simply say that the system Δ is opaque for φ_{NS}.

State-based Opacity

We now discuss state-based notions of opacity in the framework of Section 3.2. Whereas LBO is defined solely in terms of the events of a language represented by an automaton, SBO is defined in terms of both events and states. As many notions of SBO implicitly consider only prefix-closed behavior, we consider a system modeled by an automaton $A = (X, E, f, X_0, X_m)$ with $X_m = X$. In this chapter, we use the convention that automata denoted by A represent state-based behavior, while automata denoted by G and H_{NS} represent languages. To express state-based opacity in the framework of Section 3.2, we must first define the relevant behavior of the automaton.

One obvious choice for the behavior is the set of runs over strings of the automaton, for example interleaved as $q_0 s_0 q_1 \cdots s_{n-1} q_n \in (Q\Sigma)^* Q$. Alternatively, as we are mostly interested in capturing existing notions of SBO which depend on whether a state is secret or not, we can obtain a more compact representation by only modeling events and state labels. In this case we can view A as

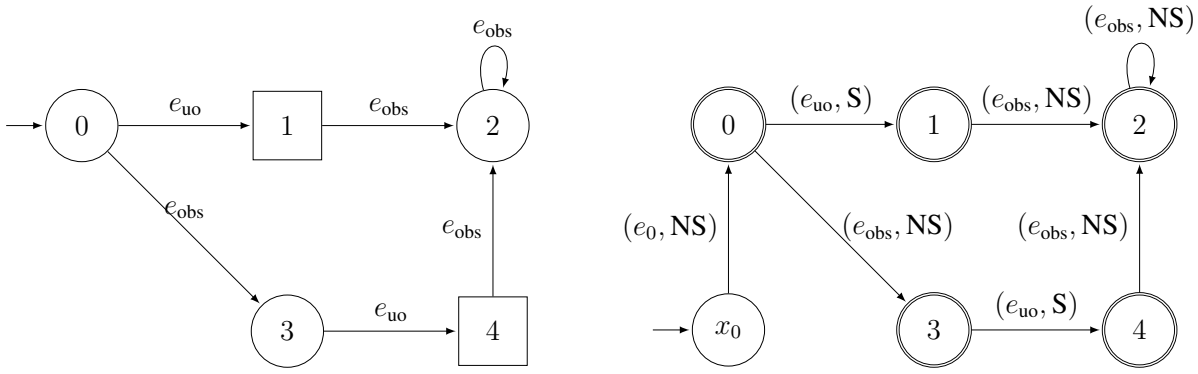

Figure 3.2: On the left, an automaton A is depicted with labels ℓ defined by labeling square states with variable S. On the right, the state-transform automaton $G = \mathcal{T}(A, \ell)$ is depicted.

a labeled transition system. Formally, consider a map $\ell : X \to A$ assigning labels to the states of A. Then runs of the system can be written as sequences of event and label pairs; (these can be thought of as input/output pairs). In order to balance the number of events and labels, we introduce an event e_0 representing the system turning on. For example consider the automaton A depicted in Fig. 3.2. The run starting at state 0, transitioning with event σ_u to state 1 with label S, then transitioning with event σ_o to state 2, would be represented as $s = (e_0, \text{NS})(\sigma_u, \text{S})(\sigma_o, \text{NS})$. In this way, the state-based behavior of A can be written as a language over $\Sigma = (E \cup \{e_0\}) \times A$. We can represent this language with an automaton as follows.

Definition 3.6. *Given automaton $A = (X, E, f, X_0, X_m)$ with labels $\ell : X \to A$, the* label-transform *of A is an automaton $\mathcal{T}(A, \ell) = (Q, \Sigma, \delta, Q_0, Q_m)$, where $Q = X \cup \{q_{init}\}$, $\Sigma = (E \cup \{e_0\}) \times A$, $Q_0 = \{q_{init}\}$, $Q_m = X_m$, and transitions defined by*

$$\delta = \{(x, (e_0, a), x') \mid a = \ell(x') \ \wedge \ (x = q_{init} \cup x' \in X_0 \vee (x, e, x') \in f)\}. \tag{3.10}$$

An example of this transformation is depicted in Fig. 3.2. This transformation adds only a single state and permits the following definition.

Definition 3.7. *The* IO sequences of A under ℓ *are defined as $\mathcal{LIO}(A, \ell) = \mathcal{L}_m(\mathcal{T}(A, \ell))$.*

We will show that notions of SBO over A can be expressed as LBO over $\mathcal{T}(A, \ell)$.

With the state-based behavior of the automaton defined, we can now express notions of SBO like CSO and ISO. Although it is already known that CSO and ISO can be transformed into LBO [101], we include this discussion as a demonstration of our opacity framework to provide insight to how more complex notions of SBO can be expressed.

Recall that CSO and ISO concern visits to a set of secret states X_{S}. This set is encoded with the labeling $\ell : X \to A$ with labels $A = \{\text{S}, \text{NS}\}$, defined for a state $x \in X$ by $\ell(q) = \text{S} \ \Leftrightarrow \ x \in X_{\text{S}}$.

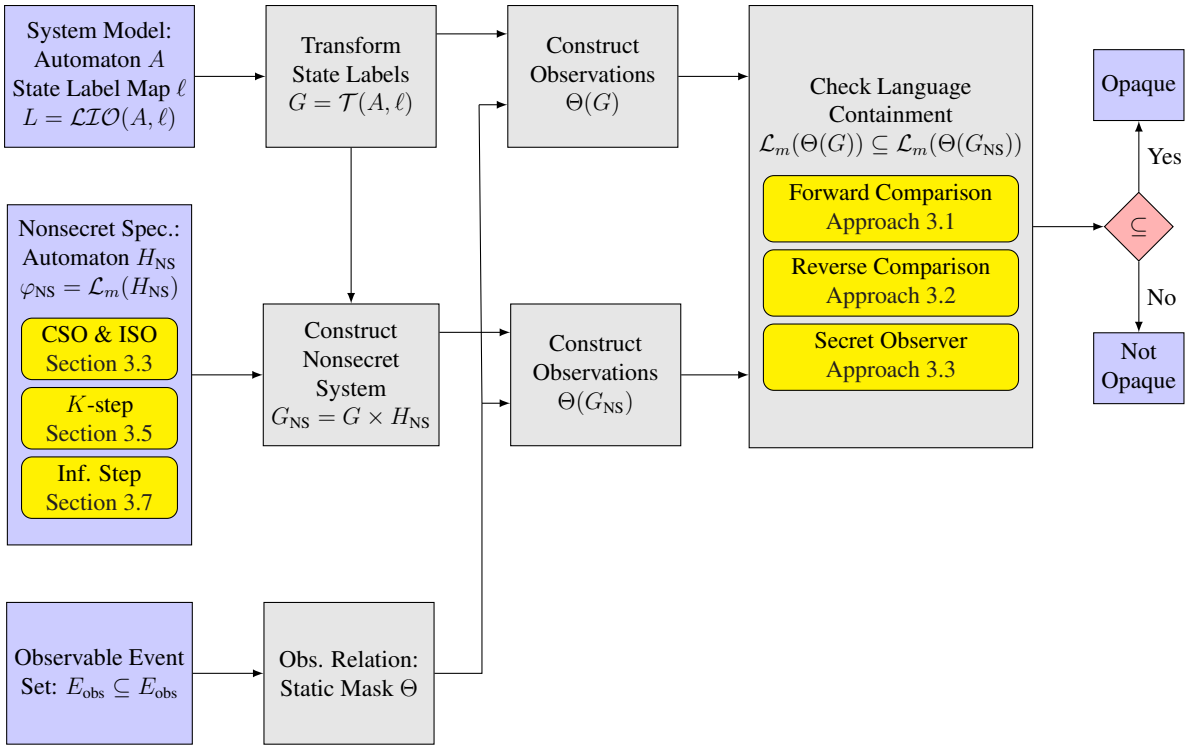

Figure 3.3: The proposed method for verifying state-based opacity by transforming to language-based opacity.

While in general these state labels could be observed, definitions of opacity in DES typically assume observations in the form of the natural projection of observable events $E_{\text{obs}} \subseteq E$. We define a corresponding static mask Θ over Σ with the convention that e_0 is observable, i.e., the observer can see the system turn on. Formally, the event $\sigma = (e, \nu) \in \Sigma$ is observed as $\Theta(\sigma) = e$ if $e \in E_{\text{obs}} \cup \{e_0\}$ and $\Theta(\sigma) = \epsilon$ otherwise. Likewise, in DES it is typically assumed that the observer knows the system model so $\hat{G} = G$ and $\hat{\Theta} = \Theta$. Under these assumptions, all that remains to express notions like CSO and ISO is constructing an automaton H_{NS} capturing the appropriate notion of secrets. To do this, we note that a visit to a secret state $x \in X_{\text{S}}$ in A corresponds to the occurrence of a event $\sigma = (e, \ell(x)) = (e, \text{S})$ in G. This will allow us to define H_{NS} in terms of the *secret events* $\Sigma_{\text{S}} = (E \cup \{e_0\}) \times \{\text{S}\}$ and *nonsecret events* $\Sigma_{\text{NS}} = (E \cup \{e_0\}) \times \{\text{NS}\}$.

Current state opacity describes the inability of an intruder to deduce that the current state of the system is secret. It can be defined as follows.

Definition 3.8 (Current-State Opacity [32]). *An automaton $A = (X, E, f, X_0, X_m)$ is said to be* current-state opaque *with respect to the secret states $X_S \subseteq X$ and observable events $E_{obs} \subseteq E$ if*

$$\begin{aligned} &\forall x_0 \in X_0.\, \forall s \in \mathcal{L}(A).\ s.t.\ \exists x_S \in f(x_0, s) \cap X_S, \\ &\exists x_0' \in X_0.\, \exists s' \in \mathcal{L}(A).\ \mathrm{P}_{(E \cup \{e_0\})}(s) = \mathrm{P}_{(E \cup \{e_0\})}(s') \ \wedge\ \exists x_{NS} \in f(x_0', s') \cap X \setminus X_S\,. \end{aligned} \tag{3.11}$$

In words, runs of A ending with a visit to a secret state should look like a run ending with a visit to a nonsecret state. In terms of input-output sequences, this definition divides the behavior $L = \mathcal{LIO}(A, \ell)$ into secret and nonsecret behavior $L_{\mathrm{S}}, L_{\mathrm{NS}} \subseteq L$ defined by

$$\varphi_{\mathrm{NS}} = \Sigma^* \Sigma_{\mathrm{NS}}, \quad L_{\mathrm{NS}} = L \cap \varphi_{\mathrm{NS}}, \quad L_{\mathrm{S}} = L \setminus L_{\mathrm{NS}}\,. \tag{3.12}$$

We can use the nonsecret specification automaton H_{NS} depicted in Fig. 3.4 with $\mathcal{L}_m(H_{\mathrm{NS}}) = L_{\mathrm{NS}}$.

Then using the observation map Θ induced by the observable events E_{obs}, we can see that A is current-state opaque if and only if $\Delta = (L, \Theta)$ is opaque with respect to φ_{NS}.

Hence, we can use the language-based methods for verification.

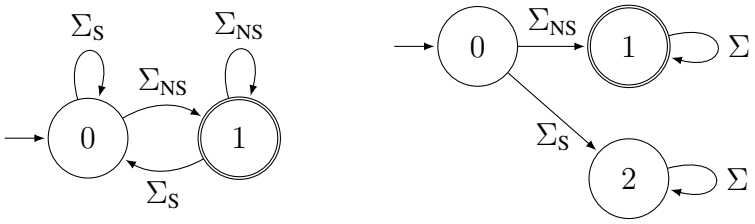

Figure 3.4: The nonsecret specification automata H_{NS} for CSO (left) and ISO (right).

To do this we first construct $G = \mathcal{T}(A, \ell)$. As $\mathcal{L}(H_{\mathrm{NS}}) = \Sigma^*$, using Theorem 3.1 we can verify CSO of A by checking if every non-initial state of the secret observer $G_{SO} = \det(\Theta(G \times H_{\mathrm{NS}}))$ is marked where $G = \mathcal{T}(A, \ell)$. As an example of this method, we verify the current-state opacity of A from Fig. 3.2 using its transformation $G = \mathcal{T}(A, \ell)$. Assuming $E_{\mathrm{obs}} = \{e_{\mathrm{obs}}\}$, we construct $G \times H_{\mathrm{NS}}$ and $G_{SO} = \det(\Theta(G \times H_{\mathrm{NS}}))$ which are depicted in Fig. 3.5. As every non-initial state of G_{SO} is marked, we deduce A is CSO.

Remark 3.2. *The construction $G = \mathcal{T}(A, \ell)$ essentially moves the state label information from the states of A to the events of G. In the product $G \times H_{NS}$, these labels are then moved from the events back to the states in the form of state markings. As a result $G \times H_{NS}$ is the same as the original automaton A where nonsecret states are marked and there are new initial states resulting from x_0 in G. In this way the secret observer method is comparable to the standard method for verifying current-state opacity [83] which checks if each state of the observer of A contains a nonsecret state. While our approach may seem convoluted for verifying CSO, the purpose of our discussion and of the above example are to demonstrate how our approach can be used to verify state-based notions of opacity in general.*

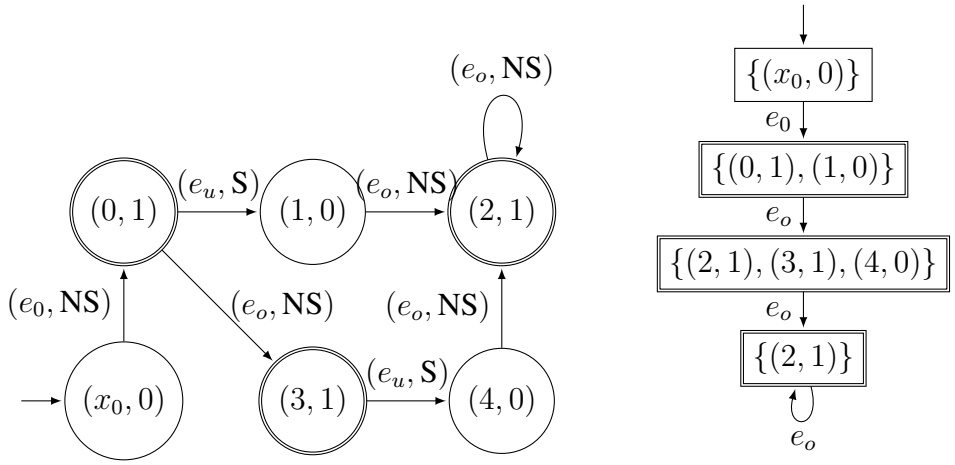

Figure 3.5: The product $G \times H_{\text{NS}}$ (left) for $G = \mathcal{T}(A, \ell)$ where A is from Fig. 3.2 and the nonsecret specification automaton H_{NS} for CSO from Figure 3.4 and the corresponding secret observer G_{SO} (right).

3.3.1 Initial-state opacity(ISO)

Next, we discuss the notion of initial-state opacity. Initial-state opacity describes the inability of an intruder to deduce that the initial-state of a run was secret. It can be defined as follows.

Definition 3.9 (Initial-State Opacity [101]). *The automaton $A = (X, E, f, X_0, X_m)$ is said to be* initial-state opaque *with respect to secret states $X_S \subseteq X_0$ and observable events $E_{obs} \subseteq E$ if*

$$\begin{aligned} &\forall x_0 \in X_S.\, \forall s \in \mathcal{L}(A)\ s.t.\ \exists x \in f(x_0, s), \\ &\exists x_0' \in X_{NS},\ \exists s' \in \mathcal{L}(A),\ \mathrm{P}_{E_{obs}}(s) = \mathrm{P}_{E_{obs}}(s')\ \wedge\ \exists x' \in f(x_0', s')\,. \end{aligned} \tag{3.13}$$

Similar to the discussion of current-state opacity, we see that the initial-state opacity of A is equivalent to the opacity of the system of L and the observation relation Θ induced by E_{obs} with respect to

$$\varphi_{\text{NS}} = \Sigma_{\text{NS}}\Sigma^*, \quad L_{\text{NS}} = L \cap L_{\text{NS}}, \quad L_{\text{S}} = L \setminus L_{\text{NS}}\,. \tag{3.14}$$

We can construct H_{NS} as in Fig. 3.4 so that $\mathcal{L}_m(H_{\text{NS}}) = L_{\text{NS}}$ and $\mathcal{L}(H_{\text{NS}}) = \Sigma^*$. Applying the secret observer method in this case is similar to transforming initial-state opacity to current-state opacity as in [101] and using the standard approach to verify current-state opacity. Furthermore, we can take advantage of the specific structure of H_{NS} to obtain more efficient verification methods. Namely, applying the reverse comparison method is similar to verifying ISO using the reversed initial-state estimator of [101] which is significantly more efficient than the initial method proposed in [84].

3.4 Opacity Specification

While current-state opacity captures the notion of hiding current secrets, K-step and infinite step opacity capture the notion of hiding past secrets. In this section, we define language-based notions of K-step and infinite step opacity over automata. We then show how these relate to the existing notions.

3.4.1 Language-based K-step opacity

Consider a system as in Section 3.3 consisting of a language $L = \mathcal{L}_m(G)$ and static mask observation relation $\Theta \subseteq \Sigma^* \times \Sigma_{\text{obs}}^*$. Motivated by our transformation of current-state opacity to language-based opacity, we consider a partition of the events Σ into secret events Σ_{S} and nonsecret events Σ_{NS}. We propose a notion of language-based K-step opacity which concerns occurrences of these secret events during the last K observations made by the intruder. We use the term *observation epoch* to refer to the system's behavior between observations. More specifically, the epoch starts when an observation is made and ends right before another observation is made or at the end of the run. We consider two types of secret behavior that can be exhibited in an observation epoch. In the first type, which we call **type 1**, *at least one secret event occurs*. In the second type, which we call **type 2**, *only secret events occur*.

We formally describe the observation epochs in terms of the set of events which do not produce an observation through Θ which is given by $\Sigma_{\text{sil}} = \Theta^{-1}(\epsilon) \cap \Sigma$. We call these events *silent* in reference to the silent transitions described by [43]. The remaining events $\Sigma \setminus \Sigma_{\text{sil}}$ are called *nonsilent*. With this we make the following definition.

Definition 3.10. *The set of* observation epochs *is defined to be* $\varphi_{ep} = (\Sigma \setminus \Sigma_{sil})\Sigma_{sil}^*$. *The sets of* observation epochs exhibiting type 1 or type 2 secrets*, respectively, are defined by*

$$\varphi_{ep,S,1} = \varphi_{ep} \cap (\Sigma^* \setminus \Sigma^*[NS]), \quad \varphi_{ep,S,2} = \varphi_{ep} \cap \Sigma_S^* \,. \tag{3.15}$$

Likewise the sets of type 1 and type 2 nonsecret epochs are defined by

$$\begin{aligned} \varphi_{NS,ep,1} &= \varphi_{ep} \setminus \varphi_{ep,S,1} = \varphi_{ep} \cap \Sigma_{NS}^*, \\ \varphi_{NS,ep,2} &= \varphi_{ep} \setminus \varphi_{ep,S,2} = \varphi_{ep} \cap (\Sigma^* \setminus \Sigma_S^*) \,. \end{aligned} \tag{3.16}$$

For simplicity, we will assume that every behavior starts with a nonsilent event, so that we may write $L \subseteq (\Sigma \setminus \Sigma_{\text{sil}})\Sigma^* = \varphi_{\text{ep}}^+$. This means any run $s \in L$ can uniquely be written as a concatenation of observation epochs, i.e., $\exists M > 0$, $r = s_{\text{ep},0} \cdots s_{\text{ep},M-1}$ with $s_{\text{ep},i} \in \varphi_{\text{ep}}$ for all $i < M$. We refer to the epoch $s_{\text{ep},M-k-1}$ as the epoch k^{th} from the end or as k epochs ago. For K-step opacity, we

define different classes of secret and nonsecret behavior for each epoch in the past, up to K epochs ago. For $k \leq K$ and type $j \in \{1, 2\}$ secrets, we define

$$\begin{aligned} \varphi_{\mathrm{S},j}(k) &= \varphi_{\mathrm{ep}}^{*} \varphi_{\mathrm{ep,S},j} \varphi_{\mathrm{ep}}^{k} , \\ \varphi_{\mathrm{NS},j}(k) &= \varphi_{\mathrm{ep}}^{+} \setminus \varphi_{\mathrm{S},j}(k) = (\varphi_{\mathrm{ep}}^{*} \varphi_{\mathrm{NS,ep},j} \varphi_{\mathrm{ep}}^{k}) \cup \bigcup_{i=1}^{k} \varphi_{\mathrm{ep}}^{i} . \end{aligned} \tag{3.17}$$

We refer to $\varphi_{\mathrm{S},j}(k)$ and $\varphi_{\mathrm{NS},j}(k)$ as the *k-delayed secret and nonsecret behavior* specifications, respectively. Note that a run consisting of fewer than $k+1$ observation epochs is by definition not an element of $\varphi_{\mathrm{S},j}(k)$, as a secret could not have occurred $k+1$ epochs ago. The k-delayed secret and nonsecret behavior of L with type $j \in \{1, 2\}$ secrets are then defined

$$L_{\mathrm{S},j}(k) = L \cap \varphi_{\mathrm{S},j}(k), \qquad L_{\mathrm{NS},j}(k) = L \setminus \varphi_{\mathrm{S},j}(k) = L \cap \varphi_{\mathrm{NS},j}(k) . \tag{3.18}$$

By considering these secrets jointly, we can model an intruder deducing *if* a secret occurred within K epochs ago (or when a secret ever occurred in the case where $K = \infty$).

Definition 3.11. *For $K \in \mathbb{N} \cup \{\infty\}$, we say the system $\Delta = (L, \Theta)$ with nonsecret events $\Sigma_{NS} \subseteq \Sigma$ is* jointly K-step opaque with type j secrets *if it is jointly opaque with respect to $\{\varphi_{NS,j}(k)\}_{k=0}^{K}$ as defined in* (3.17)*.*

By considering these secrets separately, we can model an intruder deducing *when* a secret occurred within K epochs ago (or when a secret ever occurred in the case that $K = \infty$).

Definition 3.12. *For $K \in \mathbb{N} \cup \{\infty\}$, we say the system $\Delta = (L, \Theta)$ with nonsecret events $\Sigma_{NS} \subseteq \Sigma$ is* separately K-step opaque with type j secrets *if it is separately opaque with respect to $\{\varphi_{NS,j}(k)\}_{k=0}^{K}$ as defined in* (3.17)*.*

For $K = \infty$ we refer to these definitions as *infinite step opacity*. While separate K-step opacity involves $\varphi_{\mathrm{NS},j}(k)$ and hence $\varphi_{\mathrm{NS},j}(k)$ for $k \leq K$, joint opacity only involves their intersections. For convenience we define for $K \in \mathbb{N}$

$$\varphi_{\mathrm{NS},j}^{\mathrm{J}}(K) = \bigcap_{k=0}^{K} \varphi_{\mathrm{NS},j}(k) = \varphi_{\mathrm{ep}}^{*} \varphi_{\mathrm{NS,ep},j}^{K+1} \cup \bigcup_{k=1}^{K} \varphi_{\mathrm{NS,ep},j}^{k} , \tag{3.19}$$

so that $\bigcap_{k=0}^{K} L_{\mathrm{NS},j}(k) = L \cap \varphi_{\mathrm{NS},j}^{\mathrm{J}}(K)$. In the joint sense, a run is secret if it consists entirely of nonsecret epochs or its last nonsecret epoch was at least $K+1$ epochs ago.

By comparing the nonsecret specification languages, we can relate the different notions of K-step opacity for $K \in \mathbb{N} \cup \{\infty\}$. Because $\varphi_{\mathrm{NS,ep},1} \subseteq \varphi_{\mathrm{NS,ep},2}$, it holds that $\varphi_{\mathrm{NS},1}(K) \subseteq \varphi_{\mathrm{NS},2}(K)$ and

thus $\varphi_{\text{NS},1}(K) \subseteq \varphi_{\text{NS},2}(K)$. Hence joint and separate K-step opacity with type 1 secrets imply joint and separate K-step opacity with type 2 secrets, respectively. Additionally, we see that joint K-step opacity with type $j \in \{1, 2\}$ secrets implies separate K-step opacity with type j secrets. These implications are depicted in Fig. 3.6. This figure also depicts the relation to the existing notions of K-step opacity derived in the next section. Furthermore for $K \leq K'$, joint and separate K'-step opacity with type $j \in \{1, 2\}$ secrets implies joint and separate K-step opacity with type j secrets.

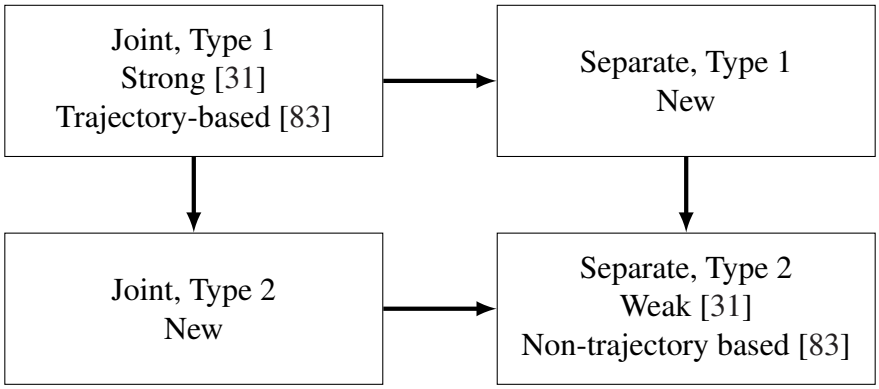

Figure 3.6: Types of K-step opacity. Arrows indicate logical implication. For example, joint type 1 K-step opacity implies separate type 1 K-step opacity.

3.4.2 Relation to existing notions of K-step opacity

Now we show how these definitions relate to the existing state-based notions of K-step opacity (for finite K). These notions were originally defined over deterministic finite automata, so for consistency we derive these relations in this setting. Consider a deterministic automaton $A = (X, E, f, \{x_0\})$ and interpret f as a partial function $f : X \times E \to X$. Consider a set of secret states $X_{\text{S}} \subseteq X$ and nonsecret states $X_{\text{NS}} = X \setminus X_{\text{S}}$ as well as a set of observable events $E_{\text{obs}} \subseteq E$.

The first form of K-step opacity was developed in [83], and was later referred to as non-trajectory-based K-step opacity in [85] and weak K-step opacity in [32].

Definition 3.13 (K-step Weak Opacity [32]). *The automaton A is* weakly K-step opaque *with respect to X_S and E_{obs} if*

$$
\begin{aligned}
&(\forall uv \in \mathcal{L}(A)\ s.t.\ |\,\mathrm{P}_{E_{obs}}(v)| \leq K\ \wedge f(x_0, u) \in X_S) \\
&(\exists u'v' \in \mathcal{L}(A)) \\
&(\mathrm{P}_{E_{obs}}(uv) = \mathrm{P}_{E_{obs}}(u'v')\ \wedge\ \mathrm{P}_{E_{obs}}(u) = \mathrm{P}_{E_{obs}}(u')\ \wedge\ f(x_0, u') \in X_{NS})\,.
\end{aligned}
$$

The second version we consider is referred to as trajectory-based K-step opacity in [85] and strong K-step opacity in [32].

Definition 3.14 (K-step Strong Opacity [32]). *The automaton A is* strongly K-step opaque *with respect to X_S and E_{obs} if*

$$
\begin{aligned}
&(\forall t \in \mathcal{L}(A)) \\
&(\exists t' \in \mathcal{L}(A),\ \forall u', v'\ s.t.\ t' = u'v') \\
&(\mathrm{P}_{E_{obs}}(t) = \mathrm{P}_{E_{obs}}(t')\ \wedge\ (|\,\mathrm{P}_{E_{obs}}(v')| \leq K\ L \rightarrow\ f(x_0, u') \in X_{NS})
\end{aligned}
$$

Weak K-step opacity describes the inability of the intruder to deduce an exact time of a visit to a secret state within the last K observations. Strong K-step opacity describes the inability of the intruder to deduce there was a visit to a secret state within the last K observations. With this intuition we can relate weak to separate and strong to joint opacity.

Theorem 3.2. *Consider a deterministic automaton A with labeling map ℓ defined by the secret states X_S and observable events E_{obs}. Let Δ denote the system with behavior L marked by $G = \mathcal{T}(A, \ell)$ and observation relation Θ induced by E_{obs}. Let $\Sigma_{NS} = \{(e, NS) \mid e \in (E \cup \{e_0\})\}$. Then*

1. *Weak K-step opacity of A is equivalent to separate K-step opacity with type 2 secrets of Δ with nonsecret events Σ_{NS}.*
2. *Strong K-step opacity of A is equivalent to joint K-step opacity with type 1 secrets of Δ with nonsecret events Σ_{NS}.*

Proof. Because the automaton A is deterministic, there is a unique sequence of states associated with each string in $\mathcal{L}(A)$. This defines a bijection $h : L \rightarrow \mathcal{L}(A)$ such that

$$
\forall s \in L. \quad \mathrm{P}^I(s) = e_0 \cdot h(s), \quad \Theta(s) = e_0 \cdot \mathrm{P}_{E_{\text{obs}}}(h(s))\,. \tag{3.20}
$$

Then note that we can write for $k \leq K$

$$
\begin{aligned}
h(L_{\text{NS},1}(k)) = \{t \in \mathcal{L}(A) \mid\ \forall i \leq |t|,\ |\,\mathrm{P}_{E_{\text{obs}}}(t_i \cdots t_{|t|-1})| = k\ L \rightarrow \\
f(x_0, t_0 \cdots t_{i-1}) \in X_{\text{NS}}\}
\end{aligned} \tag{3.21}
$$

$$
\begin{aligned}
h(L_{\text{NS},2}(k)) = \{t \in \mathcal{L}(A) \mid\ |\,\mathrm{P}_{E_{\text{obs}}}(t)| < k \vee \exists i \leq |t|\ |\,\mathrm{P}_{E_{\text{obs}}}(t_i \cdots t_{|t|-1})| = k \wedge \\
f(x_0, t_0 \cdots t_{i-1}) \in X_{\text{NS}}\}\,.
\end{aligned} \tag{3.22}
$$

Suppose A is weakly K-step opaque and let $k \leq K$. Consider a run of A given by $s \in L$. If $|\Theta(s)| < k$ then by definition $s \in L_{\text{NS},2}(k)$. Otherwise consider $t = h(s)$ so $|\,\mathrm{P}_{E_{\text{obs}}}(t)| \geq k$. Let

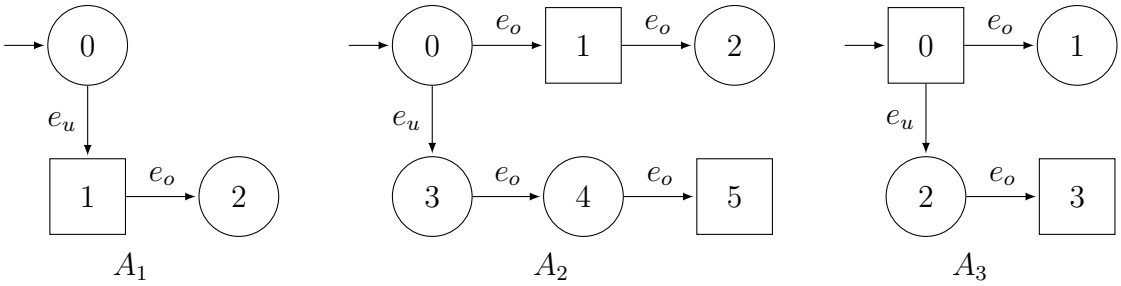

Figure 3.7: Automata demonstrating the differences in the various notions of K-step opacity. Here square states denote secret states. The observable event set is $E_{\text{obs}} = \{e_{\text{obs}}\}$.

$i \leq |t|$ be such that $|\,\mathrm{P}_{E_{\text{obs}}}(t_i \cdots t_{|t|-1})| = k$ and define $u = t_0 \cdots t_{i-1}$ and $v = t_i \cdots t_{|t|-1}$. By weak opacity of A, there must exist $t' = u'v'$ such that $\mathrm{P}_{E_{\text{obs}}}(t) = \mathrm{P}_{E_{\text{obs}}}(t')$, $|\,\mathrm{P}_{E_{\text{obs}}}(v')| = k$, and $f(x_0, u') \in X_{\text{NS}}$. Thus for $s' = h^{-1}(t')$ it holds that $s' \in L_{\text{NS},2}(k)$ and $\Theta(s) = \Theta(s')$. Hence Δ is separately K-step opaque with type 2 secrets. The proof of the converse is similar.

Now we consider strong K-step opacity. Suppose that A is strongly K-step opaque. Consider a run of A given by $s \in L$ and define $t = h(s)$. By strong K-step opacity of A, there exists $t' \in \mathcal{L}(A)$ with $\mathrm{P}_{E_{\text{obs}}}(t) = \mathrm{P}_{E_{\text{obs}}}(t')$ where for every $i' \leq |t'|$ such that $|\,\mathrm{P}_{E_{\text{obs}}}(t'_{i'} \cdots t'_{|t'|-1})| \leq K$ it holds that $f(x_0, t'_0 \cdots t'_{i'-1}) \in X_{\text{NS}}$. Thus for $s' = h^{-1}(t')$ it holds that $s' \in L_{\text{NS},1}(k)$ for all $k \leq K$ and $\Theta(s') = \Theta(s)$. Thus Δ is jointly K-step opaque with type 1 secrets. The proof of the converse is similar. □

The other notions of joint opacity with type 2 secrets and separate opacity with type 1 secrets, to our knowledge, have not been previously proposed. Formally, we say that the automaton A is K-step opaque with type j secrets if the transformation $G = \mathcal{T}(A, \ell)$ is K-step opaque with type j. The differences between the proposed notions of K-step opacity stem from how secrets interact with unobservable behavior. To demonstrate how these notions differ, consider the automata A_1, A_2, A_3 from Fig. 3.7 and secret states $X[\mathrm{S}]$ given by the square states and observable event set $E_{\text{obs}} = \{e_{\text{obs}}\}$. In A_1 for example, there are no type 2 secret epochs possible as a visit to secret state 1 must be preceded by a visit to nonsecret state 0 in the same epoch. Hence A_1 is jointly and separately 1-step opaque with type 2 secrets. We can verify the various notions of 1-step opacity for all of these automata as depicted in Table 3.1.

To paraphrase, joint K-step opacity with type 1 secrets reflects the inability of the intruder to deduce *if* there was a period between observations where a *single* secret state was visited, while joint K-step opacity with type 2 secrets reflects the inability of the intruder to deduce *if* there was a period between observations where *only* secret states were visited. Likewise, separate K-step opacity with type 1 secrets reflects the inability of the intruder to deduce *when* there was a period between observations where a *single* secret state was visited, while separate K-step opacity with type 2 secrets reflects the inability of the intruder to deduce *when* there was a period between

1-Step Opacity Type	A_1	A_2	A_3
Separate Type 2	Yes	Yes	Yes
Separate Type 1	No	Yes	No
Joint Type 2	Yes	No	No
Joint Type 1	No	No	No

Table 3.1: The results of verifying joint and separate 1-step opacity with type 1 and type 2 secrets for the automata A_1, A_2, A_3 from Fig. 3.7.

observations where *only* secret states were visited. So we see for automata without unobservable events, type 1 and type 2 secrets are equivalent and these new notions of joint and separate reduce to the existing notions of strong and weak. While these new notions may only reflect differences in the modeling of unobservable events in some sense, they demonstrate how the proposed approach can be used to formulate precise notions of opacity appropriate for a given problem.

3.5 Verification methods for finite K-step opacity

In this section we will present methods for verification of K-step opacity for finite K. First, we construct automata specifying nonsecret behavior. Then we show how to use these automata to verify joint K-step opacity and separate K-step opacity.

3.5.1 Nonsecret specification automata

In order to use language-based methods to verify K-step opacity, we must first construct automata that mark the corresponding nonsecret specification languages. To do this, we will use the automata depicted in Fig. 3.8 as building blocks.. Note that $\mathcal{L}_m(H_*) = \Sigma^*$, $\mathcal{L}_m(H_{\text{ep}}) = \varphi_{\text{ep}}$, $\mathcal{L}_m(H_{\text{ep},1}) = \varphi_{\text{NS,ep},1}$, and $\mathcal{L}_m(H_{\text{ep},2}) = \varphi_{\text{NS,ep},2}$ [1]. To efficiently represent the nonsecret specifications languages, we note that

$$\varphi_{\text{NS},j}(k+1) = \varphi_{\text{NS},j}(k)\varphi_{\text{ep}} \cup \varphi_{\text{ep}}, \quad \varphi^{\text{J}}_{\text{NS},j}(k+1) = \varphi^{\text{J}}_{\text{NS},j}(k)\varphi_{\text{NS,ep},j} \cup \varphi_{\text{NS,ep},j}\,. \tag{3.23}$$

So by appropriately defining the initial states and concatenating the automata from Fig. 3.8, we can construct automata that specify the nonsecret runs. While the standard concatenation construction adds an epsilon-transition between the marked states of one automaton and the initial states of the next, we can reduce the resulting number of states by merging these states as in the following construction.

Definition 3.15. *Let $H^i = (Q^i, \Sigma, \delta^i, Q^i_0, Q^i_m)$ for $i \in \{1, 2\}$ be such that $Q^2_0 \cap Q^2_m = \emptyset$. Let $Q^{\cup} = Q^1 \sqcup Q^2 \setminus Q^2_0$, $Q^{\cup}_0 = Q^1_0 \cup Q^1_m$, and $Q^{\cup}_m = Q^2_m$. Here $\sqcup$ denotes the disjoint union. We define*

[1] While $H_{\text{ep},2}$ could be designed to be deterministic, our nondeterministic $H_{\text{ep},2}$ offers reduced complexity.

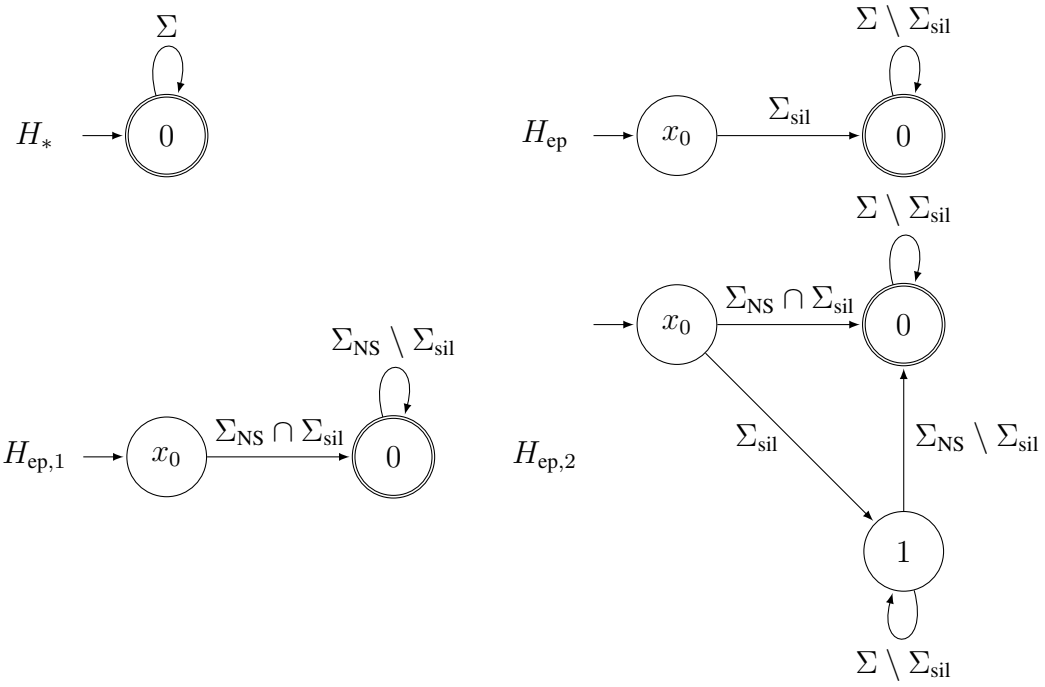

Figure 3.8: Automata used to construct nonsecret specification automata for K-step opacity defined over input-output pairs Σ categorized into nonsecret pairs Σ_{NS}, and observable and unobservable pairs $\Sigma_{\text{sil}}, \Sigma \setminus \Sigma_{\text{sil}}$.

the concatenated automaton $H^1 \cup H^2 = (Q^{\cup}, \Sigma, \delta^{\cup}, Q_0^{\cup}, Q_m^{\cup})$ *where for all* $\sigma \in \Sigma$*,*

$$\begin{aligned} \forall q^1 \in Q^1 \setminus Q_m^1.\ \delta^{\cup}(q^1, \sigma) &= \delta^1(q^1, \sigma) \\ \forall q^2 \in Q^2 \setminus Q_0^2.\ \delta^{\cup}(q^2, \sigma) &= \delta^2(q^2, \sigma) \\ \forall q^1 \in Q_m^1.\ \delta^{\cup}(q^1, \sigma) &= \delta^1(q^1, \sigma) \cup \bigcup_{q^2 \in Q_0^2} \delta^2(q^2, \sigma)\,. \end{aligned}$$

This construction merges the marked states of H^1 *with the initial states of* H^2*. Note that* $\mathcal{L}_m(H^1 \cup H^2) = (\mathcal{L}_m(H^1) \cup \mathcal{L}_{mm}(H^1)) \cdot \mathcal{L}_m(H^2)$*, where* $\mathcal{L}_{mm}(H^1) = \mathcal{L}_m(H^1, Q_m^1)$ *is the marked language of* H^1 *starting at the marked states of* H^1*.*

Based on the relation in (3.23), we use this $\cup$ to construct specification automata.

