Dynamic Observation
in Discrete-Event Systems

by

David Sears

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen’s University
Kingston, Ontario, Canada
February 2018

Copyright © David Sears, 2018
Abstract

This dissertation studies problems of dynamic observation in discrete-event systems. In discrete-event systems, agents exist that observe events generated by a system. The agents have some task to fulfill that requires observing the behaviour of the system, such as control of the system or diagnosis of failures in the system. Traditionally in discrete-event systems, agents observe all occurrences of a given event, or no occurrences of that event.

In problems of dynamic observation, certain occurrences of a given event may be observed whereas other occurrences may not be observed. An example scenario of dynamic observation is when an agent uses sensors for observing events which the agent may turn on or off at its leisure. Another example scenario is when the agent receives messages from another agent indicating that certain events in the system have occurred.

This dissertation studies problems related to the above two scenarios, as well as more general scenarios where dynamic observation occurs. First, we study the problem of computing a finite-state representation of how an agent uses its sensors for making event observations. We show that such a representation may be computed in polynomial-time when certain restrictions on how an agent may use its event sensors are imposed. We show that a simple generalization of these restrictions results in the
problem of computing the representation being PSPACE-complete.

Second, we study the problem of computing indistinguishable state pairs of finite-state automata. Indistinguishable state pairs are used in the verification of properties involved in solutions to problems of dynamic observation. We demonstrate how indistinguishable state pairs may be computed and how they may be used to verify such properties.

Third, we study cases in dynamic observation where observing more event occurrences permits agents to more precisely estimate the state of the system. We apply these results in the solution to a problem where two agents must accomplish their tasks while communicating as little as possible to one another.

Note that this dissertation is presented in a manuscript-style.
Co-Authorship

• Chapter 3
  


• Chapter 4


• Chapter 5


Acknowledgements

First and foremost, I thank my supervisor, Dr. Karen Rudie. I’ve had many ideas over the past seven years, most of which have lead to dead ends or I felt were not worthwhile enough to develop further. I have a tendency to overcomplicate things, and Karen has been burdened with interpreting my overly-complicated explanations of my ideas. I am thankful for her feedback for simplifying these ideas.

Karen has been invaluable for her knowledge of academic writing. My writing, especially when it comes to formal content, can be very messy and hard to follow. Karen has provided excellent recommendations for organizing and cleaning up my writing.

Karen has provided me with tremendous support over these last seven years. She provided me with financial support nearly every year since I started my Ph.D. She supported me even when I was past the four-year mark of my Ph.D and between jobs, for which I am extremely grateful. Last, but not least, I thank her for her moral encouragement, as I have hit many lows in the last seven years, and have often struggled to pick myself back up.

I thank the School of Computing. The School provided me with several TAships (five in total), even when I was well past the four-year mark of my Ph.D. I acquired some great teaching experience working under Dr. Robin Dawes and Margaret Lamb,
some of the best instructors in the School.

I thank the examiners for my Ph.D. defense: Dr. John Mullins, Dr. Kai Salomaa, Dr. Ram Murty, Dr. Farhana Zulkernine and Dr. Tucker Carrington. I especially thank Dr. John Mullins and Dr. Kai Salomaa for their very useful feedback on ways the thesis could be improved, which has helped to shape the final version of the thesis.

For funding, I thank the Natural Sciences and Engineering Research Council of Canada (NSERC). NSERC has been the primary source of funding for Karen’s research group, and hence the primary source of funding for myself through the first few years of my studies.

I thank the Mathematics and Computer Science department at the Royal Military College of Canada. Specifically, I thank professors Alain Gosselin, Francois Rivest and Rachad Shoucri for giving me my first opportunity as a lecturer for a course in Data Structures and Algorithms, and for helping me along the way.

Last, I thank my parents Bryan and Debra, my sister Laura, my dog Billie, the family cat Leo, and my cousins Dan Hendry and Geoff Mackay for all of their support and encouragement.

This thesis is dedicated to my late dog, Dylan.
Statement of Originality

I, David Sears, certify that all of the work described within this thesis is the original work of the author, unless otherwise noted. Any published (or unpublished) ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

(February, 2018)
# Table of Contents

- **Abstract** ................................................................. i
- **Co-Authorship** .......................................................... iii
- **Acknowledgements** ..................................................... v
- **Statement of Originality** .............................................. vii
- **Table of Contents** ....................................................... viii
- **List of Figures** ........................................................... ix

### Chapter 1: Introduction .......................... 1
- 1.1 Background ......................................................... 1
- 1.2 Thesis Statement .................................................. 6
- 1.3 Contributions ....................................................... 6
- 1.4 The Big Picture ..................................................... 9
- 1.5 Thesis Organization ................................................. 14

### Chapter 2: Preliminaries ................. 18

### Chapter 3: Literature Review – Minimal Sensor Activation and Minimal Communication in Discrete-Event Systems .... 29
- 3.1 Preliminaries ....................................................... 30
- 3.2 Minimal Sensor Activation for Purposes of State Disambiguation and State Estimation ................................................. 32
- 3.3 Minimal Communication for Purposes of State Disambiguation ......................................................... 62
Chapter 4: Computing Deterministic Finite Automaton Representations of Sensor Activation Maps in Discrete-Event Systems

4.1 Introduction ............................................. 86
4.2 Preliminaries ............................................. 90
4.3 Computing decisions for policies satisfying a strong notion of feasibility ............................................. 95
4.4 Computing decisions when active event sensors will eventually detect event occurrences ............................................. 101
4.5 Computing decisions using state equivalence classes ............................................. 114
4.6 Conclusions ............................................. 145

Chapter 5: Computation and Applications of Indistinguishable State Pairs in Discrete-Event Systems

5.1 Introduction ............................................. 146
5.2 Preliminaries ............................................. 149
5.3 A known algorithm for computing indistinguishable clusters ............................................. 151
5.4 Computing Indistinguishable State Pairs using the Product of NFA ............................................. 154
5.5 Computing Indistinguishable State Pairs in the Presence of Unobservable Cycles ............................................. 167
5.6 Example Demonstrating Computation of Indistinguishable State Pairs ............................................. 174
5.7 Applications of Computing Indistinguishable State Pairs ............................................. 176
5.8 Verifying if one state is distinguished from another ............................................. 201
5.9 Conclusions ............................................. 204

Chapter 6: Monotonic Observation Maps ............................................. 205

6.1 Introduction ............................................. 205
6.2 Preliminaries ............................................. 206
6.3 When Observing More Event Occurrences Permits More Precise Estimates ............................................. 210
6.4 Minimal Observation for Satisfying Feasibility and Specifications ............................................. 254
6.5 Conclusions ............................................. 279

Chapter 7: Conclusions and Future Work ............................................. 280

Bibliography ............................................. 284
# List of Figures

3.1 Automaton generator of $\mathcal{L}(\tilde{G})$ for which sensor activation map $\tilde{\omega}$ is not implementable. ............................................ 36

3.2 Automaton generator of $\mathcal{L}(\tilde{G})$ for which sensor activation map $\tilde{\omega}$ is implementable. ............................................ 36

3.3 Automaton used to demonstrate the approach of Wang et al. [53] ............................................ 46

3.4 Automaton $G_{rev}$ used to compute $T_{\tilde{\omega}_{\text{spec}}}$ when $\Sigma_o = \{a, c, f\}$. ............................................ 46