Definition 3.16. *We define the* nonsecret specification automata for K-step opacity *iteratively as follows. Let* $H_{NS,j}(0) = H_{NS,j}^J(0) = H_* \cup H_{ep,j}$ *and for* $k \geq 0$ *define*

$$H_{NS,j}(k+1) = H_{NS,j}(k) \cup H_{ep}, \qquad H_{NS,j}^J(k+1) = H_{NS,j}^J(k) \cup H_{ep,j}\,. \tag{3.24}$$

The following result relates these nonsecret specification automata to the K-delayed nonsecret

behavior defining K-step opacity.

Lemma 3.3. *For every $K \in \mathbb{N}$ it holds that*

$$\begin{aligned} \varphi_{ep}^{+} \cap \mathcal{L}_m(H_{NS,j}(K)) &= \varphi_{NS,j}(K), \\ \varphi_{ep}^{+} \cap \mathcal{L}_m(H_{NS,j}^{J}(K)) &= \varphi_{NS,j}^{J}(K)\,. \end{aligned} \tag{3.25}$$

Proof. We show this for $H_{\text{NS},2}(K)$. The proofs for the other cases are similar. We claim that for all $k \leq K$ that $\mathcal{L}_{mm}(H_{\text{NS},2}(k)) = \Sigma_{\text{sil}}^{*}$ and

$$\mathcal{L}_m(H_{\text{NS},2}(k)) = \Sigma^{*}\varphi_{\text{NS,ep},2}\varphi_{\text{ep}}^{k} \cup \Sigma_{\text{sil}}^{*} \bigcup_{i=1}^{k} \varphi_{\text{ep}}^{i}\,. \tag{3.26}$$

Note that this condition holds for $k = 0$ as

$$\mathcal{L}_m(H_{\text{NS},2}(0)) = \Sigma^{*}\varphi_{\text{NS,ep},2}, \qquad \mathcal{L}_{mm}(H_{\text{NS},2}(0)) = \Sigma_{\text{sil}}^{*}\,. \tag{3.27}$$

Now assume that condition (3.26) holds for some $k < K$. Then by definition of $\cup$ we have

$$\begin{aligned} \mathcal{L}_m(H_{\text{NS},2}(k+1)) &= (\mathcal{L}_m(H_{\text{NS},2}(k)) \cup \mathcal{L}_{mm}(H_{\text{NS},2}(k)))\mathcal{L}_m(H_{\text{ep}}) \\ &= (\Sigma^{*}\varphi_{\text{NS,ep},2}\varphi_{\text{ep}}^{k} \cup \Sigma_{\text{sil}}^{*} \cdot \bigcup_{i=1}^{k} \varphi_{\text{ep}}^{i} \cup \Sigma_{\text{sil}}^{*})\varphi_{\text{ep}} \\ &= \Sigma^{*}\varphi_{\text{NS,ep},2}\varphi_{\text{ep}}^{k+1} \cup \Sigma_{\text{sil}}^{*} \cdot \bigcup_{i=1}^{k+1} \varphi_{\text{ep}}^{i} \end{aligned}$$

Hence by induction, condition (3.26) holds for all $k \leq K$. Then note because $\varphi_{\text{ep}} = \Sigma_{\text{obs}}\Sigma_{\text{sil}}^{*}$ that

$$\begin{aligned} \varphi_{\text{ep}}^{+} \cap \mathcal{L}_m(H_{\text{NS},2}(k)) &= \varphi_{\text{ep}}^{*}\varphi_{\text{NS,ep},2}\varphi_{\text{ep}}^{k} \cup \bigcup_{i=1}^{k} \varphi_{\text{ep}}^{i} \\ &= \varphi_{\text{NS},2}(k)\,. \end{aligned}$$

□

We can then use the automata $H_{\text{NS},j}(K)$ and $H_{\text{NS},j}^{\text{J}}(K)$ in specifying K-step opacity. As before, consider an automaton A with secret states labeled by ℓ with behavior given by the input-output pairs $L = \mathcal{LIO}(A, \ell)$. Then in terms of equation (3.18),

$$L \cap \varphi_{\text{NS},j}(K) = \varphi_{\text{NS},j}(K), \qquad L \cap \varphi_{\text{NS},j}^{\text{J}}(K) = \bigcap_{k=0}^{K} \varphi_{\text{NS},j}(k)\,. \tag{3.28}$$

So $H_{\text{NS},j}(k)$ for $k \leq K$ can be used as nonsecret specification automata for verification of separate

K-step opacity with type j secrets. Likewise, $H^{\mathrm{J}}_{\mathrm{NS},j}(K)$ can be used for joint K-step opacity with type j secrets. In any case, it holds that $\mathcal{L}(H_{\mathrm{NS},j}(K)) = \mathcal{L}(H^{\mathrm{J}}_{\mathrm{NS},j}(K)) = \Sigma^*$ so we will be able to apply the secret observer method later on.

Remark 3.3. *By expanding the recursive definitions of $H = H_{NS,j}(K)$ or $H = H^{J}_{NS,j}(K)$, we can write H in the form $\bigcup_{i=0}^{K+1} H^i$. To avoid ambiguity due to redundant state names, we refer to the state q of H^i by (q, i) when embedded in H.*

3.5.2 Verification of joint K-step opacity

We now discuss how to verify K-step opacity using the specification automata we have constructed in conjunction with the approaches described in Section 3.3. Formally, we consider a system represented by an automaton G and static observation relation Θ. As $H^{\mathrm{J}}_{\mathrm{NS},j}(K)$ encodes all of the behavior considered nonsecret for K-step opacity with type j secrets, we can verify this form of opacity by simply applying the aforementioned methods for verifying language-based opacity with the nonsecret specification automaton $H^{\mathrm{J}}_{\mathrm{NS},j}(K)$, where $K \in \mathbb{N}$ with type j secrets. Due to the equivalence described in subsection 3.4.2, we can verify the state-based notions of K-step opacity in the same way after applying the label transform as follows.

Approach 3.4 (Joint K-step opacity verification). *Given A, ℓ, E_{obs}, and $K < \infty$, construct the label-transform $G = \mathcal{T}(A, \ell)$, the nonsecret specification automaton $H^{J}_{NS,j}(K)$, and the static mask Θ induced by E_{obs}. We can then apply any of the language-based methods from Section 3.3 to G, $H^{J}_{NS,j}(K)$, and Θ to verify the joint K-step opacity with type j secrets of A.*

For example we depict $H^{\mathrm{J}}_{\mathrm{NS},1}(2)$ in Fig. 3.9. Recall this automaton is constructed by concatenating H_* and three copies of $H_{\mathrm{ep},1}$. We apply the secret observer method to the automaton A from Fig. 3.2 using its label transform $G = \mathcal{T}(A, \ell)$ also depicted in Fig. 3.2. The construction of $G_{SO} = \det(\Theta(G \times H^{\mathrm{J}}_{\mathrm{NS},1}(2)))$ is depicted in Fig. 3.10. We see that the string $e_0 e_{\mathrm{obs}} e_{\mathrm{obs}}$ is not marked in G_{SO}. Hence by the secret observer method, A is not jointly 2-step opaque with type 1 secrets. Upon observing $e_{\mathrm{obs}} e_{\mathrm{obs}}$ we can deduce that A traversed the states $0, 1, 2, 2$ or $0, 3, 4, 2$ which both pass through secret states.

3.5.3 Verification of separate K-step opacity

Verification of separate K-step opacity is less straightforward than joint opacity. By the definition of separate opacity, it suffices to verify that the system is opaque for the specification $\varphi_{\mathrm{NS},j}(k))$ for each $k \leq K$ using the proposed language-based approaches. However, these tests can be combined into a single comparison as in the joint case by taking advantage of the structure of the specification automata. As each comparison involves determinizing a different automaton, this alternative

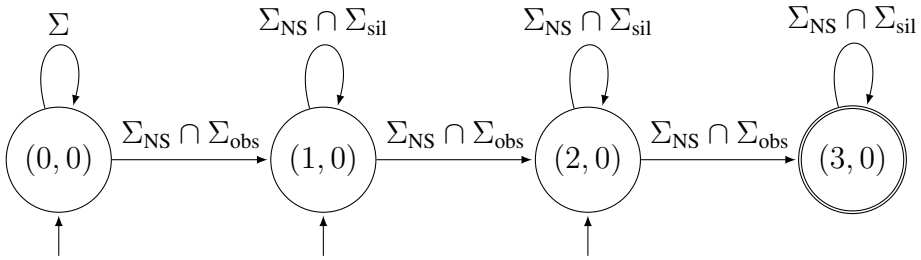

Figure 3.9: The nonsecret specification automaton $H^{\text{J}}_{\text{NS},1}(2)$ for 2-step joint opacity with type 1 secrets.

approach is significantly more efficient. By construction, $H_{\text{NS},j}(k)$ is embedded within $H_{\text{NS},j}(K)$ as a type of subautomaton for $k \leq K$. So we can use $H_{\text{NS},j}(K)$ to specify the nonsecret runs $\varphi_{\text{NS},j}(k)$ for $k \leq K$ for separate K-step opacity. As in Remark 3.3, we can write $H_{\text{NS},j}(k) = \bigcup\!\!\bullet_{i=0}^{k+1} H^i$ where $H^0 = H_*,\ H^1 = H_{\text{NS,ep},j}$, and $H^i = H_{\text{ep}}$ for $i \geq 2$. Recall using the convention of Remark 3.3, the marked states of $H_{\text{NS},j}(k)$ are simply the marked states of H^k denoted by $Q^{k+1}_{\text{NS},m}$ embedded into $H_{\text{NS},j}(k)$ as $Q^{k+1}_{\text{NS},m} \times \{k+1\}$. Hence it holds that $\mathcal{L}_{Q^{k+1}_{\text{NS},m} \times \{k+1\}}(H_{\text{NS},j}(K)) = \mathcal{L}_m(H_{\text{NS},j}(k))$. This yields the following approach.

Approach 3.5 (Separate K-step opacity verification using secret observer). *Given A, ℓ, E_{obs}, and $K < \infty$, construct the label-transform $G = \mathcal{T}(A, \ell)$, the nonsecret specification automaton $H_{NS,j}(K)$, and the static mask Θ induced by E_{obs}. Recall that A is separate K-step opaque with type j secrets if the k-delayed behavior with type j secrets is opaque for each $k \leq K$. We can verify this by applying the secret observer method for each $k \leq K$ to G, $H_{NS,j}(K)$, and Θ where we redefine the marked states of $H_{NS,j}(K)$ to be $Q^{k+1}_{NS,m} \times \{k+1\}$. Each of these tests involves analyzing the states of the same automaton $G_{SO} = \det(\Theta(G \times H_{NS,j}(K)))$ under different notions of state markings. As a result, we must only determinize a single automaton to apply this approach.*

However, the idea of this approach is not applicable to the reverse comparison method as this would require considering multiple sets of initial states. Alternatively, we can avoid multiple determinizations by utilizing the fact that the intruder's knowledge of the system's behavior only increases as they make more observations. Informally, if the intruder deduces a secret happened within the last $K-1$ observations, after making another observation they can still deduce a secret happened within the last K observations. So if the intruder can always make more observations, it suffices to consider secrets that occurred exactly K observations ago for the purposes of verification. This is similar to the results of Lemma 2 in [83]. We will show under some conditions that it suffices to verify the opacity of L to $\varphi_{\text{NS},j}(K)$ in order to verify separate K-step opacity with type j secrets. Here we say that A is **observation extendable** with respect to Θ if for every $s \in L = \mathcal{LIO}(A, \ell)$, there exists $s_{suf} \in \Sigma^*_{\text{sil}}(\Sigma \setminus \Sigma_{\text{sil}})$ so that $(s \cdot s_{suf}) \in L$. With this we claim the following result.

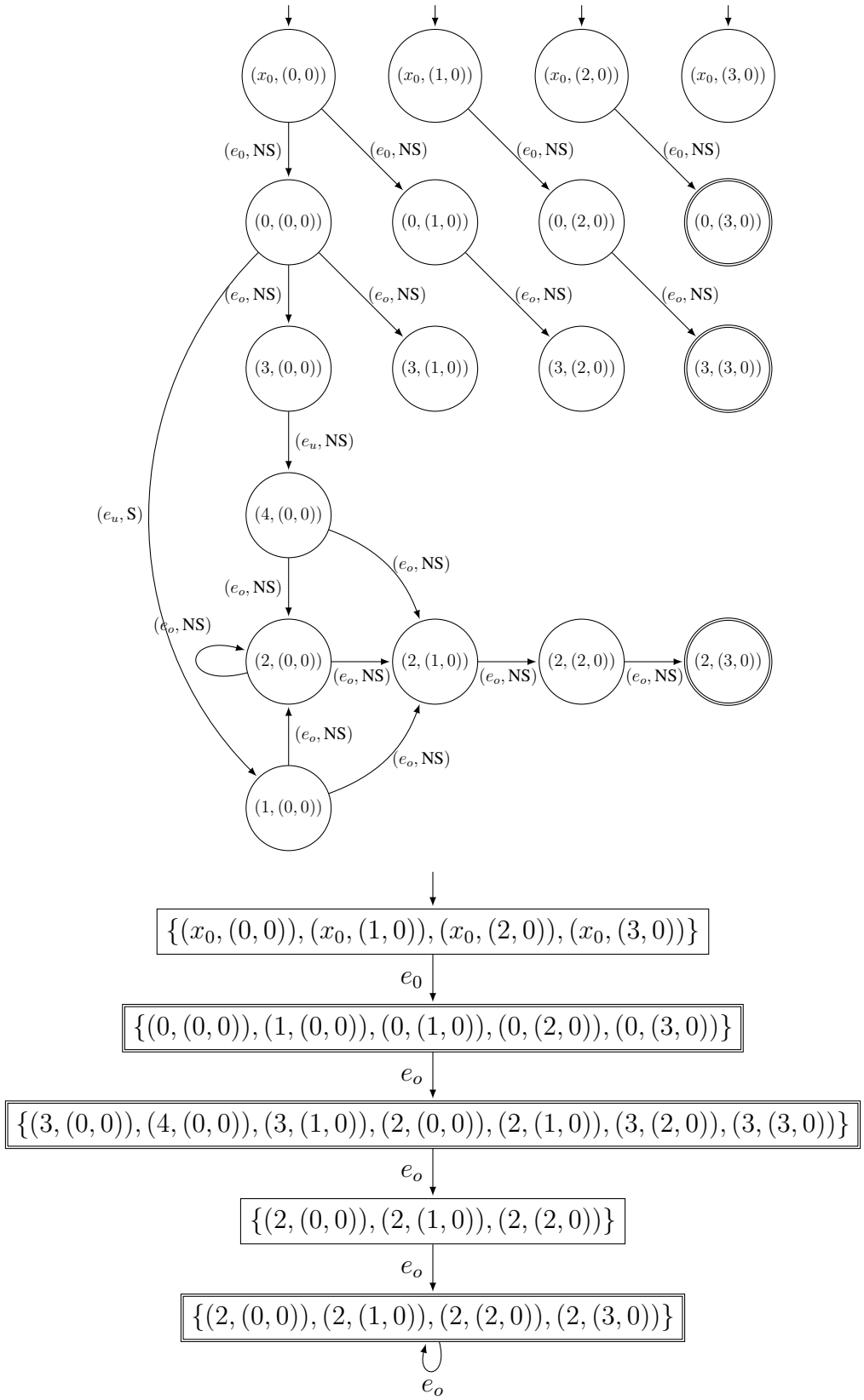

Figure 3.10: The product(top) of G from Fig. 3.2 with the nonsecret specification $H^{\text{J}}_{\text{NS},1}(2)$ and the corresponding secret observer G_{SO} (bottom).

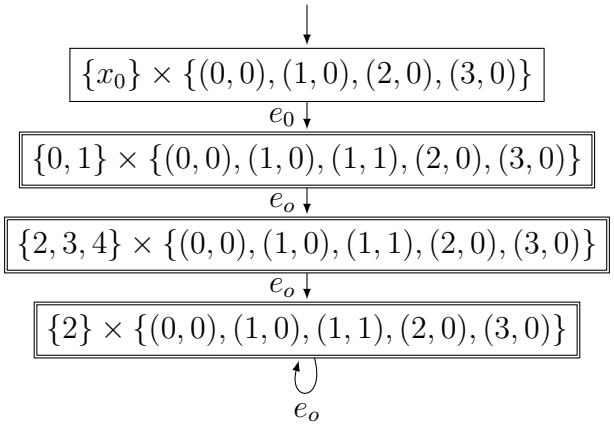

Figure 3.11: The secret observer G_{SO} constructed for the automaton A from Fig. 3.2 with the nonsecret specification $H_{\text{NS},2}(2)$.

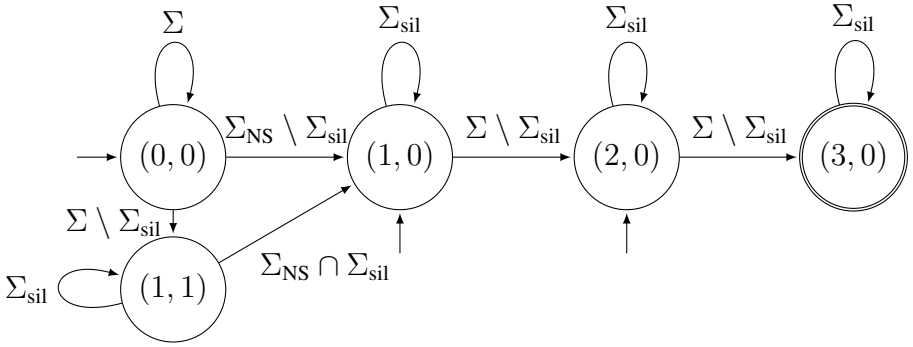

Figure 3.12: The nonsecret specification automaton $H_{\text{NS},2}(2)$ for separate 2-step opacity with type 2 secrets.

Theorem 3.4. *If A is observation extendable, then A is separate K-step opaque with type j secrets if and only if the system with behaviors $L = \mathcal{LIO}(A, \ell)$ and observation relation Θ induced by E_{obs} is opaque for $\varphi_{NS,j}(K)$.*

Proof. Suppose that A is separately K-step opaque with type j secrets. Let $s \in L$. By the separate opacity of A, there exists a run $s' \in L \cap \varphi_{\text{NS},j}(K)$ with $\Theta(s) = \Theta(s')$. Hence the system (L, Θ) is opaque to $\varphi_{\text{NS},j}(K)$ is opaque to Θ.

Conversely, suppose that (L, Θ) is opaque for $\varphi_{\text{NS},j}(K)$. Then let $s \in L$ and $k \in \{0, \cdots, K\}$. As L is observation extendable, there exists an extended run $s_{ext} = s \cdot s_{suf}$ so that $s_{ext} \in L$ and $|\Theta(s_{suf})| = K - k$. By hypothesis, there exists a run $s'_{ext} \in \varphi_{\text{NS}} = \varphi_{\text{NS},j}(K)$ with $\Theta(s'_{ext}) = \Theta(s_{ext})$. By defining s'_{suf} to be the last $K-k$ observation epochs of s'_{ext}, we can write $s'_{ext} = s' \cdot s'_{suf}$ with $|\Theta(r'_{suf})| = K - k$. Then we see that $s' \in \varphi_{\text{NS},j}(k)$ and $\Theta(s') = \Theta(s)$. Hence A is separately K-step opaque with type j secrets. □

So when the system is observation extendable, we can verify separate K-step opacity in the following way.

Approach 3.6 (Separate K-step opacity verification for observation extendable systems). *Given A, ℓ, E_{obs}, and $K < \infty$ where A is observation extendable with respect to the static mask Θ induced by E_{obs}, construct the label-transform $G = \mathcal{T}(A, \ell)$ and the nonsecret specification automaton $H_{NS,j}(K)$. We can verify the separate K-step opacity with type j secrets of A by applying any of the language-based approaches to G, Θ, and $H_{NS,j}(K)$.*

Remark 3.4. *While it may not be the case that A is observation extendable (for example if A is deadlocked), we can always modify A to be observation extendable while preserving K-step opacity. To do this we define a new automaton A_{ext} by adding an artificial observable event σ_{ext} as a self-loop for every state in A. Then one can show that A_{ext} will be separately K-step opaque if and only if A is. Then by construction L_{ext} will be observation extendable, and so we can apply Approach 3.6 to A_{ext}.*

Using Approach 3.6 we can verify separate K-step opacity using the reverse comparison or secret observer method. For example consider the system A from Fig. 3.2 which is observation extendable and the nonsecret specification automaton $H_{\text{NS},2}(2)$ which is depicted in Fig. 3.12. The resulting secret observer $G_{SO} = \det(\Theta(G \times H_{\text{NS},2}(2)))$ for $G = \mathcal{T}(A, \ell)$ is depicted in Fig. 3.11. As every state except the initial state is marked, we see that A is separately 2-step opaque with type 2 secrets.

3.6 Complexity of K-step opacity verification

In this section, we analyze the complexity of the proposed methods for verifying K-step opacity for finite K for an automaton A with labeling map ℓ. These methods use the transformed automaton $G = \mathcal{T}(A, \ell)$. First we analyze the secret observer using Approach 3.4 for joint opacity and Approach 3.5 for separate opacity. Then we analyze the reverse language comparison using Approach 3.6. Finally, we compare the secret observer methods to existing verifiers for K-step opacity known as the K-delayed state and trajectory estimators [31,86]. For separate K-step we also compare with the two-way observer method [53, 109]. These results are summarized in Table 3.2 and Table 3.3. Note we allow the automaton A to be nondeterministic in general, but require A to be deterministic when comparing with existing methods as they only consider deterministic automata.

3.6.1 Secret observer complexity

Recall that applying the secret observer method in Approaches 3.4, 3.5, or 3.6 to verify K-step opacity involves constructing the automaton $G_{SO} = \det(\Theta(G \times H_{\text{NS}}))$ for an appropriate choice of

H_{NS}. We will bound the number of reachable states in this automaton to bound state complexity of these verification approaches. A naive upper bound for the number of states in the power set construction for determinization of an automaton with n states is simply 2^n. Using the known structure of H_{NS}, we can obtain a tighter bound for determinizing the automaton $\Theta(G \times H_{\text{NS}})$. To do this, we will analyze which states of H_{NS} can be reached by runs that reach a fixed state of G in the following observation.

Observation 3.1. *Consider two automata* $G = (Q, \Sigma, \delta, Q_0, Q_m)$ *and* $H = (Q_{NS}, \Sigma, \delta_{NS}, Q_{NS,0}, Q_{NS,m})$ *with a static mask* $\Theta \subseteq \Sigma^* \times \Sigma^*$. *For convenience for* $s \in \Sigma^*$ *let* $\delta(s) = \bigcup_{q \in Q_0} \delta(q, s)$ *and* $\delta_H(s) = \bigcup_{q_H \in Q_{H,0}} \delta_H(q_H, s)$. *Suppose we are given sets* $F \subseteq 2^{Q_H}$ *and* $C \subseteq \Sigma^*$ *such that* F *is closed under union,* $\emptyset \in F$, *and for all* $s \in \mathcal{L}(G \times H)$ *such that* $\Theta(s) \in C$ *it holds that* $\delta_H(s) \in F$. *Then for every* $\gamma \in C$ *we can define the function* $w_\gamma : Q \to F$ *by*

$$w_\gamma(q) = \bigcup_{\substack{s \in obs^{-1}(\gamma) \\ s.t.\ q \in \delta(s)}} \delta_H(s) \tag{3.29}$$

Then denote the automaton $\Theta(G \times H)$ *As*

$$\Theta(G \times H) = (Q_{\Theta(G\times H)}, \Sigma \cup \{\epsilon\}, \delta_{\Theta(G\times H)}, Q_{0,\Theta(G\times H)}, Q_{m,\Theta(G\times H)}) \,. \tag{3.30}$$

For $\gamma \in C$ *it holds that*

$$\delta_{\Theta(G\times H)}(\gamma) = \bigcup_{s \in obs^{-1}(\gamma)} \delta(s) \times \delta_H(s) = \bigcup_{q \in Q} (\{q\} \times w_\gamma(q)) \,. \tag{3.31}$$

Hence the number of states in $\det(\Theta(G \times H))$ *reached by a string in* C *is bounded by the number of functions from* Q *to* F, *of which there are* $|F|^{|Q|}$.

We can apply this observation to bound the complexity of the secret observer method. As $G_{SO} = \det(\Theta(G \times H_{\text{NS}}))$ is deterministic, it has a single initial state reached by ϵ. So all states of G_{SO} other than the initial state are reached by $C = \Sigma^+$. Then to apply Observation 3.1, we must determine a set $F \supseteq \{\delta_H(s) \mid s \in C\}$ which is also closed under union and contains the empty set. We claim in verifying joint K-step opacity that

- for $H = H^{\text{J}}_{\text{NS},1}(K)$, we can choose $|F| = K + 3$ with

$$F = \{\emptyset\} \cup \{\{0, \cdots, k\} \times \{0\}\}_{k=0}^{K+1} \,, \tag{3.32}$$

- for $H = H^{\mathrm{J}}_{\mathrm{NS},2}(K)$, we can choose $|F| = 2(K+1)+1$ with

$$F = \{\emptyset\} \cup \{\{(0,0),(k+1,1)\} \cup (\{1,\cdots,k\} \times \{0,1\})\}_{k=0}^{K} \cup \{\{(0,0)\} \cup (\{1,\cdots,k+1\} \times \{0,1\})\}_{k=0}^{K}. \quad (3.33)$$

If we denote the number of states of the original automaton A as $n = |X|$, then the number of states of $G = \mathcal{T}(A,\ell)$ is $n+1$, including the artificial initial state. Observation 3.1 then shows the number of states of G_{SO} other than the initial state is bounded by $|F|^n$. These bounds are given by $(K+3)^n$ for $H^{\mathrm{J}}_{\mathrm{NS},1}(K)$ and $(2K+3)^n$ for $H^{\mathrm{J}}_{\mathrm{NS},2}(K)$. For separate opacity, we use the naive power set bounds of $2^{n(K+2)}$ for $H_{\mathrm{NS},1}(K)$ and $2^{n(K+3)}$ for $H^{\mathrm{J}}_{\mathrm{NS},2}(K)$. These bounds are summarized in Table 3.2 and Table 3.3.

3.6.2 Reverse comparison complexity

We can use the same approach to analyze the reverse comparison method as in Approach 3.4 and Approach 3.6 to verify K-step opacity. These approaches require constructing the automaton $G_R = \mathrm{rev}(\Theta(G)) \times \mathrm{comp}(\det(\mathrm{rev}(\Theta(G \times H_{\mathrm{NS}}))))$ for an appropriate choice of H_{NS}. By observing that $\mathrm{rev}(\Theta(G \times H_{\mathrm{NS}})) = \Theta(\mathrm{rev}(G) \times \mathrm{rev}(H_{\mathrm{NS}}))$, we can use Observation 3.1, to bound the number of reachable states of this automaton. For the nonsecret specification automata H_{NS} use for K-step opacity, the reachable sets of $\mathrm{rev}(H_{\mathrm{NS}})$ are simpler than H_{NS}. Consider a string $s \in \mathrm{rev}(\varphi^{+}_{\mathrm{ep}})$ with $k = \max(0, K+1-|\Theta(s)|)$. Using the notation from Remark 3.3, we can see that H_{NS} must reach a state corresponding to H^{k}_{NS}. Consider the set $C_k = \Sigma^{K+1-k}$ with $1 \le k \le K$ and $C_0 = \Sigma^{K+1}\Sigma^*$. Then we determine a set $F_k \supset \{\delta_{\mathrm{rev}(H_{\mathrm{NS}})}(s) \mid s \in C_k\}$ that is closed under union and contains the empty set. We claim that

- for $H_{\mathrm{NS}} = H^{\mathrm{J}}_{\mathrm{NS},1}$ or $H_{\mathrm{NS}} = H_{\mathrm{NS},1}$ we can choose $|F_k| = 2$ with

$$F_k = \{\{(k,0)\},\emptyset\}. \quad (3.34)$$

- for $H_{\mathrm{NS}} = H^{\mathrm{J}}_{\mathrm{NS},2}$ or $H_{\mathrm{NS}} = H_{\mathrm{NS},2}$ we can choose $|F_k| = 3$ with

$$F_k = \{\{(k,0)\},\{(k,0),(k,1)\},\emptyset\} \quad (3.35)$$

So by Observation 3.1 for C_k, the number of states of $\det(\Theta(\mathrm{rev}(G) \times \mathrm{rev}(H_{\mathrm{NS}})))$ reached by a string $\gamma \in \Sigma^+$ with $k = \max(0, K+1-|\gamma|)$ is bounded by $|F_k|^{n+1}$ where $n = |X|$ is the number of states in the original automaton A. Hence the number of states of $\det(\mathrm{rev}(\Theta(G \times H_{\mathrm{NS}})))$ is $O((K+1)2^n)$ for type 1 secrets and $O((K+1)3^n)$ for type 2 secrets. So then the number of states of

K	Forward ($n=4$)	Reverse ($n=4$)	Forward ($n=6$)	Reverse ($n=6$)
0	5	6	7	8
2	53	29	187	67
4	293	45	3007	147
8	2117	77	114487	275
16	16517	141	T/O	531

K	Forward ($n=4$)	Reverse ($n=4$)	Forward ($n=6$)	Reverse ($n=6$)
0	5	6	7	8
2	35	29	137	67
4	137	45	1547	147
8	749	77	36047	275
16	4949	141	1071767	531

Figure 3.13: The number of states in the forward secret observer automata $G_{SO}(n)$ and reverse automata $G_R(n)$ constructed from $G(n) = \mathcal{T}(A(n), \ell_n)$. The bottom table uses $H_{\text{NS}} = H^{\text{J}}_{\text{NS},1}(K)$ and the top table uses $H_{\text{NS}} = H_{\text{NS},2}(K)$. Here T/O denotes a timeout where the automaton could not be constructed.

$G_R = \text{rev}(\Theta(G)) \times \text{comp}(\det(\text{rev}(\Theta(G \times H_{\text{NS}}))))$ is $O(n(K+1)2^n)$ for $H = H_{\text{NS},1}(K), H^{\text{J}}_{\text{NS},1}(K)$ and $O(n(K+1)3^n)$ for $H = H_{\text{NS},2}(K), H^{\text{J}}_{\text{NS},2}(K)$. These bounds are depicted in Table 3.2 and Table 3.3. From these bounds, we see that the reverse comparison method is distinguished by the factor K entering linearly into the bound. This may indicate for systems with many states but small value of K, the reverse comparison method may be more efficient.

To demonstrate the advantage of the reverse language comparison, consider the following family of automata. For $n > 1$ define $A(n) = (X_n, E_n, f_n, X_{0,n}, X_{m,n})$ where $X_n = \{0, \cdots, n-1\}$, $E_n = \{e_0, \cdots, e_{n-1}\}$, $f_n(i, e_j) = (i + j) \bmod n$, $X_{0,n} = X_n \setminus \{0\}$, and $X_{m,n} = X_n$. Likewise, define the labels $A = \{\text{S}, \text{NS}\}$ with $\ell_n(0) = \text{S}$ and $\ell_n(i) = \text{NS}$ for $i \neq 0$. We define all events to be observable $E_{\text{obs}} = E_n$. After constructing $G(n) = \mathcal{T}(A(n), \ell_n)$ for various n, we compute the number of states in the secret observer automaton $G_{SO}(n) = \det(\Theta(G(n) \times H_{\text{NS}}))$ and in the reverse automaton $G_R(n) = \text{rev}(\Theta(G(n))) \times \text{comp}(\det(\text{rev}(\Theta(G(n) \times H_{\text{NS}}))))$ for $H_{\text{NS}} = H^{\text{J}}_{\text{NS},1}(K)$ and $H_{\text{NS}} = H_{\text{NS},2}(K)$ across various values of K. These results are depicted in Fig. 3.13. The number of states in the forward automata increases roughly exponentially with K while the number of states in the reverse automata increases linearly.

3.6.3 Comparison to K-delay State & trajectory estimators

We can compare our secret observer method with some existing methods for verification of K-step opacity. The first proposed verification methods for weak and strong K-step opacity are called the K-delay state estimator and K-delay trajectory estimator. These K-delay state estimators construct an automaton that estimates the possible states sequences over the last K observations

Separate Type 2 (Weak)	
Algorithm	State Complexity
Secret Observer	$O(2^{n(K+3)})$
Reverse Comparison	$O(n(K+1)3^n)$
State Estimator [86]	$O((\lvert E_{\text{obs}}\rvert+1)^K 2^n)$
Two-way Observer [109]	$O(\min(2^n, \lvert E_{\text{obs}}\rvert^K)2^n)$

Table 3.2: State complexities of verification methods for separate K-step opacity with type 2 secrets (weak K-step opacity) of an automaton with n states. The discussion in subsection 3.6.3 implies that the secret observer method has state complexity no worse than the K-delay state estimator.

from which one can deduce if weak opacity has been violated. The K-delay trajectory estimator augments this structure with a sequence of binary variables representing whether a secret state was visited between the observations to deduce if strong opacity has been violated. Their complexities are depicted in Table 3.2 and Table 3.3. By analyzing our Approach 3.5 for verifying weak K-step opacity, we see the resulting automaton estimates only the current state and whether a secret state has been visited during each of the last K observations. Likewise, the automaton from our Approach 3.4 for strong K-step opacity estimates the current state and whether only secret states have been visited during each of the last K observations. As we can deduce whether secret states have been visited from the possible sequences of past states, we can view our secret observer automata as quotients of the K-delay estimators. In this way, the complexity of our proposed methods are at most that of the K-delay estimators for the respective forms of K-step opacity. We have provided a formal proof of this result in longer version of this chapter [97].

To demonstrate this result, we construct a family of automata $A(i)$ where the secret observer method has significantly reduced complexity compared to the delayed state/trajectory method for verification of strong/weak K-step opacity. For $i > 1$ define the deterministic automaton $A(i) = (X_0, E_i, f_i, \{2\})$ where $X_0 = \{1, \cdots, i\}$, $E_i = \{\sigma_1, \cdots, \sigma_i\}$, $E_{\text{obs}} = E$, and the transition function defined by $f_i(j, \sigma_k) = k$. Consider the labeling map $\ell_i : X_0 \to A$ where $A = \{\text{S}, \text{NS}\}$ defined by $\ell_i(1) = \text{S}$ and $\ell_i(j) = \text{NS}$ for $j \neq 1$. Note that $A(i)$ recognizes a run along every state sequence in $\{2\} \cdot (X_0)^*$. Hence, we see the K-delayed state observer states correspond to $\bigcup_{k=0}^{K} \{2\} \times (X_0)^k$, of which there are $\sum_{k=0}^{K} i^k = \frac{1-i^{K+1}}{1-i} = O(i^K)$ states. Let $G(i) = \mathcal{T}(A(i), \ell_i)$. The secret observer $G_{SO}(i) = \det(\Theta(G(i) \times H_{\text{NS},2}(K)))$ estimates the current state and the secrecy of the past $K+1$ epochs. We can verify that the number of states in $G_{SO}(i)$ is $O(i2^K)$. So we see that the secret observer method can be significantly less complex than the delayed state estimator for verification of weak K-step opacity. A similar result holds for strong K-step opacity.

Joint Type 1 (Strong)	
Algorithm	State Complexity
Secret Observer	$O((K+3)^n)$
Reverse Comparison	$O(K2^n)$
Trajectory Estimator [31]	$O((\lvert E_{\text{obs}}\rvert+1)^K 2^n)$

Table 3.3: State complexities of verification methods for joint K-step opacity with type 1 secrets (strong K-step opacity) of an automaton with n states. The discussion in subsection 3.6.3 implies that the secret observer method has state complexity no worse than the K-delay trajectory estimator.

3.7 Infinite step opacity

Now we consider K-step opacity for $K = \infty$, also called infinite step opacity. The results of Theorem 3.2 can be extended to the infinite step case. In particular our notion of separate infinite-step opacity with type 2 secrets corresponds to the existing notion of infinite step opacity as in [85, 109]. We will discuss how the previous verification methods for finite K can be adapted to this case.

Recall our definition of infinite step opacity involves an infinite number of nonsecret language specifications, i.e. the k-delayed nonsecret behavior $L_{\text{NS},j}(k)$ for $k \in \mathbb{N}$ as defined in (3.18). Recall in the finite case we were able to reduce the multiple language comparison checks into a single check for verifying separate opacity. In Approach 3.5, we constructed one automaton that encompassed all of the nonsecret behavior, but this automaton would necessarily be infinite for $K = \infty$. In Approach 3.6, under the condition of observation extendability we showed it suffices to consider secret behavior occurring exactly K epochs ago, but there is no clear analog for this for $K = \infty$. Hence it appears that we cannot directly use our methods for verification of separate infinite step opacity. However we can use a result of [109] that states that infinite step opacity (separate opacity with type 2 secrets) is equivalent to K-step opacity for $K = 2^n$ where n denotes the number of states of the automaton in question. With this observation, we can verify separate infinite step opacity with type 2 secrets by verifying separate 2^n-step opacity. Alternatively, the two-way observer could be used to directly verify separate infinite step opacity [109].

We can more effectively apply our methods to joint infinite step opacity as this involves only one language comparison by definition. Note that we can define

$$\varphi^{\text{J}}_{\text{NS},j}(\infty) = \bigcap_{i=0}^{\infty} \varphi_{\text{NS},j}(i) = \varphi^{+}_{\text{NS,ep},j} \tag{3.36}$$

As in the finite case, we can construct an automaton to specify this nonsecret behavior. Consider the automata depicted in Fig. 3.14. To obtain a smaller complexity bound, we will apply the forward comparison method instead of the secret observer method. Recall in this case that we do not require

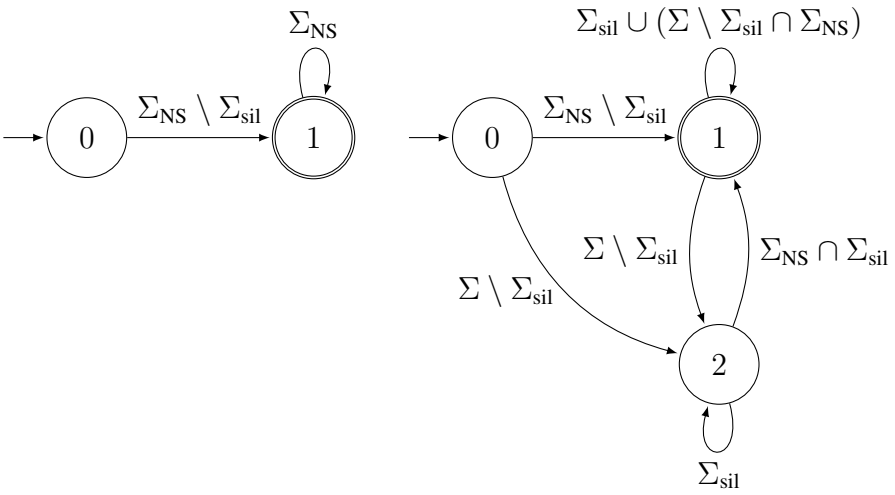

Figure 3.14: The nonsecret specification automata $H^{\text{J}}_{\text{NS},1}(\infty)$ (left) and $H^{\text{J}}_{\text{NS},2}(\infty)$ (right) for joint infinite step opacity.

$\mathcal{L}_m(H^{\text{J}}_{\text{NS},j}(\infty)) = \Sigma^*$. As before, we will analyze the complexity using Observation 3.1. Consider the set $C = \Sigma^+$. Then we determine a set $F \supset \{\delta_H(s) \mid s \in C_k\}$ that is closed under union and contains the empty set. We claim that

- for $H = H^{\text{J}}_{\text{NS},1}(\infty)$ we can choose $|F| = 2$ with

$$F = \{\emptyset, \{1\}\}\,. \tag{3.37}$$

- for $H = H^{\text{J}}_{\text{NS},2}(\infty)$ we can choose $|F| = 3$ with

$$F = \{\emptyset, \{2\}, \{1,2\}\} \tag{3.38}$$

So by Observation 3.1 for C, the number of states of $\det(\mathbf{obs}(G \times H_{\text{NS}}))$ reached by a string $\gamma \in \Sigma^+$ is bounded by $|F|^n$ with $n = |X|$ where $G = \mathcal{T}(A, \ell)$. Then the number of states in the automaton $G_F = \mathbf{obs}(G) \times \det(\mathbf{obs}(G \times H_{\text{NS}}))^c$ other than the initial state is $O(n2^n)$ for type 1 secrets and $O(n3^n)$ for type 2 secrets. To the best of our knowledge, verification of joint infinite step opacity has not been reported in the literature previously.

3.8 Results

We evaluate the effectiveness of our verification methods for K-step opacity with numerical experiments. We compare the time and space complexity of the proposed methods with existing methods for verifying the existing notions of strong and weak K-step opacity. Recall these

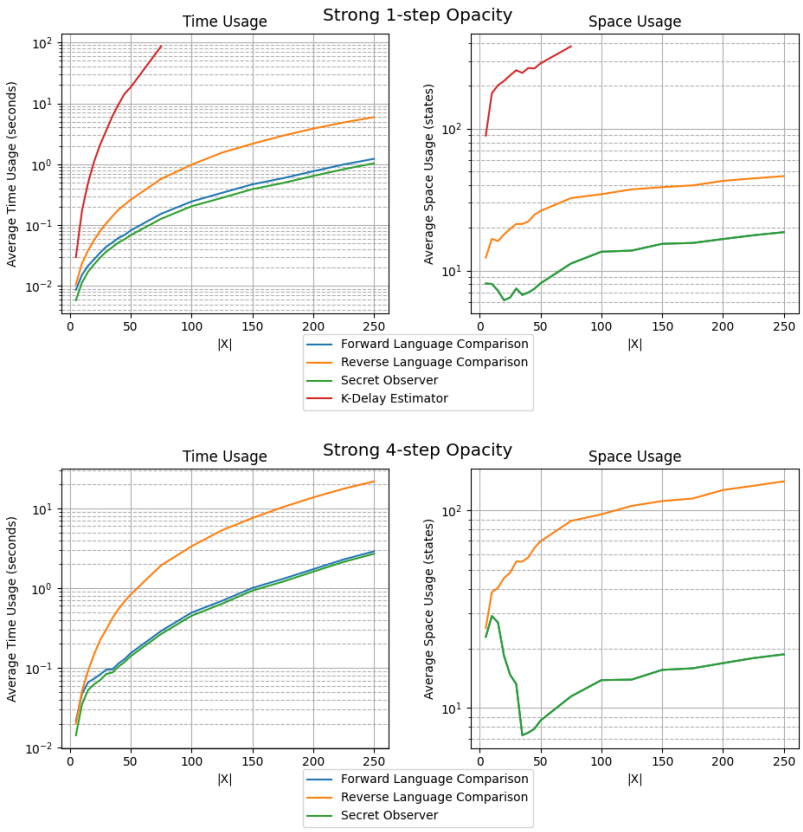

Figure 3.15: Plots of average runtime (time usage) and the number of states in the verifier automata (space usage) versus the number of states in the random automata system model ($|X|$) for several methods for verifying strong K-step opacity.

correspond to the notions of joint K-step opacity with type 1 secrets and separate K-step opacity with type 2 secrets, respectively. It should be noted while the existing methods were originally described for deterministic automata, there is a natural extension to the nondeterministic automata considered here. We compare the runtimes and number of states in the final verifier automata for an implementation of each method. In order to show how these methods scale with the size of the original system and the value of K, we verify the opacity of systems represented by randomly generated automata with secret states. We generate these automata in two ways. We present the runtimes and number of states in the verification automata averaged over 100 systems for fixed system sizes up to 250 states. These methods were implemented in the *MDESops* library [68].

3.8.1 First random generation approach

For the first experiment, we generate automata with a fixed number of states with a random number of outgoing transitions to random states. There are 18 events total with 6 observable events.

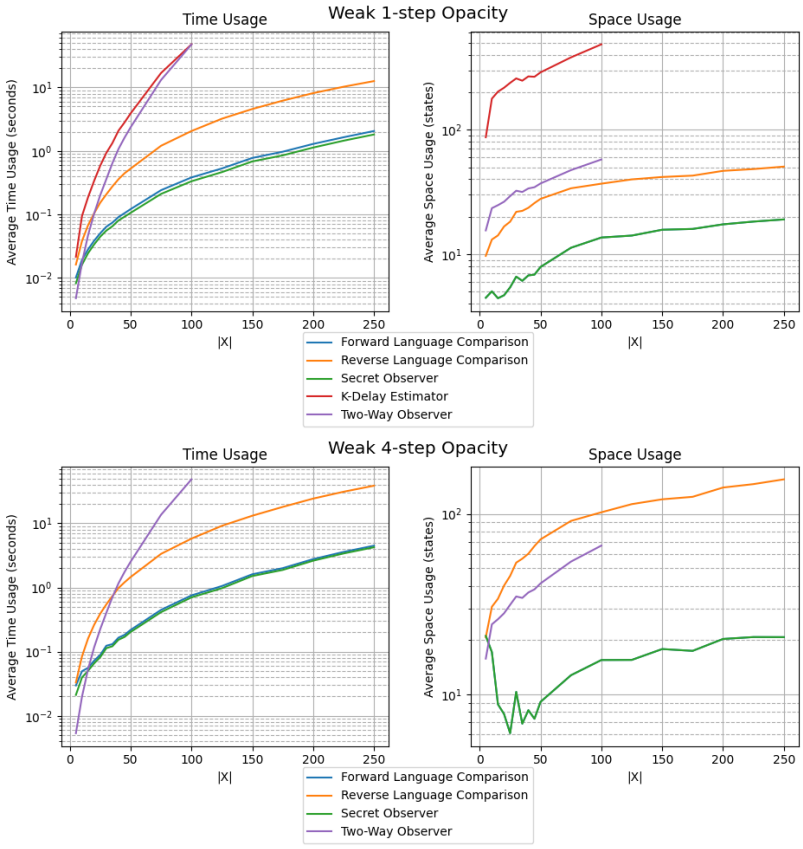

Figure 3.16: Plots of average runtime (time usage) and the number of states in the verifier automata (space usage) versus the number of states in the random automata system model ($|X|$) for several methods for verifying weak K-step opacity.

All states are considered to be initial, and one state is labeled as secret.

For strong K-step opacity, we compare the proposed forward comparison, reverse comparison, and secret observer methods with the existing K-delay trajectory estimator. We consider both $K = 1$ and $K = 4$. The average results over the randomly generated automata for verifying strong K-step opacity are depicted in Fig. 3.15. Due to the long runtime of the K-delay trajectory estimator ($> 100s$), we do not evaluate this method for large automata in the $K = 1$ case and remove it entirely in the $K = 4$ case. In these examples, the forward comparison method performed nearly identically to the secret observer method, which is why it does not appear in the space usage plots. From these plots, we see that the proposed methods for verification perform significantly faster than the existing method. This supports the claim of subsection 3.6.3, stating that the complexity of the secret observer method for verifying K-step opacity is less than that of the K-delay trajectory estimator. It is also interesting to note that the secret observer method outperforms the reverse language comparison for the small values of K investigated. This indicates the linear scaling with K in the complexity of this method is only significant for large values of K.

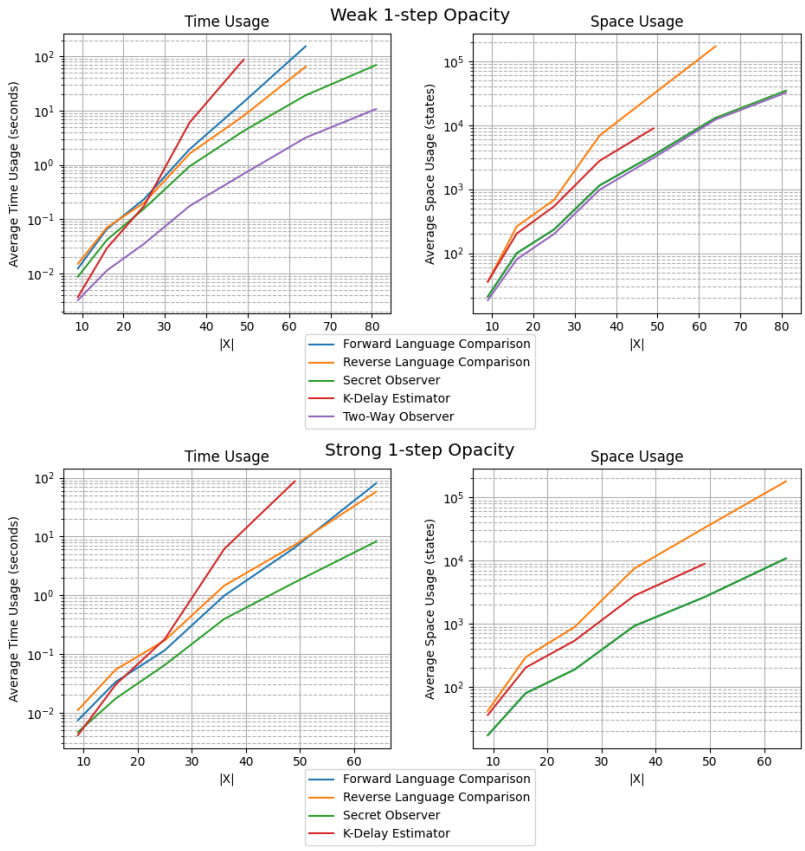

Figure 3.17: Plots of average runtime (time usage) and the number of states in the verifier automata (space usage) versus the number of states in the system model ($|X|$) for several methods for verifying strong and weak 1-step opacity for the grid-based automata.

For weak K-step opacity, we compare the proposed forward comparison, reverse comparison, and secret observer methods with the existing K-delay state estimator and the two-way observer [109]. For the secret observer method, Approach 3.5 is used, while for the forward and reverse comparison methods, Approach 3.6 is used. As in the strong case, we consider both $K = 1$ and $K = 4$. The average results over the randomly generated automata for verifying weak K-step opacity are depicted in Figure 3.16. Due to the long runtime of the K-delay state estimator and two-way observer in some cases, we omit these results when necessary. As in the strong case, the forward comparison method performed nearly identically to the secret observer method. From these plots, we see that the proposed methods for verification outperform the existing K-delay state estimator in average runtime and size in all cases. While the runtime in applying the two-way observer is smaller for small-sized automata, the secret observer method outperforms it on the average in time and space for larger automata (> 15 states). It should be noted that one property of this method for generating random automata is that for larger system sizes, nearly all of the automata generated were opaque for each notion of K-step opacity. We consider a more balanced and structured method

for generation next.

3.8.2 Second random generation approach (grid-based)

In the second experiment, we generate automata as a square grid where states can transition to the 4 adjacent states. These transitions are then randomly removed or labeled with a random event. The number of observable events and secret states are scaled logarithmically with the system size. Again, all states are considered initial. The generation of these automata was tuned to provide a balance of automata that were opaque and not opaque across all system sizes.

We present results for verifying weak and strong 1-step opacity in Fig. 3.17. These results show similar trends to the previous method for generating random automata. One notable difference is that the two-way observer method for verifying weak K-step opacity offers slightly improved performance over the proposed secret observer method.

3.9 Conclusion

While advances in algorithms verifying different notions of opacity offer improved performance in practice, such as those presented in this chapter, there remains a fundamental limit imposed by the inherent complexity of the verification problem (PSPACE-hardness).

We have presented several new results for the information-flow property of opacity in the context of discrete event systems. We presented a general framework of opacity to unify the many existing notions across a variety of system and intruder models. We used this framework to discuss notions of opacity over automata, both language-based and state-based. We provided several methods for verification of language-based opacity. We then developed a general approach for specifying state-based notions of opacity with automata and a transformation of these notions to language-based ones. Together, we used these results to describe existing notions of opacity like current-state opacity and initial-state opacity. We demonstrated how our approach unifies existing methods for opacity by showing the resulting language-based verification methods for these notions embody the existing verification methods. We further demonstrated the effectiveness of this approach in our investigation of K-step and infinite step opacity.

Using the intuition of K-step opacity with our approach, we derived a uniform view of four notions of K-step and infinite-step opacity. Two of these notions correspond to the existing notions of strong and weak K-step opacity, while the other two are new and meaningful notions. We developed appropriate specification automata for these notions, allowing verification with the language-based methods. We formally analyzed the complexity of these methods for K-step and infinite step opacity, showing these methods compare favorably in some instances to existing methods. In particular, we showed that the proposed secret observer method outperforms the

existing K-delay estimators for verifying strong and weak K-step opacity. Finally, we performed numerical experiments with randomly-generated automata to compare the verification methods. These results showed that the proposed verification methods offer increased performance over existing methods.

CHAPTER 4

Opacity Against Observers with a Bounded-Memory

4.1 Introduction

The practice of formal verification of cyber-physical systems requires many assumptions on the system itself and its environment. For example, in expressing privacy as opacity, we require that an observer cannot deduce sensitive information about the system's behaviors. In this setting it is typically assumed that the observer of the system has partial observation of behaviors but impose no other constraints. This hides the implicit assumption that the in the process of their deductions, observers have *perfect recall*, always deducing a secret correctly if possible. While this approach provides strong theoretical guarantees of privacy, it presents a number of challenges in practice. First, an observer may have limited computational resources to perform deduction, especially in an embedded setting. Second, the computational resources required to verify these privacy guarantees may be prohibitive as we must consider every possible way information may leak. Indeed, the aforementioned notions of opacity over automata are all readily transformed into one another [6, 97, 101], and the common verification problem is known to be PSPACE-complete [16]. This high complexity is observed in the poor exponential scalability of verification algorithms in practice.

In this chapter, we address these challenges by proposing a new notion of opacity reflecting an additional constraint on the observer: the amount of memory available to them. We characterize opacity from the observer's point of view, modeling their deductions with a nondeterministic automaton that marks observations deemed secret. In this form, we can impose a bound $k \in \mathbb{N}$ on the size of this automaton, representing the memory available to the observer. Our proposed notion of k-bounded memory opacity (k-BMO) requires that no such automaton exists. We establish basic properties of this notion, including that the verification problem is co-NP-complete, reduced from the PSPACE-completeness for LBO. In addition to these results, we develop a verification approach using an encoding into the Boolean satisfiability problem (SAT) and demonstrate it on a number of examples.

4.2 Problem Formulation

In this section, we present an alternative characterization of opacity from the viewpoint of the observer which we use to propose a new notion of opacity against observers with bounded memory. Formally, we consider a system modeled by an NFA $G = (Q, \Sigma, \delta, Q_0, Q_m)$ marking the system's behavior $L = \mathcal{L}_m(G)$ which is observed through an observation relation $\Theta \subseteq \Sigma^* \times \Sigma_{\text{obs}}^*$ induced by a static mask. Similarly, we assume an observer of this system also models the system with an NFA $\hat{G} = \left(\hat{Q}, \Sigma, \hat{\delta}, \hat{Q}_0, \hat{Q}_m\right)$ marking the nominal behavior $\hat{L} = \mathcal{L}_m(\hat{G})$ which is observed through another observation relation $\hat{\Theta} \subseteq \Sigma^* \times \Sigma_{\text{obs}}^*$ also induced by a static mask. Using the framework of Chapter 3, we will consider the language-based opacity of the system $\Delta = (L, \Theta)$ for the nominal system $\hat{\Delta} = (\hat{L}, \hat{\Theta})$ with a nonsecret specification φ_{NS}.

Recall that we refer to the observations $s \in \Theta(L)$ *violating* if $s \notin \Theta(\hat{L} \cap \varphi_{\text{NS}})$, and that opacity corresponds to the inability of the observer to deduce an observation was violating. In deducing if an observation s is violating, they attempt a version of the regular language acceptance problem, i.e., is s an element of $\hat{\Theta}(\hat{L} \cap \varphi_{\text{NS}})$? We will formulate a new notion of opacity capturing a restriction on the algorithms the observer uses to solve this problem, namely the amount of available *memory states*. As noted in [58], these algorithms can be viewed as passive *attacks* on the system which do not alter the system's behavior. We adopt this terminology, and model these attacks with an NFA $A = (Q_A, \Sigma_{\text{obs}}, \delta_A, Q_{A,0}, Q_{A,m})$ which tracks observations of the system G and marks some which are violating. We can use this notion of attacks to provide an alternative characterization of opacity.

Proposition 4.1. *The system $\Delta = (L, \Theta)$ is* not *opaque for nominal system $\hat{\Delta} = (\hat{L}, \hat{\Theta})$ and nonsecret specification φ_{NS} if and only if there exists an attack NFA A with the following properties*

1. Correctness*: The attack marks only violating observations*

$$\mathcal{L}_m(A) \cap \hat{\Theta}(\hat{L} \cap \varphi_{NS}) = \emptyset\,. \tag{4.1}$$

2. Nontriviality*: The attack marks some observation*

$$\mathcal{L}_m(A) \cap \Theta(L) \neq \emptyset\,. \tag{4.2}$$

Proof. If there exists a correct and nontrivial attack A, then there exists a string $s \in \mathcal{L}_m(A)$ contained by $\Theta(L)$ but not by $\hat{\Theta}(\hat{L} \cap \varphi_{\text{NS}})$. Hence s is violating and thus G is not LBO. Conversely if G is not LBO, then there exists such a violating string. Thus the attack A marking this single string is necessarily correct and nontrivial. □

Note that we do not require attacks to deduce *all* violating observations, just one. Indeed, due to nondeterminism an attack can be correct and nontrivial even if it rejects a violating observation

on one run but accepts it on another. Because of this, the proposed memory bound is not directly related to the standard notion of space complexity for the regular language acceptance problem. Alternatively, the smallest attack which deduces every violating observation can be computed by applying state minimization to the forward comparison automaton G_F constructed in Section 4.4. The number of states in an attack automaton A represents the number of memory states utilized by a corresponding nondeterministic deduction algorithm. We can then define a notion of opacity capturing a restriction on the memory available to an observer as a bound on the number of such states.

Definition 4.1. *Given $k \in \mathbb{N}$, system Δ, nominal system $\hat{\Delta}$, and nonsecret specification φ_{NS}, we say that the system is* k-bounded memory opaque *(or k-BMO for short) if there is* no *correct and nontrivial attack A with k states.*

4.2.1 Properties of Bounded Memory Opacity

We now observe a number of simple properties about this notion of opacity.

Proposition 4.2. *Consider the setting of Definition 4.1.*

1. *If Δ is $(k+1)$-BMO, then Δ is k-BMO.*
2. *If Δ is LBO, then Δ is k-BMO.*
3. *There exists a $k \in \mathbb{N}$ so that if Δ is k-BMO, then Δ is LBO.*

Proof.

1. We can add unreachable states to an attack without altering its correctness or nontriviality.
2. If Δ is LBO, then any correct attack cannot be nontrivial.
3. If Δ is not LBO, then the forward comparison automaton with marked states swapped to accept secret observations is a correct and nontrivial attack. □

In general, we are interested in the smallest bound k for which a system is k-BMO or equivalently, the size of *minimal* attacks. While we can always construct attacks recognizing the smallest violating observation or attacks that are deterministic, the following examples show that such attacks may not be minimal.

Example 4.1. *Consider the DFA G depicted in Fig. 4.1 with $M+1$ states and all events observable. We assume the eavesdropper knows the system $\hat{\Delta} = \Delta$, and consider the nonsecret specification given by $\varphi_{NS} = \mathcal{L}_{Q_{NS}}$ using the indicated nonsecret states Q_{NS}. Then the violating strings are*

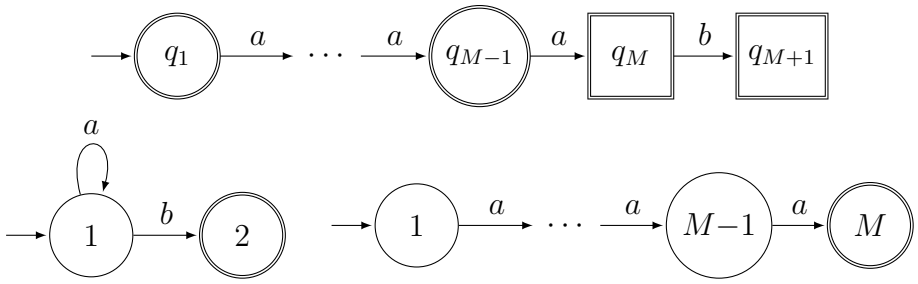

Figure 4.1: A system NFA G (top) and corresponding attack NFAs A (bottom left) and A' (bottom right). States $Q_{\mathrm{S}} = \{q_M, q_{M+1}\}$ act as secret states in G as in Example 4.1 while the remaining states $Q_{\mathrm{NS}} = Q \setminus Q_{\mathrm{S}}$ are nonsecret.

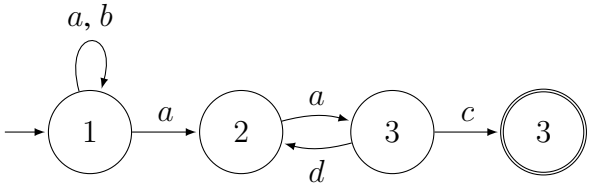

Figure 4.2: A nondeterministic attack A from which we construct the system G in Example 4.2.

$s_1 = a^{M-1}$ *and* $s_2 = a^{M-1}b$*. The attack* A *depicted in Fig. 4.1 is minimal, marking the string* s_2*. Furthermore, it is clear that any correct attack marking* s_1*, such as* A *depicted in Fig. 4.1, must have at least* M *states, counting the occurrences of* a*. So the shortest violation of opacity may not always correspond to a minimal attack.*

Example 4.2. *Consider the nondeterministic attack* A *with* 4 *states depicted in Fig. 4.2. We will construct a system for which this attack is minimal. Let* G' *denote the complement of the determinization of* A *with all unmarked states removed. As a result* $\mathcal{L}_m(G') = \mathcal{L}(G')$ *contains strings whose prefixes are not in* $\mathcal{L}_m(A)$*, i.e.,* $\mathcal{L}(G') \cap \mathcal{L}_m(A) = \emptyset$*. Let* G *denote the union NFA construction for* G' *and an automaton generating the secret string* $abaadac \in \mathcal{L}_m(A)$ *and marking its strict prefixes. We do not depict* G *here due to its large size. We suppose that all events are observable and that the eavesdropper knows the system model* $\Delta = \hat{\Delta}$*. By construction,* A *is correct and nontrivial, yet applying the verification method developed later in Section 4.4, we determine that there is no deterministic attack with size* 4*. So in general, there may be no minimal attack that is deterministic.*

4.3 Problem Complexity

In this section, we discuss the problem of verifying k-BMO. We show this problem is co-NP-complete, or equivalently, that falsifying k-BMO by synthesizing an attack is NP-complete. In addition, we present a verification approach based upon an encoding into SAT. Formally, we state

the problem of verifying k-BMO as follows.

Problem 4.1. *Given NFAs G and $\hat{G}$ marking the behaviors L and $\hat{L}$, observation relations Θ and $\hat{\Theta}$ given by static masks, an NFA H_{NS} marking the language φ_{NS}, and a $k \in \mathbb{N}$, determine if the system $\Delta = (L, \Theta)$ is k-BMO for the nominal system $\Delta = (\hat{L}, \hat{\Theta})$ and nonsecret specification φ_{NS}.*

4.3.1 Verifying k-BMO is co-NP

It is well-known that the complexity of verifying opacity in general is PSPACE-complete [16]. By relaxing the requirement that an observer never deduces a secret, i.e., LBO, to the requirement that one with a bounded memory never does, i.e., k-BMO, we reduce the complexity of the verification problem.

Theorem 4.3. *Verifying k-BMO is* co-NP.

Proof. Let G, $\hat{G}$, Θ, $\hat{\Theta}$, H_{NS}, and k be as in Problem 4.1 which serve as input to the verification problem. Let n and m denote the number of states and events of G, respectively. Likewise, let $\hat{n}$ and n_{NS} denote the number of states $\hat{G}$ and H_{NS}, respectively. To show the problem is co-NP, it suffices to show that we can check if an attack A with size k (serving as a certificate) is correct and nontrivial in polynomial time. Using properties of the parallel composition, A is correct if the following equivalent conditions hold

$$\mathcal{L}_m(A) \cap \hat{\Theta}(\hat{L} \cap \varphi_{\text{NS}}) = \emptyset \;\Leftrightarrow\; \mathcal{L}_{Q_{A,m} \times \hat{Q}_m \times Q_{\text{NS},m}}(A \times \hat{\Theta}(\hat{G} \times H_{\text{NS}})) = \emptyset\,. \tag{4.3}$$

Likewise, A is nontrivial if the following equivalent conditions hold

$$\mathcal{L}_m(A) \cap \Theta(L) \neq \emptyset \;\Leftrightarrow\; \mathcal{L}_{Q_{A,m} \times Q_m}(A \times \Theta(G)) \neq \emptyset\,. \tag{4.4}$$

The language of an NFA is nonempty if and only if its marked states are reachable from initial ones, which can be checked using breadth-first search in linear time in the number of edges. So the first condition can be checked over $A \times \hat{\Theta}(\hat{G} \times H_{\text{NS}})$ with time

$$O\left((|Q_A| \cdot |\hat{Q}| \cdot |Q_{\text{NS}}|)^2 |\Sigma|\right) = O(k^2 \hat{n}^2 n_{\text{NS}}^2 m)\,.$$

Similarly, the second condition can be checked over $A \times \hat{\Theta}(G)$ with time

$$O\left((|Q_A| \cdot |Q|)^2 |\Sigma|\right) = O(k^2 n^2 n^2 m)\,.$$

So the total time complexity is $O\left(k^2(\hat{n}^2 n_{\text{NS}}^2 + n^2)m\right)$. By convention, we represent k in unary so that its representation as an input is proportional to the value of k. In this case, we see that checking

that an attack is correct and nontrivial can be done in polynomial time. □

While we show next that this problem is co-NP-complete, the result of Theorem 4.3 is significant as such problems can often be solved in practice with SAT solvers. To demonstrate this, we develop a SAT encoding for verification in Section 4.4 whose performance is evaluated in Section 4.5.

Remark 4.1. *While we have proposed a bound on the memory of a potential attacker, one may instead consider verifying opacity over strings with a bounded length, similar to the concept of* bounded-model checking *[21]. While such methods can be very efficient using symbolic techniques [69], it is not immediately clear how long the strings considered must be in order to achieve some privacy requirement. As demonstrated in Fig. 4.1, there may a simple attack to deduce a secret occurred while the shortest violating string is arbitrarily long (bounded by the number of states in the system). However, we can note for any attack A, a minimal string marked by the attack will only visit the states of the product $A \times \Theta(G) \times \hat{\Theta}(\hat{G} \times H_{NS})$ at most once. Hence letting $k = |A|$, $n = |G|$, $\hat{n} = |\hat{G}|$, $n_{NS} = |H_{NS}|$, if there are no strings violating opacity with length $(kn\hat{n}n_{NS}) - 1$, the system is k-BMO.*

4.3.2 Verifying k-BMO is co-NP-Complete

To show that verifying k-BMO is co-NP-complete, we adapt the proof of PSPACE-completeness for verifying LBO [16]. This proof constructs a reduction from the universality problem which asks if an NFA G marks every string, i.e., $\Sigma^* \subseteq \mathcal{L}_m(G)$? Without loss of generality, we may assume that $\mathcal{L}(G) = \Sigma^*$. We consider the system $\Delta = (L, \Theta)$ with $L = \mathcal{L}(G)$ and Θ defined by all events being observable. Furthermore, we suppose the nominal system $\hat{\Delta}$ is equal to the true system Δ, and the nonsecret specification is simply $\varphi_{\text{NS}} = \mathcal{L}_m(G)$. In which case, Δ is LBO if and only if G is not universal. This completes the reduction from the universality problem which is known to be PSPACE-complete [89]. To show the NP-completeness of our problem, we consider a variant of the universality problem over bounded strings. The *bounded nonuniversality* problem asks for an NFA G and bound $n \in \mathbb{N}$, does G not mark all strings of length at most n, i.e. $\Sigma^{\leq n} \not\subseteq \mathcal{L}_m(G)$? We now present a reduction to falsifying k-BMO from the bounded nonuniversality problem which is known to be NP-complete [20]. This reduction and the original PSPACE reduction are depicted in Fig. 4.3.

Theorem 4.4. *Verifying k-BMO is co-NP-complete.*

Proof. Consider an NFA G' and bound $n \in \mathbb{N}$ represented in unary which serve as inputs to the problem. Let $G_{\leq n}$ denote an automaton with $n + 1$ states generating all strings with length at most n and marking none of them. Let G be the union automaton of G' and $G_{\leq n}$ restricted to strings of length n so $\mathcal{L}(G) = \Sigma^{\leq n}$ and $\mathcal{L}_m(G) = \mathcal{L}(G') \cap \Sigma^{\leq n}$. By construction, G is nonuniversal with bound n if and only if there exists a string $s \in \mathcal{L}(G)$ but $s \notin \mathcal{L}_m(G)$, i.e., s is violating. We consider

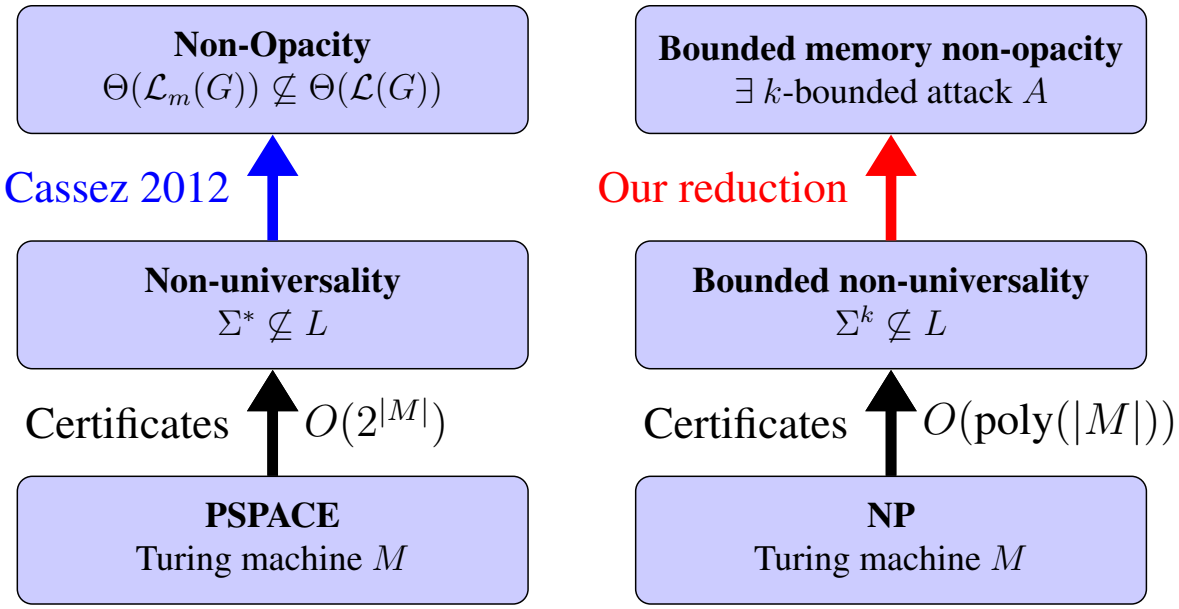

Figure 4.3: The reductions proving the PSPACE-hardness of falsifying CSO (left) and the NP-hardness of falsifying k-BMO (right).

the same setting of LBO for the PSPACEreduction described above. As we can construct an attack with $n+1$ states that only marks s, we see G' is nonuniversal with bound n if and only if G is k-BMO for $k = n+1$. As G may be constructed in polynomial time, this procedure describes a reduction from the bounded nonuniversality problem to falsifying bounded memory opacity. □

Remark 4.2. *To show the general universality problem is* PSPACE*-complete, [89] construct an NFA marking invalid computations of a nondeterministic Turing machine with a polynomial space bound. Due to this polynomial space, nontrivial computations may be exponentially long in the input. In the reduction, this corresponds to the shortest violations of opacity that exist being exponentially long in the size of the system. Similarly, to show the bounded nonuniversality problem is* NP*-complete, [20] perform a similar construction for nondeterministic Turing machines running in polynomial time. As a result of this time bound, the length of computations is polynomial in the input and the bound. This corresponds to the fact that the shortest violations of opacity marked by an attack are polynomial in the size of the system and attack, rather than exponential like in the general case.*

4.4 Verification Approach

In order to verify k-BMO effectively, we can express it as a Boolean satisfiability problem. That is we can encode an attack A with propositional variables and develop constraints modeling

correctness and nontriviality. The overall approach is depicted in Fig. 4.4. Formally given some $k \in N$, we consider an attack A with states $Q_A = \{0, \cdots, k-1\}$. Without loss of generality, we assume the initial state is 0 and that the attack has a single marked state given by $k-1$. For simplicity, we describe the encoding for current-state opacity which can be extended to the more general case. In particular we assume all states of G are marked, the nominal model $\hat{\Delta}$ is equal to the true system Δ, and the nonsecret specification is simply $\varphi_{\text{NS}} = \mathcal{L}_{Q_{\text{NS}}}(G)$ for some set of nonsecret states $Q_{\text{NS}} \subseteq Q$. We introduce the variables $\tau_A(q_A, \sigma, q'_A)$ meaning the corresponding transition is present in A, i.e., $(q_A, \sigma, q'_A) \in \delta_A$. From equations (4.3)-(4.4), correctness and nontriviality of A correspond to language emptiness/nonemptiness in the composition $A \mid\mid G$. In order to encode this composition, we encode the observability of an event $\sigma \in \Sigma$ with a formula $O(\sigma)$. Then the presence of a transition in the composition from (q_A, q) to (q'_A, q') over event σ where $(q, \sigma, q') \in \delta$ is given by the formula $\tau(q_A, q, \sigma, q'_A, q')$ defined by

$$(\neg O(\sigma) \wedge (q_A = q'_A)) \vee (O(\sigma) \wedge \tau_A(q_A, \sigma, q'_A)) \,. \tag{4.5}$$

To encode language emptiness, we recall that the language marked by an automaton is empty if and only if its marked states are not reachable from the initial state in the underlying graph. We can represent reachability in the composition with the variables $R(q_A, q)$, whose truth indicates that $(q_A, q) \in Q_A \times Q$ is reachable from the initial state $(q_{A,0}, q_0)$. To encode reachability in SAT, we use constraints similar to [74] based upon acyclicity over auxiliary variables T. These constraints require the initial state to be reachable, i.e., $R(q_{A,0}, q_0)$, and for all other states (q_A, q) that

$$R(q'_A, q') \leftarrow \bigvee_{\substack{\sigma \in \Sigma, q_A \in Q_A \\ (q, \sigma, q') \in \delta}} R(q_A) \wedge \tau(q_A, q, \sigma, q'_A, q') \tag{4.6}$$

$$R(q'_A, q') \rightarrow \bigvee_{\substack{\sigma \in \Sigma, q_A \in Q_A \\ (q, \sigma, q') \in \delta}} R(q_A) \wedge \tau(q_A, q, \sigma, q'_A, q') \wedge T(q_A, q, q'_A, q') \tag{4.7}$$

$$\text{Acyclic}(T) \,. \tag{4.8}$$

Constraint (4.6) ensures R is true for reachable states while constraints (4.7)-(4.8) ensure R is true only for reachable states.

Here, the constraint $\text{Acyclic}(T)$ denotes a formula that is satisfied when the graph over nodes $Q_A \times Q$ with edges encoded by T is acyclic. We can think of this graph as a spanning tree of the reachable set rooted at the initial state. Critically, this formulation for reachability results in a total number of constraints that is linear in the number of transitions in the composed automaton (viewing acyclicity as a single constraint which is natively supported by solvers like [74]). From

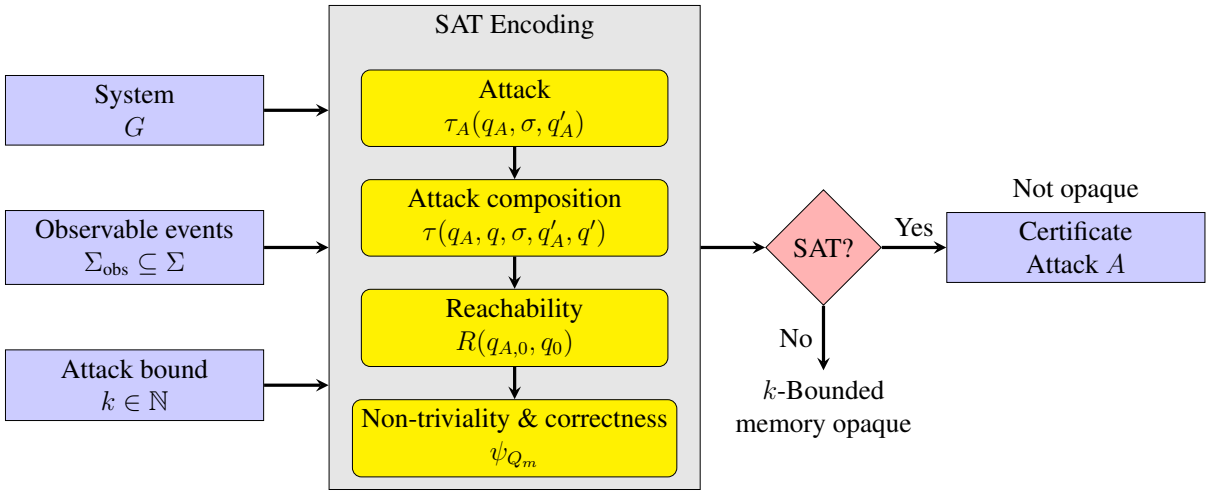

Figure 4.4: The proposed verification approach for k-BMO utilizing the SATencoding.

equations (4.3)-(4.4), we see the encoded attack is correct and nontrivial if the following constraint is satisfied

$$\psi_{Q_m} = \bigwedge_{q \in Q_m} \neg R(k{-}1,\, q) \wedge \bigvee_{q \in Q \setminus Q_m} R(k{-}1,\, q), \tag{4.9}$$

where $k{-}1$ is the marked state of A. Then there exists an attack A encoded by these variables that is correct and nontrivial, i.e., G is not k-BMO, if and only the constraints (4.5)-(4.9) are satisfiable. Furthermore, the total number of constraints is $O(k^2n^2m)$ where $n = |Q|$ and $m = |\Sigma|$. This formulation of verification as a constraint satisfaction problem has many advantages. In particular, it is easy to incorporate extensions or additional constraints on the attacks as demonstrated in the following section.

4.5 Results

In this section, we investigate the performance of the proposed SAT encoding for verifying k-BMO. We first demonstrate its superior scalability in comparison to a standard approach for verifying LBO on randomly generated automata. We then present an example showing how the SAT encoding can be easily extended to solve more general problems. This example system models server load-balancing with quantitative constraints on observations available to an attacker.