3.5 Automaton $G$ used to demonstrate the approach of Shu et al. [37] ............................................ 58

3.6 Detector automaton for $G$ under $\Omega_1^{\text{rev}}$. ............................................ 58

3.7 Detector automaton for $G$ under $\Omega_2^{\text{rev}}$. ............................................ 58

3.8 Automaton $R_1$ ............................................ 68

3.9 Automaton $R_2$ ............................................ 68

3.10 Automaton $R_1 \| R_2$ ............................................ 70

3.11 Automaton $R_1(C_{21})$. ............................................ 70

3.12 Example demonstrating lack of monotonicity of feasible communication policies. ............................................ 72

4.1 Plant and policy demonstrating that $T_\Omega$ is not necessarily transitive. ............................................ 94

4.2 Plant and policy which satisfies (4.4) but does not satisfy (4.6). ............................................ 96

4.3 Plant and policy which satisfies (4.6) used to demonstrate how sensor activation decisions are computed. ............................................ 100

4.4 Plant and policy which satisfies (4.10) where the sensor for $e_2$ is activated at $x_1$ but $e_2$ is not observed on occurrence of transition $(x_1, e_2)$. ............................................ 104

4.5 Plant and policy defined which satisfies (4.4) and (4.11) but not (4.6). ............................................ 106

4.6 Plant $G$ and policy $\Omega$ which satisfies (4.4) and (4.11). Computing observer automaton is exponential whereas computing $|G|$ is polynomial. ............................................ 114

4.7 Plant and policy which satisfies (4.4) and where scenario 2 occurs. ............................................ 123

4.8 DFA $DET_{\Omega}(G)$ constructed from applying subset construction on plant of Fig. 4.7 when transitions not in $\Omega$ are relabeled with $\varepsilon$. ............................................ 123

5.1 CLUSTER-TABLE executes in $\Theta(|X|^4 \cdot |\Sigma|)$ for this example. ............................................ 153

5.2 Automaton $G$. Transitions in set $\delta_e$ are labeled by $\varepsilon$. ............................................ 175

5.3 Automaton $G/\rho$. ............................................ 175
5.4 Automaton $G/\rho \otimes G/\rho$.

5.5 A visual aid to help demonstrate that $F(\mathcal{V}_1' \cup \mathcal{V}_2')$ if and only if $F(\mathcal{V}_1' \cup \mathcal{V}_2' \cup \mathcal{Y}_j')$. A dashed, undirected edge labeled by a set denotes that a pair formed by the two states with which the edge is incident is a member of the set, e.g., $(y_1, y_4) \in \mathcal{Y}_1'$. A directed edge labeled by $\varepsilon^*$ denotes that the sink vertex is in the $\varepsilon$-closure of the source vertex in $G/\rho$.

5.6 Example used to demonstrate that one could omit testing some pairs in $\mathcal{Y}_f'$.

6.1 $I_{[O_2 \cup C_1]}$.

6.2 $I_{[O_1 \cup C_2]}$.

6.3 $I_{[O'_1 \cup C'_1]}$.

6.4 $I_{[O'_1 \cup C'_2]}$.
Chapter 1

Introduction

1.1 Background

A Discrete-Event System (DES) is a system with a discrete state set and event-driven dynamics. Events occur sequentially and asynchronously in such systems. As an example, an elevator can be modeled as a DES. The events would be the opening and closing of doors, the pressing of buttons, the arrival at a floor, etc. The states of the system would be defined by information such as the current floor of the elevator, the set of floors in which the elevator is to travel, the direction it is moving, etc. As events occur sequentially in the model, the simultaneous pressing of two buttons is not modeled (but, in reality, the low-level event processing unit ensures the sequencing of events). As events occur asynchronously, the time between two event occurrences is not captured in the model.

We consider DES which may be modeled by (possibly nondeterministic) Finite Automata. A finite automaton has a state set and event set which are both finite. Events label transitions between states of a finite automaton. The occurrence
of an event $\sigma$ at a given state $x$ of a finite automaton triggers those transitions from $x$ which are labeled by $\sigma$.

The monitoring or controlling of discrete-event systems by agents external to the system is of practical interest. Left uncontrolled, such systems may execute in a manner that results in undesirable behaviour. The control of DES was initiated by Ramadge and Wonham in their works [20, 21, 57] where agents referred to as supervisors observe the events of a DES and, depending on the sequence of events that they observe, disable certain events from occurring in the system to satisfy a specification of desirable behaviour.

When the control of a DES by agents is overly-restrictive or cannot be implemented it may instead suffice to monitor the operation of the DES to determine when faults or violations of desirable behaviour occur. The monitoring and diagnosing of faults in DES by external agents was initially considered in Lin [15] and Sampath et al. [25].

In problems of control and diagnosis agents may not have complete observation of the system. Some events generable by the system may be observed by an agent while others may not. In such cases we say that the agent’s observation of the DES is static. To further complicate matters, individual occurrences of a given observable event may be observed while others may not. In such cases we say that the agent’s observation of the DES is dynamic.

In this dissertation, we study problems related to the monitoring of DES where an agent’s observation is dynamic. However, the observation of event occurrences is not defined arbitrarily. For the most part, we consider that the observation of an event occurrence depends on if an agent’s sensor for the event is activated (i.e., turned on)
or if the event is communicated to the agent by some other agent. In some cases we consider more general settings that include the above two cases.

In problems of sensor activation, an agent has associated with it sensors responsible for detecting the occurrence of events observable to the agent. Furthermore, the agent is capable of activating (i.e., turning on) or deactivating (i.e., turning off) these sensors following the observation of individual event occurrences. When the sensor for observable event \( \sigma \) is on (respectively, off) any occurrence of \( \sigma \) will (resp., will not) be observed by the agent. In such problems it is assumed that agents have control over their own sensors and not over the sensors of other agents. The criteria for determining when an event sensor should be activated (resp., deactivated) are dependent on the problem at hand and an agent’s estimate of the current state of a given DES. In the literature, when sensor activation is concerned, usually cost is associated with the activation of sensors and the problems considered require that sensors be used as little as possible to reduce cost. Studies in minimizing the activation of sensors are motivated by increasing the life span and availability of sensors, by the need to conserve power when only limited battery power is available, or by the need to maintain security.

Problems of sensor activation are different from problems of sensor selection which have been considered in the literature (e.g., [7, 22, 59, 60]). Sensor selection concerns computing a set of observable events having minimum cost or minimum cardinality to satisfy some goal (e.g., to satisfy observability or diagnosability of DES, concepts which will be discussed in the literature review). Sensor selection problems for diagnosability, observability and normality (a property that is stronger than observability, one of the requirements for centralized control over partially-observed DES) have been
proven to be NP-complete \cite{59}. However, many of the problems of sensor activation that have been considered may be solved in polynomial time, as will be seen in the literature review.

In problems of communication in decentralized and distributed DES, information about the DES required for satisfying a given objective (e.g., control or diagnosis of DES) is distributed among agents and typically agents individually do not have enough information for satisfying the given objective. Then communication of some of the distributed information among the agents is required. Typically the information communicated consists of observed event occurrences of the DES. In the literature review and in Chapter \ref{chap:literature}, we consider that communication of observed event occurrences between agents is reliable, in-order and without loss or delay. We consider that an agent can communicate with any other agent (i.e., the communication topology is complete). In the literature review, we recall some cases where more restrictive communication topologies are used. Similar to problems of sensor activation, problems of communication usually require that communication among agents be minimized in some way. Studies in minimizing communication are motivated by the need to reduce network bandwidth, by the need to conserve power when only limited battery power is available or by the need to maintain security.