4.5.1 Comparing Opacity Verification Methods

We compare an implementation[1] of the proposed SAT encoding for verifying k-BMO with a standard method for verifying LBO based upon constructing the observer (the NFA G is LBO if all

[1]Implementation available at `https://gitlab.eecs.umich.edu/M-DES-tools/bounded-opacity`

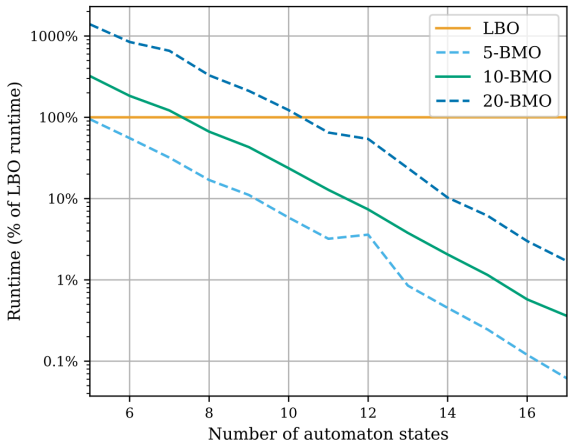

Figure 4.5: The runtimes to verify different notions of opacity as a percentage of the runtime to verify LBO.

states in its observer are marked). In particular, we encode the constraints for verifying k-BMO developed in Section 4.4 into the solver GraphSAT [74] which natively supports the acyclicity constraint (4.8). We evaluate the runtime of both implementations on randomly generated automata. For a given number of states n and a fixed number of events $m = 10$, transitions are included in the automata independently with a fixed probability. The probability is selected such that the expected number of transitions is $2nm$ where the exponential blowup of the observer construction is encountered [92]. We do this to demonstrate for a fixed bound k that the proposed method performs well in the worst-case scenario for verifying LBO.

The resulting runtimes for verification were averaged over 30 instances[2] for each size n ranging from 5 to 17. As the absolute runtimes are sensitive to details of each implementation, we depict the verification runtimes as a percentage of the time to verify LBO in Fig. 4.5. We observe that for a fixed k, this ratio for verifying k-BMO decreases steadily indicating that the proposed method, while exponential itself, scales exponentially slower with automata size than the observer construction for LBO. While there are more efficient approaches to verify LBO such as modular [54] or antichain-based methods [27], it is likely similar trends will exist in comparison to verifying k-BMO due to the different complexity classes.

4.5.2 Server Load Hiding

Many cyber-physical systems operate, in part, over the public Internet, using remote servers to offer critical services. Security for these critical services is often based on *location hiding*,

[2]The SAT solver failed to terminate on 11 out of the 1170 of instances within a five minute timeout which are not included in our analysis. We note that the unpredictability of the runtime of SAT solvers presents a limitation to our approach in practice.

e.g., hiding the true address of a server behind a network of proxies [52]. While this can provide protection against distributed denial of service (DDoS) attacks, resourceful attackers can learn the structure of simple, static networks to bypass these measures [50, 93]. When the server location cannot be hidden, it may be desirable to instead hide which servers are under a heavy load as such servers are attractive targets for DDoS attacks.

We consider the problem of verifying that these loads are hidden in the simplified load-balancing system depicted in Fig. 4.6 in which users send requests to the balancer which then assigns these requests to servers. We model the load balancer with a DFA G_L that arbitrarily assigns requests to available servers. The overall system G is given by the parallel composition of the load balancer G_L with n_U users and n_S servers with a capacity $C \in \mathbb{N}$ modeled by the DFAs $G_{U,i}$ and $G_{S,j}$ depicted in Fig. 4.7

$$G = G_{U,1} \parallel \cdots \parallel G_{U,n_U} \parallel G_{S,1} \parallel \cdots \parallel G_{S,n_S} \parallel G_L \,. \tag{4.10}$$

We will extend the SAT constraints developed in Section 4.4 to encode opacity against attackers that can compromise user devices with a k-bounded memory. Formally, if the attacker has compromised user i, the events $\mathbf{req}_i$ and $\mathbf{res}_i$ become observable. We can incorporate this choice of observability for event σ by viewing $O(\sigma)$ from constraint (4.5) as a decision variable. The attacker then aims to solve a kind of *optimal sensor placement* problem, choosing users i to observe with uniform cost $c_i = 1$. By incorporating the constraints for k-BMO in the MAX-SAT framework, we can model these costs with soft clauses $\neg(O(\mathbf{req}_i) \vee O(\mathbf{res}_i))$ with weight c_i. We require that the attacker cannot deduce that a specific server is heavily loaded, i.e., at secret state C. To model this, we let $Q_{m,j}$ denote states of G where server j does not pass through state C. Then, we can encode opacity with respect to all of the server secrets by replacing the constraint ψ_{Q_m} from (4.9) in the SAT encoding with the constraint $\psi = \bigwedge_{j=1}^{n_S} \psi_{Q_{m,j}}$.

By solving the resulting instance of MAX-SAT, we can determine the minimum cost of an attack with size k (if one exists) as the total weight of the solution clauses. This corresponds to the number of users that must be compromised to deduce when a server is heavily loaded. We report the results for solving this problem over a variety of parameters in Table 4.1. As we would expect, there must be sufficiently many users for an attack to exist, i.e. $n_U \geq C n_S$. Interestingly, the size of an attack may be smaller than the number of users that it monitors. Similar to Example 4.1, the minimum cost attack in the first system utilizes the events of all 5 users but itself has only 4 states.

4.6 Conclusion

In this chapter, we have presented a new notion of opacity expressing privacy from an observer with a bounded memory. We derived a number of its basic properties, including the co-NP-completeness of its verification problem. We demonstrated the applicability of this notion on a

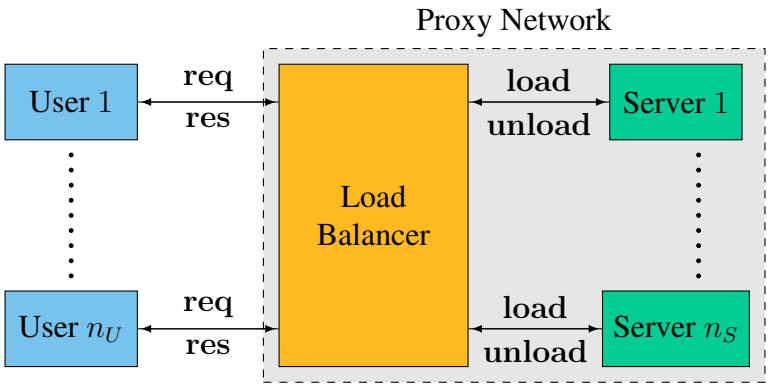

Figure 4.6: The architecture of the load-balancing system.

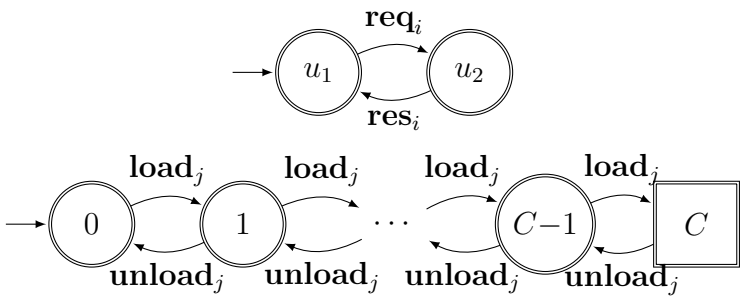

Figure 4.7: The automata $G_{U,i}$ (top) modeling user i and $G_{S,j}$ (bottom) modeling server j.

n_U	n_S	C	$\lvert Q\rvert$	k	Time(s)	#Clauses	Opaque	Cost
5	1	5	248	4	20.3	3.5×10^6	No	5
5	1	6	248	5	29.4	6.9×10^6	Yes	n/a
4	2	2	419	3	22.3	3.9×10^6	Yes	n/a
4	2	2	419	4	55.5	9.7×10^6	No	3

Table 4.1: Verification results for the server load-hiding system.

number of experiments utilizing a SAT encoding for verification.

CHAPTER 5

Enforcement of Opacity with Obfuscation

5.1 Introduction

Oftentimes, it can be verified that an existing system is not opaque, but it is desired to *enforce* opacity upon the system. The problem of opacity enforcement has been approached using a variety of mechanisms in the literature. For example, supervisory control can enforce opacity by restricting behavior that would reveal secrets to an eavesdropper. The synthesis of such controllers has been investigated in [28, 82] for instance. However, it is not always feasible to alter existing system behavior, e.g., human behavior in a cyber-physical system. Obfuscation has been proposed as an alternative to control to enforce opacity by altering the information ultimately available to the eavesdropper. Several works have investigated the synthesis of *edit functions* which selectively insert and delete outputs from the system [100, 102]. These approaches synthesize edit functions, which effectively enforce current-state opacity, as winning strategies to finite two-player reachability games as in [26]. Related obfuscation strategies, such as delaying observations at runtime [32] and dynamic masks [110], have also been studied.

In this chapter, we consider the problem of synthesizing edit functions to enforce K-step opacity. To the best of our knowledge, this problem has not yet been explicitly considered in the literature. As noted in [100], synthesis methods for current-state opacity can be applied to enforce other notions of opacity that can be transformed into current-state opacity, provided the resulting enforcement strategy for current-state opacity can be mapped back to the original notion of opacity under consideration. For example, it was found in [101] that initial-state and language-based opacity can be transformed into current-state opacity. Recently, a novel language-based formulation of the various notions of K-step and infinite step opacity was established, for the first time, in [97]. While [97] focuses on the verification of K-step opacity over finite automata, in this chapter we focus on the enforcement of K-step opacity with edit edit functions. We show how safe edit functions can be synthesized by suitably leveraging existing methods for current-state opacity. We demonstrate this approach on a case study by synthesizing edit functions enforcing location privacy on a system modeling the motion of individuals using a contact-tracing app whose data is available

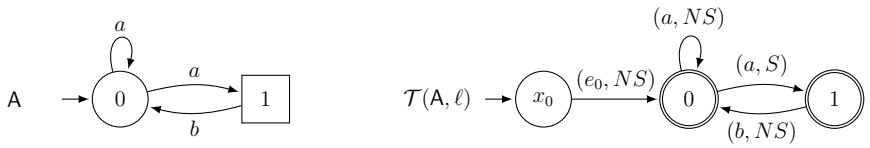

Figure 5.1: An automaton A (left) where the square state 1 is considered secret so $\ell(0) = \text{NS}$ and $\ell(1) = \text{S}$. The label-transform $\mathcal{T}(A, \ell)$ (right) recognizes the input-output sequences of A under ℓ.

to a malicious eavesdropper.

5.2 Problem Formulation

In this setting, we assume that all observers of the system (both intended and unintended) have access to the same observations. These observations are produced not directly from the system's sensors, but rather through the obfuscator.

Fig. 5.2 depicts the flow of information we consider.

5.2.1 Obfuscation Model

Recall that opacity is violated when the eavesdropper observes information output by the system that reveals secret behavior. This situation can be avoided with *obfuscation*, i.e., altering the information output by the system to fool the eavesdropper while maintaining the system's utility. As the outputs we consider are strings, we model this type of obfuscator as a nondeterministic edit function as in [49].

Definition 5.1. *A* nondeterministic edit function *over the events* Σ^* *is a function* $\mathbf{Obf} : \Sigma^* \to 2^{\Sigma^*}$ *or equivalently a relation* $\mathbf{Obf} \subseteq \Sigma^* \times \Sigma^*$.

We interpret the edit function $\mathbf{Obf}$ as follows: if the edit function has already received the string of inputs $s \in \Sigma^*$, upon receiving the new input event $\sigma \in \Sigma$, it produces a string of outputs from the set $\mathbf{Obf}(s\sigma)$. Similarly, before receiving any input, it produces a string of outputs in $\mathbf{Obf}(\epsilon)$. With this interpretation, the edit function defines the following observation relation between its inputs and outputs

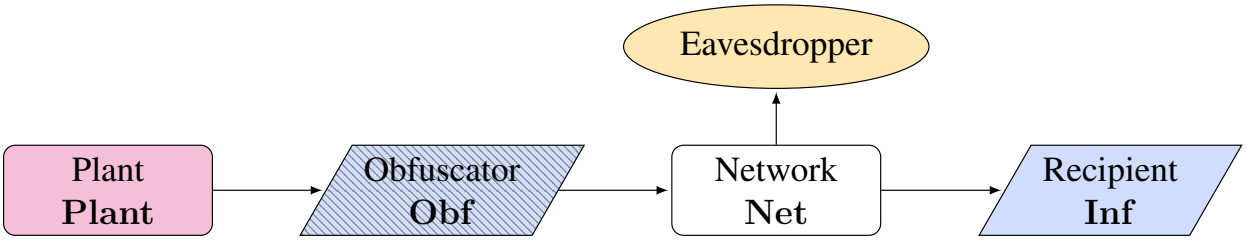

Figure 5.2: Network architecture for enforcement of opacity with obfuscation.

Definition 5.2. *Given the edit function* $\mathbf{Obf}$ *over events* Σ*, the non-prefix observation relation induced by* $\mathbf{Obf}$ *denoted by* Θ^n_{Obf} *is the smallest relation satisfying*

$$\begin{aligned} (\epsilon, t) \in \mathbf{Obf} &\implies (\epsilon, t) \in \Theta^n_{Obf} \\ (s, s') \in \Theta^n_{Obf} \wedge (s\sigma, t) \in \mathbf{Obf} &\implies (s\sigma, s't) \in \Theta^n_{Obf}. \end{aligned}$$

The prefix observation relation induced by $\mathbf{Obf}$ *denoted by* Θ_{Obf} *is then the smallest relation satisfying*

$$\begin{aligned} (\epsilon, t) \in \mathbf{Obf} \wedge t' \in \overline{t} &\implies (\epsilon, t') \in \Theta_{Obf} \\ (s, s') \in \Theta^n_{Obf} \wedge (s\sigma, t) \in \mathbf{Obf} \wedge t' \in \overline{t} \setminus \{\epsilon\} &\implies (s\sigma, s't') \in \Theta_{Obf}. \end{aligned}$$

An edit function can be understood as selectively deleting events as they occur and possibly inserting fictitious ones. Such an edit function can be implemented in many ways. For example, some edit functions can be represented as finite state transducers [80], i.e., a finite state transducer marks the induced observation relation. In this case, we say the edit function is *finite*. For simplicity, in this chapter we will only discuss such finite edit functions.

5.2.2 Notions of opacity under enforcement

We are interested in designing obfuscators which enforce language-based opacity in the framework of Chapter 3. In this framework, we model the behaviors of the system as a language L over the alphabet Σ. However, instead of making observations of these behaviors through a static mask as in Chapter 3 and Chapter 4, the eavesdropper now makes observations through the obfuscator. Formally, if obfuscation is implemented with an edit function $\mathbf{Obf}$, we consider the induced observation relation Θ_{Obf} which defines the system $\Delta = (L, \Theta_{\text{Obf}})$. As in the general framework, we consider that the eavesdropper may have varying degrees of knowledge about the system encoded with a nominal model $\hat{\Delta} = (\hat{L}, \hat{\Theta})$ where $\hat{L}$ is a language over Σ. Given some nonsecret specification φ_{NS}, we consider opacity with respect to different nominal observation relations which correspond to different levels of information *safety* or privacy ensured by the enforcement mechanism. The implementation of the edit function $\mathbf{Obf}$ may ultimately be unknown to the eavesdropper, or we say the obfuscator is *private*. In this case, opacity requires that the obfuscated observations of the real system are consistent with the nonsecret behavior, We represent this with the nominal observation relation given by the identity $\mathcal{I} = \{(s, s) \mid s \in \Sigma^*\}$.

Definition 5.3. *Given a behaviors* L*, nominal behaviors* $\hat{L}$*, and nonsecret specification* φ_{NS}*, we say an edit function* $\mathbf{Obf}$ *enforces* private safety *if* $\Delta = (L, \Theta_{Obf})$ *is opaque for* $\hat{\Delta} = (\hat{L}, \mathcal{I})$ *and*

φ_{NS}, *i.e.,*

$$\Theta_{Obf}(L) \subseteq \hat{L}\,. \tag{5.1}$$

Alternatively, the edit function $\mathbf{Obf}$ may be learned by the eavesdropper after it is implemented, or the obfuscator is *public*. In this case, opacity requires that the obfuscated observations of the real system are consistent with the obfuscated observations of nonsecret behavior.

Definition 5.4. *Given a behaviors L, nominal behaviors $\hat{L}$, and nonsecret specification φ_{NS}, we say an edit function* $\mathbf{Obf}$ *enforces* public safety *if $\Delta = (L, \Theta_{Obf})$ is opaque for $\hat{\Delta} = (\hat{L}, \Theta_{Obf})$ and φ_{NS}, i.e.,*

$$\Theta_{Obf}(L) \subseteq \Theta_{Obf}(\hat{L})\,. \tag{5.2}$$

These notions of public and private safety generalize the definitions of public and private safety for CSO from [49]. Additionally, given multiple nonsecret specifications, we can make corresponding private and public definitions of joint opacity (Definition 3.2) and separate opacity (Definition 3.3). In this way, we can define both private and public notions of K-step opacity as described in Section 3.4. We recall that K-step opacity informally requires hiding secret behavior over K of the last "steps" of the system. In formalizing a definition, we must identify which steps we are referring to, e.g., original or obfuscated behavior. For simplicity, we consider steps corresponding to events of the nominal model of behavior possessed by the eavesdropper. Formally, the specifications for nonsecret behavior are given by the k-delayed behavior defined by equation (3.17) for the observation relation $\hat{\Theta} = \mathcal{I}$. As we consider all events to be observable $\Sigma_{\text{obs}} = \Sigma$, there is no distinction between type 1 and type 2 secrets, so we denote these k-delayed behaviors as $\{\varphi_{\text{NS}}(k)\}_{k=0}^{K}$.

5.3 Enforcement Approach

In this section, we describe how edit functions enforcing private safety for joint and separate K-step opacity, as in Definitions 3.2 and 3.3, can be synthesized using existing methods for CSO such as [48, 102, 104]. Abstracting away details specific to each implementation, these synthesis methods can broadly be described as solving the following problem.

Problem 5.1 (Private Safety Enforcement). *Given an NFA $G = (Q, \Sigma, \delta, Q_0, Q_m)$ and a class of edit functions $\mathcal{S}_{Obf} \subseteq \{\mathbf{Obf} : \Sigma^* \times \Sigma^*\}$, find an edit function $\mathbf{Obf} \in \mathcal{S}_{Obf}$ such that*

$$\Theta_{Obf}(\mathcal{L}(G)) \subseteq \mathcal{L}_m(G)\,. \tag{5.3}$$

In this problem, the marked states Q_m encode admissible or nonsecret states, and unobservable events are represented with ϵ transitions. The set $\mathcal{S}_{\text{Obf}}$ describes the edit functions that satisfy

additional constraints that can be imposed in a given synthesis method. For example, the methods of [48, 102] synthesize edit functions with a bound on the number of consecutive insertions. After determinizing G, the methods of [104] also consider *utility constraints* which limit the difference between original observations and their obfuscations.

Remark 5.1. *Other mechanisms for enforcement, such as the dynamic masks considered in [110], can be viewed as a specific type of edit function. Additionally while Problem 5.1 describes private safety for CSO, there is a similar problem for public safety with corresponding synthesis methods that could be applied to K-step opacity. For example synthesis of edit functions which are both publicly and privately safe is discussed in [48] while the R-enforcers of [32] can be viewed as edit functions enforcing public safety using delay.*

By transforming an automaton A with secret label map ℓ and refining the state space by nonsecret specification automaton H_{NS} into the automaton $G = \Theta(\mathcal{T}(A, \ell) \times H_{\text{NS}})$ as in Section 3.3, we can express K-step opacity with as language-based opacity as in Section 3.4. Using this result, we can then enforce K-step opacity over A by solving Problem 5.1 over G.

5.3.1 Enforcing Joint K-step Opacity

As joint K-step opacity is defined in terms of only one class of nonsecret runs, namely the intersection of the sets $\varphi_{\text{NS}}(k)$ for $k \in \{0, \cdots, K\}$, we can apply synthesis methods for current-state opacity directly.

Theorem 5.1. *Let $G = \Theta(\mathcal{T}(A, \ell) \times H^{J}_{NS}(K))$. An edit function $\mathbf{Obf} \in \mathcal{S}_{Obf}$ is privately safe for joint K-step opacity over A and ℓ if and only if it is a solution to Problem 5.1 for G, $\mathcal{S}_{Obf}$.*

Proof. We see that

$$\begin{aligned} \mathcal{L}(G) &= \Theta(\mathcal{L}(\mathcal{T}(A, \ell)) \cap \mathcal{L}(H^{\text{J}}_{\text{NS}}(K))) \\ &= \Theta(\mathcal{L}(\mathcal{T}(A, \ell)) \cap \Sigma^*) = \Theta(L)\,. \end{aligned} \tag{5.4}$$

Additionally,

$$\begin{aligned} \mathcal{L}_m(G) &= \Theta(\mathcal{L}_m(\mathcal{T}(A, \ell) \times H^{\text{J}}_{\text{NS}}(K))) \\ &= \Theta\left(\bigcap_{k=0}^{K} L_{\text{NS}}(K)\right)\,. \end{aligned} \tag{5.5}$$

□

So we see edit functions that are solutions to Problem 5.1 are exactly those which enforce joint private opacity as in the sense of Definition 5.3.

If a deterministic automaton is required, the same result holds for the determinization $\det(G)$ instead of G. Here the automaton $\det(G)$ corresponds to the secret observer automaton G_{SO} for joint opacity from [97]. So to enforce K-step opacity for an automaton A, we construct G as in Theorem 5.1, synthesize an edit function $\mathbf{Obf}$ as a solution to Problem 5.1 for G, and apply $\mathbf{Obf}$ directly to the outputs of the original system. This is possible as the language of G is the set of observations produced by the original automaton A. We demonstrate this in the following example.

Example 5.1. *In this example we construct an edit function enforcing joint* 1*-step opacity with type* 1 *secrets. Consider the automaton* A*, secret label map* ℓ*, and their label-transform* $\mathcal{T}(A, \ell)$ *depicted in Fig. 5.1. Consider the nonsecret specification automaton* $H^J_{NS}(1)$ *as depicted in Fig. 5.3 constructed for* A *and* ℓ*. This automaton marks sequences of input-output pairs corresponding to runs that have not visited a secret state since* 1 *observation ago. We construct the automaton* $G = \mathbf{obs}(\mathcal{T}(A, \ell) \times H^J_{NS}(1))$ *which is depicted in Fig. 5.4. This automaton* G *marks the observations of the behavior of* A *that do not violate joint* 1*-step opacity. To better understand these observations we construct* $\det(G)$ *which is language equivalent and is depicted in Figure 5.5. We can see that the observation* ab *is not marked and hence is unsafe. This is because if the eavesdropper observes the event* b*, they reason that the system was in the secret state* 1 *exactly* 1 *step ago.*

If knowing the total number of events executed in the system is important, we can consider the following class of edit functions

$$\mathcal{S}_{Obf} = \{\mathbf{Obf} : \Sigma^* \to \Sigma^* \mid \forall s \in \mathcal{L}(G),\ |\Theta_{Obf}(s)| = |s|\}\,. \tag{5.6}$$

Then a solution to Problem 5.1 for G*,* $\mathcal{S}_{Obf}$ *is the edit function* $\mathbf{Obf}$ *which replaces all occurrences of* a *with* b *so* $\forall s \in \mathcal{L}(G),\ \Theta_{Obf}(s) = a^{|s|}$*. We can verify that* $\mathbf{Obf}$ *is a solution to Problem 5.1 as the set of obfuscated observations is* $\Theta_{Obf}(\mathcal{L}(G)) = \{a\}^*$ *which is safe. Thus by Theorem 5.1, the edit function* $\mathbf{Obf}$ *is privately safe for* 1*-step opacity. This makes sense:* $\mathbf{Obf}$ *hides the only event* b *that would reveal the secret to the eavesdropper.*

5.3.2 Separate Case

Enforcing separate K-step opacity is complex as we must consider multiple classes of nonsecret behavior $\varphi_{\text{NS}}(k)$ for $k \in \{0, \cdots K\}$. However, like it suffices to consider observations of the intersection of these nonsecret behaviors $\Theta\left(\bigcap_{k=0}^{K} \varphi_{\text{NS}}(k)\right)$ for joint K-step opacity, it suffices to consider the intersection of the observations of these nonsecret behaviors $\bigcap_{k=0}^{K} \Theta(\varphi_{\text{NS}}(k))$ for separate K-step opacity in terms of private safety. While we could represent this latter set of

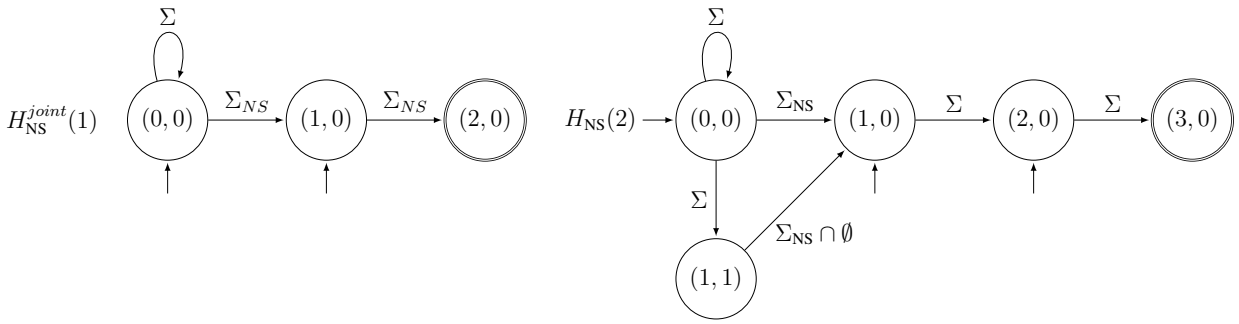

Figure 5.3: The nonsecret specification automata $H_{\text{NS}}^{joint}(1)$ and $H_{\text{NS}}(2)$. These automata are defined over the input-output pairs $\Sigma = ((E \cup \{e_0\})) \times \{\text{S}, \text{NS}\}$. Here $\Sigma_{NS} = ((E \cup \{e_0\})) \times \{NS\}$.

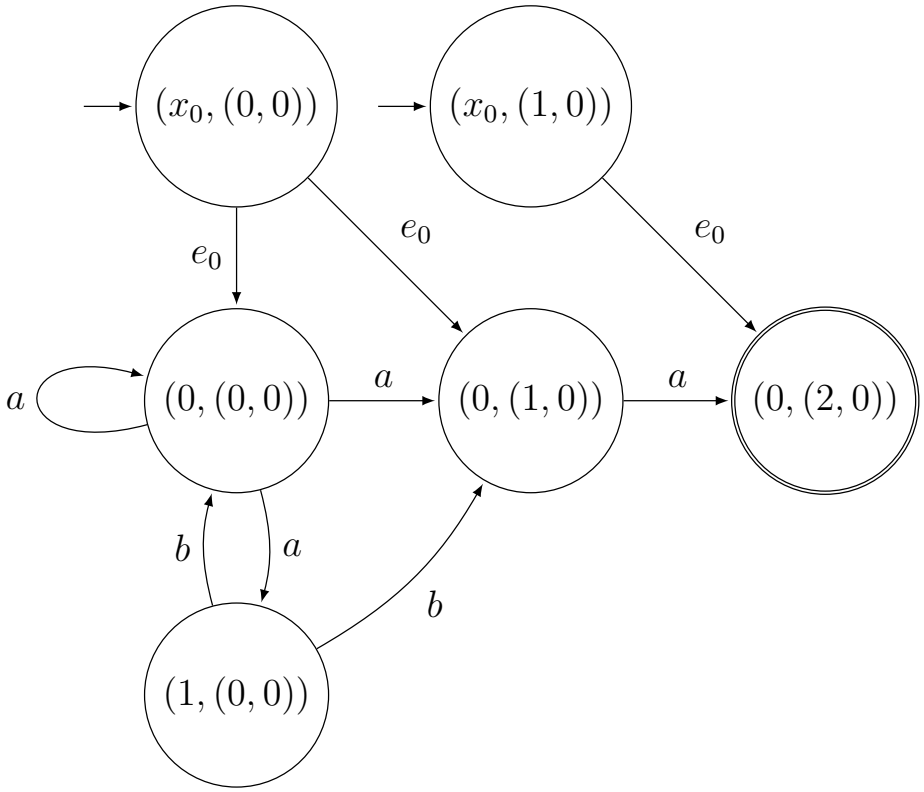

Figure 5.4: The automaton $G = \Theta(\mathcal{T}(A, \ell) \times H_{\text{NS}}^{\text{J}}(1))$ from Example 5.1.

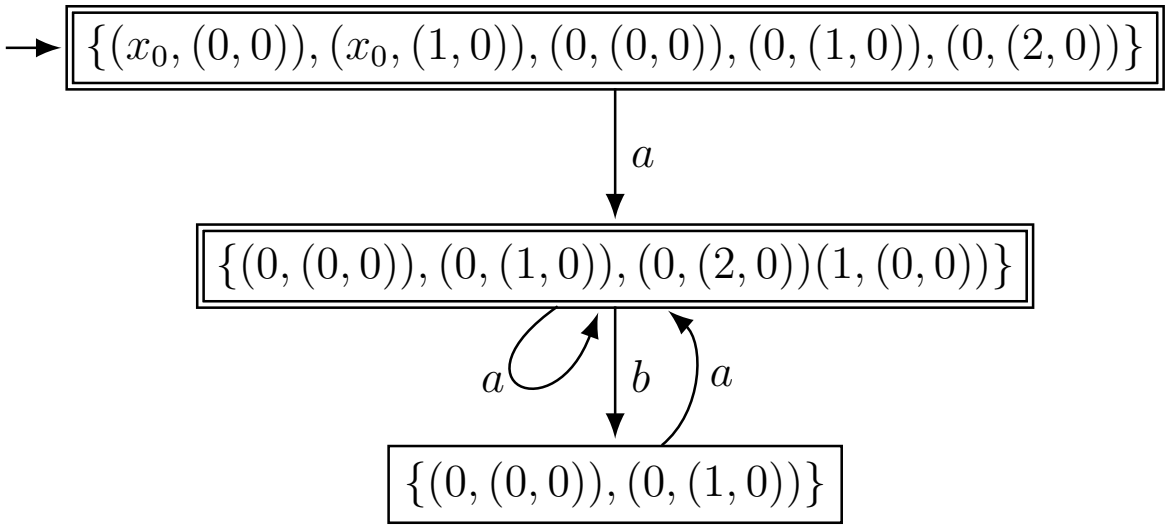

Figure 5.5: The automaton $\det(G)$ from Example 5.1.

observations with the automaton

$$\prod_{k=0}^{K} \Theta(\mathcal{T}(A,\ell) \times H_{\text{NS}}(k))\,, \tag{5.7}$$

we can avoid this product by utilizing the fact that $H_{\text{NS}}(k)$ is a subautomaton of $H_{\text{NS}}(K)$ for $k \leq K$. For each $k \in \{0,\cdots,K\}$ there exists a set $Q_{m,H,k}$ of states of $H_{\text{NS}}(K)$ such that the language of $H_{\text{NS},j}(K)$ marked by the states $Q_{m,H,k}$ is $\mathcal{L}_m(H_{\text{NS}}(k))$ [97]. For example in the automaton $H_{\text{NS}}(2)$ depicted in Figure 5.3, these sets are given by $Q_{m,H,0} = \{(1,0)\}$, $Q_{m,H,1} = \{(2,0)\}$, $Q_{m,H,2} = \{(3,0)\}$.

Theorem 5.2. *Let $G = \det(\Theta(\mathcal{T}(A,\ell) \times H_{NS}(K)))$ using the power set construction for determinization. Let the marked states Q_m of G be redefined as*

$$Q_m = \{\overline{q} \mid \forall k \in \{0,\cdots,K\}\ \exists q \in \overline{q} \cap Q \times Q_{m,H,k}\}\,. \tag{5.8}$$

Then an edit function $\mathbf{Obf} \in \mathcal{S}_{Obf}$ is privately safe for separate K-step opacity over A and ℓ if and only if it is a solution to Problem 5.1 for G, $\mathcal{S}_{Obf}$.

Proof. Similar to the proof for Theorem 5.1, we see

$$\begin{aligned} \mathcal{L}(G) &= \Theta(\mathcal{L}(\mathcal{T}(A,\ell)) \cap \mathcal{L}(H_{\text{NS}}(K))) \\ &= \Theta(L \cap \Sigma^*) = \Theta(L)\,. \end{aligned} \tag{5.9}$$

Likewise

$$\begin{aligned}\mathcal{L}_m(G) &= \bigcap_{k=0}^{K} \Theta(\mathcal{L}_m(\mathcal{T}(A,\ell) \times H_{\text{NS}}(k))) \\ &= \bigcap_{k=0}^{K} \Theta(L_{\text{NS}}(k))\,.\end{aligned} \tag{5.10}$$

So we see edit functions that are solutions to Problem 5.1 are exactly those which satisfy Definition 5.3 for private safety for separate opacity. □

The automaton G in this theorem corresponds to the secret observer automaton G_{SO} for separate opacity from [97]. We can then synthesize edit functions by solving Problem 5.1 as in the joint case.

5.4 Results

In this section, we demonstrate an application of K-step opacity enforcement in the context of contact tracing smartphone apps, developed in response to the COVID-19 pandemic, that record proximity between users to a centralized server. These apps raise a number of privacy concerns as described in [17, 51, 78]. We propose the use of obfuscation to enforce privacy for users of these apps while maintaining the utility of these apps for public health. We demonstrate this through a small example where a malicious user who gains access to this information may be able to combine it with partial location information to determine that another user has visited some secret location. Then we show how to enforce K-step opacity with a privately safe edit function using our approach, specifically, by leveraging Theorem 5.1. This edit function was synthesized with the methods from [104] as implemented in the *EdiSyn* library[1]. The automata constructions presented in this work and an interface to the EdiSyn library are provided by the *MDESops* library [68].

5.4.1 Modeling

Our model assumes that there are some number of users, including one user that acts as a malicious eavesdropper, that can move freely between a shared set of locations. We suppose that the malicious user has full knowledge of their own movements, in addition to partial knowledge of the other users' movements based on a partition of the map into discrete regions. This partial knowledge is modeled similarly to the automaton model for location-based services described in [105], but with event labels based on the destination node instead of the source node. We further assume that the malicious user has access to information from the contact tracing app from which they can deduce

[1] `https://gitlab.eecs.umich.edu/M-DES-tools/EdiSyn`

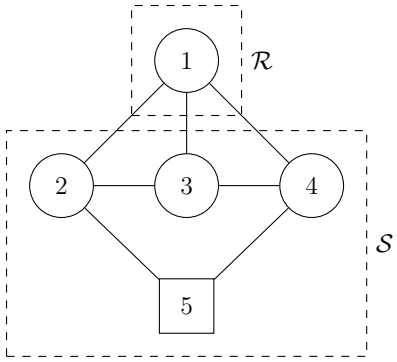

Figure 5.6: The graph Γ that represents the locations and the physical paths between them. $\mathcal{R}$ and $\mathcal{S}$ represent the partition of locations into distinct regions. Location 5 is considered to be secret.

the sizes of contact clusters that exist at any given time, where we consider a contact cluster to be a set of multiple users sharing the same location. The malicious eavesdropper does not know the locations of the contact clusters, or which users are in which contact clusters.

Our procedure requires as input a simple undirected graph $\Gamma = (\mathcal{V}, \mathscr{E})$ that represents locations by the set of vertices $\mathcal{V}$, and the physical paths between locations by the set of edges $\mathscr{E}$. It also requires a partition P of $\mathcal{V}$ into regions, a fixed number of users n, and some definition of secret behavior. We additionally define **region** : $\mathcal{V} \to P$ so that for $v \in \mathcal{V}$, **region**(v) is the region containing v.

Throughout this section, we illustrate our procedure using Γ as shown in Fig. 5.6, with $P = \{\mathcal{R}, \mathcal{S}\}$ as shown. We also suppose that $n = 3$, and user 2 visiting location 5 as the secret state of the system.

5.4.2 Mobility Model

Using Γ, we first construct the map automaton $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Delta, \mathcal{V}_0)$, where $\mathcal{V}$ is the set of states, $\mathcal{E}$ is the set of events, $\Delta : \mathcal{V} \times \mathcal{E} \to 2^{\mathcal{V}}$ is the transition function, and $\mathcal{V}_0$ is the set of initial states. We define $\mathcal{V}_0 = \mathcal{V}$, $\mathcal{E} = \{\tau\}$, and $\Delta(v, \tau) = \{v\} \cup \{u \in \mathcal{V} \mid (u, v) \in \mathscr{E}\}$ for all $v \in \mathcal{V}$.

For each $i \in \{1, 2, \ldots, n\}$, we construct an individual automaton $\mathcal{G}_i$, which represents all movements that user i is allowed to make in a single time step. We construct $\mathcal{G}_1$ from $\mathcal{G}$ by removing the secret location, since we assume the malicious user is not allowed to enter the secret location. For $i > 1$, we let $\mathcal{G}_i = \mathcal{G}$, since other users are unrestricted in their movement. The individual automaton $\mathcal{G}_1$ for our example is shown in Fig. 5.7. The individual automata $\mathcal{G}_2$ and $\mathcal{G}_3$ are not shown, but are similar with $\mathcal{G}_1$, with the additional inclusion of the secret location 5. We then construct $\mathcal{H} = \mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_n$, which gives the full model of all possible movements by all users within the system. The state set of $\mathcal{H}$ is then $\mathcal{V}^n$, and for each state $x \in \mathcal{V}^n$ with $x = (v_1, ..., v_n)$, we have that v_i is the location of user i.

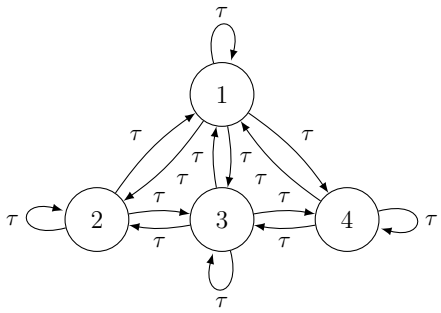

Figure 5.7: The individual automaton $\mathcal{G}_1$, representing all possible movements by the malicious user in a single time step. The individual automata $\mathcal{G}_2$ and $\mathcal{G}_3$ are similar, but also include the secret state 5.

Given a state $x \in \mathcal{V}^n$, we define

$$\alpha(x) = (\alpha_1, ..., \alpha_n) \tag{5.11}$$

where for each i,

$$\alpha_i = \begin{cases} v_1, & i = 1 \\ \textbf{region}(v_i), & i > 1. \end{cases} \tag{5.12}$$

The result is that $\alpha(x)$ denotes the locations of each user, as observed by the malicious eavesdropper, when the system is in state x. We also define the multiset

$$\beta(x) = \{|\mathcal{B}_v| : |\mathcal{B}_v| > 1\} \tag{5.13}$$

where for each $v \in \mathcal{V}$,

$$\mathcal{B}_v = \{i \in \{1, \ldots, n\} : v_i = v\}. \tag{5.14}$$

The result is that $\beta(x)$ represents the sizes of the contact clusters that exist when the system is in state x. Note that in general, $\beta(x)$ must be a multiset since there may be more than one contact cluster of a given size, but since also the contact clusters are indistinguishable and thus their sizes should be unordered. For our example with $n = 3$, we note that $\beta(x)$ has only three possible values: $\beta(x) = \varnothing$ if all states in x are distinct, $\beta(x) = \{2\}$ if two states of x are identical and the third is distinct, or $\beta(x) = \{3\}$ if all states of x are identical.