In some work on sensor activation and communication, the decision to activate a sensor or communicate an event is based on the transitions of a given finite automaton representation of the DES. For example, Wang et al. \cite{54} and Rudie et al. \cite{23} are seminal works on sensor activation and communication in DES, respectively. In these works, and others to be examined in the literature review, observations and the decisions to make observations (i.e., turning on sensors and communicating events) are
modeled using the transitions of the DES. There are some shortcomings to modeling observations using the transitions of a DES which we investigate in this dissertation. One problem is that, after determining which transitions should be observed (i.e., which sensors should be turned on for events and when), the agent needs to know which sensors it should turn on and off unambiguously. Traditionally this is solved using a nondeterministic finite automaton to deterministic finite automaton conversion. However, this conversion is well-known to be in exponential-time in the worst-case. Second, determining which transitions of a DES should be observed and which transitions of a DES should be communicated from one agent to another cannot be done arbitrarily. Certain requirements need to be met, e.g., defining the transitions in such a manner that the agent can unambiguously know when it should turn a sensor on or off or unambiguously know when to communicate the occurrence of an event to another agent. Verifying if a set of chosen transitions satisfies the required properties may be done by computing the indistinguishable state pairs of the DES. We address this problem in Chapter 5.

Problems for computing sensor activation decisions or communications between agents typically are based on a property that we refer to as monotonicity of observations. Roughly speaking, this property is that, as an agent observes more event occurrences, it should be able to more precisely estimate the current state of the DES or behaviour (i.e., string or trace) generated by the DES. Very few results exist in the literature regarding the property of monotonicity of observations. We review some of these properties in the literature review. We introduce several new results in this dissertation which exhibit this property, as stated in the Thesis Statement.
1.2 Thesis Statement

1. We identify a shortcoming related to existing work on sensor activation that is computationally expensive, specifically, the problem of computing deterministic finite representations of sensor activation maps and sensor activation policies. We examine specific cases for this problem which permit solutions to be computed in polynomial-time.

2. We review an existing algorithm for computing indistinguishable state pairs of a given finite automaton that is used to verify properties used in the solutions to sensor activation and communication problems. We propose a simple algorithm for solving the same problem and demonstrate how this algorithm may be used to verify such properties.

3. We present several results where observations are monotonic in the sense that, as an agent observes more, the agent may more precisely estimate the current state of the DES or behaviour (i.e., string or trace) generated by the DES. Such results may be used to solve problems where observation of event occurrences should be minimized. We apply these results for solving one such problem where two agents communicate with one another.

1.3 Contributions

Here we elaborate on the contributions made in this dissertation in more detail than stated in the Thesis Statement.
The first contribution is a detailed literature review on sensor activation and communication problems in DES and their solutions. Specifically, we consider problems where the use of event sensors should be minimized and where the communication of event occurrences between agents should be minimized. This literature review is an abbreviated version of the published survey paper \cite{32}, which is the first of its kind to treat this topic so exhaustively.

Second, we study sensor activation policies defined over the transitions of a given DES automaton. We recall the problem of computing a deterministic, finite-state representation of a sensor activation policy. Typically this problem is solved using conversion from nondeterministic to deterministic finite automata, which is well-known to be in exponential-time. We identify special cases where the finite-state representation of the sensor activation policy can be computed in polynomial-time. Specifically, we consider two strong variants of feasibility (also called consistency), a property that, roughly speaking, states that the decision to turn a sensor on or off (or communicate an event occurrence to another agent) should not be ambiguous when such decisions are made based on what the agent has observed. Feasibility is a property that is required to be satisfied by solutions to all problems of sensor activation and communication. We show that one of the cases studied is not particularly restrictive, as a simple generalization of this case results in the problem of computing the finite-state representation of a sensor activation policy being PSPACE-complete. In the two cases studied, we detail how the associated finite-state representation of a sensor activation policy may be computed.

The third contribution is the following. Computing solutions to sensor activation
and communication problems requires verifying properties such as feasibility, observability and coobservability. Such properties can be verified by computing the indistinguishable state pairs of automata. We review an existing algorithm for computing indistinguishable state pairs and demonstrate its asymptotic complexity for a particular case. We introduce a straightforward algorithm for computing indistinguishable state pairs, which is derived directly from the product of two nondeterministic finite automata. We demonstrate how observability, coobservability, and a condition that we study in one of the cases for sensor activation mentioned previously may be verified using indistinguishable state pairs. We also demonstrate that the problem of determining if there exists a sequence of events generated by the DES automaton that leads to a given state, and this sequence does not appear the same as sequences leading to another given state is PSPACE-complete.

Fourth, we provide a general formal setting for studying problems of dynamic observation, which is derived from the works studied in the literature. We introduce the notion of an observation map, which generalizes maps such as sensor activation maps and communication maps that have been studied previously in the literature. In this setting, we demonstrate that the monotonicity of observations property holds for many different cases involving agents turning their sensors on or off and communicating with one another. Some of these cases generalize previous results where monotonicity of observations is demonstrated (e.g., [54] Theorem 1 when concerning sensor activation). We also investigate some cases where monotonicity of observations does not hold, which generalize a case studied previously in the literature ([46] Section 2.2.6). In all of the cases studied we require that at least some of the observation maps (i.e., sensor activation maps, communication maps, and generalizations of
them) considered satisfy various notions of feasibility (i.e., the decision to do something must unambiguously be determined using the observations made by an agent). We introduce a predicate defined over observation maps that we refer to as a specification. We introduce a problem of minimizing the communication of event occurrences between two agents for purposes of satisfying a given specification and feasibility of observation maps. Solving this problem in the general case of multiple agents is left open.

1.4 The Big Picture

Each of the chapters that constitute the body of this dissertation are very technical, and most of the focus is placed on the precise details rather than the big picture ideas within each respective chapter. Here we attempt to provide a sketch of the big picture problems and ideas in the context of each chapter.

In Chapter 4 we consider automata models of DES and suppose that whether or not an event occurrence is observed by some external agent is dependent on certain features of the model, namely, the transitions of the automaton model. Several existing works consider automaton models of DES and this model of observations, and are covered in the literature review of Chapter 3. However, modeling observations over the transitions of an automaton model is useless to an external agent unless we can define how the DES is observed under such a model of observations. When an external observer observes a DES, it only sees events that occur in the DES, not the state that the DES is in or transition that is made from one state to another in the DES. However, the agent can use the sequence of events that it observes to infer what state the DES may be in or what transitions have occurred in the DES.
So Chapter 4 is motivated by the problem of computing a deterministic representation of the different ways the DES can be observed when observations are defined over transitions of the automaton model. Solutions to this problem are difficult to compute in general (i.e., in the worst-case, the standard approach used for solving this problem is in exponential-time). So we study special cases in which the solutions can be computed efficiently (i.e., in polynomial-time). The compromise we make to achieve this is that these special cases are particularly strong, and most ways in which observations can be defined over automaton transitions will not belong to these special cases. Also, the constructions that are used in these cases cannot be used to compute a precise estimate of the current state of the DES. That is, these constructions are only useful for making observations of the DES.

However, the constructions that are used for representing how an agent observes a DES are very small (not minimal in general, but very close to minimal). As such, they are appealing when we have control over what the external agent observes, and wish to construct a representation of the agent’s observations that is small, and construct this representation quickly. A hypothetical application is where an external agent wants to observe some system, but it is hazardous to observe the system directly. In this case, we may want to compute (possibly many) inexpensive observers that we can drop into the vicinity of the system for making observations of the system. These observers would relay the observations of event occurrences that they observe back to the external agent. If one of the observers is destroyed while in the vicinity of the DES, then we can quickly construct another observer and drop it in to replace the destroyed observer if necessary. Further, the observers may have only a limited amount of energy that they can use to observe the system. When an observer’s energy
reserves are depleted, then it can no longer make observations of the system. So it is also desirable to make sure that an observer can last as long as possible (assuming that it is not destroyed in the hazardous setting) while still being able to make the observations of the system that are necessary for the external agent.