Finally, we construct A from $\mathcal{H}$ by relabeling each transition as the concatenation $\alpha(x)\beta(x)$ where x is the state at which the transition ends, and α and β are the location and contact markings as defined in equations (5.11) and (5.13). We let the observable event set of A be $\Sigma_{\text{obs}} = \Sigma$. We additionally mark states with $v_2 \in \mathcal{V}_S$ as secret. For our example, A contains 100 states and 5,054 transitions, and so is unable to be shown in full. However, a small subautomaton of A is shown in

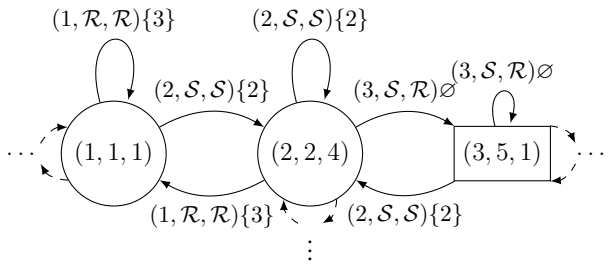

Figure 5.8: A small subautomaton of A. Events are labeled according to their target state. The state $(3,5,1)$ is secret since $v_2 \in \mathcal{V}_S = \{5\}$.

Fig. 5.8, illustrating the result of the event relabeling.

5.4.3 Opacity Enforcement

Using existing methods of opacity verification, we can determine that the system is current-state opaque, but that it is neither separately nor jointly 1-step opaque. Note that since $\Sigma_{\text{obs}} = \Sigma$, there is no distinction between type 1 or type 2 secrets. One event sequence that violates 1-step opacity is

$$o = \Big((1,\mathcal{S},\mathcal{R})\{2\},\ (3,\mathcal{S},\mathcal{S})\varnothing,\ (4,\mathcal{S},\mathcal{S})\{3\}\Big), \tag{5.15}$$

which corresponds to the state-estimate sequence

$$\begin{aligned} X_1 &= \{(1,2,1),(1,3,1),(1,4,1),(1,5,1)\}, \\ X_2 &= \{(3,2,4),(3,5,4),(3,4,2),(3,5,2)\}, \\ X_3 &= \{(4,4,4)\}. \end{aligned} \tag{5.16}$$

Each X_i contains at least one nonsecret state, and thus current-state opacity is not violated by o. However, the only state in X_2 from which the event $o_3 = (4,\mathcal{S},\mathcal{S})\{3\}$ can occur is the secret state $(3,5,4)$, and thus 1-step opacity is violated.

To enforce joint 1-step opacity, we first use A to construct G as defined in Theorem 5.1, with $K = 1$. We additionally constrain the set $\mathcal{S}_{\text{Obf}}$ of allowable edit functions in two ways. First, we only allow the edit function to replace events, i.e. deleting events entirely or inserting new events where none previously existed is not allowed. Second, we enforce a utility constraint as is defined in [104]. This constraint prevents the obfuscator from changing any of the location information observed by the eavesdropper directly. To construct this constraint, we first define the utility distance

$D_A : \mathcal{V}^n \times \mathcal{V}^n \to \{0, 1\}$ over the automaton A such that for $x, y \in \mathcal{V}^n$, we have

$$D_A(x, y) = \begin{cases} 0, & \alpha(x) = \alpha(y) \\ 1, & \text{otherwise.} \end{cases} \tag{5.17}$$

where α is as defined in equation (5.11). We then transform D_A into a utility distance D_G for G so that $D_G(s, t) = 0$ if and only if for every A component x in s, there exists an A component y in t such that $D_A(x, y) = 0$. The utility constraint on the edit function $\mathbf{Obf}$ over G then requires that for any observation $o \in \mathbf{obs}(\mathbb{R})$, the state s of G reached by o and the state t of G reached by the obfuscated observation $M(o)$ satisfy $D_G(s, t) = 0$.

Now we construct an edit function $\mathbf{Obf}$ by applying EdiSyn to solve Problem 5.1 for G, $\mathcal{S}_{\text{Obf}}$. The resulting edit function is encoded by an obfuscator automaton, similar to a string transducer, containing 440 states and 11,290 transitions. Therefore it is not possible to include the full result in here. However, we consider again the violating event sequence o defined in equation (5.15). This is mapped by the edit function to the obfuscated event sequence

$$M(o) = \Big((1, \mathcal{S}, \mathcal{R})\{2\},\ (3, \mathcal{S}, \mathcal{S})\{3\},\ (4, \mathcal{S}, \mathcal{S})\{3\}\Big), \tag{5.18}$$

which corresponds to the new state-estimate sequence

$$\begin{aligned} X_1' &= \{(1, 2, 1), (1, 3, 1), (1, 4, 1), (1, 5, 1)\} \\ X_2' &= \{(3, 3, 3)\} \\ X_3' &= \{(4, 4, 4)\}. \end{aligned} \tag{5.19}$$

Since X_1' contains nonsecret states from which the event $M(o)_2 = (3, \mathcal{S}, \mathcal{S})\{3\}$ can occur, and since X_2' and X_3' each contain only nonsecret states, then joint 1-step opacity is not violated by $M(o)$. Additionally, this is a valid edit since each event that may occur from a state in X_2 may also occur from a state in X_2'.

5.5 Conclusion

In this chapter, we considered the problem of synthesizing edit functions that enforce the various notions of K-step opacity. By transforming K-step opacity into a language-based notion, we can apply existing synthesis methods for CSO. To the authors' knowledge, synthesis methods for edit functions enforcing K-step opacity have not been proposed before. We demonstrate this approach on a novel contact-tracing system model. We focused on the effectiveness of edit functions that are not known to the eavesdropper, but a similar approach can be used assuming the edit function is

public as in [48]. Additionally, the language-based formulation of K-step opacity could be used to study the problem of synthesizing supervisory control to enforce K-step opacity.

CHAPTER 6

Enforcement of Opacity and Utility with Distributed Synthesis

6.1 Introduction

Obfuscation provides a powerful mechanism for opacity enforcement as we have seen in the previous chapter. In particular edit functions, which selectively insert and delete system outputs, *edit functions* can effectively hide secrets from everyone on an open network [100]. Because these methods do not distinguish between intended and unintended recipients., information that is available to some is available to all. This limits the utility of this approach as it forces a strict trade off between privacy and utility.

In this chapter we propose an obfuscation framework that allows an intended recipient to infer sensitive information that cannot be deduced by unintended recipients. Similar to encryption, this is possible by designing a "key" that is provided to the intended recipient to recover information about the plant from their obfuscated observations. Whereas encryption achieves privacy by conspicuously altering data to ensure it is impossible or at least computationally difficult to recover, our proposed method of obfuscation achieves privacy by inconspicuously altering data to *deceive* recipients with partial knowledge of the system. In this sense, our goals for obfuscation are orthogonal to those of encryption. We provide an automatic method for the simultaneous design of obfuscation and inference policies for dynamic systems subject to security and utility requirements. In this setting, plants can be modeled by finite automata over which we can specify requirements with temporal logics. We consider obfuscation policies similar to edit functions which can produce a positive number of outputs given a single input from the plant. By modeling the obfuscation and inference policies as processes in a distributed system, we can leverage techniques from distributed synthesis [38] to design solutions with formal guarantees of privacy and utility. We motivate our solution to this problem with the following simple example.

Example 6.1. *Consider a company operating a research facility whose layout is depicted in Figure 6.2. Smart devices report employee's locations throughout the building to a server over an internal network. However, the devices' accuracies are limited and only report an approximate location represented by a region of the building: $\mathcal{R}$ is the lobby, $\mathcal{S}$ is the offices, and $\mathcal{T}$ is the electronics lab*

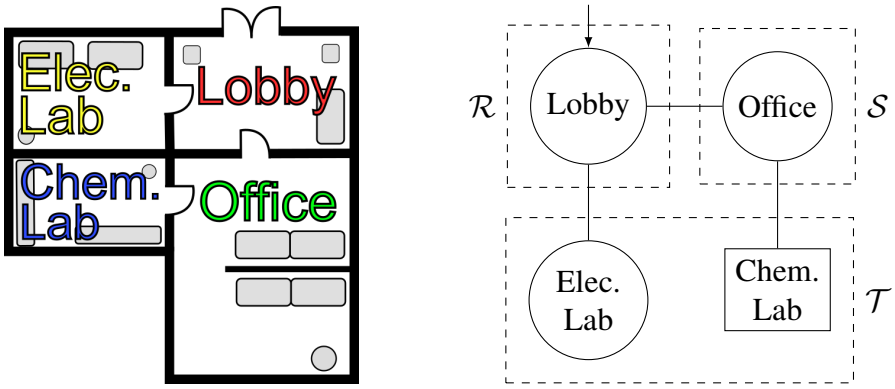

Figure 6.1: The graph $\Gamma = (\mathcal{V}, \mathscr{E})$ that represents the locations $\mathcal{V}$ and the physical paths between them $\mathscr{E}$. Regions $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$ represent the approximate locations the smart device can report and partition the space $\mathcal{V}$. The chemistry lab is considered to be a *secret* location.

Figure 6.2: The graph $\Gamma = (\mathcal{V}, \mathscr{E})$ that represents the locations $\mathcal{V}$ and the physical paths between them $\mathscr{E}$. Regions $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$ represent the approximate locations the smart device can report and partition the space $\mathcal{V}$. The chemistry lab is considered to be a *secret* location.

and chemical lab. The company is concerned this information may reveal to their competitors how they allocate their resources, i.e., when an employee enters the chemistry lab. However, the company does not want to restrict employees' movements or alter their schedules. While encryption can secure this information over the network, it alerts competitors to the existence of a secret and may prompt them to investigate using other means. Instead the company chooses to deceive competitors by reporting obfuscated employee locations, but these must be consistent with the building layout in order to not raise suspicion.

By expressing the requirement to hide their location as opacity, obfuscation techniques can be applied to the dynamic model induced by the building layout for each employee. However, the company also utilizes an emergency response service that must be informed when individuals are using the chemical lab. Existing methods of obfuscation cannot guarantee this form of utility as they view all recipients equally: if this information is hidden to competitors, it must be hidden from everyone. Instead, the company realizes they could provide information about their obfuscation to the emergency service, so that the service can infer the relevant information from the obfuscated locations. In summary, the company wants to design obfuscation and inference policies that are consistent with the dynamics induced by the building layout, guarantee privacy from their competitors, and maintain utility with the emergency service.

6.2 Problem Formulation

In this section, we formulate the problem of obfuscation synthesis with formal models for the plant, obfuscator, and recipients. We express security and utility requirements through opacity and a new notion of inference.

6.2.1 System Model & Requirements

We consider a plant that sequentially outputs over a set of Boolean variables O_{env}. This behavior is defined by a set of finite traces $L_{\text{env}} \subseteq \left(2^{O_{\text{env}}}\right)^*$. The obfuscator $\mathbf{Obf}$ receives a subset $I_{\text{Obf}} \subseteq O_{\text{env}}$ of the plant's output variables as input and produces a sequence of outputs on the Boolean variables O_{Obf} at each step. We only consider obfuscators that are *deterministic*, i.e., producing a single sequence of outputs on a single input, and *non-silent*, i.e., never producing the empty sequence. We call the implementation of an obfuscator an *obfuscation policy*.

Definition 6.1. *A deterministic and non-silent* obfuscation policy *is a function* $\mathbf{Obf} : \left(2^{I_{Obf}}\right)^+ \to \left(2^{O_{Obf}}\right)^+$ *that maps input histories to a sequence of produced outputs.*

This definition is similar to the concept of an *edit functions* as in [49]. Given an input history $i = i_0 \cdots i_n \in \left(2^{I_{\text{Obf}}}\right)^+$, the corresponding output histories consist of the complete output sequences made for i_0 through i_{n-1} followed by a partial output sequence made for i_n. We define the set of such *output histories* $\mathbf{Hist}(\mathbf{Obf}, i)$ by

$$\mathbf{Hist}(\mathbf{Obf}, i) = \{\mathbf{Obf}(i_0) \cdots \mathbf{Obf}(i_0 \cdots i_{n-1})\} \cdot (\overline{\mathbf{Obf}(i_0 \cdots i_n)} \setminus \{\epsilon\}) \,. \tag{6.1}$$

For example, if $\mathbf{Obf}(i_0) = o_0 o_1$ and $\mathbf{Obf}(i_0 i_1) = o_2 o_3$ then the output histories are $\mathbf{Hist}(\mathbf{Obf}, i_0 i_1) = \{o_0 o_1 o_2,\ o_0 o_1 o_2 o_3\}$.

Recipients passively observe a subset $I_{\text{Inf}} \subseteq O_{\text{Obf}}$ of the obfuscator's output variables and try to reason about the state of the plant. Importantly we assume that recipients *only observe the outputs of the obfuscator sequentially, not how they are produced*, e.g. they cannot distinguish outputting the word $\sigma\sigma$ on one input and outputting σ twice over two inputs. This motivates our definition of the output histories from equation (6.1). Privacy and security requirements express limits on the knowledge deduced by unintended recipients. In the context of opacity [49], the notion of *private safety* describes privacy from a recipient that knows the plant but is unaware of obfuscation. In this case the recipient's observations should correspond to observations of nonsecret behavior in the plant. In this work, we more generally require these observations to belong to some *nonsecret* language L_{NS}.

Definition 6.2. *We say an obfuscation policy* $\mathbf{Obf}$ *enforces* private safety *with respect to the plant behavior* L_{env} *and* nonsecret *output histories* $L_{NS} \subseteq \left(2^{O_{Obf}}\right)^*$ *if*

$$\forall e \in L_{env} \setminus \{\epsilon\},\ \forall h \in \mathbf{Hist}(\mathbf{Obf}, e|_{I_{Obf}}) :\ h \in L_{NS}\,. \tag{6.2}$$

In the next subsection, we show how this notion of private safety can express the inability of a recipient with uncertain knowledge of the obfuscation policy to infer information about the plant. For instance, we construct L_{NS} for Example 6.1 as the set of location histories that do not visit the chemistry lab. In this case, private safety ensures that competitors can never deduce when employees enter the chemistry lab. For simplicity, we will assume that both L_{env} and L_{NS} satisfy the conditions of Lemma 6.4 so that they can be generated by finite automata without deadlock.

While obfuscation ensures security against unintended recipients, the *intended recipient* must be able to *infer* certain information about the plant's behavior using the outputs of the obfuscator. So the inputs to the recipient are $I_{\text{Inf}} = O_{\text{Obf}}$ while their outputs O_{Inf} encode their inferences about the plant with Boolean variables. The recipient's reasoning is then modeled by an *inference policy* $\mathbf{Inf} : \left(2^{I_{\text{Inf}}}\right)^+ \to 2^{O_{\text{Inf}}}$ mapping the recipient's history of observations to their inferences. While the recipient could reason about arbitrary predicates over the plant behavior, we consider only their ability to always infer some fixed function $\mathbf{Data} : 2^{O_{\text{env}}} \to 2^{O_{\text{Inf}}}$ of the current plant output. This means if the current plant output is $e \in 2^{O_{\text{env}}}$, then the recipient's inference policy should output $\mathbf{Data}(e)$. The number of possible inferences is given by $2^{|O_{\text{Inf}}|}$. In the context of Example 6.1, the emergency service must infer when the user is in the chemistry lab, so we define $\mathbf{Data}$ to output true when there is a user present and false otherwise.

Definition 6.3. *Given the plant behavior* $L_{env} \subseteq \left(2^{O_{env}}\right)^*$*, obfuscation policy* $\mathbf{Obf} : \left(2^{I_{Obf}}\right)^+ \to \left(2^{O_{Obf}}\right)^+$*, and data function* $\mathbf{Data} : 2^{O_{env}} \to 2^{O_{Inf}}$*, an inference policy* $\mathbf{Inf} : \left(2^{I_{Inf}}\right)^+ \to 2^{O_{Inf}}$ *is correct if*

$$\forall e = e_0 \cdots e_n \in L_{env} \setminus \{\epsilon\},\ \forall h \in \mathbf{Hist}(\mathbf{Obf}, e|_{I_{Obf}}) :\ \mathbf{Inf}(h) = \mathbf{Data}(e_n)\,. \tag{6.3}$$

With the models of the system defined along with the notions of correct inferences and private safety, we can now define the obfuscation synthesis problem.

Problem 6.1 (Obfuscation synthesis). *Given a plant with behavior* L_{env}*, nonsecret output histories* L_{NS}*, and information* $\mathbf{Data}$*, find an obfuscation policy* $\mathbf{Obf}$ *that enforces private safety and an inference policy* $\mathbf{Inf}$ *that is correct.*

After designing the obfuscation and inference policies, the inference policy can be securely transferred to the intended recipient. Critically, this allows the intended recipient to infer information about the plant that unintended recipients cannot.

6.2.2 Modeling Security Requirements

We now show how to construct the language L_{NS} to express security properties as private safety. In the context of opacity, reference [49] provides a definition of L_{NS} expressing the inability of an observer unaware of any obfuscation to deduce the plant is currently at a secret state. We extend this definition by considering recipients with some fixed but uncertain knowledge of the plant and obfuscation policy *a priori*. Namely they believe the plant behavior belongs to the set $L'_{\text{env}} \subseteq \left(2^{O_{\text{env}}}\right)^+$ and that this behavior is altered with an obfuscation policy in the class $\mathcal{S}_{\text{Obf}} \subseteq \{\mathbf{Obf}' : \left(2^{I_{\text{Obf}}}\right)^+ \to \left(2^{O_{\text{Obf}}}\right)^+\}$. We consider the security requirement that the output of the obfuscator must be consistent with the recipient's model of L'_{env} and $\mathcal{S}_{\text{Obf}}$. Additionally, the recipient must not be able to deduce that the plant behavior did not belong to some set of nonsecret behavior $L'_{\text{env,NS}} \subseteq L'_{\text{env}}$. We clarify that while the unintended recipient may know that their observations have been altered, we assume they do not know for what purpose or how the obfuscation is designed[1]. In this case it suffices to ensure any output history of the true obfuscation policy $\mathbf{Obf}$ over the plant behaviors L_{env} is also the output history of some obfuscation policy $\mathbf{Obf}' \in \mathcal{S}_{\text{Obf}}$ applied to a nonsecret plant trace $e'_{\text{NS}} \in L'_{\text{env,NS}}$. Formally,

$$\forall e \in L_{\text{env}} \setminus \{\epsilon\},\ \forall h \in \mathbf{Hist}(\mathbf{Obf}, e|_{I_{\text{Obf}}}),$$
$$\exists e'_{\text{NS}} \in L'_{\text{env,NS}},\ \exists \mathbf{Obf}' \in \mathcal{S}_{\text{Obf}} :\ h' \in \mathbf{Hist}(\mathbf{Obf}', e'_{\text{NS}}|_{I_{\text{Obf}}})\,. \quad (6.4)$$

A class of obfuscation policies $\mathcal{S}_{\text{Obf}} = \{\mathbf{Obf}' : \left(2^{I_{\text{Obf}}}\right)^+ \to \left(2^{O_{\text{Obf}}}\right)^+\}$ defines a relation $\mathbf{R}_{\mathcal{S}_{\text{Obf}}} \subseteq \left(2^{I_{\text{Obf}}}\right)^+ \times \left(2^{O_{\text{Obf}}}\right)^+$ between plant behaviors and the corresponding possible output histories defined by

$$\mathbf{R}_{\mathcal{S}_{\text{Obf}}} = \bigcup_{e' \in \left(2^{I_{\text{Obf}}}\right)^+} \bigcup_{\text{Obf}' \in \mathcal{S}_{\text{Obf}}} \{e'\} \times \mathbf{Hist}(\mathbf{Obf}', e')\,. \quad (6.5)$$

We call such a relation *regular* if it can be represented by a finite transducer, i.e., $\mathbf{R}_{\mathcal{S}_{\text{Obf}}}$ is the set of all pairs of input and output words accepted by the transducer. These relations are also called rational transductions in [80]. The *composition* $\mathbf{R}(L)$ of a relation $\mathbf{R}$ to a language L is defined by

$$\mathbf{R}(L) = \{h \mid \exists e \in L :\ (e, h) \in \mathbf{R}\}\,. \quad (6.6)$$

We assume that the behavior $L'_{\text{env,NS}}$ satisfies the conditions of Lemma 6.4 and the class of obfuscation policies $\mathcal{S}_{\text{Obf}}$ defines a *regular relation*. Under these assumptions, we have the following result.

[1] In future work, we can consider obfuscators that are secure to recipients aware of the design requirements and synthesis method. This would be similar to the notion of *public safety* [49].

Theorem 6.1. *Let $L'_{env,NS}$ be a language satisfying the conditions of Lemma 6.4 and let $\mathcal{S}_{Obf}$ be such that $\mathbf{R}_{\mathcal{S}_{Obf}}$ is regular. Then $L_{NS} = \mathbf{R}_{\mathcal{S}_{Obf}}(L'_{env,NS} \setminus \{\epsilon\})$ also satisfies the conditions of Lemma 6.4. Furthermore, an obfuscation policy* $\mathbf{Obf}$ *is privately safe with respect to L_{env} and L_{NS} if and only if unintended recipients never deduce the plant behavior did not belong to $L'_{env,NS}$, i.e, condition* (6.4) *holds.*

Proof. By assumption, there exists a finite automaton G with $L(G) = L'_{\text{env,NS}}$. Then as $\mathbf{R}_{\mathcal{S}_{\text{Obf}}}$ is a regular relation, by the results of [80] the composition $L_{\text{NS}} = \mathbf{R}_{\mathcal{S}_{\text{Obf}}}(L'_{\text{env,NS}} \setminus \{\epsilon\})$ can be represented by an automaton constructed as the product of the finite transducer representing $\mathbf{R}_{\mathcal{S}_{\text{Obf}}}$ with G. As both G and the transducer are both finite and deadlock-free (as obfuscation policies are defined over all inputs), it follows that this product automaton is also deadlock-free in the sense of Lemma 6.4. Hence L_{NS} satisfies the conditions of Lemma 6.4.

Next observe by the definition of L_{NS} and $\mathbf{R}_{\mathcal{S}_{\text{Obf}}}$ that

$$\begin{aligned} h \in L_{\text{NS}} &\overset{(6.6)}{\iff} \exists e' \in L'_{\text{env,NS}} : \ (e', h) \in \mathbf{R}_{\mathcal{S}_{\text{Obf}}} \\ &\overset{(6.5)}{\iff} \exists e' \in L'_{\text{env,NS}}, \ \exists \mathbf{Obf}' \in \mathcal{S}_{\text{Obf}} : \ h \in \mathbf{Hist}(\mathbf{Obf}', e') \,. \end{aligned}$$

Hence condition (6.4) is equivalent to private safety. □

By explicitly modeling the knowledge of unintended recipients, we have control over the level of security the obfuscator guarantees. We also note that the more uncertain the unintended recipient, i.e., the larger $L'_{\text{env,NS}}$ and $\mathcal{S}_{\text{Obf}}$ are, the easier it is to enforce private safety. For example if the unintended recipient is not aware that the plant outputs are altered, we consider the class $\mathcal{S}_{\text{Obf}}$ with only the identity map. This corresponds to the existing notion of private safety in [48]. If instead the unintended recipient is more uncertain and believes that the outputs could be altered but there can only be k consecutive outputs for a single input, we consider the class $\mathcal{S}_{\text{Obf}} = \{\mathbf{Obf} \mid \forall e : \ |\mathbf{Obf}(e)| \leq k\}$. In this case, any solution for the first case will also be a solution for the second case. In both bases, we can also see that the induced relations $\mathbf{R}_{\mathcal{S}_{\text{Obf}}}$ are regular.

Example 6.2. *We transform the problem described in Example 6.1 into an instance of Problem 6.1 as follows. First we model the movement of the employee around the building layout graph Γ from Figure 6.2 and the regions detected by the smart devices as the plant behavior L_{env}. We define this behavior over the possible regions and whether or not the current location is the secret one (the chemistry lab). So we define $O_{env} = \{\mathcal{R}, \mathcal{S}, \mathcal{T}, S\}$ where S denotes the secret*[2]*. We construct an*

[2]For simplicity, we encode each region with a single variable. As they are disjoint, they could more efficiently be encoded with two variables rather than three total.

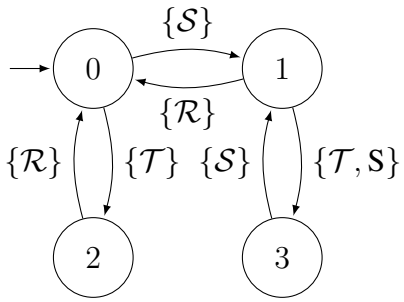

Figure 6.3: An automaton G generating the plant behavior L_{env} corresponding to the employee's movement throughout the building depicted in Figure 6.2. Each region of the building is encoded with its own variable $\mathcal{R}, \mathcal{S}, \mathcal{T}$ along with the secret status of the room encoded with the variable S .

automaton that generates the behavior L_{env} with states given by the locations in Γ. Then for each movement from one location to another given by an edge in Γ, we add a transition labeled with the region and secret status of the destination. This results in the automaton G depicted in Figure 6.3.

Next, we model the obfuscator. Recall the obfuscator only observes the current region (and not the secret output S) so we have $I_{Obf} = \{\mathcal{R}, \mathcal{S}, \mathcal{T}\}$. Likewise the obfuscator outputs regions which we represent with copies of the region variables $O_{Obf} = \{\mathcal{R}', \mathcal{S}', \mathcal{T}'\}$ (variables across processes must be disjoint). We can then express the privacy requirement as private safety. Recall the unintended recipients are competitors that should not be able to deduce that an employee is in the chemistry lab. As the competitor knows the true layout of the building, their model of the plant is also given by $L'_{env} = L(G)$. The nonsecret behavior $L'_{env,NS}$ is then given by the language generated by G after removing the secret state, i.e., $L'_{env,NS} = \overline{\{\mathcal{TR}, \mathcal{SR}\}^*}$. As we assume the competitor will not be aware of this obfuscation, we consider the class $\mathcal{S}_{Obf}$ consisting only of the identity map (mapping regions to their copy). We can then construct $L_{NS} = \mathbf{R}_{\mathcal{S}_{Obf}}(L'_{env,NS}) = \overline{\{\mathcal{T}'\mathcal{R}', \mathcal{S}'\mathcal{R}'\}^*}$ as in Theorem 6.1 to define the appropriate notion of private safety.

Finally, we model the intended recipient. Recall the intended recipient is the emergency service that observes outputs from the obfuscator and must be able to infer when an employee is in the chemistry lab. So we define $I_{Inf} = O_{Obf}$ and $O_{Inf} = \{S'\}$ and $\mathbf{Data} : 2^{O_{env}} \to 2^{O_{Inf}}$ with $\mathbf{Data}(e) = \{S'\}$ if $S \in e$ and $\mathbf{Data}(e) = \emptyset$ otherwise. Here S' is a copy of the plant variable S that is true when the recipient infers an employee is in the chemistry lab.

The company then desires to solve Problem 6.1 to design an obfuscation policy $\mathbf{Obf}$ *and inference policy* $\mathbf{Inf}$*. By having each employee obfuscate their location with* $\mathbf{Obf}$ *on their smart device, they ensure their competitors will not think they visit the chemistry lab. Then by securely distributing* $\mathbf{Inf}$ *to the emergency service, they ensure they will be able to know when they visit the chemistry lab.*

6.3 Obfuscation Synthesis in Distributed Systems

In this section, we show how to transform Problem 6.1 for obfuscation synthesis into an instance of Problem 2.1 for pipeline synthesis that we know how to solve. While the system of Problem 6.1 resembles the pipeline architecture of Problem 2.1 with the plant feeding into the obfuscation policy feeding into the inference policy, it is not "synchronous": the obfuscation policy produces a variable-length sequence of outputs on consuming a single input. To address this issue we *unfold* the obfuscation system and specifications for private safety and correct inferences into synchronous ones. Then solutions to Problem 6.1 can be found by folding solutions found to an instance of Problem 2.1 over the unfolded system.

6.3.1 Unfolding the System

We must *unfold* our the obfuscation system so that one output is produced by the obfuscation policy in each step. This unfolded system has the same plant outputs O_{env} as the original or *folded* system. We represent the obfuscator with a process $p_0 = (I_0, O_0)$ with the same inputs $I_0 = I_{\text{Obf}}$ but with outputs $O_0 = O_{\text{Obf}} \cup \{\text{yield}\}$ augmented with a variable yield indicating the sequence of outputs has completed. As the inference with an inference policy is already synchronous, we can represent it directly with the process $p_1 = (I_1, O_1)$ with $I_1 = I_{\text{Inf}}$ and $O_0 = O_{\text{Inf}}$. Here the yield variable is output by the obfuscator process p_0 because the obfuscation policy controls the length of its output sequences; however, the yield variable is not available to the inference process p_1 as input because recipients do not observe these lengths. This defines a pipeline architecture $A = (O_{\text{env}}, p_0, p_1)$. We now show how to unfold behavior of the original or *folded* system into behavior over A.

In a single step of the folded system, the plant generates an output $v \in 2^{O_{\text{env}}}$, the obfuscation policy outputs the sequence $o_0 \cdots o_n \in \left(2^{O_{\text{env}}}\right)^+$, and finally the inference policy makes a corresponding sequence of inferences $p_0 \cdots p_n \in \left(2^{O_{\text{Inf}}}\right)^+$. By defining $f = (v, (o_0 \cup p_0) \cdots (o_n \cup p_n))$ we can view a step of the folded system as elements of the set

$$F = 2^{O_{\text{env}}} \times \left(2^{O_{\text{Obf}} \cup O_{\text{Inf}}}\right)^+ . \tag{6.7}$$

To unfold this step of the original system, we break each output of the obfuscation policy into a single step. The auxiliary output yield is used to indicate that output has completed. As there must be one environment output each step, we have it output v on the first step followed by $\emptyset$ on

subsequent steps. So we unfold the step, as $\mathbf{unfold}(f) = t_0 \cdots t_n \in \left(2^{V(A)}\right)^+$ where

$$t_0 = v \cup o_0 \cup p_0, \qquad \forall j \in \{1, \cdots, n-1\} : \; t_j = o_j \cup p_j, \tag{6.8}$$

$$t_n = o_n \cup p_n \cup \{\text{yield}\}\,, \tag{6.9}$$

when $n > 0$ and $t_0 = v \cup o_0 \cup p_0 \cup \{\text{yield}\}$ when $n = 0$.

Now we extend this notion from a single step to infinite traces. To this end, we denote the flattening or concatenating of a sequence of empty words into a sequence of letters by $\mathbf{flat} : (\Sigma^+)^\omega \to \Sigma^\omega$ so that for $s = (s_j)_{j\in\mathbb{N}}$ with $s_j \in \Sigma^+$, it holds $\mathbf{flat}(s) = s_0 s_1 \cdots$. We can then define the *traces* of the folded system by

$$\begin{aligned} \mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf}) = \{(e_j, a_j)_{j=0}^\infty \in F^\omega \mid \forall j \in \mathbb{N} : \; & a_j|_{O_{\text{Obf}}} = \mathbf{Obf}(e_0 \cdots e_j|_{I_{\text{Obf}}}), \\ & \text{for } (\tilde{a}_k)_{k\in\mathbb{N}} = \mathbf{flat}((a_j)_{j\in\mathbb{N}}), \\ & \forall k \in \mathbb{N} : \; \tilde{a}_k|_{O_{\text{Inf}}} = \mathbf{Inf}(\tilde{a}_0 \cdots \tilde{a}_k|_{I_{\text{Inf}}})\}\,. \end{aligned} \tag{6.10}$$

We denote the set of words possible after unfolding a single step by $U = \mathbf{unfold}(F) \subseteq \left(2^{V(A)}\right)^+$. So to unfold a trace in F^ω, we unfold each each step to an element of U and flatten the result. Formally, we define

$$P_U = \mathbf{flat}(U^\omega) \subseteq \left(2^{V(A)}\right)^\omega\,, \tag{6.11}$$

and $\mathbf{unfold} : F^\omega \to P_U$ by

$$\mathbf{unfold}(f_0 f_1 \cdots) = \mathbf{unfold}(f_0)\mathbf{unfold}(f_1) \cdots\,. \tag{6.12}$$

To demonstrate unfolding, consider a behavior of the plant from Example 6.2 as depicted in Figure 6.3 where the employee moves from the lobby to the office to the chemistry lab, i.e., $\{\mathcal{S}\} \to \{\mathcal{T}, \mathrm{S}\}$. We consider the obfuscation and inference policies $\mathbf{Obf}$ and $\inf$ depicted in Figure 6.5. Over this path, the obfuscator first outputs movement to office and back to the lobby, i.e., $\{\mathcal{S}'\} \to \{\mathcal{R}'\}$, then movement to the electronics lab and back to the lobby $\{\mathcal{T}'\} \to \{\mathcal{R}'\}$. We see as the inference policy is correct, for the first two locations it infers that they are not in the chemistry lab, i.e., $\emptyset$, then on the next two that they are, i.e., $\{\mathrm{S}'\}$. This corresponds to the finite folded trace

$$f = (\underbrace{\{\mathcal{S}\}}_{e}, \underbrace{\{\mathcal{S}'\}}_{a_0}\underbrace{\{\mathcal{R}'\}}_{a_1})\,(\{\mathcal{T}, \mathrm{S}\}, \{\mathcal{T}', \mathrm{S}'\}\{\mathcal{R}', \mathrm{S}'\})\,.$$

This is unfolded to

$$\mathbf{unfold}(f) = \underbrace{\{\mathcal{S}, \mathcal{S}'\}}_{t_0}\underbrace{\{\mathcal{R}', \text{yield}\}}_{t_1}\{\mathcal{T}, \mathrm{S}, \mathcal{T}', \mathrm{S}'\}\{\mathcal{R}', \mathrm{S}', \text{yield}\}\,.$$

The following result shows that **unfold** can be inverted to *fold* traces. Importantly, this defines a transformation between behaviors in the folded setting of Problem 6.1 and unfolded setting of Problem 2.1.

Lemma 6.2. *The map* $\mathbf{unfold} : F^\omega \to P_U$ *is a bijection.*

Proof. From the definition for a single step, we see **unfold** defines a bijection between F and U. Then as $\mathbf{unfold}(f_0 f_1 \cdots) = \mathbf{unfold}(f_0)\mathbf{unfold}(f_1)\cdots$ we see **unfold** is surjective onto P_U. Finally as every word in U ends with yield, there is a unique way to write traces in P_U as sequences of words in U (delimited by yield). So as **unfold** is injective for a single step, **unfold** is injective over F^ω. Hence **unfold** is a bijection. □

In addition to unfolding the traces of an obfuscation policy $\mathbf{Obf} : \left(2^{I_{\text{Obf}}}\right)^+ \to \left(2^{O_{\text{Obf}}}\right)^+$ and inference policy $\mathbf{Inf} : \left(2^{I_{\text{Inf}}}\right)^+ \to 2^{O_{\text{Inf}}}$, we can also unfold the functions themselves into strategies $s_0 : \left(2^{I_0}\right)^+ \to 2^{O_0}$ and $s_1 : \left(2^{I_1}\right)^+ \to 2^{O_1}$ implementing the architecture A. Note that strategies are defined for all input sequences, even those violating the yield behavior encoded in P_U. These violating traces do not correspond to traces in the folded system. As such, we say the strategies s_0 and s_1 are an *unfolding* of **Obf** and **Inf** if

$$\mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})) = \mathrm{Tr}(s_0, s_1) \cap P_U\,. \tag{6.13}$$

As obfuscation policies can only insert a finite sequence of outputs, we should only consider strategies s_0 that *always eventually yield*, i.e.,

$$\forall i \in \left(2^{I_0}\right)^+,\ \exists v \left(2^{I_0}\right)^+ :\ \text{yield} \in s_0(iv)\,. \tag{6.14}$$

In this case we have the following results.

Theorem 6.3.
Unfolding*: Every obfuscation policy* **Obf** *and inference policy* **Inf** *has an unfolding in the sense of* (6.13) *given by strategies* s_0 *and* s_1 *where* s_0 *always eventually yields and* $s_1 = \mathbf{Inf}$.
Folding*: For every strategy* s_0 *that always eventually yields and strategy* s_1*, there exists a unique obfuscation policy* **Obf** *and inference policy* **Inf** *such that* s_0 *and* s_1 *are an unfolding in the sense of* (6.13) *of* **Obf** *and* **Inf** *with* $s_1 = \mathbf{Inf}$.

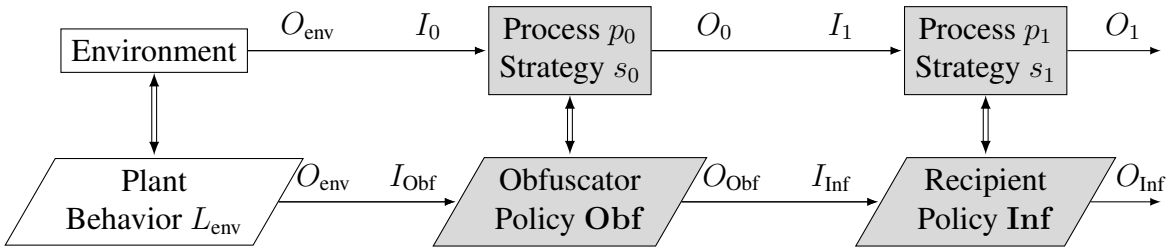

Figure 6.4: The pipeline architecture A (top) and the structure of the obfuscation system (bottom).

Proof. See appendix. □

This result shows that we can transform between policies over the folded system and strategies implementing the unfolded distributed pipeline architecture A as in Figure 6.4. Similar to strategies, the obfuscation policy can be represented by a transducer, and when this transducer is finite we say the policy is finite. As unfolding and folding preserve ω-regularity, this theorem implies that a solution to Problem 6.1 is finite if and only if it has an unfolding that is finite.

6.3.2 Unfolding the Specifications

In order to perform distributed synthesis on the unfolded system, we must map specifications for the folded system onto the unfolded one. To do this, we observe that private safety of the obfuscation policy $\mathbf{Obf}$ and the correctness of the inference policy $\mathbf{Inf}$ can be expressed as properties over the traces $\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})$. Then by unfolding these traces, we can express these requirements as ω-regular properties over the traces $\mathrm{Tr}(s_0, s_1)$ where s_0 and s_1 are an unfolding of $\mathbf{Obf}$ and $\mathbf{Inf}$. We define the properties of the folded system representing the plant behavior, private safety, and correct inferences, respectively, by

$$
\begin{aligned}
P_{\text{env}} &= \{(e_j, a_j)_{j=0}^{\infty} \in F^{\omega} \mid e_0 e_1 \cdots \in \lim L_{\text{env}}\} \\
P_{\text{NS}} &= \{(e_j, a_j)_{j=0}^{\infty} \in F^{\omega} \mid a_0 a_1 \cdots|_{O_{\text{Obf}}} \in \lim L_{\text{NS}}\} \\
P_{\text{Inf}} &= \{(e_j, a_j)_{j=0}^{\infty} \in F^{\omega} \mid \forall j \in \mathbb{N} :\ a_j|_{O_{\text{Inf}}} = \underbrace{\mathbf{Data}(e_j) \cdots \mathbf{Data}(e_j)}_{|a_j| \text{ times}}\} .
\end{aligned}
$$

Next, we construct Büchi automata that accept the unfolding of each of these properties.

To unfold the plant behavior, we recall that in the unfolded system the plant only progresses after the outputs have yielded and outputs $\emptyset$ otherwise. We utilize the following simple result:

Lemma 6.4. *If $L = L(G)$ where $G = (Q, \Sigma, \delta, Q_0)$ is finite and has no deadlocked state, i.e., each state has an outgoing transition, then $\lim(L)$ is closed and ω-regular, being accepted by the finite Büchi automaton $H = (Q, \Sigma, \delta, Q_0, Q)$, and $\overline{\lim L} = L$.*

As L_{env} satisfies the conditions of Lemma 6.4, there exists a finite Büchi automaton $H_{\text{env}} = (Q_{\text{env}}, 2^{O_{\text{env}}}, \delta_{\text{env}}, Q_{\text{env},0}, Q_{\text{env}})$ accepting $\lim L_{\text{env}}$. Let $H'_{\text{env}} = (Q'_{\text{env}}, 2^{V(A)}, \delta'_{\text{env}}, Q'_{\text{env},0}, Q'_{\text{env},m})$ where $Q'_{\text{env}} = Q_{\text{env}} \times \{0,1\}$, $Q'_{\text{env},0} = Q_{\text{env},0} \times \{0\}$, $Q'_{\text{env},m} = Q_{\text{env}} \times \{0\}$, and

$$\delta'_{\text{env}}((q,b),v) = \begin{cases} \delta_{\text{env}}(q, v|_{O_{\text{env}}}) \times \{0\}, & b = 0 \wedge \text{yield} \in v \\ \delta_{\text{env}}(q, v|_{O_{\text{env}}}) \times \{1\}, & b = 0 \wedge \text{yield} \notin v \\ \{(q,0)\}, & b = 1 \wedge \text{yield} \in v \wedge v|_{O_{\text{env}}} = \emptyset \\ \{(q,1)\}, & b = 1 \wedge \text{yield} \notin v \wedge v|_{O_{\text{env}}} = \emptyset \\ \emptyset, & \text{otherwise} \end{cases}$$

The first component q of the states of H'_{env} follows a stuttered path of the plant automaton H_{env}, repeating the current plant state until yield has occurred. This occurrence is tracked by the second component b of the states of H'_{env}, where $b = 0$ indicates that yield has occurred and the plant can transition. Only the states with $b = 0$ are accepting as the obfuscation policy only outputs finite sequences, i.e., yield occurs infinitely often. By construction $\mathbf{unfold}(P_{\text{env}}) = L(H'_{\text{env}})$. Furthermore, if H_{env} is deterministic then H'_{env} is as well and the size of H'_{env} is polynomial in the size of H_{env}.