Also, the constructions provided in Chapter 4 are interesting in principle, and may also be of interest outside of the application area concentrated on in Chapter 4 (sensor activation). For example, such constructions have not been studied in automata theory to the knowledge of the author. In automata theory, such constructions can provide a way to compute a deterministic finite automaton that generates the same language as a non-deterministic finite automaton in polynomial-time when the requirements for these constructions that we study in Chapter 4 are satisfied. In general, the standard procedure for computing such a deterministic finite automaton (i.e., the subset construction or powerset construction) is in exponential-time. An example is provided in Chapter 4 which demonstrates that our construction is in polynomial-time, whereas the standard procedure is in exponential-time.

In Chapter 5 we consider again automaton models of DES and that observations are modeled over the transitions of the DES. The problem considered in Chapter 5 is the computation of the pairs of states that are reached by event sequences in the DES that appear identical according to the observation model. Such state pairs are useful in the development of solutions to different supervisory control problems. For example, observability, a condition that is required for the control of DES by a single agent that does not see all event occurrences, can be reduced to verifying which pairs of states of a DES are indistinguishable. Generalizing further, coobservability, a condition that is required for the control of DES by two or more agents where each
agent may not observe all event occurrences, can similarly be reduced to verifying which pairs of states of a DES are indistinguishable to multiple agents. We demonstrate how these conditions can be verified by verifying which pairs of states of a DES are indistinguishable in Chapter 5. We also demonstrate how indistinguishable state pairs may be used to verify if the observation model belongs to one of the special cases studied in Chapter 4.

Chapter 5 proposes a straightforward algorithm for computing the product of automata where observations are modeled over the transitions of the respective automata. Though the algorithm is straightforward, an in-depth analysis of the correctness of the algorithm and the complexity of the algorithm are provided. To the author’s knowledge, though the product of automata is used in nearly all standard texts on automata theory and DES, no formal treatment of any algorithm for computing the product exists (surprisingly!).

Chapter 5 also addresses a problem of security in DES. Suppose that a DES is modeled as an automaton with a specific, finite state set. Information about the DES is encoded in each state of the automaton. Some of the states may contain privileged information that we wish to keep hidden from external agents. However, external agents can observe some event occurrences of the DES, and may even understand how the DES is modeled. In this setting, it is natural to consider the problem of verifying if an external agent can unambiguously determine if the DES is in one of its secret states at any point in time. A result in Chapter 5 shows that this problem is difficult to solve using any algorithm. Specifically, the problem is PSPACE-complete.

In Chapter 6 a formal, abstract, general setting for studying how DES may be observed by agents is introduced. This setting is an extension of settings used in
previous works, e.g., it is a generalization of all settings used in all works covered in the literature review of Chapter 3. We use our new setting to study different scenarios where seeing more event occurrences of the DES permits an agent to more precisely estimate the behaviour of the DES. We note that this property of knowing more as an agent observes more does not hold in general, so specific scenarios need to be considered.

The need to know more as an agent observes more is motivated by problems of reducing the observations that an agent makes of a DES. We review many such problems in the literature review of Chapter 3. The primary contribution of Chapter 6 is that the scenarios we study are more general or incomparable with properties that exist in the literature that permit an agent to know more as it observes more. Thus the scenarios we study may be useful for solving problems of reducing an agent’s observation of a system that have not been considered in the literature before. We provide an example of such a problem where it is required to minimize communication between two agents for satisfying some goal. The problem we propose is unlike any other studied in the literature before. Also, the problem, and its solution that we propose, are very general. This generality comes at the expense of being impractical. However, it does serve as the basis upon which one can derive practical solutions when one considers that DES are modeled by any machine model that is not a general model of computation, not just finite-state automata. For example, if one were to consider pushdown automata as a model of DES (similarly, linear-bounded automata, Petri nets, etc.), then we conjecture that a more specific problem and its solution could be derived easily from the more general problem and solution that we propose. However, it remains impractical for machine models of recursively-enumerable languages, such
as Turing machine models.

We believe that extending the framework of Chapter 6 and using it to derive more scenarios where an agent knows more as it observes more is the most important avenue for future work in this dissertation. Such results could be used for solving problems where agent observations of a system should be reduced for one purpose or another.

1.5 Thesis Organization

First, we provide a preliminary set of concepts and notation that are used throughout the dissertation in Chapter 2. We cull from many concepts used in the DES literature and some concepts from the Automata Theory literature.

In Chapter 3 we conduct a literature review of minimal sensor activation and minimal communication in DES. We review works on sensor activation and communication for purposes of state disambiguation and state estimation of DES. In problems of state disambiguation, an agent uses its observation of events generated by the DES to maintain an estimate of the actual state that the DES is in. It is required that an agent observe events generated by the DES in such a manner that the agent’s state estimate of the DES does not contain certain pairs of states of the DES. That is, certain pairs of states of the DES should not be indistinguishable (also called confusable) to the agent. The determination of which pairs of states should not be indistinguishable is dependent on the specific problem at hand. For instance, problems of supervisory control or diagnosis may be cast as state disambiguation problems [50]. In problems of state estimation, it is required that an agent observe
events generated by the DES in such a manner that the agent can always or eventually know the actual state of the DES. That is, under certain conditions, the agent’s state estimate contains only the actual state of the DES. Problems of state estimation are typically instances of problems of state disambiguation. The body of the chapter comprises three sections, each regarding a different application or observation method. In Section 3.1 we provide a set of preliminary concepts necessary for reviewing the works in the literature review. In Section 3.2 we review approaches for minimal sensor activation for purposes of state disambiguation and state estimation of DES. In Section 3.3 we review approaches for minimal communication for purposes of state disambiguation and state estimation of DES. For a more general review on problems of sensor activation and communication (including supervisory control) we refer the reader to our technical report [30].

The remaining chapters constitute the body of the dissertation. The chapters are self-contained and can be read independent of one another.

Chapter 4 reviews the problem of computing deterministic finite automaton representations of a given sensor activation policy. We consider three classes of sensor activation policies of increasing generality. Each class is defined by a type of feasibility condition, i.e., a condition requiring an agent’s sensor activation decisions to be made unambiguously based on the event observations made by the agent. For each class, we demonstrate ways to compute deterministic, finite representations of sensor maps in polynomial time. However, for the last class considered, we demonstrate that verifying if an arbitrary sensor activation policy belongs to this class is PSPACE-complete. An introduction to the chapter is provided in Section 4.1. A preliminaries section introducing the requirements for understanding the remainder
of the chapter is provided in Section 4.2. In Section 4.3 we consider sensor activation policies which satisfy a very strong notion of feasibility. This notion of feasibility is generalized in Section 4.4. A further generalization characterizing the last class of sensor activation policy studied is treated in Section 4.5. Finally, we summarize the results of the chapter in Section 4.6.

Chapter 5 investigates the computation of indistinguishable state pairs of nondeterministic finite automata with observable and unobservable transitions. In Section 5.1 we provide a brief introduction to the chapter. In Section 5.2 we provide the concepts and notation used throughout the chapter. The problem of computing indistinguishable state pairs has been considered previously [50]. In Section 5.3 we review an existing algorithm for computing indistinguishable state pairs, and demonstrate its asymptotic complexity for a particular case. In Section 5.4 we recall the product of nondeterministic finite automata, denoted throughout by $\otimes$. We demonstrate an equivalence between the indistinguishable state pairs of an automaton $G$ and the states of $G \otimes G$. We propose an algorithm for the computation of $G_1 \otimes G_2$ for given automata $G_1, G_2$. In Section 5.5 we define the construction of a quotient automaton when automaton $G$ contains cycles of unobservable transitions. We demonstrate an equivalence between the indistinguishable state pairs of automaton $G$ and the states of the product of the quotient automaton with itself. In Section 5.6 we provide an example demonstrating computation of the quotient automaton and the product. In Section 5.7 we demonstrate how different properties studied in DES may be verified using our algorithm for computing indistinguishable state pairs. In Section 5.8 we consider the problem of determining if a sequence of events leads to one state of an automaton, but not another state in the automaton. We demonstrate that this
problem is PSPACE-complete. We summarize the chapter in Section 5.9.

In Chapter 6, we investigate settings under which observing more permits an agent to better estimate the current state or behaviour of a DES. The setting used is a very general language-based setting, independent of any particular model that may be used for a DES. We provide an overview of the chapter in Section 6.1. In Section 6.2, we formalize the concepts necessary for the chapter, specifically, the concept of an observation map, information map (similar to information maps studied prior in the literature and in Chapter 4), as well as feasibility of observation maps with respect to given information maps. In Section 6.3, we present several results where monotonicity of observations holds for different types of observation maps, information maps, and feasibility of observation maps with respect to those information maps. In Section 6.4, we introduce a predicate referred to as a specification, and formalize a problem of minimal communication between two agents. We formalize an algorithm for solving this problem, which relies on some of the monotonicity results demonstrated in Section 6.3.

Finally, we reflect on the contributions of the dissertation and highlight some possibilities for future work in Chapter 7.

Note that this dissertation is presented in a manuscript-style. That is, each of the chapters are presented independent of one another. Also, most chapters are derived heavily from work that is published by the author and co-authors, or has been submitted for publication. The association between chapters and the published or unpublished manuscripts from which they are derived is as follows:
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Manuscript</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
</tr>
</tbody>
</table>
Chapter 2

Preliminaries

In this chapter we provide the concepts and notation that are used throughout the thesis. Also, note that each subsequent chapter contains a preliminaries section with concepts and notation unique to that chapter.

Some basics on formal languages and automata used in DES are recalled. We refer the reader to the monograph of Wonham [56] and text of Cassandras & Lafortune [1] for a more comprehensive and detailed coverage of concepts used in DES. Also, familiarity with asymptotic time complexity characterizations of algorithms and their associated notation (i.e., Big-O) is required for chapters 3, 4 and 5.

Given a set, $S$, we denote the cardinality of $S$ by $|S|$. An alphabet (also called event set) is a finite set of distinct symbols (also called events, for our purposes). Let $\Sigma$ represent an alphabet. Let $\Sigma^+$ denote the set of all finite sequences of symbols in $\Sigma$ of the form $\sigma_1\sigma_2\ldots\sigma_k$ where $k \geq 1$ is arbitrary and $\sigma_i \in \Sigma$ for all $i \in \{1, \ldots, k\}$. We refer to a sequence of length zero (i.e., consisting of no symbols) as the empty string, denoted by $\varepsilon$. We assume $\varepsilon$ is not a member of any alphabet $\Sigma$. The Kleene-closure of $\Sigma$, denoted by $\Sigma^*$, is defined as $\Sigma^* = \{\varepsilon\} \cup \Sigma^+$. An element of $\Sigma^*$ is a string or
**CHAPTER 2. PRELIMINARIES**

**event sequence** over the alphabet $\Sigma$.

For $s \in \Sigma^*$, $t \in \Sigma^*$ is a *prefix* of $s$, denoted by $t \leq s$, if $s = tu$ for some $u \in \Sigma^*$. Thus $\varepsilon \leq s$ and $s \leq s$ for all $s \in \Sigma^*$. The set of all prefixes of string $s$ is denoted by $\overline{s}$. Let $K$ denote a formal language where $K \subseteq \Sigma^*$. The *prefix-closure* of language $K$ is defined as $\bigcup_{s \in K} s$ and denoted by $\overline{K}$. We refer to $K$ as being a *prefix-closed language* if and only if $K = \overline{K}$.

A nondeterministic finite automaton (NFA), $G$, is denoted by a tuple $G = (X, \Sigma, \delta, x_0)$ where $X$ is the nonempty set of states, $\Sigma$ is the alphabet, $x_0$ is the initial state, and $\delta : X \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^X$ is the (state-)transition function, a partial function which determines the transitions from state to state on occurrence of symbols in $\Sigma \cup \{\varepsilon\}$. We extend $\delta$ to a function $\delta : X \times \Sigma^* \rightarrow 2^X$ by induction on length of strings. We use $\delta(x,s)!$ to denote that $\delta(x,s)$ is defined for $x \in X$ and $s \in \Sigma^*$. The language generated by $G$ is $L(G) = \{s \in \Sigma^* : \delta(x_0,s)!\}$. We refer to $G$ as being a *generator* of $L(G)$.

Automaton $G$ may contain a set of distinguished states called *marked* states (or *accepting* states). When this is the case, automaton $G$ is denoted by a tuple $G = (X, \Sigma, \delta, x_0, X_m)$ where $X_m \subseteq X$. When $G$ contains marked states, the language that is marked by (equivalently, accepted by) $G$ is denoted by $L_m(G) = \{s \in \Sigma^* : \delta(x_0,s) \cap X_m \neq \emptyset\}$. When $G$ contains marked states, we refer to $G$ as being an *acceptor* of $L_m(G)$. We consider automata with marked states in Chapters 3 and 5.

Automaton $G$ is more specifically a deterministic finite automaton (DFA) if $\delta : X \times \Sigma \rightarrow X$ (i.e., $\delta$ is deterministic), and $\delta$ is a partial function. When $X$ is finite, a state-transition graph representation of $G$ can be constructed [10].

Automaton $G$ is reachable if any state in $X$ is reachable from $x_0$ via some sequence
CHAPTER 2. PRELIMINARIES

of transitions in \(\delta\) (i.e., for all \(x \in X\), there exists \(s \in \mathcal{L}(G)\) such that \(x \in \delta(x_0, s)\)). We only consider reachable automata in this dissertation.

In Chapters 4 and 5 we sometimes treat the transition function of an automaton as a set rather than a function. We say that \((x, \sigma, y) \in \delta\) if and only if \(y \in \delta(x, \sigma)\). When automaton \(G\) is deterministic, sometimes we exclude the exit state of a transition when referring to a transition, and simply refer to the transition by its input state and event, i.e., \((x, \sigma)\), instead of \((x, \sigma, y)\) where \(y\) denotes the exit state of the associated transition.

When automaton \(G\) is an NFA, an algorithm known as the subset construction can be used on input \(G = (X, \Sigma, \delta, x_0)\) for constructing a DFA \(G_{\text{obs}}\) where \(\mathcal{L}(G_{\text{obs}}) = \mathcal{L}(G)\) (also, \(\mathcal{L}_m(G_{\text{obs}}) = \mathcal{L}_m(G)\) if \(G\) is an acceptor) [10]. We define automaton \(G_{\text{obs}}\) next. Automaton \(G_{\text{obs}} = (Q, \Sigma, \xi, q_0)\). The state space of \(G_{\text{obs}}\) is a subset of the power set of \(X\) (i.e., \(Q \subseteq 2^X\)). State \(q_0 = \delta(x_0, \epsilon)\). For all \(s \in \mathcal{L}(G)\), \(\xi(q_0, s) = \delta(x_0, s)\).

Throughout this dissertation, a discrete-event system (sometimes called a plant) is modeled using either a DFA, an NFA, or a formal language (usually a prefix-closed language).