Next, we unfold behavior representing private safety. As L_{NS} satisfies the conditions of Lemma 6.4, there exists a finite Büchi automaton

$$H_{\text{NS}} = (Q_{\text{NS}}, 2^{O_{\text{Obf}}}, \delta_{\text{NS}}, Q_{\text{NS},0}, Q_{\text{NS}}) \tag{6.15}$$

accepting $\lim L_{\text{NS}}$. Let $H'_{\text{NS}} = (Q'_{\text{NS}}, 2^{V(A)}, \delta'_{\text{NS}}, Q'_{\text{NS},0}, Q'_{\text{NS},m})$ where $Q'_{\text{NS}} = Q_{\text{NS}}$, $Q'_{\text{NS},0} = Q_{\text{NS},0}$, $Q'_{\text{NS},m} = Q_{\text{NS}}$, and

$$\delta'_{\text{NS}}(q,v) = \delta_{\text{NS}}(q, v|_{O_{\text{Obf}}})\,. \tag{6.16}$$

The automaton H'_{NS} simply accepts traces of the unfolded system whose restriction to the obfuscation outputs O_{Obf} are in L_{NS}. As unfolding does not alter these outputs, it holds that $\mathbf{unfold}(P_{\text{NS}}) = P_U \cap L(H'_{\text{NS}})$. Also, clearly H'_{NS} has the same number of states as H_{NS}.

Finally, we unfold behavior representing correct inferences, i.e, the inferred output is equal to the $\mathbf{Data}$ function of the current plant output. In the unfolded system, the "current" plant output corresponds to the value of the variables O_{env} after the most recent yield (or the initial value). So we construct H'_{Inf} accepting these traces as follows. Let $H'_{\text{Inf}} = (Q'_{\text{Inf}}, 2^{V(A)}, \delta'_{\text{Inf}}, Q'_{\text{Inf},0}, Q'_{\text{Inf},m})$

where $Q'_{\text{Inf}} = \{q_0\} \cup 2^{O_{\text{Inf}}}$, $Q'_{\text{Inf},0} = \{q_0\}$, $Q'_{\text{Inf},m} = Q'_{\text{Inf}}$, and

$$\delta'_{\text{Inf}}(q, v) = \begin{cases} v|_{O_{\text{Inf}}}, & \text{yield} \notin v \ \wedge \ v|_{O_{\text{Inf}}} = q \\ v|_{O_{\text{Inf}}}, & \text{yield} \notin v \ \wedge \ q = q_0 \ \wedge v|_{O_{\text{Inf}}} = \mathbf{Data}(v|_{O_{\text{Inf}}}) \\ q_0, & \text{yield} \in v \ \wedge \ v|_{O_{\text{Inf}}} = q \\ q_0, & \text{yield} \in v \ \wedge \ q = q_0 \ \wedge v|_{O_{\text{Inf}}} = \mathbf{Data}(v|_{O_{\text{Inf}}}) \end{cases}$$

The state of automaton H'_{Inf} tracks the "current" plant output until yield, and the automaton accepts only traces with inferences matching this output. Then it holds that $\mathbf{unfold}(P_{\text{Inf}}) = P_U \cap L(H'_{\text{Inf}})$. Also we note that the size of H'_{Inf} is polynomial in the number of possible inferences.

Additionally, we express the requirement of finite output sequences, represented by always eventually yielding as in (6.14) in the unfolded system, with a finite Büchi automaton H'_{yield}. Let $H'_{\text{yield}} = (Q'_{\text{yield}}, 2^{V(A)}, \delta'_{\text{yield}}, Q'_{\text{yield},0}, Q'_{\text{yield},m})$ where $Q'_{\text{yield}} = \{0, 1\}$, $Q'_{\text{yield},0} = \{0\}$, $Q'_{\text{yield},m} = \{0\}$ and

$$\delta'_{\text{yield}}(q, v) = \begin{cases} 0, & \text{yield} \in v \\ 1, & \text{yield} \notin v \,. \end{cases} \tag{6.17}$$

Then

$$L(H'_{\text{yield}}) = \{t \in \left(2^{V(A)}\right)^{\omega} \mid \forall j \in \mathbb{N}, \ \exists k \geq j : \ \text{yield} \in t_k\} \,. \tag{6.18}$$

Also note H'_{yield} has two states. We combine these properties to create a specification capturing the desired behavior of the unfolded system

$$\varphi = L(H'_{\text{yield}}) \cap (L(H'_{\text{env}})^c \cup (L(H'_{\text{NS}}) \cap L(H'_{\text{Inf}}))) \,. \tag{6.19}$$

Using standard constructions for the union, product, and complement of Büchi automata [4], we can construct a finite Büchi automaton that accepts this specification. Assuming the automaton generating the plant behavior L_{env} is deterministic, the size of the specification automaton is polynomial in the size of the automata generating the plant behavior L_{env}, nonsecret behavior L_{NS}, and number of possible inferences. With this specification we present our main results.

Theorem 6.5. *Consider an obfuscation policy* $\mathbf{Obf}$ *and inference policy* $\mathbf{Inf}$ *with an unfolding given by* s_0 *and* s_1 *where* s_0 *always eventually yields in the sense of* (6.14)*. Then* $\mathbf{Obf}$ *and* $\mathbf{Inf}$ *are solutions to Problem 6.1 with respect to* L_{env}*,* L_{NS}*, and* $\mathbf{Data}$ *if and only if* s_0 *and* s_1 *are solutions to Problem 2.1 for the architecture* A *and specification* φ*.*

Proof. From Definitions 6.3 and 6.2, we see that $\mathbf{Obf}$ enforces private safety and $\mathbf{Inf}$ is correct

with respect to L_{env}, L_{NS}, and $\mathbf{Data}$ if and only if

$$\overline{\text{Tr}(\mathbf{Obf}, \mathbf{Inf})} \cap \overline{P_{\text{env}}} \subseteq \overline{P_{\text{Inf}}} \cap \overline{P_{\text{NS}}} \,. \tag{6.20}$$

As $\text{Tr}(\mathbf{Obf}, \mathbf{Inf})$, P_{env}, P_{Inf}, and P_{NS} are all closed, this is equivalent to the infinite trace inclusion

$$\text{Tr}(\mathbf{Obf}, \mathbf{Inf}) \cap P_{\text{env}} \subseteq P_{\text{Inf}} \cap P_{\text{NS}} \,. \tag{6.21}$$

As $\mathbf{unfold}$ is a bijection, we can unfold each side of this inclusion. So by assumption as $\mathbf{unfold}(\text{Tr}(\mathbf{Obf}, \mathbf{Inf})) = P_U \cap \text{Tr}(s_0, s_1)$, the inclusion is equivalent to

$$\text{Tr}(s_0, s_1) \cap P_U \cap \mathbf{unfold}(P_{\text{env}}) \subseteq \mathbf{unfold}(P_{\text{NS}}) \cap \mathbf{unfold}(P_{\text{Inf}}) \,. \tag{6.22}$$

In turn by the definitions of $H_{\text{env}}, H_{\text{NS}}, H_{\text{Inf}}$, this inclusion is equivalent to

$$\text{Tr}(s_0, s_1) \cap P_U \cap L(H_{\text{env}}) \subseteq P_U \cap L(H_{\text{NS}}) \cap L(H_{\text{Inf}}) \,. \tag{6.23}$$

Rearranging terms and using the fact that $L(H_{\text{env}}) \subseteq P_U$, this is equivalent to

$$\text{Tr}(s_0, s_1) \subseteq L(H_{\text{Inf}}) \cap L(H_{\text{NS}}) \cup L(H_{\text{env}})^c \,. \tag{6.24}$$

By assumption as s_0 always eventually yields, it must be that $\text{Tr}(s_0, s_1) \subseteq L(H_{\text{yield}})$. Hence by the definition of φ, the inclusion is equivalent to $\text{Tr}(s_0, s_1) \subseteq \varphi$. □

Applying the results of Theorem 6.3, we see if there is a solution to Problem 6.1 then there must be an unfolding that is a solution to Problem 2.1. Conversely, if there is a solution to Problem 2.1 the synthesis method [38] finds a finite solution. Applying the results of Theorem 6.3 that this solution can be folded into a solution to Problem 6.1 that is also finite. Hence if the obfuscation synthesis problem has a solution, finite obfuscation and inference policies solving the problem can be found by solving the corresponding pipeline synthesis problem and folding the result. Applying this approach to the problem from Example 6.2 yields the policies depicted in Figure 6.5. Assuming the automaton generating the plant behavior L_{env} is deterministic, the size of this solution is double-exponential in the size of the automata generating L_{env} and L_{NS} and number of possible inferences.

6.4 Case Study: Contact Tracing

In this section, we demonstrate how our framework can be used in the context of smartphone apps developed for contact tracing. We model apps that record proximity between users to a centralized server. These apps raise a variety of privacy concerns as described in [17, 78], including

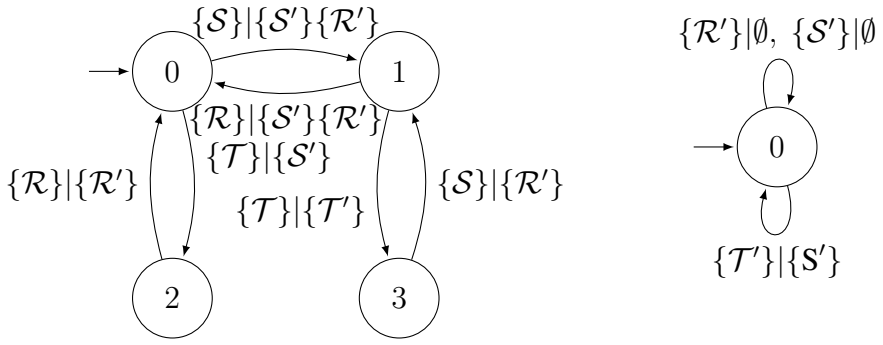

Figure 6.5: The solution to the obfuscation problem described in Example 6.2 given by transducers representing the obfuscation policy (left) and inference policy (right).

the disclosure of user location information due to unsecured networks. Our synthesis method provides solutions that enforce location privacy from the malicious actors while maintaining utility for public health by providing professionals with relevant contact information. We demonstrate this approach on small model similar to the one developed in [95] where a malicious user has gained access to the app. Our presentation of this model is condensed. More detail on the construction can be found in [95].

6.4.1 Modeling

In our model, we consider a number of normal users and one user that is malicious. As in Example 6.1, their movement is constrained by a graph $\Gamma = (\mathcal{V}, \mathcal{E})$ depicted in Figure 6.6 representing the layout of their city. The users' smart devices would report their approximate location given by their current region in a partition P of the graph Γ similar to the model proposed in [105]. As such, we define **region** : $\mathcal{V} \to P$ so that for $v \in \mathcal{V}$, **region**(v) is the region containing v. We assume the malicious user has compromised the users' location-based service apps and has access to the reported regions, and also knows their own location exactly. We further assume that this user has compromised the contact tracing app which reports which users are in contact with each other, i.e., share the same location. As a privacy measure, the app does not report the location of the contact.

We suppose that there are $n = 3$ users and consider any user visiting the secret location 5 as secret in the plant. Using Γ, we first construct the map automaton $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Delta, \mathcal{V}_0)$, where $\mathcal{V}$ is the set of states, $\mathcal{E}$ is the set of events, $\Delta : \mathcal{V} \times \mathcal{E} \to 2^{\mathcal{V}}$ is the transition function, and $\mathcal{V}_0$ is the set of initial states. We define $\mathcal{V}_0 = \mathcal{V}$, $\mathcal{E} = \{\tau\}$, and $\Delta(v, \tau) = \{v\} \cup \{u \in \mathcal{V} : (u, v) \in \mathscr{E}\}$ for all $v \in \mathcal{V}$.For each user denoted $i \in \{1, 2, \ldots, n\}$, we construct an automaton $\mathcal{G}_i$ modeling their movement. We let user 1 denote the malicious user and construct $\mathcal{G}_1$ from $\mathcal{G}$ by removing the secret location (if the malicious user can enter the secret location, the problem trivially has no solution). For users $i > 1$, we let $\mathcal{G}_i = \mathcal{G}$ as their movements are unrestricted. For example, the individual

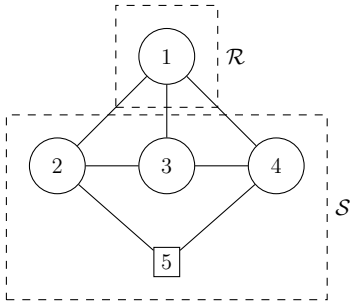

Figure 6.6: The graph Γ that represents the locations and the physical paths between them. Regions $P = \{\mathcal{R}, \mathcal{S}\}$ represent the approximate locations reported by smart phones. Location 5 is considered to be secret.

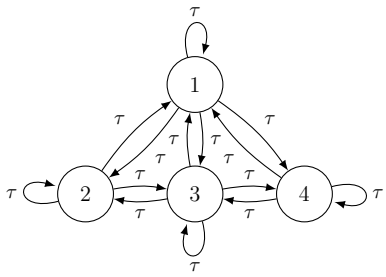

Figure 6.7: The automaton $\mathcal{G}_1$, representing the movement of the malicious user 1.

automaton $\mathcal{G}_1$ is shown in Figure 6.7. The synchronous movement of all the users is then described by the product automaton $\mathcal{H} = \mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_n$. Using the product construction, $\mathcal{H}$ has states $x = (v_1, ..., v_n) \in \mathcal{V}^n$ where v_i represents the current location of user i.

Given a state $x \in \mathcal{V}^n$, we define

$$\alpha(x) = (\alpha_1, ..., \alpha_n) \tag{6.25}$$

where for each i,

$$\alpha_i = \begin{cases} v_1, & i = 1 \\ \textbf{region}(v_i), & i > 1. \end{cases} \tag{6.26}$$

When the system enters state x, the malicious user observes the information in $\alpha(x)$, i.e, their own location and the regions of other users. We also define the set

$$\beta(x) = \{(i, j) \mid 1 \leq i < j \leq n, \;\; v_i = v_j\}, \tag{6.27}$$

which represents the contact information of the users at state x, i.e. the pairs of users that share the same location. To put this model into the setting of synthesis, we consider Boolean encodings of the

outputs of α and β denoted by α_b and β_b, respectively. Finally, we construct an automaton G_{env} from $\mathcal{H}$ by labeling each transition to a state x with the variables $\alpha_b(x) \cup \beta_b(x)$. In order to construct the specification φ, we also determinize G_{env} using the powerset construction. The automaton G_{env} generates the plant behavior $L_{\text{env}} = L(G_{\text{env}})$ which satisfies the conditions of Lemma 6.4, i.e., G_{env} is finite and has no deadlocked states. Recall for privacy, the malicious user must not determine users are at the secret location, i.e., if in the current state of G_{env} any user is in the secret location which we call a secret state. As such we define the malicious user's nonsecret plant model $L'_{\text{env,NS}}$ as the language generated by the automaton G_{env} after removal of the secret states. As we assume they are not aware of obfuscation, their class of possible obfuscation policies $\mathcal{S}_{\text{Obf}}$ is just the identity map. Finally, as the contact information represented by β_b should be inferred, we define the function $\mathbf{Data}$ for the plant output $e = \alpha(x)_b \cup \beta_b(x)$ by $\mathbf{Data}(e) = \tilde{\beta}_b(x)$, where $\tilde{\beta}_b(x)$ is a copy of the variables in $\beta_b(x)$ (to ensure variables are disjoint). Together these components define an instance of Problem 6.1 which by Theorem 6.5 can be transformed into a corresponding instance of Problem 2.1. In this form, we also add the additional ω-regular constraint that the obfuscation policy cannot alter the location of the malicious user encoded in α_b as this would alert them to the existence of obfuscation.

6.4.2 Implementation and Results

While the pipeline synthesis problem can be solved directly using automata theoretic methods, due to a lack of available tools and to take advantage of the performance of bounded synthesis methods, we use a different approach. There is a straightforward reduction of a distributed synthesis problem to a decidable hyperproperty satisfiability problem [34, 35]. Hyperproperties are properties of a system quantified over multiple traces of the system. The key idea of the reduction is to encode the variable dependence induced by the distributed architecture into a hyperproperty. Specifically, this hyperproperty ensures that any traces of the system with the same input up to a point must have the same output. Using this reformulation, we have implemented our synthesis method using the bounded synthesis tool *BoSy*[3]. We construct the Büchi automaton accepting the specification φ from (7.10), and perform minimization to reduce the number of states of this automaton while maintaining the specification language in order to improve performance. We then provide the tool with this automaton as well as the pipeline architecture encoded in HyperLTL, a temporal logic for hyperproperties. When a solution exists, the tool returns the smallest solution encoded as a finite automaton. Again, strategies for the obfuscation policy and the inference policy represented as transducers are readily extracted from this monolithic automaton.

The automaton G_{env} representing the plant behavior consists of 60 states after minimization. From this, we construct the Büchi automaton accepting the specification φ which has 968 states.

[3] `https://www.react.uni-saarland.de/tools/bosy/`

With this automaton as input, the tool was able to synthesize a solution within 25 minutes on a machine with typical specs. As an automaton encoding both the obfuscation and inference policies, the solution mirrors the structure of the plant automaton G_{env}, possessing a corresponding 60 states. After extraction from the solution, the obfuscation and inference policies constructed guarantee privacy in the form of private safety while maintaining utility in the form of providing correct contact information.

6.5 Conclusion

Balancing privacy and utility within a networked dynamic system presents an interesting challenge. To achieve this, we propose a framework of obfuscation with an inference policy that allows intended recipients to interpret obfuscated information. We present a method for automatically designing both obfuscation and inference policies using techniques from distributed synthesis. This approach allows for a variety of specifications for utility and privacy in the form of temporal logic. We also developed a software implementation of this approach and demonstrate its effectiveness in enforcing privacy on a contact-tracing system model.

We remark that our proposed obfuscation framework is orthogonal to cryptographic methods for network privacy. While cryptography achieves privacy by ensuring outputs appear arbitrary, obfuscation achieves privacy by deception, i.e., ensuring outputs are consistent with a recipient's model of nonsecret behavior. In this way our obfuscation framework is similar to the notion of *network steganography* [62]. Additionally, our approach can be applied in cases where encryption cannot.

CHAPTER 7

Integrating Control and Obfuscation

7.1 Introduction

Privacy modeled as opacity can be enforced with a variety of mechanisms. However, most works on opacity enforcement consider a specific mechanism with a limited scope of application. For example, it may be impractical to restrict some aspects of a system's behavior with supervisory control, such as human actions in a physical system. Likewise, it may not be possible to completely alter a system's outputs with obfuscation, such as when those outputs are publicly observable. As such, many practical applications may require the integration of multiple mechanisms to enforce privacy.

In altering either the system's behavior or observations, these mechanisms must also maintain the system's *utility*, e.g., a controller must maintain safe behavior or an observer must monitor the system accurately. For example, utility constraints on obfuscators in [104] require that observers can infer some specified information about the system's state; however, it is assumed that all observers, both intended and unintended, have the same capabilities and access to information. This may not be the case when controllers and observers are distributed across a network, or when intended recipients need access to sensitive information while unintended ones do not. Chapter 6 considers the enforcement of privacy and utility in this distributed setting, but is limited to obfuscation within a simple linear network topology (pipeline) like the one depicted in Fig. 7.1. A similar problem with the same topology but with alternative assumptions is also considered in [61]. The addition of control imposes a new challenge for privacy, as an eavesdropper may observe all information transmitted across the network including both sensor outputs and control commands.

In this chapter, we address the problem of privacy and utility enforcement in distributed systems utilizing both obfuscation and control. We develop a solution approach for components modeled by automata by leveraging distributed reactive synthesis [38]. While this problem is undecidable for general network architectures, we demonstrate our approach on three representative networked control problems which we show to be decidable. We discuss how privacy and various utility constraints can be expressed with the ω-regular specifications used in our approach. In particular

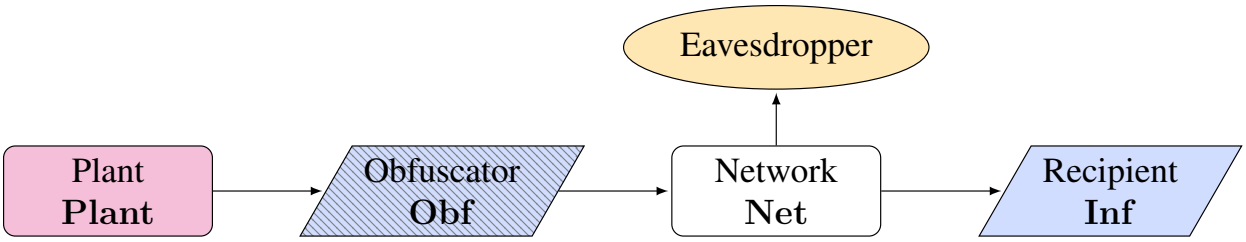

Figure 7.1: The architecture for obfuscation and inference without control considered in Chapter 6.

we consider language-based opacity [57] which has been used to express many other existing notions of opacity [97] in DES. By explicitly modeling the eavesdropper's beliefs about the system's implementation, this framework can precisely express many practical privacy requirements. *Our main contribution is the formulation of privacy and utility enforcement with obfuscation and control as a distributed reactive synthesis problem, demonstrated over a number of practical network architectures*.

A similar problem of opacity enforcement is considered in [90] utilizing control, obfuscation, and dynamic masks in a fixed network topology. Our work differs in a few key aspects. Importantly, our approach is applicable to general network architectures, and furthermore, it is *complete* for the three problems we focus on. In contrast, the synthesis method of [90] is incomplete, potentially not finding a solution when one exists. Secondly, in [90] they consider control with supervisors which can realize nondeterministic behavior, whereas our controllers are deterministic as implementations of reactive processes. The complex relation between these two control approaches are thoroughly discussed in [88]. Thirdly, [90] employs dynamic masks and obfuscation with general edit functions, while for simplicity we limit our discussion to replacement functions.

We motivate our problem and approach with the following running example.

Example 7.1. *Consider the building depicted in Fig. 7.2 whose doors are equipped with keypad sensors and controllable locks. At each door, the keypad reports entry attempts to an authorization server which responds with a signal to open the door or not. These signals may contain sensitive information which raises privacy concerns if the server is remote. For example, an eavesdropper may use their knowledge of the building's layout to deduce room occupancy from their observation of entry attempts at the keypads. While this risk can be mitigated by keeping all information at the local site, it may be infeasible to alter the system's existing network architecture.*

In this case, obfuscation can be employed to alter both the keypad outputs sent to the server and control outputs sent back to the building. At the same time, the system must maintain its utility. This may concern a remote user's access to information, for example to diagnose a faulty door. Additionally, utility may require that rooms are accessible by authorized users, i.e., after using the keypad, the door is eventually signaled to open. Our problem is then to design both obfuscators and

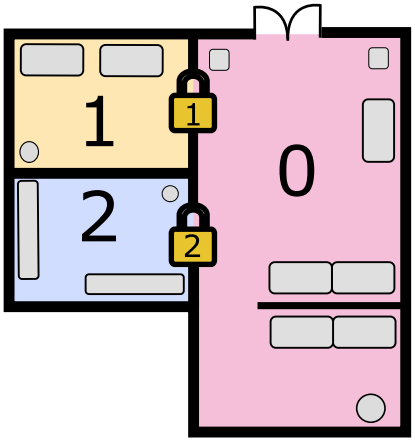

Figure 7.2: The layout of a building with two electronically-controlled locked doors. At each door a keypad, shared by both of its sides, controls the lock via a potentially remote authorization server.

controllers for a given network architecture to enforce privacy and utility reactively as an individual moves about the building.

The rest of this chapter is organized as follows. In Chapter 2, we review concepts from formal languages and discuss our methodology using results from distributed reactive synthesis. Next, we develop our modeling and synthesis approach for privacy and utility enforcement in Section 7.2, over a networked system employing both obfuscation and a local controller. Then in Section 7.3, we apply this approach to the problems of designing a remote controller for utility and securing an existing remote controller, respectively, alongside obfuscation for privacy. Finally, we discuss our implementation of this approach in Section 7.4 in application to the building access control system.

7.2 Integrating Obfuscation and Control Locally

In this section, we present the first of three problems which integrate obfuscation and control in a networked system to enforce privacy and utility. We will discuss modeling with distributed systems, then specifications for privacy and utility, and finally, a method for synthesis. We build upon the system architecture discussed in [96] and depicted in Fig. 7.1, in which a plant dynamically produces outputs which are obfuscated before being broadcast on a network and acted upon by a recipient. Here, we additionally consider that some behavior of the plant may be restricted or controlled locally in order to better enforce privacy and maintain utility for the plant and recipient. This controller uses the observable outputs from the plant as feedback to generate inputs to the plant. The components of this networked system including the controller are modeled by processes in a distributed system whose architecture, referred to as Architecture 1, is depicted in Fig. 7.3. Note that the control action is assumed to be communicated locally, i.e., not broadcast on the network. As

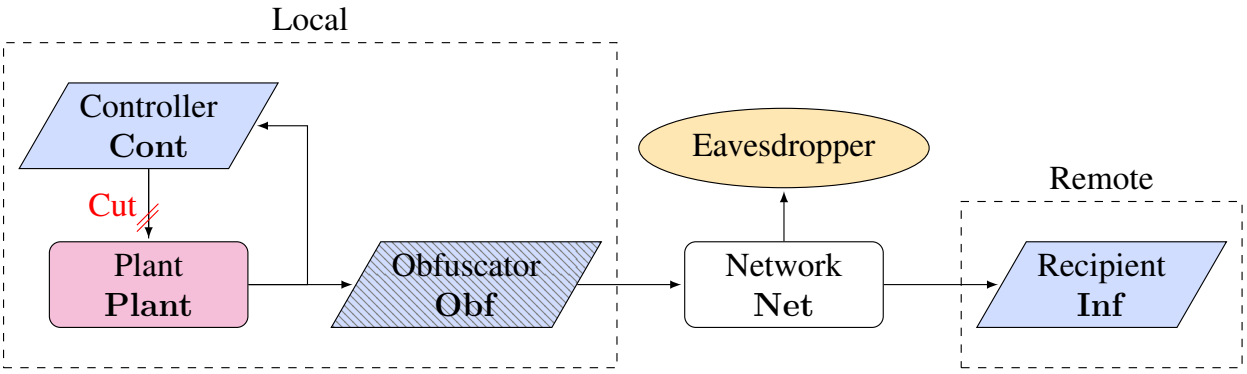

Figure 7.3: **Architecture 1** featuring control and obfuscation at the local site which transmit information to the recipient at the remote site. The edge labeled *cut* indicates the feedback eliminated in the transformation used for synthesis in subsection 7.2.4. The black-box processes to be synthesized are represented by parallelograms. The striped parallelograms denote processes unknown by the eavesdropper.

such, we are not concerned with the control action leaking information as is considered in Sections 7.3.1 and 7.3.2.

7.2.1 System Model

The overall system is modeled as a distributed system with architecture $A = (P, W, \mathbf{env}, E, O, H)$. The plant, obfuscator, controller, network, and recipient are represented by the processes $P = \{\mathbf{Plant}, \mathbf{Obf}, \mathbf{Cont}, \mathbf{Net}, \mathbf{Inf}\}$. The interconnection of these processes $E \subseteq P \times P$ is depicted in Fig. 7.3.

The plant drives the system, nondeterministically producing outputs which must be conveyed to the recipient. As such, it acts as the environment process, i.e., $\mathbf{env} = \mathbf{Plant}$. The observable outputs of the plant are communicated to both the obfuscator and controller, i.e., $I_{\text{Obf}} = I_{\text{Cont}} \subseteq O_{\text{Plant}}$, while its hidden outputs represent its internal state. The controller process $\mathbf{Cont}$ provides feedback to the plant $I_{\text{Plant}} = O_{\text{Cont}}$ in order to enforce privacy (e.g., restricting secret-revealing behavior) and utility (e.g., restricting unsafe behavior). We assume the dynamics relating the outputs from the plant to inputs from the controller are described by the ω-regular language M_{Plant} over the plant variables V_{Plant}.

The obfuscator process $\mathbf{Obf}$ modifies the outputs of the plant before they are broadcast on the network to enforce privacy. As the obfuscator seeks to mimic the plant, its outputs are copies of the plant's outputs. Formally, we define $O_{\text{Obf}} = \{o_{\text{Obf}} \mid o \in I_{\text{Obf}}\}$ where o_{Obf} denotes a distinct copy of the plant output variable o which may take on different values. We emphasize that an implementation of the obfuscator is a strategy $s_{\text{Obf}} : \left(2^{I_{\text{Obf}}}\right)^+ \to 2^{O_{\text{Obf}}}$ which in each step replaces a single input from the plant with a single obfuscated output. This corresponds to the notion of a

deterministic *edit function* using only replacement [104].

The network broadcasts the outputs it receives from the obfuscator to all recipients on the network, both intended and unintended. In order to capture the potential dynamics of the network, such as a delay or bandwidth limitation, we model the network as a white-box process $\mathbf{Net}$. In Architecture 1 the network receives input from the obfuscator $I_{\text{Net}} = O_{\text{Obf}}$ and transmits copies $O_{\text{Net}} = \{o_{\text{Net}} \mid o \in I_{\text{Net}}\}$. We assume the network has a fixed implementation as a deterministic strategy $s_{\text{Net}} : \left(2^{I_{\text{Net}}}\right)^+ \to 2^{O_{\text{Net}}}$. As such, the network $\mathbf{Net}$ is the only white-box process $W = \{\mathbf{Net}\}$ with $S_W = \{s_{\text{Net}}\}$. In the case that the network directly broadcasts its inputs without delay as considered in our examples, we may omit it from the architecture, directly connecting the obfuscator to the recipient.

The final process $\mathbf{Inf}$ in the system models the actions of the intended recipient of the plant's information, for example *inferring* sensitive plant information. It utilizes the obfuscated outputs from the network $I_{\text{Inf}} = O_{\text{Net}}$ to take action at the remote site modeled by its outputs O_{Inf}.

Example 7.2. *In the building system from Example 7.1, the authorization server controlling the locks may be local to the building, but outputs from the keypads are shared over a network with their manufacturers in order to diagnose faults requiring maintenance. In particular, door 2 may experience a fault preventing it from opening. We can model this system with Architecture 1 utilizing obfuscation of the keypad signals for privacy. In this model, the plant receives inputs I_{Plant} given by the control c^j from the server signaling door j to unlock for $j \in \{1, 2\}$. Likewise, the plant produces outputs O_{Plant} consisting of o^j indicating door j is open, k^j indicating keypad j is pressed, and f indicating door 2 is faulty with $j \in \{1, 2\}$.*

We assume that locally, the keypad and door outputs k^j, o^j are observed while the fault f is hidden. In addition, we assume that the outputs of the plant are delayed by one step before observation which can be represented by introducing delayed copies of these variables; however, in an abuse of notation we simply write $I_{Cont} = I_{Obf} = \{k^j, o^j\}$. The outputs of the keypads, but not whether the doors are opened, are nominally communicated to the manufacturer to infer a fault with door 2. The obfuscator replicates these outputs over the network with its own set of outputs O_{Obf} given by k^j_{Obf} for $j \in \{1, 2\}$. For simplicity, we assume the network communicates these values unmodified without delay, i.e., in an abuse of notation $I_{Net} = O_{Net}$. Then finally, the manufacturer is the recipient of these obfuscated outputs $I_{Inf} = O_{Obf}$ and produces a single output $O_{Inf} = \{f_{Inf}\}$ whenever the fault has been inferred.

Now we describe the plant dynamics M_{Plant}. We assume the user starting from room 0, always remains at a keypad once pressed until the corresponding door is signaled to open, moving through the door if it has opened. Furthermore, the doors always open immediately once signaled unless the fault forces door 2 to remain closed forever. In order to allow the fault to be diagnosed, we must also assume a kind of liveness that the user always eventually presses accessible keypads, i.e., if

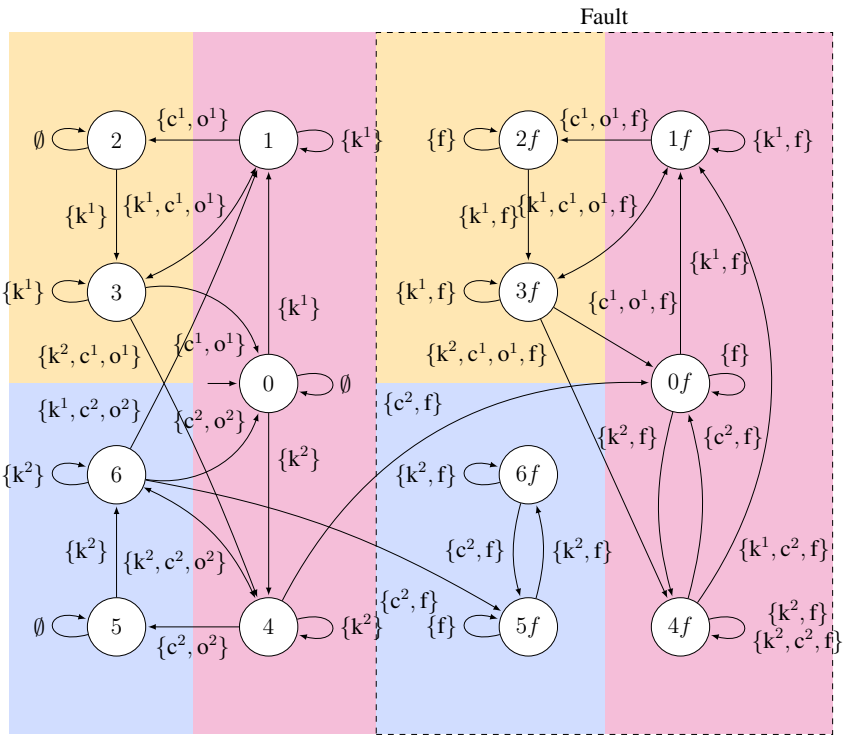

Figure 7.4: An automaton encoding the plant from Example 7.2. Not depicted are transitions accepting invalid control outputs, e.g., when no keypad is pressed. Background colors indicate which room from Fig. 7.2 corresponds to each state.

the plant visits a state with a transition labeled with k^i infinitely often, then k^i must occur infinitely often. The resulting dynamics are represented by the automaton depicted in Fig. 7.4. Selected traces of the system under this plant behavior are depicted in Fig. 7.5.

Remark 7.1. *Beyond replacement, obfuscation can be implemented with edit functions which also delete outputs or insert fictitious ones. In [96], distributed synthesis is used to design obfuscators with insertion. This work transformed the system so the obfuscator outputs one insertion in each step to fit the standard framework of* synchronous *systems considered by distributed synthesis. To this end the obfuscator was augmented with an additional output yield to indicate the end of insertions, and the specification was modified to ensure the plant holds its outputs until yield occurs. This approach is conceptually similar to introducing a local controller as in Architecture 1 with a single output yield. We may utilize a similar approach to design obfuscators with insertion; however, synchronization is more complex in the presence of multiple feedback paths. So for simplicity, we only discuss obfuscation by replacement.*

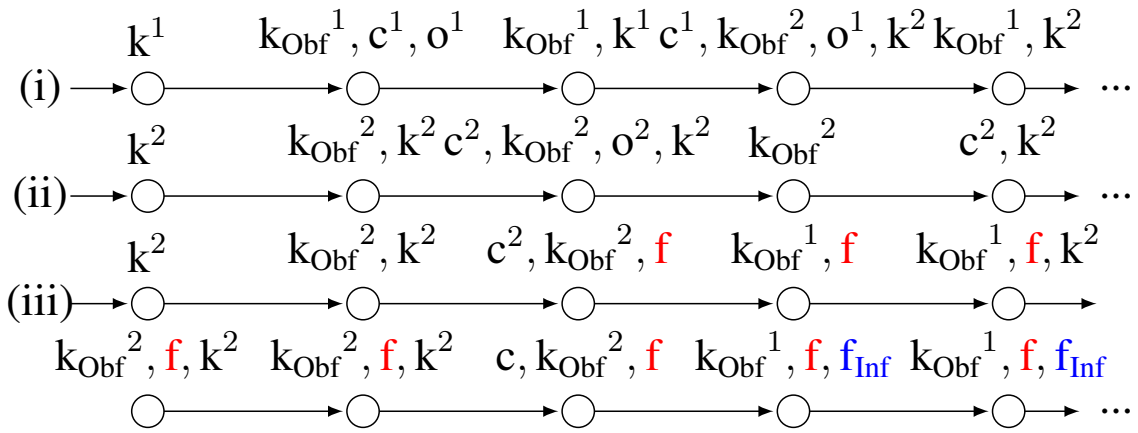

Figure 7.5: Possible traces of the building system in Architecture 1. In the first trace, the user passes through door 1 to room 1 and then returns the same way to room 0. During this movement, the obfuscator violates privacy. In step 4, the eavesdropper believes keypad 1 was used immediately after keypad 2 was used from room 0. This could only happen in the original system if a fault prevented door 2 from opening. The remaining traces are drawn from this system implemented as in Fig. 7.6. The second trace represents non-faulty behavior as the user passes through door 2, whereas the third trace, displayed over two lines, contains the fault f. After the user tries to use door 2 again, the recipient has observed 5 occurrences of $k_{Obf}{}^2$ (an odd number) followed by $k_{Obf}{}^1$, and thus correctly infers a fault has occurred and output f_{Inf}.

7.2.2 Privacy Requirements

Knowledge about the current behavior of the plant or system may be used by a malicious agent to damage the system or harm its users. As such, we consider requirements on the privacy of the plant's behavior. We model privacy as the opacity of a set of behaviors identified as *secret*. Opacity requires that the occurrence of these behaviors can never be deduced by an eavesdropper. In particular, we consider the notion of *private safety* [49], where the eavesdropper is aware of the plant's dynamics but not of obfuscation, i.e., the implementation and goals of obfuscation are *private* knowledge. We adapt this notion of private safety to infinite traces over distributed systems with obfuscation and control, expressed as an ω-regular specification. In this more general setting, we assume that the eavesdropper possesses a *nominal model* of the system without obfuscation. Privacy requires that the eavesdropper does not deduce secrets within this model, regardless of what observations are made of the obfuscated system.

In particular, we model these nominal and secret behaviors as ω-languages, adapting of the notion of *language-based opacity* (LBO) [57] to infinite strings. Formally, it is assumed that the eavesdropper's nominal model of the system is given by a known language $\hat{M} \subseteq \left(2^V\right)^\omega$. Within this model, we must ensure they cannot deduce some secret aspects of the behavior given by the *secret language* $M_S \subseteq \left(2^V\right)^\omega$. The strings in the language $\hat{M} \setminus M_S$ are called *nonsecret*. We assume

the eavesdropper observes a subset of the output variables $V_{\text{obs}} \subseteq V$ shared between the nominal and actual system, i.e., a string $s \in \left(2^V\right)^\omega$ is observed as $s|_{V_{\text{obs}}}$. Using this nominal model, the eavesdropper deduces that they have observed secret behavior if their observation could not have resulted from nonsecret behavior. Formally, upon the occurrence of the string s, the eavesdropper *cannot* deduce if it was secret if $s|_{V_{\text{obs}}} \in (\hat{M} \setminus M_{\text{S}})|_{V_{\text{obs}}}$. So we make the following definition for language-based privacy.