Given two alphabets, \(\Sigma_1, \Sigma_2\), the (natural) projection from \(\Sigma_1\) to \(\Sigma_2\) is a function which takes as input a string \(s \in \Sigma_1^*\) and erases all symbols from \(s\) that are not in \(\Sigma_2\). We consider that one or more agents observes events generated by \(G\). An agent may not be able to “observe” all events in \(\Sigma\) generated by \(G\). This induces a partition of \(\Sigma\) into events observable (resp., not observable) by the agent. In the case of one agent, we denote its set of observable events by \(\Sigma_o \subseteq \Sigma\), and set of unobservable events by \(\Sigma_{uo} = \Sigma \setminus \Sigma_o\). When \(\Sigma_{uo} \neq \emptyset\) the agent’s observation of strings generated by \(G\) is modeled by the natural projection \(P : \Sigma^* \to \Sigma_o^*\). Specifically, \(P(\epsilon) = \epsilon\) (i.e., nothing
is observed) and, given $\sigma \in \Sigma$,$$
P(\sigma) = \begin{cases} 
\sigma & \text{if } \sigma \in \Sigma_o 
\varepsilon & \text{otherwise (i.e., nothing is observed)}.
\end{cases}
$$

Given $s\sigma \in \mathcal{L}(G)$ where $\sigma \in \Sigma$, $P(s\sigma) = P(s)P(\sigma)$. Given $s \in \mathcal{L}(G)$, the agent observes $P(s)$.

Given DFA $G$, event set $\Sigma_o$, that the agent observes $\mathcal{L}(G)$ using its natural projection $P$ and that the agent knows or has provided to it the automaton model of $G$, one can construct the agent’s observer DFA which generates $P(\mathcal{L}(G)) = \{P(s) : s \in \mathcal{L}(G)\}$. For any string $s \in \mathcal{L}(G)$, the state of the agent’s observer DFA reached by $P(s)$ represents the subset of states in $X$ that are reached by strings in the set $
\{s' \in \mathcal{L}(G) : P(s') = P(s)\}.$ That is, the state provides a precise “state estimate” of the state that $G$ is in given that the agent observes all occurrences of events in $\Sigma_o$. The observer DFA is constructed by replacing events in $\Sigma \setminus \Sigma_o$ labeling transitions in $G$ by $\varepsilon$ then applying the subset construction described previously.

In Chapters 3, 4 and 5 we consider generalizations of the natural projection for modeling an agent’s observation of a DES. Specifically, we consider that an agent’s observations may be defined instead using the transitions of a given automaton representation of a DES. However, even in this more general case, the construction of the observer DFA can still be used for providing a precise estimate of the current state of the DES.

The observer DFA is used to provide state estimates in the case that the agent only has bounded memory for storing estimates of $G$. If the agent has unbounded memory then it can store the entire sequence of events it observes. Let $s \in \Sigma_o^*$ denote such a sequence. From $s$ one may define the set of strings $
\{s' \in \mathcal{L}(G) : P(s') = s\}.$
This set is referred to as the agent’s “string estimate” when the agent has observed \( s \). String estimates provide a more precise estimate of the system behaviour than state estimates. However, string estimates require substantially more memory to maintain. We consider string estimates of a DES in Chapter 6.

Given an agent’s set of observable events \( \Sigma_o \), we say that a pair \( (s, s') \in \mathcal{L}(G) \times \mathcal{L}(G) \) (resp., \( (s', s) \)) is indistinguishable (also confusable) to the agent if \( P(s) = P(s') \). Note that, for strings \( s, s', s'' \in \mathcal{L}(G) \), if \( (s, s') \) is indistinguishable and \( (s, s'') \) is indistinguishable then \( (s', s'') \) is indistinguishable (i.e., indistinguishability of strings is transitive). We say that a pair of states \( (x, y) \in X \times X \) (resp., \( (y, x) \)) is indistinguishable (also confusable) to the agent if \( x \) and \( y \) are reached by strings which are indistinguishable. That is, there exist strings \( s, s' \in \mathcal{L}(G) \) such that \( P(s) = P(s') \), \( \delta(x_0, s) = x \) and \( \delta(x_0, s') = y \). Equivalently, a pair \( (x, y) \) is indistinguishable if the observer DFA constructed from \( G \) and \( \Sigma_o \) has a state \( x' \) which contains both \( x \) and \( y \). Note that if \( (x, y) \) is indistinguishable and \( (y, z) \) is indistinguishable then it is not necessarily the case that \( (x, z) \) is indistinguishable (i.e., indistinguishability of states is not necessarily transitive). We typically denote the set of indistinguishable state pairs of \( G \) by \( \Pi \) for a given \( G \) and \( \Sigma_o \), but sometimes we refer to this set as \( T_{conf} \).

Formally, when concerning the projection \( P \),

\[
\Pi = \{(x, y) \in X \times X : \exists s, s' \in \mathcal{L}(G), x \in \delta(x_0, s), y \in \delta(x_0, s'), P(s) = P(s')\}.
\]

Given \( G, \Sigma_o \), set \( \Pi \) can be computed using the CLUSTER-TABLE algorithm of Wang et al. [50] or by the product \( \otimes \) studied in Chapter 5.

Natural projections result in what we referred to in the introduction as static observation. That is, for any given event, every single occurrence of the event is either observed by the agent, or is not observed by the agent. We generalize natural
projections to observation maps where this property does not necessarily hold. We consider that observation maps may be defined over the states of $G$, transitions of $G$ or strings of $\mathcal{L}(G)$ in this dissertation. Using such observation maps, the observation of any two occurrences of an event $\sigma \in \Sigma_o$ may differ. That is, following some string $s \in \mathcal{L}(G)$ an occurrence of event $\sigma$ may be observed but following some other string $s' \in \mathcal{L}(G)$ an occurrence of $\sigma$ may not be observed by an agent. We refer to such observation maps as dynamic observation maps.

In general, an agent’s (dynamic) observation map is a map $\omega$ of the form $\omega : \mathcal{L}(G) \to 2^{\Sigma_o}$. Observation map $\omega$ specifies which events would be observed by the agent upon their occurrence following a given string in $\mathcal{L}(G)$. Given an agent’s observation map $\omega$, we can define the agent’s corresponding information map $\theta : \mathcal{L}(G) \to \Sigma_o^*$. Information map $\theta$ specifies how an agent observes a given string in $\mathcal{L}(G)$ given the agent’s observation map $\omega$. Map $\theta$ is analogous to the natural projection $P$ but where observation of an event occurrence is determined by $\omega$. Information map $\theta$ is defined inductively using $\omega$ as follows. For the empty string $\varepsilon$, $\theta(\varepsilon) = \varepsilon$, and for all $s\sigma \in \mathcal{L}(G)$ with $\sigma \in \Sigma$

$$\theta(s\sigma) = \begin{cases} 
\theta(s)\sigma & \text{if } \sigma \in \omega(s) \\
\theta(s) & \text{otherwise.}
\end{cases}$$

Note that the natural projection $P$ is a specific case of an information map $\theta^{all}$ corresponding to observation map $\omega^{all}$ defined as follows. For all $s \in \mathcal{L}(G)$, $\omega^{all}(s) = \Sigma_o$.

We note that all concepts introduced here and throughout this dissertation that concern natural projections also apply to the more general (dynamic) observation maps and associated information maps introduced above.
We recall computational complexity classes from the computer science literature that are referenced in this dissertation. The Turing machine model of computation is widely recognized as a universal model of computation, and is commonly used to characterize the computational complexity of decision problems. We provide a characterization of complexity classes studied in this dissertation in terms of the Turing machine model.