Definition 7.1 (Privacy). *Let $\hat{M} \subseteq \left(2^V\right)^\omega$ be the nominal language, $M_S \subseteq \left(2^V\right)^\omega$ the secret language, and $V_{obs} \subseteq V$ the observed variables. Then we say the language $M \subseteq \left(2^V\right)^\omega$ enforces* privacy *if*

$$M|_{V_{obs}} \subseteq (\hat{M} \setminus M_S)|_{V_{obs}} . \tag{7.1}$$

The right side of this inclusion represents observations that are both consistent with the eavesdropper's model of behavior and nonsecret. From this definition, we can observe some simple results about monotonicity.

Proposition 7.1. *Assume that M enforces privacy for secrets M_S in a fixed nominal model $\hat{M}$. If $M' \subseteq M$ and $M'_S \subseteq M_S$ then M' enforces privacy for M'_S.*

We require that our distributed system enforce privacy, i.e., the infinite traces of the system are consistent with nonsecret behavior.

$$\varphi_{\text{priv}} = \left((\hat{M} \setminus M_{\text{S}})|_{V_{\text{obs}}}\right)|^V . \tag{7.2}$$

While it is not the focus of this work, the standard notion of LBO with respect to a language of finite nominal behaviors $\hat{L} = \overline{\hat{M}}$ and finite secret behaviors L_{S} [57] can be expressed as a safety property φ_{priv} with finite prefixes given by

$$\overline{\varphi_{\text{priv}}} = \left((\hat{L} \setminus L_{\text{S}})|_{V_{\text{obs}}}\right)|^V . \tag{7.3}$$

As shown in [97] many existing notions of opacity may be expressed as LBO. Examples include current-state opacity, initial-state opacity, and notions of K-step opacity.

In general, the nominal model can be any language representing the eavesdropper's beliefs about the system's behavior. These beliefs may be uncertain. For example, an eavesdropper may know the plant behaviors M_{Plant} resulting in a nominal model satisfying $\hat{M}|_{V_{\text{Plant}}} \subseteq M_{\text{Plant}}$. On the other hand, beliefs about the system may also be incorrect. For example, the eavesdropper may be unaware of some processes in the system's architecture, such as the obfuscator. We can model this nominal belief by "shorting out" these processes, treating them as white-box processes with a fixed implementation that directly passes inputs to their corresponding output copies. In this way, we

can construct a nominal model over the same variables as the true system model which captures knowledge of the plant dynamics but only a subset of the system's processes denoted $\hat{P}$ as follows.

Definition 7.2. *Given an eavesdropper aware of some nominal processes $\hat{P}$ in the architecture A with plant dynamics M_{Plant}, this knowledge is captured by the* base nominal model *defined by*

$$\hat{M}_0 = M_{Plant}|^V \cap \bigcap_{p \in W \cup \hat{P}} \mathrm{Tr}(s_p)|^V \,, \tag{7.4}$$

where the implementations s_p of processes $p \in P \setminus \hat{P}$ pass their inputs to corresponding output copies.

We demonstrate constructing the privacy specification with the following example.

Example 7.3. *Returning to the building system, we suppose the eavesdropper is unaware of obfuscation in Architecture 1, but is aware of the plant dynamics M_{Plant}. So they model the system as in Fig. 7.3 with the indicated process* Obf *unknown. Formally, this means $P \setminus \hat{P} = \{\mathbf{Obf}\}$ which defines the base nominal language $\hat{M}_0$ as in Definition 7.2. Likewise, they may assume that the doors are eventually controlled to open after the keypad is pressed. This requirement is expressed by the LTL formula*

$$\varphi_{Cont} = \bigwedge_{j \in \{1,2\}} \mathrm{G}\left(k^j \Rightarrow \mathrm{F}\, c^j\right) \tag{7.5}$$

Then the nominal language reflecting their beliefs is given by $\hat{M} = \hat{M}_0 \cap \varphi_{Cont}$. If the occurrence of the fault f should be hidden from the eavesdropper for security reasons, we consider the secret language M_S expressed by the LTL formula $\mathrm{F} f$. One can show that this induces the admissible language

$$(\hat{M} \setminus M_S)|_{V_{obs}} = \left(L_1^+ L_2^+\right)^\omega \cup \left(L_2^+ L_1^+\right)^\omega \,, \tag{7.6}$$

where $L_i = \left(\{\emptyset\}^* \{\{k_{Obf}{}^i\}\}^+\right)^2$ corresponds to observations of the user going through door $i \in \{1,2\}$ twice. Roughly this requires the eavesdropper to observe at least two presses of the keypad at a door before one at the other door, and that both keypads are pressed infinitely often. Otherwise, the eavesdropper knows the user left the keypad before the door was opened or that the user was not able to open the door, so a fault is believed to have occurred. This is used to construct the specification φ_{priv} as in Equation (7.2)*, which describes traces of the system enforcing privacy.*

Remark 7.2. *While this framework can include knowledge about obfuscation, conceptually, one must be careful that the nominal model accurately reflects this knowledge. An issue arises if the eavesdropper knows* why *obfuscation is being implemented, i.e., the privacy specification. In this case, a direct description of the nominal model and privacy specification are self-referential: the*

nominal model encompasses systems satisfying the privacy requirement which is in turn defined with respect to the nominal model. A similar challenge arises when the eavesdropper knows the specification for the controller, which we address in subsection 7.3.1. One way to resolve the issue with the privacy requirement is to make stronger assumptions on the eavesdropper's knowledge, namely that the implementation of obfuscation is public knowledge. *Privacy in this case is referred to as* public safety *in [49] which also presents corresponding synthesis methods.*

7.2.3 Utility Requirements

Utility refers both to desirable behavior of the plant as well as the recipient's access to information. For example the *utility constraints* proposed in [104], roughly require that all observers unaware of obfuscation are still able to infer which region of states the system currently inhabits. In order to allow for information about the plant to be hidden from unintended observers unaware of obfuscation yet revealed to intended ones that are aware, [96] proposed an alternative utility requirement which we discuss now.

To model this type of specification, we identify a subset of the plant outputs $\mathbf{Data} \subseteq O_{\text{Plant}}$ that should be inferred by the recipient. We model the inference of the recipient explicitly with the output of the process $\mathbf{Inf}$. As such the recipient outputs should match the plant outputs $\mathbf{Data}$. To ensure the output sets are disjoint, we define the outputs $O_{\text{Inf}} = \{o_{\text{Inf}} \mid o \in \mathbf{Data}\}$ as copies of the variables in $\mathbf{Data}$. The requirement that the recipient infers the outputs $\mathbf{Data}$ from the plant can be modeled with the LTL formula

$$\varphi_{\text{Data}} = \mathrm{G} \bigwedge_{o \in \text{Data}} (o \;\Leftrightarrow\; \mathrm{X}\, o_{\text{Inf}}) \; . \tag{7.7}$$

Instead of requiring the recipient to infer the current plant output after a one step delay, we may consider more temporally complex relations as demonstrated in the following example.

Example 7.4. *In the building system, we require that the remote recipient must be able to monitor the building and diagnose door lock faults. Formally, they must eventually infer if f occurs in the current trace, i.e.,* $\mathbf{Data} = \{f\}$. *The diagnosis specification can then be expressed with the LTL formula*

$$\varphi_{diag} = \mathrm{F} f \;\Leftrightarrow\; \mathrm{F} f_{Inf} \, . \tag{7.8}$$

In addition, we will also require that the controller eventually signals the doors to open if the keypads have been pressed. This is captured by the previous specification φ_{Cont} defined in equation (7.5). *The utility requirement is then the combination*

$$\varphi_{util} = \varphi_{diag} \cap \varphi_{Cont} \tag{7.9}$$

7.2.4 Synthesis

With this system model and these specifications we can state the design problem for Architecture 1.

Problem 7.1 (Architecture 1). *Given an instance A of Architecture 1 and ω-regular privacy and utility specifications φ_{priv} and φ_{util}, find an implementation S for* **Obf**, **Cont**, *and* **Inf** *solving the distributed synthesis problem for A with specification*

$$\varphi = \varphi_{priv} \cap \varphi_{util} \,. \tag{7.10}$$

To apply the distributed synthesis algorithm we must transform our problem to match the conditions of Theorem 2.1. Specifically, we must eliminate the constraints on the plant and the feedback from the controller while maintaining the same set of solutions. To do this, we utilize two ideas from [38] for simplifying system architectures. First, due to the strict order of informed processes, feedback edges from less informed to more informed processes may be eliminated as they are redundant. Intuitively, as the non-environment processes are deterministic, such feedback outputs from them can simply be predicted. In particular as the environment (the plant in our case) is the most informed processes as the source of non-determinism in the system, feedback from our controller is redundant. Second, white-box processes may be eliminated by combination with more informed processes and encoding their implementation as part of the specification. Likewise, we can incorporate our plant dynamics in the specification. With these ideas, we transform the architecture and specification to the form used in Theorem 2.1 without altering the set of solution implementations. We describe this transformation in general now.

Let $A = (P, W, \mathbf{env}, E, O, H)$ be an architecture over V with a finite white-box implementation S_W, environment traces $M_{\text{env}} \subseteq \left(2^{V_{\text{env}}}\right)^\omega$ and an ω-regular specification $\varphi \subseteq \left(2^V\right)^\omega$. We create a new architecture by cutting any feedback to the environment process. Outputs communicated from a process only to the environment are replaced by hidden outputs in the process. For a given process $p \neq \mathbf{env}$, let $H'_p = H_p \cup \left(O_{(p,\text{env})} \setminus \bigcup_{p' \neq \text{env}} O_{(p,p')}\right)$, $O'_{(p,\text{env})} = \emptyset$, and $O'_{(p,p')} = O_{(p,p')}$ for $p' \neq \mathbf{env}$. Likewise, let $E' = \{(p, p') \in E \mid p' \neq \mathbf{env}\}$, $O' = \{O'_e \mid e \in E'\}$, and $H' = \{H'_p \mid p \in P\}$. The transformed architecture is denoted by $A' = (P, W, \mathbf{env}, E', O', H')$. Note that the output set of each process is unchanged, while only the inputs of the environment process were changed by removal. As such, the two architectures support the same implementations.

To capture the environment dynamics M_{env}, observe that the solution only needs to enforce the specification φ over traces agreeing with M_{env}. So we define a new specification

$$\varphi' = \varphi \cup \text{comp}\left(M_{\text{env}}|^V\right) \,. \tag{7.11}$$

Thus, this new specification is also ω-regular as it is constructed from the union, complement, and restriction of ω-regular languages. Then as desired, the environment of the transformed system is unconstrained, i.e., $M_{\text{env}}{}' = \left(2^{O_{\text{env}}}\right)^{\omega}$. We then have the following result.

Theorem 7.2. *Given a distributed synthesis problem over A, S_W, M_{env}, φ, consider the transformed problem $A', S_W, M_{env}{}', \varphi'$ be constructed as described above. Then the two problems have the same set of solution implementations.*

Proof. Consider an implementation S for both systems. Note that the trace sets for each strategy in the implementations are the same for both architectures. So from the definition of the trace set for distributed systems, we see

$$\text{Tr}(A, S, S_W, M_{\text{env}}) = \text{Tr}(A', S, S_W, M_{\text{env}}{}') \cap M_{\text{env}}|^V .$$

Hence we can compare satisfaction of the specifications:

$$\text{Tr}(A, S, S_W, M_{\text{env}}) \subseteq \varphi \quad \Leftrightarrow \quad \text{Tr}(A', S, S_W, M_{\text{env}}{}') \subseteq \varphi \cup \text{comp}\left(M_{\text{env}}|^V\right) = \varphi' .$$

So by the definition of Problem 2.1, S is a solution for the original problem if and only if it is a solution for the transformed one. □

As a consequence of this result, if the transformation results in a decidable problem, then we can apply distributed synthesis to find a solution implementation.

Theorem 7.3. *Problem 7.1 can be solved in 2-exponential time.*

Proof. After eliminating the feedback in the transformation of Architecture 1, the only potential fork is between the $\mathbf{Obf}$ and $\mathbf{Cont}$. However, as they observe the same inputs and hence possess the same information, this is not an information fork. Thus we may apply the results of Theorem 2.1 to the transformed architecture. Note the number of the information levels excluding the environment is 2, one for $\mathbf{Inf}$ and one for both $\mathbf{Obf}$ and $\mathbf{Cont}$ as they share the same information. Thus the synthesis algorithm runs in 2-exponential time in the size of the automata representing M_{Plant}, s_{Net}, φ_{priv}, and φ_{util}. If a solution implementation S is found, Theorem 7.2 states that it is also a solution for Problem 7.1. Likewise if no solution is found, Problem 7.1 has no solution. □

Thus distributed synthesis provides a sound and complete method for designing the obfuscator, controller, and actions for the recipient simultaneously to enforce privacy and utility. We demonstrate this procedure with the building system.

Example 7.5. *A solution to the distributed synthesis problem for the building system in Architecture 1 is depicted in Fig. 7.6. In particular, we see that after the corresponding keypad is pressed, the controller immediately opens door 1 while it delays opening door 2. This delay demonstrates coordination of the controller with the obfuscator, allowing the obfuscator to fabricate an extra press of the keypad providing deniability to which room the user is in. Indeed, distributed synthesis could be employed to verify that there exists no solution utilizing control or obfuscation alone. From the solution, we also see that the obfuscator always outputs an even number of presses to keypad 2, unless a fault has occurred, which is precisely what allows the intended recipient to eventually diagnose the fault. These behaviors are demonstrated by the selected traces of the system in Fig. 7.5.*

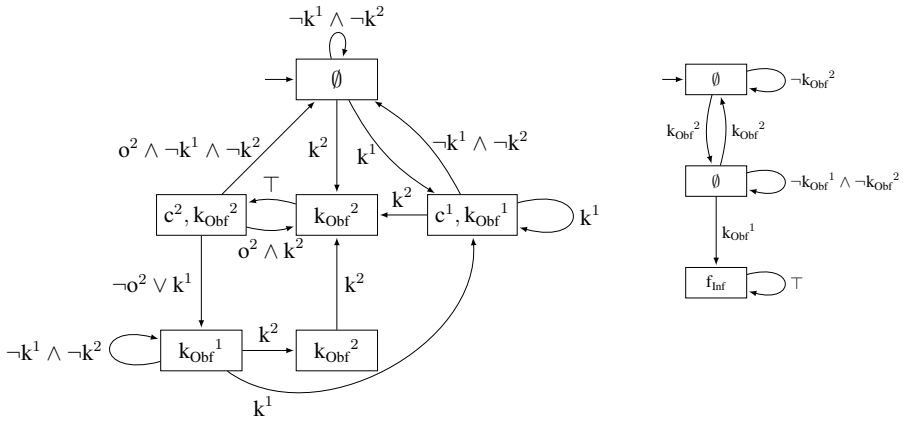

Figure 7.6: Automata implementing the combined obfuscator and controller (left) and inference function (right) in the solution to Example 7.5. For compactness, transitions are labeled by formulas over the plant variables rather than the corresponding sets which satisfy the formulas. The symbol $\top$ denotes *true*, the formula accepting all labels.

7.3 Integrating Obfuscation and Control Remotely

In this section, we discuss how obfuscation can be used when a controller is implemented at a remote site.

7.3.1 Remote Network Control and Obfuscation

In some cases, the infrastructure to implement a controller for the plant may only be present at a remote site. As the inputs to the controller and the feedback from the controller are transmitted over the network, they may potentially need to be obfuscated to preserve privacy. In Architecture 2 depicted in Fig. 7.7, the controller takes the place of the recipient from Architecture 1, implicitly inferring information from obfuscated plant outputs and selecting a control action which is then

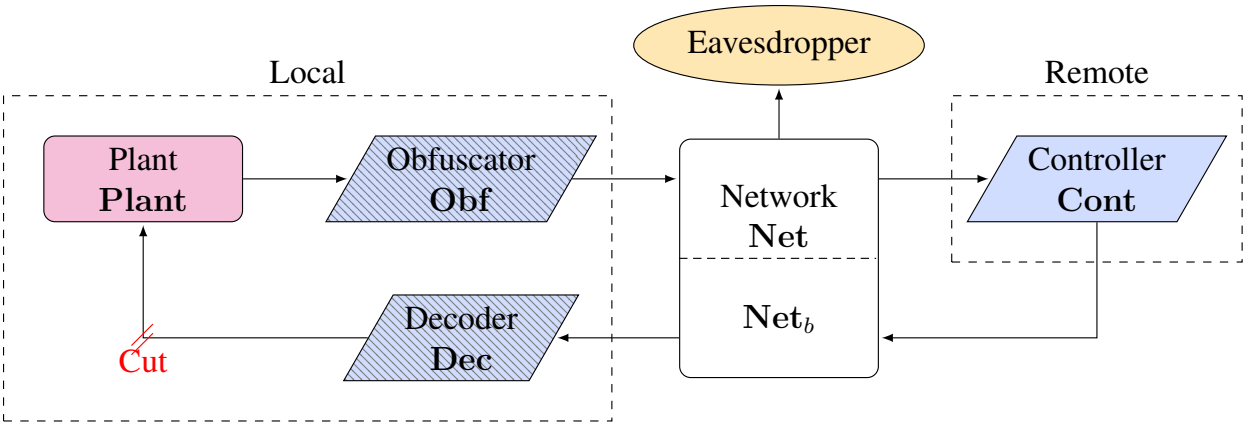

Figure 7.7: **Architecture 2** featuring a controller at the remote site which operates on obfuscated data produced by the obfuscator and consumed by the decoder at the local site. The processes are styled as in Fig. 7.3.

obfuscated. To allow the plant to interpret obfuscated outputs from the controller, an additional process called a *decoder* is introduced that processes outputs from the controller before passing them along to the plant.

System Model and Specifications

As with Architecture 1, we model the system with a distributed architecture $A = (P, W, \mathbf{env}, E, O, H)$ with processes $P = \{\mathbf{Plant}, \mathbf{Obf}, \mathbf{Net}, \mathbf{Net}_b, \mathbf{Cont}, \mathbf{Dec}\}$ representing the plant, obfuscator, both directions of the network, the controller, and the decoder. Later on, we will discuss why the two channels of the network are modeled with separate processes. The interconnections E are depicted in Fig. 7.7.

The sets of input and output variables of $\mathbf{Plant}$, $\mathbf{Obf}$, and $\mathbf{Net}$ are the same as in Architecture 1. Now instead of receiving inputs from the plant directly, the controller receives obfuscated inputs from the plant over the network, i.e., $I_{\text{Cont}} = O_{\text{Net}}$. Conversely, it produces outputs which are fed back through the network, i.e. $I_{\text{Net}_b} = O_{\text{Cont}}$. The eavesdropper may observe these control outputs on the network and use them to refine their beliefs about the plant behavior. As such, it may be advantageous for the controller to output obfuscated commands which are then interpreted at the local site by the decoder. So as the control outputs should mimic the plant inputs, we define $O_{\text{Cont}} = \{i_{\text{Cont}} \mid i \in I_{\text{Plant}}\}$. Likewise, the return network process reports the controller outputs to the decoder, so we define $O_{\text{Net}_b} = \{i_{\text{Net}_b} \mid i \in I_{\text{Plant}}\}$. Finally, the decoder de-obfuscates these outputs to provide to the plant with $O_{\text{Dec}} = I_{\text{Plant}}$. Again, the only white-box processes are from the network $W = \{\mathbf{Net}, \mathbf{Net}_b\}$ which each have fixed implementations.

We now discuss the specifics of formulating privacy and utility specifications in this architecture. In particular, we suppose the controller is designed such that the closed-loop behavior of the system

satisfies some utility requirement φ_{util} reflecting safety or liveness in the plant for example. As the controller is more apparent in this architecture, the eavesdropper may possess knowledge about both it and the plant. In particular, we assume the eavesdropper knows the utility requirement φ_{util}. We would like to construct the privacy specification φ_{priv} capturing privacy with respect to this new knowledge. Unaware of obfuscation, the eavesdropper derives their nominal model from the plant dynamics and Architecture 2 in Fig. 7.7 with the indicated unknown processes $P \setminus \hat{P} = \{\mathbf{Obf}, \mathbf{Dec}\}$. This defines the base nominal model $\hat{M}_0$ as in Definition 7.2. At this point, the eavesdropper expects behavior from $\hat{M}_0$ that not only satisfies φ_{util}, but belongs to an implementation solving the corresponding distributed synthesis problem for Architecture 2 over the nominal processes $\hat{P}$. The correct nominal model should then consist of the union of the trace sets of all such solutions.

In this case where the plant is in a simple feedback loop with the controller, we can use ideas from ω-supervisory control [88] to construct the nominal model $\hat{M}$ as an ω-regular language. If the plant is completely observed by the controller, the union of all deterministic solutions to the reactive synthesis problem form a maximally permissive supervisor. Specifically, we take $\hat{M}$ to be the supremal closed-loop behavior with respect to $\hat{M}_0$ describing the *plant* and the language φ_{util} describing the specification. More generally in the partial observation case, there is no unique maximal solution. However, for regular specifications over finite strings, the union of all correct supervisors can be constructed as a regular language as shown in [46] and Theorem 2 of [112]. Under mild conditions, these results may be extended to the ω-regular case. With this, we can now discuss the building example in detail.

Example 7.6. *We now consider that the building is instead controlled remotely as in Architecture 2 subject to the same specification φ_{Cont} from Example 7.3. Removing the requirement that the remote site infers the fault, the utility specification is simply $\varphi_{util} = \varphi_{Cont}$. We construct the privacy specification in a manner similar to before, identifying the fault as the secret behavior and assuming the processes related to obfuscation, i.e.,* $\mathbf{Obf}$ *and* $\mathbf{Dec}$*, are unknown to the eavesdropper. The key differences are that now the eavesdropper observes the control actions over the network and knows about the control specification φ_{util}. Thus the nominal model is constructed as all closed-loop behaviors of all implementations ensuring the base nominal model $\hat{M}_0$ satisfies the specification φ_{util} as discussed above. In this case, it turns out that this nominal model $\hat{M}$ and secret language L_S are the same as the one from Example 7.3 for Architecture 1 (up to the addition and renaming of variables). However, as the obfuscated controls ${c_{Obf}}^1$ and ${c_{Obf}}^2$ are now observed by the eavesdropper, the admissible observations are different, now given by*

$$(\hat{M} \setminus M_S)|_{V_{obs}} = \left(\left(L_1^+ L_2^+\right)^\omega \cup \left(L_2^+ L_1^+\right)^\omega\right) , \tag{7.12}$$

where $L_i = \left(\{\emptyset\}^* \{\{k_{Obf}{}^i\}\}^* \{\{c_{Obf}{}^i, k_{Obf}{}^i\}\}\right)^2$ *corresponds to observations of the user going through door* $i \in \{1, 2\}$ *twice. This defines the privacy specification* φ_{priv} *as in Equation* (7.2).

Synthesis

With the system model and specifications we can state the design problem for Architecture 2.

Problem 7.2 (Architecture 2). *Given an instance* A *of Architecture 2 and* ω*-regular privacy and utility specifications* φ_{priv} *and* φ_{util}*, find an implementation* S *for* $\mathbf{Obf}$*,* $\mathbf{Cont}$*, and* $\mathbf{Dec}$ *solving the distributed synthesis problem for* A *with specification* $\varphi = \varphi_{priv} \cap \varphi_{util}$*.*

As before, we can transform the system with Theorem 7.2 to match the conditions of Theorem 2.1.

Theorem 7.4. *Problem 7.2 can be solved in 3-exponential time.*

Proof. After eliminating the feedback from the decoder as depicted in Fig. 7.7, the resulting architecture is a pipeline, free of information forks. Thus we may apply the results of Theorem 2.1 to the transformed architecture. As there are 3 black-box processes in the pipeline, the synthesis algorithm runs in 3-exponential time in the size of the automata representing M_{Plant}, s_{Net}, s_{Net_b}, φ_{priv}, and φ_{util}. □

Here we see why the two channels of the network must be modeled as separate processes. If instead they are modeled with a single process, there is an information fork between the decoder and controller rooted at the hypothetical merged network process. This is because the network may transmit different information from the plant to the controller and decoder. Alternatively, we can also avoid information forks with a single network process by requiring it to transmit the same outputs to each receiving process.

Example 7.7. *A solution to the distributed synthesis problem for the building system with Architecture 2 is depicted in Fig. 7.8. From the solution, we see the obfuscator communicates to the controller the use of keypad 1 by the absence of output and of keypad 2 by two consecutive* $k_{Obf}{}^2$*. Similarly, the controller communicates the open command to the decoder for door 1 by the absence of output and for door 2 by two consecutive* $c_{Obf}{}^2$*. All the while these processes intersperse the outputs* $k_{Obf}{}^1$ *and* $c_{Obf}{}^1$ *to mimic the nominal system without obfuscation. In addition by removing fault diagnosis from the utility specification, we can observe that the remote site can no longer infer the occurrence of the fault.*

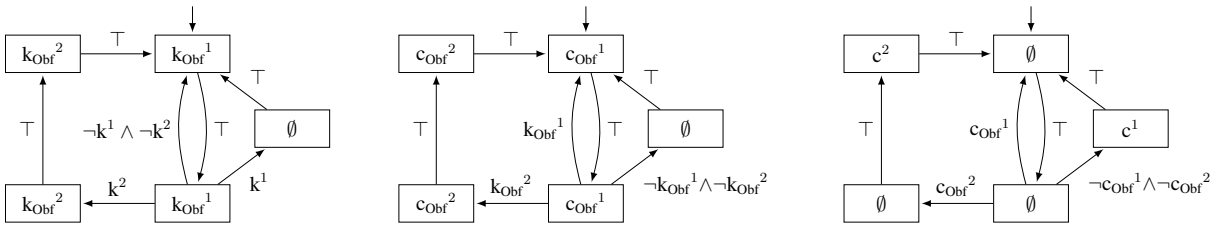

Figure 7.8: Automata implementing the obfuscator (left), controller (middle), and decoder (right) in the solution to Example 7.7.

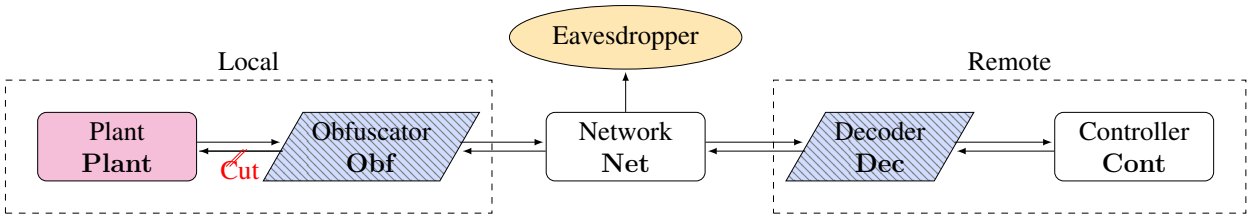

Figure 7.9: **Architecture 3** featuring a controller with a fixed implementation at the remote site which must be secured by combination obfuscator-decoders at both the local and remote site. The processes are styled as in Fig. 7.3.

7.3.2 Securing an Existing Remote Controller with Obfuscation

In this section, as in the previous one, we consider a plant that is controlled from the remote site. However, we assume that this controller has an existing implementation which cannot be altered. As this implementation may not have been designed with security in mind, it may leak sensitive information. We now consider the problem of securing such a controller by obfuscating its outputs in addition to those of the plant while maintaining the original closed-loop behavior. As in Architecture 2, the obfuscated controller outputs must be de-obfuscated before they can be input into the plant. Conversely, because the controller has a fixed implementation that was designed without obfuscation, its inputs must also be de-obfuscated. Unlike Architecture 2, for simplicity we model both obfuscation and decoding with a single process at each site. Similarly, we model both channels of the network with a single process as well. By merging these processes, we can avoid the information fork between the obfuscator and decoder present in Architecture 2. While both of these processes perform obfuscation and decoding, for consistency with Architecture 2 we will refer to the local process as the obfuscator $\mathbf{Obf}$ and the remote process as the decoder $\mathbf{Dec}$. We refer to this architecture as depicted in Fig. 7.9 as Architecture 3.

System Model and Specifications

We model the system with a distributed architecture $A = (P, W, \mathbf{env}, E, O, H)$ with processes $P = \{\mathbf{Plant}, \mathbf{Obf}, \mathbf{Net}, \mathbf{Dec}, \mathbf{Cont}\}$ representing the plant, obfuscator, network, obfuscator, and

controller. These obfuscator processes also take the role of the decoder from Architecture 2. The interconnection of these processes E is depicted in Fig. 7.9.

While the plant remains unchanged, the obfuscator, decoder, and network now receive inputs and produce outputs mirroring both the inputs and outputs of the plant. Formally, $O_{\text{Obf,Net}}$, $O_{\text{Net,Dec}}$, and $O_{\text{Dec,Cont}}$ are distinct copies of O_{Plant}. Likewise $O_{\text{Net,Obf}}$, $O_{\text{Dec,Net}}$, and $O_{\text{Cont,Dec}}$ are distinct copies of I_{Plant}.

Unlike in the previous architectures, the controller is added as a white-box process $W = \{\mathbf{Cont}, \mathbf{Net}\}$ with a fixed implementation $s_{\text{Cont}} : \left(2^{I_{\text{Cont}}}\right)^+ \to 2^{O_{\text{Cont}}}$. We assume that the closed-loop behavior for this controller must be maintained by the obfuscators. We can express this utility requirement with a specification φ_{util} ensuring the plant and controller inputs are exactly recovered by corresponding decoders after obfuscation. We can express this with an LTL formula

$$\varphi_{\text{util}} = \bigwedge_{i \in I_{\text{Obf}}} \underbrace{\mathrm{G}\left(i \leftrightarrow \mathrm{X}\, i_{\text{Cont}}\right)}_{\substack{\text{Control input is} \\ \text{delayed plant output}}} \wedge \bigwedge_{i \in I_{\text{Plant}}} \underbrace{\mathrm{G}\left(i \leftrightarrow i_{\text{Dec}}\right)}_{\substack{\text{Plant input is} \\ \text{control output}}} . \tag{7.13}$$

Example 7.8. *We again consider the building utilizing a controller at the remote site; however, we now assume that its implementation is fixed due to practical considerations. This implementation simply immediately sends the signal for the door to open once the corresponding keypad signal is received. Without obfuscation, this controller satisfies the utility specification from Example 7.6, requiring doors be signaled to open after the keypad is pressed. We will assume that this specification forms the eavesdropper's knowledge about the controller. Given the processes related to obfuscation, i.e., $P \setminus \hat{P} = \{\mathbf{Obf}, \mathbf{Dec}\}$, are unknown to the eavesdropper, this results in the same privacy specification φ_{priv} as in Example 7.6 (up to renaming variables). In order to maintain the existing closed-loop behavior, we must ensure the plant and controller recover their respective inputs exactly from obfuscation. As explained above, this is captured by the utility specification φ_{util} from Equation* (7.13).

Synthesis

With the system model and specifications we can state the design problem for Architecture 3.

Problem 7.3 (Architecture 3). *Given an instance A of Architecture 3 and ω-regular privacy and utility specifications φ_{priv} and φ_{util}, find an implementation S for* $\mathbf{Obf}$ *and* $\mathbf{Dec}$ *solving the distributed synthesis problem for A with specification $\varphi = \varphi_{priv} \cap \varphi_{util}$.*

As before, we can transform the system with Theorem 7.2 to match the conditions of Theorem 2.1.

Theorem 7.5. *Problem 7.3 can be solved in 2-exponential time.*

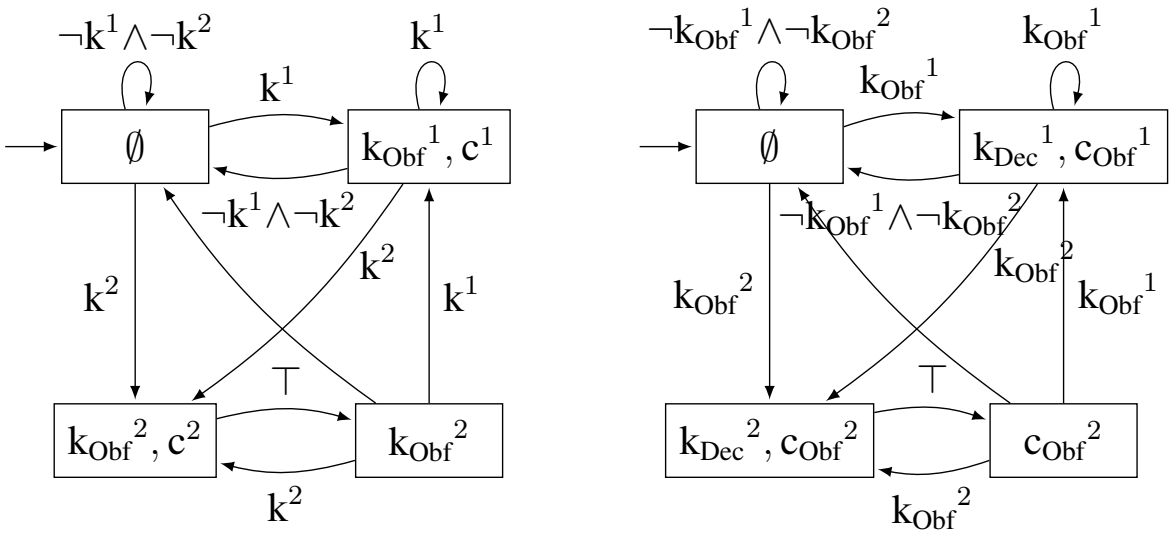

Figure 7.10: Automata encoding the implementation of the obfuscator **Obf** (left) and decoder **Dec** (right) for Example 7.9.

Proof. While there are many forks in this architecture, none of them constitute information forks. This is because information from the plant, i.e., the environment, propagates linearly to the other processes inducing a strict order on the processes levels of information. Indeed, after removing the feedback edges which are redundant and applying the transformation in Theorem 7.2, the resulting architecture is a pipeline. Thus we may apply the results of Theorem 2.1 to the transformed architecture. As there are 2 black-box processes in the pipeline, the synthesis algorithm runs in 2-exponential time in the size of the automata representing M_{Plant}, s_{Net}, s_{Cont}, φ_{priv}, and φ_{util}. □

Example 7.9. *Analysis of the building system with Architecture 3, shows that there are in fact no solutions to the distributed synthesis problem. We note while under non-faulty conditions, there are enough possible messages that the obfuscator can send to convey which key pads have been pressed: the obfuscator can transmit its inputs without modification. However, such a solution is no longer possible once we consider the occurrence of a fault. This possibility in the source of information, i.e., the plant, along with decrease in bandwidth from mimicking non-faulty behavior renders the problem unfeasible. Alternatively, if we assume that the fault has already occurred which reduces the amount of information that needs to be conveyed, a solution exists which is depicted in Fig. 7.10. In fact this solution guarantees privacy even when the eavesdropper knows the implementation of the controller.*

While we were able to ensure privacy against an eavesdropper with the exact model of plant and controller, such a requirement may be too restrictive. In this case, as the controller is deterministic, the eavesdropper can predict its output on any obfuscated input from the plant. As such, it is often

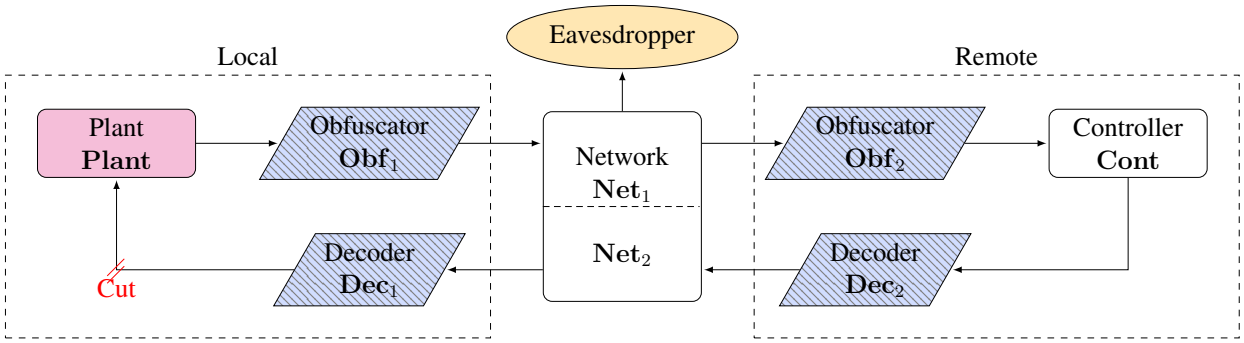

Figure 7.11: An alternative to Architecture 3 for securing an existing network controller.

necessary for the nominal model to contain uncertainty about the controller implementation in order to make the problem feasible.

Remark 7.3. *Instead of combining obfuscation and decoding into a single process in Architecture 3, we may instead maintain two separate processes like in Architecture 2. This alternative architecture is depicted in Fig. 7.11. While conceptually similar, there are a number of practical differences between these architectures. This alternative ensures, for example, that the local decoder only uses information transmitted from the remote obfuscator. In contrast, decoding at the local site in Architecture 3 may use all information available locally, including the direct outputs from the plant. In addition, the alternative architecture contains 4 black-box process arranged linearly after eliminating feedback to the environment. As a result, the synthesis algorithm for the alternative requires 4-exponential time, compared to the 2-exponential time required for Architecture 3.*

7.4 Results

In this section, we describe how distributed reactive synthesis problems can be solved in practice. In particular, we explain at a high level the approach used to design the solutions for the building access control examples presented in the previous sections. Additional details about the implementation can be found in the appendix.

We observed that the original algorithm for distributed synthesis proposed in [38, 76] does not scale to the example problems considered here. This is due in part to the algorithm's explicit construction of automata of n-exponential size. Instead, we employed a similar approach to [96] used for the synthesis of obfuscators and inference functions. This approach is based upon the reduction from distributed synthesis to synthesis for hyperproperties described in [34]. *Hypeproperties* generalize the concept of specifications for individual traces such as LTL properties, to relations of multiple traces. In short, the reduction constructs a hyperproperty encoding the information flow

Arch.	States	Hyper States	Synth. Time (s)
1	59	2	149
2	81	3	72
3	31	2	548

Table 7.1: Information for the synthesis of the reduced examples for each architecture, including the number of states in the automata describing the property and hyperproperty specifications.

induced by the distributed architecture as a relation of the input and output variables of different traces.

We then solved the reduced synthesis problem for hyperproperties using the tool BoSyHyper [35]. This tool achieves improved performance by taking advantage of the existence of small solutions with bounded synthesis as well as advanced heuristics employed within modern constraint solvers. Unfortunately, even with these optimizations, the tool was unable to synthesize solutions for the building access control problems discussed in Examples 7.5, 7.7, and 7.9. In order to obtain solutions for these problems, we manually abstracted the plant and specifications, simplifying the problems and reducing their complexity for the synthesis tool. The tool was then able to synthesize solutions for the reduced problem which were then manually lifted to the original problems. These solutions, depicted in Fig. 7.6, 7.8, and 7.10, were then formally verified for correctness in enforcing privacy and utility. Information about the synthesis for the reduced problems is provided in Table 7.1. Counterintuitively, the smaller sized examples result in longer synthesis times. This demonstrates the observed fact that smaller problem sizes do not necessarily correspond to easier problems for the constraint solvers.

In addition to improved performance, the hyperproperty approach is also more extensible. While the original explicit algorithm cannot be applied to architectures with an information fork, the information flow of arbitrary distributed architectures can be expressed with hyperproperties. This problem is undecidable in general; however, synthesis algorithms for hyperproperties provide a sound but incomplete method for more general architectures. In our framework this would enable synthesis for problems with non-deterministic network delays as well as plants distributed across multiple sites.

7.5 Conclusion

In this chapter, we have addressed the problem of enforcing privacy and utility over networked systems with obfuscation and control. By modeling the system with distributed reactive processes, we were able to leverage tools from distributed reactive synthesis to automatically design implementations. We demonstrated this approach on three problems with distinct architectures.

CHAPTER 8

Conclusion

8.1 Summary of Contributions

In this dissertation, we have presented several methods for the verification and enforcement of both privacy and utility over cyber-physical systems. In particular, we addressed the problems of computational efficiency and support for complex specifications. We modeled privacy with the formal information flow property of opacity which requires that an observer cannot deduce sensitive information about a system's behavior. Informally, verification asks "Can we prove a given system is opaque?"; whereas enforcement asks "Can we implement a mechanism ensuring a given system is opaque". We focused on the use of obfuscation as an enforcement mechanism which was later integrated with control.

Chapter 3 proposes a general framework for both specifying and verifying opacity over discrete event systems modeled with automata. We considered the possibility that an observer's model of the system's behavior and observable outputs may be uncertain or even incorrect in comparison with the actual model of the system. This possibility was reflected in our definition of opacity with separate models for the actual system and the observer's nominal model of the system along with a specification of nonsecret behavior. The use of a nominal model is critical in expressing opacity over systems with enforcement mechanisms which may or may not be known by the observer, as was considered in subsequent chapters. We showed that using this definition, opacity is equivalent to a certain regular language inclusion problem. We presented three approaches to checking this inclusion based on elementary automata constructions. To demonstrate these approaches, we expressed the complex notion of K-step opacity within the framework and evaluated the theoretical and empirical performance of the proposed verification algorithms. In order to improve the computational efficiency further, in Chapter 4 we developed a relaxation of opacity. We proved the verification problem for this relaxation is co-NP-complete, reduced from the complexity of PSPACE-completeness for the general notion of opacity. An encoding to SATwas used to provide an efficient and extensible verification method for the relaxation which was demonstrated to scale better than methods verifying the general notion.

Next, we discussed how the problem of enforcement, specifically with obfuscation, can be formulated in the framework of Chapter 3. We considered obfuscation with edit functions that dynamically delete and insert fictitious events to a system's output to confuse and mislead observers. Critically, these outputs are designed to mimic the original system. In this way, knowledge of the privacy enforcement mechanism itself can be made private. In Chapter 5, we adapted existing techniques for the synthesis of edit functions to our framework in order to enforce more complex notions of opacity, such as K-step opacity. We then proposed a stronger notion of utility for obfuscated systems in Chapter 6 by considering that some intended recipients of the system's information need more access to sensitive information than unintended ones. We employed distributed reactive synthesis in order to find privacy enforcing edit functions alongside strategies for an intended recipient to interpret the obfuscated information. Finally, we extended this framework further in Chapter 7 to more complex network architectures combining obfuscation and control. This extension was demonstrated on three representative problems over a building access-control system.

8.2 Future Work

While this dissertation presented techniques addressing the computational efficiency and expressivity of specifications for verification and enforcement problems, there are still a number of barriers limiting application of these approaches in practice. Here we discuss directions for future work to remove these barriers. In particular, we focus on the problem of privacy enforcement with obfuscation.

The main limitation of the proposed methods remains scalability to large system models. As with many applications of formal methods, the scalability of our methods can be greatly improved by first abstracting such large models into smaller ones, providing a high-level view of the system's dynamics relevant to privacy. For example, the automata models described in this dissertation can be minimized using the concept of opacity-preserving bisimulation [71]. Additionally, rather than enforcing the general notion of opacity with obfuscation, which results in exponential complexity, we may instead enforce a relaxed notion of opacity like bounded-memory opacity as defined in Chapter 4. Indeed, the SAT constraints encoding bounded memory opacity can be used with a QBF solver in order to solve the enforcement problem as in [58]. This would result in a reduced PSPACE complexity for enforcement. Alternatively, we may investigate more scalable approaches to the distributed reactive synthesis problem used to design obfuscators. Recent work on contract-based and modular synthesis may offer greatly reduced complexity in practice [37]. Yet another approach would be the synthesis of programs with syntactical structure rather than the structureless automata and transducers considered in this dissertation [63].

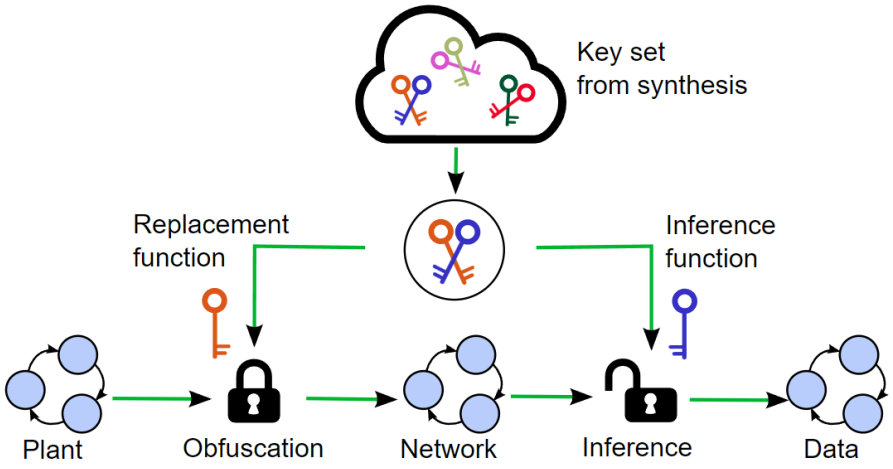

Figure 8.1: The proposed obfuscation scheme in the presence of obfuscation-aware eavesdroppers. An obfuscation implementation acting as a kind of secret key is selected from a pool at runtime to be implemented on the system. While an eavesdropper may know the members of this pool, they are unaware of the specific implementation chosen. Similar to encryption, we can design this pool so each obfuscator provides deniability for the others' observations.

While we describe at a high-level how the framework used in this dissertation can describe privacy specifications in general, it remains to investigate which notions of privacy are most applicable and practical for cyber-physical systems. For example, the idea of quantifying measures of opacity has gained interest recently. We recall that our obfuscation framework based on distributed reactive synthesis is implemented as a hyperproperty realization problem. This approach could be adapted to enforce quantitative hyperproperties [36]; instead of deniability through a single alternative explanation, deniability through many alternatives may be desired in practice. Additionally, just as we specify the secrets for opacity with formal logic, there is a need to specify the beliefs of the observer as well. This may be accomplished using *strategy logic* to describe the obfuscators an observer believes are possible as strategies in a game [18]. Likewise, it may be useful to formalize the overall privacy guarantees in a formal language more familiar to the information security community, for example using Dolev-Yao theory as in [64].

Along these lines, we may investigate how obfuscation can achieve the stronger notions of privacy guaranteed by cryptography for example. In particular, we ask how can obfuscation guarantee privacy even when observers are aware of its existence? One solution could be the use of obfuscation on top of encryption. A more integrated solution inspired by encryption is depicted in Fig. 8.1 which is partially implemented in [44]. There, a pool of obfuscators providing mutual deniability is synthesized as a solution to a different hyperproperty realization problem.

Finally, once the aforementioned barriers have been addressed, we must identify real systems that would benefit from the privacy guarantees provided by obfuscation. This dissertation has presented a

number of potential use cases, including location-based services, contact-tracing apps, and building access-control. These applications are likely to present unique challenges for implementation, such as vulnerability to timing attacks [40] which must be addressed carefully. Alternatively, rather than using obfuscation to enforce information security, we may shift our perspective to an attacker. Obfuscation is a natural model for attackers on a compromised system to exfiltrate sensitive data while remaining undetected. In this way, tools for the design of obfuscators may be used to analyze the resilience of systems to this kind of attack.

APPENDIX A

Proofs of Transformation for Insertion Function Synthesis

Proof of Theorem 6.3 (Unfolding). For convenience, we make the following definition. Given a folded trace $f = f_0 f_1 \cdots \in F^\omega$ and its unfolding $t = t_0 t_1 \cdots = \mathbf{unfold}(f)$, we say step n in f corresponds to step j in t if $|\mathbf{unfold}(f_0 \cdots f_{n-1})| < j \leq |\mathbf{unfold}(f_0 \cdots f_n)|$.

Consider an obfuscation policy $\mathbf{Obf} : \left(2^{I_{\text{Obf}}}\right)^+ \to \left(2^{O_{\text{Obf}}}\right)^+$ and inference policy $\mathbf{Inf} : \left(2^{I_{\text{Inf}}}\right)^+ \to 2^{O_{\text{Inf}}}$. Given two folded traces $f, f' \in \mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})$ let $t = \mathbf{unfold}(f)$ and $t' = \mathbf{unfold}(f')$. Suppose that at some point, the outputs in O_0 differ between t and t' so $j = \min\{k \in \mathbb{N} \mid t_k|_{O_0} \neq t'_k|_{O_0}\}$ exists. Let n denote the corresponding step in the f and f'. If yield differs in t_j and t'_j, then the length of the outputs in f_n and f'_n differ. Otherwise if the outputs in O_{Obf} differ in t_j and t'_j, then the outputs themselves differ in f_n and f'_n. Hence the inputs of I_{Obf} must differ in f and f' at some step $n' \leq n$ as they result from the deterministic obfuscation policy $\mathbf{Obf}$. Let $j' = |\mathbf{unfold}(f_0 \cdots f_{n'-1})| + 1$ be the first step in t corresponding to n' in f. By definition of $\mathbf{unfold}$, the inputs of I_0 in $t_{j'}$ and $t'_{j'}$ are given by the inputs in $f_{n'}$ and $f'_{n'}$. So $t_{j'}|_{I_0} \neq t'_{j'}|_{I_0}$. By contrapositive, we have the following result

$$\forall p, p' \in \overline{\mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf}))} : \; t|_{I_0} = t'|_{I_0} \;\Rightarrow\; p|_{O_0} = p'|_{O_0} \,. \tag{A.1}$$

So given $i \in \left(2^{I_0}\right)^+$, if

$$\exists p = p_0 \cdots p_n \in \overline{\mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf}))} : \; p|_{I_0} = i \,, \tag{A.2}$$

we can uniquely define $s_0(i) = p_n|_{O_0}$, and otherwise we define $s_0(i) = \{\text{yield}\}$. Because $\mathbf{Inf}$ is already synchronous, we simply define $s_1 = \mathbf{Inf}$. Then by construction we see that

$$\mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})) \subseteq P_U \cap \mathrm{Tr}(s_0, s_1) \,.$$

Conversely, consider a trace $t \in P_U$ but $t \notin \mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf}))$. Then as $\mathbf{unfold}$ is bijective onto P_U, this means that $t = \mathbf{unfold}(f)$ for some $f \notin \mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})$. As $\mathbf{Obf}$ is defined for all inputs, there exists a trace $f' \in \mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})$ with the same inputs in I_{Obf} as f. Let

$t' = \mathbf{unfold}(f')$ and define $j = \min\{k \mid t_k \neq t'_k\}$ which must exist as $t \neq t'$. Let n denote the corresponding step in the folded system. If yield $\notin t_{j-1} = t'_{j-1}$ then as $t, t' \in P_U$ it must hold that $t_j|_{I_0} = t'_j|_{I_0} = \emptyset$ by the definition of $\mathbf{unfold}$. Otherwise if yield $\in t_{j-1} = t'_{j-1}$ then $t_j|_{I_0}$ and $t'_j|_{I_0}$ must be given by the corresponding inputs in f_n and f'_n, respectively. But by assumption, these inputs are equal so $t_j|_{I_0} = t'_j|_{I_0}$. In either case the inputs in I_0 of t and t' are equal up to step j, but the outputs in $O_0 \cup O_1$ are unequal. As $t' \in \mathrm{Tr}(s_0, s_1)$, this implies $t \notin \mathrm{Tr}(s_0, s_1)$. This implies that $\mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})) \supseteq U^\omega \cap \mathrm{Tr}(s_0, s_1)$.

Finally, by definition, traces in P_U always eventually yield and traces outside of P_U must not satisfy the condition of (A.2) at some point, after which they must always yield by construction. In either case s_0 always eventually yields. □

Proof of Theorem 6.3 (Folding). Let $s_0 : \left(2^{I_0}\right)^+ \to 2^{O_0}$ and $s_1 : \left(2^{I_1}\right)^+ \to 2^{O_1}$ be strategies such that s_0 always eventually yields in the sense of (6.14). As s_0 always eventually yields and is defined for all inputs, for every $e = e_0 e_1 \cdots \in \left(2^{O_{\text{env}}}\right)^\omega$, we can inductively construct a trace $t \in \mathrm{Tr}(s_0, s_1)$ so that

$$t|_{O_{\text{env}}} = e_0 \emptyset^{k_0 - 1} e_1 \emptyset^{k_1 - 1} \cdots,$$

for some $k_j > 0$ so that the partial sums $\sum_{j=0}^{n} k_j$ is the index of the n^{th} occurrence of yield in t. For example $k_0 = \min\{k \mid \text{yield} \in s_0(e_0 \emptyset^{k-1})\}$. This trace t is unique as it results from the deterministic strategies s_0 and s_1. Note that t follows the yield behavior, i.e., $t \in P_U$, so there exists $f = f_0 f_1 \cdots \in F^\omega$ such that $t = \mathbf{unfold}(f)$. Additionally, by the construction of t, we see that the plant outputs in O_{env} of f are exactly e. Furthermore, similar to the proof of the unfolding case, if f' is constructed for plant outputs e' as detailed above, where e' has a common prefix with e, then f' has a corresponding common prefix with f. Hence, we can define the obfuscation policy $\mathbf{Obf}(e_0 \cdots e_n) = f_n|_{O_{\text{Obf}}}$ and inference policy $\mathbf{Inf} = s_1$ so that $\mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})) \subseteq \mathrm{Tr}(s_0, s_1) \cap P_U$.

Also similar to the proof of the unfolding case, given a trace $t \in \mathrm{Tr}(s_0, s_1$ that is not constructed as above, at some point t must violate the yield behavior so that $t \notin P_U$. Hence we also have $\mathbf{unfold}(\mathrm{Tr}(\mathbf{Obf}, \mathbf{Inf})) \supseteq \mathrm{Tr}(s_0, s_1) \cap P_U$. Thus s_0 and s_1 are an unfolding of $\mathbf{Obf}$ and $\mathbf{Inf}$. □

APPENDIX B

Implementation Details for Obfuscation and Control

This appendix details the synthesis of solutions to the building access control problems presented in Examples 7.5, 7.7, and 7.9.

B.0.1 Specifications for Examples

We now discuss the trace specifications for the example problems in detail. In particular, we represent the specifications with Büchi automata as needed by the synthesis tool. The desired specification used for each of the three examples is of the form $\varphi = \varphi_{\text{util}} \cap \varphi_{\text{priv}}$ as in Equation (7.10). Here φ_{util} describes utility requirements for each problem while privacy is described by $\varphi_{\text{priv}} = \left((\hat{M} \setminus M_{\text{S}})|_{V_{\text{obs}}}\right)|^{V}$ as in Equation (7.2) depending on the nominal model $\hat{M}$. In light of Theorem 7.2, the specification input to the synthesis tool must incorporate the plant dynamics M_{Plant} constructed as in Equation (7.11) as $\varphi' = \varphi \cup \text{comp}\left(M_{\text{env}}|^{V}\right)$.

Each example utilizes the same plant dynamics M_{Plant} encoded by the automaton depicted in Fig. 7.4 which represents a single user's movement throughout the building. The liveness condition that all persistently accessible keypads are eventually used can be expressed as a Streett acceptance condition over the states of this automaton. Formally, for $j \in \{1, 2\}$ we define B_j as the set of states where keypad j can be pressed, i.e., states with an outgoing transition labeled by k^j. Specifically, $B_1 = \{0, 1, 2, 3, 0f, 1f, 2f, 3f\}$ and $B_2 = \{0, 4, 5, 6, 0f, 4f, 5f, 6f\}$. Likewise, we define G_j as the set of states where keypad j is pressed, i.e., the destination of these transitions labeled by k^j. Specifically, $G_1 = \{1, 3, 0f, 3f\}$ and $G_2 = \{4, 6, 4f, 6f\}$. The acceptance condition for liveness requires for each j, that if the states of B_j are visited infinitely often then so are the states of G_j. In words, if keypad j can always eventually be pressed, then it is always eventually pressed. It is well-known that such Streett conditions can be transformed into a Büchi conditions [42] which are used by the synthesis tool.

Next, we discuss the construction of the privacy specification. In all three examples, we assume the eavesdropper's model of the system architecture is the plant with dynamics M_{Plant} in direct feedback with the controller with a unit delay. As such, the base nominal models $\hat{M}_0$ in each problem are the same up to the renaming of variables. Furthermore, we assume that the eavesdropper knows

that the closed-loop system satisfies the control specification φ_{Cont} defined in Equation (7.5) which requires door j to eventually be signaled to open with $\mathbf{Cont}^j$ after the corresponding keypad has been pressed with output k^j. In this case, we know that feasible traces must belong to the intersection $\hat{M} = \hat{M}_0 \cap \varphi_{\text{Cont}}$ which is used by the eavesdropper as their nominal model in the example for Architecture 1.

However, the question remains whether all of such traces of the plant satisfying the control specification are realized by a controller, as pondered by the more astute eavesdroppers considered in Architecture 2 and Architecture 3. In this case, the question can be answered in the affirmative by inspection, any trace in $\hat{M}_0 \cap \varphi_{\text{Cont}}$ can be achieved by a controller by appropriately inserting delays in opening doors as necessary. We see that the eavesdroppers in all three examples utilize the same nominal model which can be expressed by the plant automaton with an additional Streett acceptance condition modeling φ_{Cont}. In particular given the secret language M_{S} defined by the occurrence of the fault, the language $\hat{M} \setminus M_{\text{S}}$ can be expressed by the non-faulty (left) half of the plant automaton where each room must be visited infinitely often. This is described by the Streett condition with B_i is given by all states and G_i is given by the states of room $i \in \{0, 1, 2\}$. Specifically, $G_0 = \{0, 1, 4\}$, $G_2 = \{2, 3\}$, and $G_2 = \{5, 6\}$. Again, this Streett automaton is then converted to a Büchi automaton. Finally, a single Büchi automaton accepting φ' may be constructed using the standard constructions for the complement and intersection.

B.0.2 Synthesis of Examples

Next, we present the details of how solutions for the example problems were designed. While the classical tree-automaton based algorithm [38, 76] is useful for theoretic analysis of distributed reactive synthesis, its explicit construction of large automata results limits applicability. Unfortunately, there are not many tools available for the purpose of distributed synthesis. Alternatively, solving the reduction to the more general problem of synthesis for hyperproperties has been observed to improve performance. For example, this is the approach taken in [96] to synthesize obfuscation and inference functions for privacy and utility enforcement. Hyperproperties generalize the concept of trace specifications such as LTL or ω-regular properties. Whereas properties describe individual traces, hyperproperties describe relations of multiple traces. For example, HyperLTL is a formal logic for expressing hyperproperties which extends LTL with explicit trace quantifiers [23]. It is capable of expressing classical information flow properties such as non-interference as well as information flow within a distributed architecture [35].

Using this fact, we can reduce the distributed synthesis problem to a HyperLTL synthesis problem by introducing a HyperLTL formula capturing the architecture. This formula is built using the following HyperLTL formula which expresses the causal dependence of the output variables $O \subseteq V$

on the input variables $I \subseteq V$ without delay

$$\mathrm{D}_{I \to O} = \forall \pi.\, \forall \pi'. \big(\bigvee_{o \in O} O[\pi] \leftrightarrow O[\pi'] \big) \mathcal{W} \big(\bigvee_{i \in I} I[\pi] \leftrightarrow I[\pi'] \big) \,,$$

where $\mathcal{W}$ denotes the weak until operator. In words, this formula requires the outputs of any two traces to be the same until their inputs differ. A similar definition for $\mathrm{D}_{I \to O}$ is made in [35] for processes with delay. Given a distributed architecture $A = (P, W, \mathbf{env}, E, O, H)$, we can construct a HyperLTL formula encoding its information flow as

$$\Phi_A = \bigwedge_{\substack{p \in P \\ p \neq \mathrm{env}}} \mathrm{D}_{I_p \to O_p} \,. \tag{B.1}$$

We then aim to solve the synthesis problem for the trace property φ' and the hyperproperty Φ_A.

We solve this problem using the tool BoSyHyper [34], an extension of the well-known LTL bounded synthesis tool BoSy to HyperLTL [33]. In particular BoSyHyper supports synthesis for HyperLTL formulas with only universal quantifiers, which includes the formulas needed for distributed synthesis. The tool was modified to accept as input the trace specification φ' represented by an explicit Büchi automaton and the HyperLTL formula Φ_A. If found by the tool, solutions are given as a monolithic model of the realized system in the Aiger format [33]. Such solutions can be converted into automata models implementing each process of the system. The construction and manipulation of automata input and output by BoSyHyper was performed with the automata library MDESops [68].

BIBLIOGRAPHY

[1] Rajeev Alur. *Principles of Cyber-Physical Systems*. The MIT Press, Cambridge, Massachusetts, 2015.

[2] Ikhlass Ammar, Yamen El Touati, Moez Yeddes, and John Mullins. Bounded opacity for timed systems. *Journal of Information Security and Applications*, 61:102926, September 2021.

[3] Eric Badouel, Marek Bednarczyk, Andrzej Borzyszkowski, Benoit Caillaud, and Phillipe Darondeau. Concurrent Secrets. *Discrete Event Dynamic Systems*, 17(4):425–446, December 2007.

[4] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, Mass, 2008.

[5] D.E. Bakken, R. Rarameswaran, D.M. Blough, A.A. Franz, and T.J. Palmer. Data obfuscation: Anonymity and desensitization of usable data sets. *IEEE Security & Privacy*, 2(6):34–41, November 2004.

[6] Jiří Balun and Tomáš Masopust. Comparing the notions of opacity for discrete-event systems. *Discrete Event Dynamic Systems*, 31(4):553–582, December 2021.

[7] Jiří Balun and Tomáš Masopust. Verifying weak and strong k-step opacity in discrete-event systems. *Automatica*, 155:111153, September 2023.

[8] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-Preserving Encryption. In Michael J. Jacobson, Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 295–312, Berlin, Heidelberg, 2009. Springer.

[9] Béatrice Bérard, Krishnendu Chatterjee, and Nathalie Sznajder. Probabilistic opacity for Markov decision processes. *Information Processing Letters*, 115(1):52–59, January 2015.

[10] Finn Brunton and Helen Nissenbaum. *Obfuscation*. The MIT Press. MIT Press, London, England, September 2016.

[11] Jeremy Bryans, Maciej Koutny, and Peter Ryan. Modelling Opacity Using Petri Nets. *Electr. Notes Theor. Comput. Sci.*, 121:101–115, February 2005.

[12] Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, November 2008.

[13] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer International Publishing, Cham, 3 ed. edition, 2021.

[14] Franck Cassez. The Dark Side of Timed Opacity. In Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-hoon Kim, and Sang-Soo Yeo, editors, *Advances in Information Security and Assurance*, Lecture Notes in Computer Science, pages 21–30. Springer Berlin Heidelberg, 2009.

[15] Franck Cassez, Jérémy Dubreil, and Hervé Marchand. Dynamic Observers for the Synthesis of Opaque Systems. In Zhiming Liu and Anders P. Ravn, editors, *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, pages 352–367, Berlin, Heidelberg, 2009. Springer.

[16] Franck Cassez, Jérémy Dubreil, and Hervé Marchand. Synthesis of opaque systems with static and dynamic masks. *Formal Methods in System Design*, 40(1):88–115, February 2012.

[17] Justin Chan, Landon Cox, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. PACT: Privacy-sensitive protocols and mechanisms for mobile contact tracing. *IEEE Data Engineering Bulletin*, 43(2):15–35, July 2020.

[18] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, June 2010.

[19] Bo Chen, Kevin Leahy, Austin Jones, and Matthew Hale. Differential privacy for symbolic systems with application to Markov Chains. *Automatica*, 152:110908, June 2023.

[20] Sang Cho and Dung T. Huynh. The parallel complexity of finite-state automata problems. *Information and Computation*, 97(1):1–22, March 1992.

[21] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, July 2001.

[22] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[23] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal Logics for Hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust*, Lecture Notes in Computer Science, pages 265–284, Berlin, Heidelberg, 2014. Springer.

[24] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, June 2008.

[25] Christian Collberg, C. Thomborson, and Douglas Low. A Taxonomy of Obfuscating Transformations. Technical Report, Department of Computer Science, The University of Auckland, New Zealand, July 1997.

[26] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3):188–217, 2007.

[27] Laurent Doyen and Jean-Francois Raskin. Antichains for the Automata-Based Approach to Model-Checking. *Logical Methods in Computer Science*, 5(1):5, March 2009.

[28] Jeremy Dubreil, Philippe Darondeau, and Herve Marchand. Supervisory Control for Opacity. *IEEE Transactions on Automatic Control*, 55(5):1089–1100, May 2010.

[29] Cynthia Dwork. Differential Privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 1–12, Berlin, Heidelberg, 2006. Springer.

[30] Rüdiger Ehlers, Stéphane Lafortune, Stavros Tripakis, and Moshe Y. Vardi. Supervisory control and reactive synthesis: A comparative introduction. *Discrete Event Dynamic Systems*, 27(2):209–260, June 2017.

[31] Yliès Falcone and Hervé Marchand. Runtime Enforcement of K-step Opacity. In *Proceedings of the IEEE Conference on Decision and Control*, pages 7271–7278, December 2013.

[32] Yliès Falcone and Hervé Marchand. Enforcement and validation (at runtime) of various notions of opacity. *Discrete Event Dynamic Systems*, 25(4):531–570, December 2015.

[33] Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. BoSy: An Experimentation Framework for Bounded Synthesis. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, volume 10427, pages 325–332, Cham, 2017. Springer International Publishing.

[34] Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing omega-regular Hyperproperties. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 40–63, Cham, 2020. Springer International Publishing.

[35] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesizing Reactive Systems from Hyperproperties. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, volume 10981, pages 289–306. Springer International Publishing, Cham, 2018.

[36] Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. Model Checking Quantitative Hyperproperties. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 144–163, Cham, 2018. Springer International Publishing.

[37] Bernd Finkbeiner and Noemi Passing. Compositional synthesis of modular systems. *Innovations in Systems and Software Engineering*, 18(3):455–469, September 2022.

[38] Bernd Finkbeiner and Sven Schewe. Uniform Distributed Synthesis. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 321–330, Chicago, IL, USA, 2005. IEEE.

[39] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

[40] J. Giles and B. Hajek. An information-theoretic and game-theoretic study of timing channels. *IEEE Transactions on Information Theory*, 48(9):2455–2477, September 2002.

[41] J. A. Goguen and J. Meseguer. Unwinding and Inference Control. In *1984 IEEE Symposium on Security and Privacy*, pages 75–75, April 1984.

[42] Erich Grädel, Wolfgang Thomas, Thomas Wilke, G. Goos, J. Hartmanis, and J. Van Leeuwen, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[43] Christoforos N. Hadjicostis. Introduction to Estimation and Inference in Discrete Event Systems. In Christoforos N. Hadjicostis, editor, *Estimation and Inference in Discrete Event Systems: A Model-Based Approach with Finite Automata*, Communications and Control Engineering, pages 1–14. Springer International Publishing, Cham, 2020.

[44] Tzu-han Hsu, Ana Oliveira da Costa, Andrew Wintenberg, Borzoo Bonakdarpour, and Ezio Bartocci. Gray-box Runtime Enforcement of Hyperproperties. In *Computer-Aided Verification (under Review)*, 2024.

[45] Dominic Hughes and Vitaly Shmatikov. Information hiding, anonymity and privacy: A modular approach. *Journal of Computer Security*, 12(1):3–36, January 2004.

[46] Kemal Inan. Nondeterministic supervision under partial observations. In Guy Cohen and Jean-Pierre Quadrat, editors, *11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*, Lecture Notes in Control and Information Sciences, pages 39–48, Berlin, Heidelberg, 1994. Springer.

[47] Romain Jacob, Jean-Jacques Lesage, and Jean-Marc Faure. Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 41:135–146, January 2016.

[48] Yiding Ji, Yi-Chin Wu, and Stéphane Lafortune. Enforcement of opacity by public and private insertion functions. *Automatica*, 93:369–378, July 2018.

[49] Yiding Ji, Xiang Yin, and Stéphane Lafortune. Opacity Enforcement Using Nondeterministic Publicly Known Edit Functions. *IEEE Transactions on Automatic Control*, 64(10):4369–4376, October 2019.

[50] Vamsi Kambhampati, Christos Papadopolous, and Dan Massey. Epiphany: A location hiding architecture for protecting critical services from DDoS attacks. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12, June 2012.

[51] Gabriel Kaptchuk, Daniel G. Goldstein, Eszter Hargittai, Jake Hofman, and Elissa M. Redmiles. How good is good enough for COVID19 apps? The influence of benefits, accuracy, and privacy on willingness to adopt. *arXiv:2005.04343 [cs]*, May 2020.

[52] Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. SOS: Secure overlay services. *ACM SIGCOMM Computer Communication Review*, 32(4):61–72, August 2002.

[53] Hao Lan, Yin Tong, Jin Guo, and Alessandro Giua. Comments on "A new approach for the verification of infinite-step and K-step opacity using two-way observers" [Automatica 80 (2017) 162–171]. *Automatica*, 122:109290, December 2020.

[54] Bengt Lennartson, Mona Noori-Hosseini, and Christoforos N. Hadjicostis. State-Labeled Safety Analysis of Modular Observers for Opacity Verification. *IEEE Control Systems Letters*, 6:2936–2941, 2022.

[55] S. Leung-Yan-Cheong and M. Hellman. The Gaussian wire-tap channel. *IEEE Transactions on Information Theory*, 24(4):451–456, July 1978.

[56] Públio M. Lima, Lilian K. Carvalho, and Marcos V. Moreira. CONFIDENTIALITY OF CYBER-PHYSICAL SYSTEMS USING EVENT-BASED CRYPTOGRAPHY. *IFAC-PapersOnLine*, 53(2):1735–1740, January 2020.

[57] Feng Lin. Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, March 2011.

[58] Liyong Lin and Rong Su. Bounded synthesis of resilient supervisors, 2021.

[59] Liyong Lin, Yuting Zhu, and Rong Su. Towards Bounded Synthesis of Resilient Supervisors. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 7659–7664, December 2019.

[60] Rongjian Liu and Jianquan Lu. Enforcement for infinite-step opacity and K-step opacity via insertion mechanism. *Automatica*, 140:110212, June 2022.

[61] Rongjian Liu, Jianquan Lu, and Christoforos N. Hadjicostis. Opacity Enforcement via Attribute-Based Edit Functions in the Presence of an Intended Receiver. *IEEE Transactions on Automatic Control*, 68(9):5646–5652, September 2023.

[62] Józef Lubacz, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Principles and overview of network steganography. *IEEE Communications Magazine*, 52(5):225–229, May 2014.

[63] Parthasarathy Madhusudan. Synthesizing Reactive Programs. In *DROPS-IDN/v2/Document/10.4230/LIPIcs.CSL.2011.428*. Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2011.

[64] Laurent Mazaré. Using unification for opacity properties. In *In Proceedings of the Workshop on Issues in the Theory of Security (WITS'04)*, pages 165–176, 2004.

[65] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 79–93, May 1994.

[66] Rômulo Meira-Góes, Christoforos Keroglou, and Stéphane Lafortune. Towards probabilistic intrusion detection in supervisory control of discrete event systems1. *IFAC-PapersOnLine*, 53(2):1776–1782, January 2020.

[67] Rômulo Meira-Góes, Stéphane Lafortune, and Hervé Marchand. Synthesis of Supervisors Robust Against Sensor Deception Attacks. *IEEE Transactions on Automatic Control*, 66(10):4990–4997, October 2021.

[68] Romulo Meira-Góes, Andrew Wintenberg, Shoma Matsui, and Lafortune Stephane. MDESops: An Open-Source Software Tool for Discrete Event Systems Modeled by Automata. In *IFAC World Congress*, Yokohama, Japan, 2023.

[69] Artur Męski, Wojciech Penczek, Maciej Szreter, Bożena Woźna-Szcześniak, and Andrzej Zbrzezny. BDD-versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: Algorithms and their performance. *Autonomous Agents and Multi-Agent Systems*, 28(4):558–604, July 2014.

[70] Silvio Micali and Phillip Rogaway. Secure Computation. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 392–404, Berlin, Heidelberg, 1992. Springer.

[71] S. Mohajerani, Y. Ji, and S. Lafortune. Compositional and Abstraction-Based Approach for Synthesis of Edit Functions for Opacity Enforcement. *IEEE Transactions on Automatic Control*, 65(8):3349–3364, August 2020.

[72] Adam D. Moore. Privacy: Its Meaning and Value. *American Philosophical Quarterly*, 40(3):215–227, 2003.

[73] P. Moulin and J.A. O'Sullivan. Information-theoretic analysis of information hiding. *IEEE Transactions on Information Theory*, 49(3):563–593, March 2003.

[74] Binda Pandey and Jussi Rintanen. Planning for Partial Observability by SAT and Graph Constraints. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28:190–198, June 2018.

[75] Torben Pryds Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer.

[76] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, pages 746–757 vol.2, November 1990.

[77] B. Ramasubramanian, R. Cleaveland, and S. I. Marcus. A framework for opacity in linear systems. In *2016 American Control Conference (ACC)*, pages 6337–6344, July 2016.

[78] Elissa M. Redmiles. User Concerns & Tradeoffs in Technology-facilitated COVID-19 Response. *Digital Government: Research and Practice*, 2(1):6:1–6:12, November 2020.

[79] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[80] Emmanuel Roche and Yves Schabes, editors. *Finite-State Language Processing*. Language, Speech, and Communication. MIT Press, Cambridge, Mass, 1997.

[81] Ron Ross, Victoria Pillitteri, Gary Guissanie, Ryan Wagner, Richard Graubart, and Deb Bodeau. Enhanced Security Requirements for Protecting Controlled Unclassified Information: A Supplement to NIST Special Publication 800-171: A Supplement to NIST Special Publication 800-171. Technical Report NIST SP 800-172, National Institute of Standards and Technology, Gaithersburg, MD, January 2021.

[82] A. Saboori and C. N. Hadjicostis. Opacity-Enforcing Supervisory Strategies via State Estimator Constructions. *IEEE Transactions on Automatic Control*, 57(5):1155–1165, May 2012.

[83] Anooshiravan Saboori and Christoforos N. Hadjicostis. Notions of security and opacity in discrete event systems. In *2007 46th IEEE Conference on Decision and Control*, pages 5056–5061, December 2007.

[84] Anooshiravan Saboori and Christoforos N. Hadjicostis. Verification of initial-state opacity in security applications of DES. In *2008 9th International Workshop on Discrete Event Systems*, pages 328–333, May 2008.

[85] Anooshiravan Saboori and Christoforos N. Hadjicostis. Verification of infinite-step opacity and analysis of its complexity*. *IFAC Proceedings Volumes*, 42(5):46–51, 2009.

[86] Anooshiravan Saboori and Christoforos N. Hadjicostis. Verification of K-step opacity and analysis of its complexity. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*, pages 205–210, December 2009.

[87] Anooshiravan Saboori and Christoforos N. Hadjicostis. Current-State Opacity Formulations in Probabilistic Finite Automata. *IEEE Transactions on Automatic Control*, 59(1):120–133, January 2014.

[88] Anne-Kathrin Schmuck, Thomas Moor, and Rupak Majumdar. On the relation between reactive synthesis and supervisory control of non-terminating processes. *Discrete Event Dynamic Systems*, 30(1):81–124, March 2020.

[89] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(Preliminary Report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 1–9, New York, NY, USA, April 1973. Association for Computing Machinery.

[90] Ruochen Tai, Liyong Lin, Yuting Zhu, and Rong Su. Privacy-preserving co-synthesis against sensor–actuator eavesdropping intruder. *Automatica*, 150:110860, April 2023.

[91] Yin Tong, Zhiwu Li, Carla Seatzu, and Alessandro Giua. Verification of State-Based Opacity Using Petri Nets. *IEEE Transactions on Automatic Control*, 62(6):2823–2837, June 2017.

[92] Gertjan van Noord. Treatment of Epsilon Moves in Subset Construction. *Computational Linguistics*, 26(1):61–76, March 2000.

[93] Ju Wang and Andrew A. Chien. Understanding when location-hiding using overlay networks is feasible. *Computer Networks*, 50(6):763–780, April 2006.

[94] J. C. Willems. The Behavioral Approach to Open and Interconnected Systems. *IEEE Control Systems Magazine*, 27(6):46–99, December 2007.

[95] Andrew Wintenberg, Matthew Blischke, Stéphane Lafortune, and Necmiye Ozay. Enforcement of K-Step Opacity with Edit Functions. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 331–338, December 2021.

[96] Andrew Wintenberg, Matthew Blischke, Stéphane Lafortune, and Necmiye Ozay. A Dynamic Obfuscation Framework for Security and Utility. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 236–246, May 2022.

[97] Andrew Wintenberg, Matthew Blischke, Stéphane Lafortune, and Necmiye Ozay. A general language-based framework for specifying and verifying notions of opacity. *Discrete Event Dynamic Systems*, 32(2):253–289, June 2022.

[98] Andrew Wintenberg, Stéphane Lafortune, and Necmiye Ozay. Opacity From Observers With a Bounded Memory. *IEEE Control Systems Letters*, 7:2359–2364, 2023.

[99] Andrew Wintenberg, Necmiye Ozay, and Stéphane Lafortune. Integrating Obfuscation and Control for Privacy. *IEEE Transactions on Automatic Control*, Under Review, 2024.

[100] Y. Wu and S. Lafortune. Enforcement of opacity properties using insertion functions. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 6722–6728, December 2012.

[101] Yi-Chin Wu and Stéphane Lafortune. Comparative analysis of related notions of opacity in centralized and coordinated architectures. *Discrete Event Dynamic Systems*, 23(3):307–339, September 2013.

[102] Yi-Chin Wu and Stéphane Lafortune. Synthesis of insertion functions for enforcement of opacity security properties. *Automatica*, 50(5):1336–1348, May 2014.

[103] Yi-Chin Wu, Vasumathi Raman, Stéphane Lafortune, and Sanjit A. Seshia. Obfuscator Synthesis for Privacy and Utility. In Sanjai Rayadurgam and Oksana Tkachuk, editors, *NASA Formal Methods*, Lecture Notes in Computer Science, pages 133–149, Cham, 2016. Springer International Publishing.

[104] Yi-Chin Wu, Vasumathi Raman, Blake C. Rawlings, Stéphane Lafortune, and Sanjit A. Seshia. Synthesis of Obfuscation Policies to Ensure Privacy and Utility. *Journal of Automated Reasoning*, 60(1):107–131, January 2018.

[105] Yi-Chin Wu, Karthik Abinav Sankararaman, and Stéphane Lafortune. Ensuring Privacy in Location-Based Services: An Approach Based on Opacity Enforcement. *IFAC Proceedings Volumes*, 47(2):33–38, January 2014.

[106] A. D. Wyner. The wire-tap channel. *The Bell System Technical Journal*, 54(8):1355–1387, October 1975.

[107] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (Sfcs 1982)*, pages 160–164, November 1982.

[108] Xiang Yin and Stéphane Lafortune. On two-way observer and its application to the verification of infinite-step and K-step opacity. In *2016 13th International Workshop on Discrete Event Systems (WODES)*, pages 361–366, May 2016.

[109] Xiang Yin and Stéphane Lafortune. A new approach for the verification of infinite-step and K-step opacity using two-way observers. *Automatica*, 80:162–171, June 2017.

[110] Xiang Yin and Shaoyuan Li. Synthesis of Dynamic Masks for Infinite-Step Opacity. *IEEE Transactions on Automatic Control*, pages 1–1, 2019.

[111] Xiang Yin, Majid Zamani, and Siyuan Liu. On Approximate Opacity of Cyber-Physical Systems. *IEEE Transactions on Automatic Control*, 66(4):1630–1645, April 2021.

[112] Tae-Sic Yoo and Stéphane Lafortune. Solvability of Centralized Supervisory Control Under Partial Observation. *Discrete Event Dynamic Systems*, 16(4):527–553, December 2006.

[113] Jianing Zhao, Xiang Yin, and Shaoyuan Li. A Unified Framework for Verification of Observational Properties for Partially-Observed Discrete-Event Systems. *IEEE Transactions on Automatic Control*, pages 1–8, 2024.

[114] Bingzhuo Zhong, Siyuan Liu, Marco Caccamo, and Majid Zamani. Secure-by-Construction Controller Synthesis via Control Barrier Functions. *IFAC-PapersOnLine*, 56(2):239–245, January 2023.