An informal description of the Turing machine model is provided next. A Turing machine has a tape, which consists of individual cells for storing symbols. The tape is of arbitrary length. The input to a Turing machine is encoded as a sequence of symbols, and is initially written on the Turing machine’s tape. The Turing machine uses a (read-write) tape head for manipulating the tape. The tape head is only ever positioned over one cell of the tape at any point in time, but can be moved backward and forward by the Turing machine. The tape head is used for reading the symbol of the cell it is positioned over, and also for writing to that cell. Initially, the tape head is positioned on the start of the input.

Similar to finite-state automata, the Turing machine also has a finite set of states, and a transition function defined over these states. Like finite-state automata, the Turing machine has an initial state that it starts in, as well as a set of accepting states. Transitions are defined over states as well as the current cell that the tape head is located over. Specifically, the input to each transition is the symbol that is currently located under the tape head, as well as the current state of the machine. The output of the transition is the state that results from the transition, as well as the action to be taken by the tape head. For the action to be taken, the tape head can optionally write a symbol to the tape, then move backward or forward one cell
on the tape. When the transition function is deterministic (resp., non-deterministic), we refer to the Turing machine as a deterministic (resp., non-deterministic) Turing machine.

We note that the set of symbols that a Turing machine writes onto its tape may be different than the set of symbols used to encode the input. In general, there is no restriction made on the set of symbols that can be written on the tape. Also, there is no restriction as to how much tape the Turing machine uses. That is, the Turing machine is free to use any number of empty cells after the input on the tape in any way that it pleases for conducting its work.

A Turing machine is said to accept its input if and only if an accepting state is eventually reached, and no further transitions are made. Otherwise, the Turing machine is said to reject its input.

We can characterize the time that a Turing machine takes in processing its input before it halts. Time is characterized by the number of transitions made by the Turing machine. We also refer to this number as the number of steps the Turing machine takes. The time demand of a Turing machine $T$ is a function $\text{time}_T(n)$ defined as the maximum of the number of steps taken by $T$ over all possible inputs of length $n$. It is assumed that a Turing machine must read all of its input before halting with accept or reject. Thus $\text{time}_T(n) \geq n$.

We can also characterize the space that a Turing machine consumes. Space is characterized by the number of distinct tape cells written to by the Turing machine. It is assumed the Turing machine does not overwrite its input, so the length of the input does not contribute to this number. The space demand of a Turing machine $T$ is a function $\text{space}_T(n)$ defined as the maximum number of tape cells used by $T$ over
all possible inputs of length $n$. It is assumed that $space_T(n) \geq 1$, i.e. $T$ writes some output to the tape before it halts.

The time and space used by a Turing machine is used to characterize the time and space of decision problems. A decision problem is a problem whose answer is yes or no. It is well-known that any decision problem can be formalized as the problem of determining whether or not a given string belongs to a formal language. Turing machines then can be used to decide the correct yes or no answer for a decision problem.

We say that a Turing machine $T$ is in time $f(n)$ (resp., space $f(n)$) if and only if $time_T(n) \in O(f(n))$ (resp., $space_T(n) \in O(f(n))$). We say that a decision problem has deterministic time complexity (resp., deterministic space complexity) at most $f(n)$ if a deterministic Turing machine $T$ exists that can compute the correct answer to the problem in time $f(n)$ (resp., space $f(n)$). Non-deterministic time and space complexities of decision problems can be defined analogously in terms of non-deterministic Turing machines.

A decision problem is in (deterministic) polynomial-time if it has deterministic time complexity at most $f(n)$ where $f(n)$ is polynomial. One can classify a problem as being in (deterministic) exponential-time or in (deterministic) polynomial-space analogously. Also, a decision problem is in non-deterministic polynomial-time if it has non-deterministic time complexity at most $f(n)$ where $f(n)$ is polynomial. One can classify a problem as being in non-deterministic polynomial-space analogously. We consider decision problems throughout this dissertation that are in one of the following:

- (deterministic) polynomial-time;
• non-deterministic polynomial-time;
• (deterministic) exponential-time;
• (deterministic) polynomial-space;
• non-deterministic polynomial-space.

The set of decision problems solvable in polynomial-time (resp., exponential-time, non-deterministic polynomial-time, polynomial-space, non-deterministic polynomial-space) is commonly denoted by complexity class $P$ (resp., $EXPTIME$, $NP$, $PSPACE$, $NPSPACE$). When regarding time, one of the popular open problems in computer science is deciding whether or not $P = NP$. When regarding space, we have $PSPACE = NPSPACE$ by Savitch’s Theorem [27].

The above complexity classes have been studied extensively in the computer science literature. The following relations between the above complexity classes are well-known:

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$$

$$P \subset EXPTIME.$$ 

Within each complexity class there exist problems which are, in a sense, the most difficult problems for that complexity class. These problems are said to be complete for their respective complexity class. Two types of these problems are referenced throughout the dissertation: $NP$-complete problems, and $PSPACE$-complete problems. We describe these problems next.

Completeness requires the notion of a reduction from one problem to another. Informally, a (many-to-one) reduction from a decision problem $A$ to a decision problem $B$ is a function $r$ for transforming input $i$ of problem $A$ into $r(i)$, an input to
problem $B$, such that the answer for problem $B$ on $r(i)$ is equivalent to the answer for problem $A$ on $i$. A polynomial-time reduction is a reduction that can be computed in polynomial-time. Problem $A$ is $NP$-complete if and only if

- $A \in NP$;

- for all $B \in NP$, there exists a polynomial-time reduction from $B$ to $A$.

Problem $A$ is $PSPACE$-complete if and only if

- $A \in PSPACE$;

- for all $B \in PSPACE$, there exists a polynomial-time reduction from $B$ to $A$. 
Chapter 3

Literature Review – Minimal Sensor Activation and Minimal Communication in Discrete-Event Systems

Note that this chapter is derived from [32].

This chapter is an overview of past research on sensor activation and communication in discrete-event systems which has inspired work in this dissertation. In Section 3.1 we provide a set of preliminary concepts necessary for reviewing the works in the literature review. In Section 3.2 we review approaches for minimal sensor activation for purposes of state disambiguation and state estimation of DES. In Section 3.3 we review approaches for minimal communication for purposes of state disambiguation and state estimation of DES.
3.1 Preliminaries

Here some preliminary concepts and notation specific to this chapter are introduced. We refer the reader to Chapter 2 for a summary of concepts and notation used in this chapter and other chapters of the dissertation.

Recall that an NFA, $G = (X, \Sigma, \delta, x_0)$ where $X$ is the nonempty set of states, $\Sigma$ is the alphabet, $x_0$ is the initial state, and $\delta : X \times (\Sigma \cup \{\varepsilon\}) \to 2^X$ is the (state-)transition function. In this chapter, we sometimes use $p \rightarrow^\sigma q$ to denote that $q \in \delta(p, \sigma)$ for $p,q \in X$, $\sigma \in \Sigma$.

Given two DFAs, $A_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}), A_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2})$, the synchronous product (also known as parallel composition) of $A_1$ and $A_2$, denoted by DFA $A_1||A_2$, simulates the parallel execution of $A_1$ and $A_2$ where synchronization between transitions of the two automata is enforced on the occurrence of events in $\Sigma_1 \cap \Sigma_2$. Specifically, it is defined as $A_1||A_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{0,1}, q_{0,2}))$ where $\delta : (Q_1 \times Q_2) \times (\Sigma_1 \cup \Sigma_2) \to (Q_1 \times Q_2)$ is defined as follows.

Given $q_1 \in Q_1, q_2 \in Q_2$ and $\sigma \in \Sigma_1 \cup \Sigma_2$,

$$\delta((q_1, q_2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), q_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \land \delta_1(q_1, \sigma)! \\
(q_1, \delta_2(q_2, \sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \land \delta_2(q_2, \sigma)! \\
(\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \land \delta_1(q_1, \sigma)! \land \delta_2(q_2, \sigma)! \\
\text{undefined} & \text{otherwise.} \end{cases}$$

We denote the DES (also called plant) by $G$, modeled by an NFA $G = (X, \Sigma, \delta, x_0)$. Sometimes, in this chapter, we consider that a DES NFA may contain multiple initial states. That is, $G$ is of the form $G = (X, \Sigma, \delta, X_0)$ where $X_0 \subseteq X$. However, for most problems considered, $G$ is modeled as a DFA.
In case multiple agents observe $G$, we denote the set of agents in this chapter by $I$. An agent $i \in I$ has observable events in $\Sigma_{i,o} \subseteq \Sigma$ and unobservable events in $\Sigma_{i,uo} = \Sigma \setminus \Sigma_{i,o}$. Agent $i$’s observation of strings generated by $G$ may be modeled by the natural projection $P_i : \Sigma^* \rightarrow \Sigma^*_{i,o}$. That is, for $s \in \mathcal{L}(G)$, agent $i$ observes $P_i(s)$. The natural projection is used sometimes in this chapter, though many of the results reviewed consider that an agent observes a DES using a more general observation map.

Some work in DES has considered problems of state disambiguation and state estimation. In problems of state disambiguation, a set of pairs of states of $G$, denoted by $T_{spec} \subseteq X \times X$, is given. It is required that states in a pair of $T_{spec}$ be distinguished (i.e., not indistinguishable) by an agent. That is, it is required that $T_{conf} \cap T_{spec} = \emptyset$. In problems of state estimation, it is required that an agent can precisely determine the current state of $G$ or what the initial state of $G$ might have been after some bounded number of event occurrences. Typically, we can verify if a state estimation problem is solved by computing an agent’s observer DFA of $G$ and determine whether or not there exists a cycle of states where each state in the cycle contains more than one state of $G$.

We recall the general (dynamic) observation map, $\omega$, introduced in Chapter 2. In this chapter we do not consider that $\omega$ is defined arbitrarily. Specifically, $\omega$ corresponds to an agent’s sensor activation decisions in problems involving sensor activation or received communications from other agents in problems involving communication. The precise definition of $\omega$ depends on the sensor activation or communication problem at hand. We provide precise definitions in our subsequent review corresponding to the particular work being reviewed.
We note here that the following two assumptions are made in all works on sensor activation and communication.

1. An agent’s decision to do something (e.g., to turn a sensor on or communicate an event) is made on the observation of events. That is, an agent cannot change its decision to do something without having observed an event.

2. Upon observation of an event, an agent makes its next decision prior to the plant generating any more events.

### 3.2 Minimal Sensor Activation for Purposes of State Disambiguation and State Estimation

In this section we review approaches for computing minimal sensor activation maps for purposes of state disambiguation and state estimation in DES. Problems of state disambiguation involve designing sensor activation maps that result in agents being able to distinguish between certain pairs of states of a DES. Problems of state estimation involve designing sensor activation maps that result in agents being able to eventually determine precisely the current state of a plant or what the initial state of the plant may have been. We present approaches to these two types of problems together as problems of state estimation may be reduced to problems of state disambiguation. The problems reviewed consider that optimality of sensor activation maps is defined in a logical sense.
3.2.1 Minimization of dynamic sensor activation in discrete-event systems for the purpose of control

First, we review Wang et al. [54], which proposes an approach for minimizing the use of an agent’s event sensors for purposes of state disambiguation.

The problem in this paper considers the case where a single agent exists that observes a given DES. The objective is for the agent to reduce the use of the sensors it uses for observing events while still permitting the agent to distinguish between certain pairs of states of an automaton model of the DES.

The DES is given, and is modeled by a DFA $G = (X, \Sigma, \delta, x_0)$. Automaton $G$ generates a regular language, denoted by $L(G)$. A set $T_{spec}$, defined over the pairs of states of $G$, is given. That is, $T_{spec} \subseteq X \times X$. Set $T_{spec}$ is referred to as the state-based specification.

The problem considered in Wang et al. [54] is to design a sensor activation map for a single agent that permits the agent to distinguish between states $x$ and $y$ for all $(x, y) \in T_{spec}$. Next we review a language-based model of sensor activation maps and a transition-based model of sensor activation maps.

For the agent, when to activate event sensors is described by a sensor activation map $\omega: \mathcal{L}(G) \rightarrow 2^{\Sigma_o}$. Specifically, for a string $s \in \mathcal{L}(G)$, $\omega(s)$ is the subset of the events that are observable by the agent whose associated event sensors are active after $s$. Map $\omega$ is used to determine which occurrences of an event are observed by the agent. Map $\omega$ is used to define the corresponding information map $\theta^\omega$, which is used to determine how a given string appears to the agent. Given sensor activation map $\omega$, we use induction to define information map $\theta^\omega: \mathcal{L}(G) \rightarrow \Sigma_o$ as follows. For
the empty string $\varepsilon$, $\theta^\omega(\varepsilon) = \varepsilon$, and for all $s\sigma \in L(G)$ with $\sigma \in \Sigma$

$$\theta^\omega(s\sigma) = \begin{cases} 
\theta^\omega(s)\sigma & \text{if } \sigma \in \omega(s) \\
\theta^\omega(s) & \text{otherwise.}
\end{cases}$$

In words, after the occurrence of $s$, the next event $\sigma$ is seen or observed by the agent when it occurs after $s$ if and only if the agent’s sensor for $\sigma$ is active after the occurrence of $s$.

It is important to note that not all arbitrary sensor activation maps $\omega$ will be “feasible” based on the information available to the agent. To guarantee feasibility, it is required that any two strings of events that are confusable / indistinguishable to the agent must be followed by the same sensor activation decision for every event. Namely, $\omega$ must be “compatible” with the information map $\theta^\omega$ that is built from it. Formally, $\omega$ is said to be feasible if

$$\forall \sigma \in \Sigma, \forall s, s's' \in L(G), \theta^\omega(s) = \theta^\omega(s') \Rightarrow [\sigma \in \omega(s) \Leftrightarrow \sigma \in \omega(s')]. \quad (3.1)$$

Feasibility is used to capture the interdependence of the agent’s observation of the system and its sensor activation map. That is, in general, the determination of when to activate sensors depends on the “observation” of the system, and at the same time it also affects the “observation” of the system.

For example, let $L(G) = \{\varepsilon, a, aa\}$. Suppose $\omega(\varepsilon) = \emptyset$, $\omega(a) = \{a\}$. Then, $\theta^\omega(a) = \theta^\omega(\varepsilon) = \varepsilon$. That $\varepsilon$ and $a$ are both observed as $\varepsilon$ to the agent implies that, in order for the agent to unambiguously determine which sensors it should turn on in accordance with $\omega$, the decision to turn on the sensor for an event $\sigma$ following occurrence of either $\varepsilon$ or $a$ in $L(G)$ should be the same when $\sigma$ follows both $\varepsilon$ and $a$ (i.e., $\sigma,a\sigma \in L(G)$). However, event $a$ follows both $\varepsilon$ and $a$ (i.e., $a,aa \in L(G)$), $a \notin \omega(\varepsilon)$ and $a \in \omega(a)$. So the agent’s decision to turn on its sensor for $a$ when the
agent observes nothing is ambiguous. The map \( \omega \) of this example is one which does not satisfy feasibility (3.1). The following are the feasible (i.e., unambiguous) sensor activation maps for the \( \mathcal{L}(G) \) of this example:

- \( \omega(\varepsilon) = \omega(a) = \omega(aa) = \emptyset \),
- \( \omega(\varepsilon) = a, \ \omega(a) = \omega(aa) = \emptyset \),
- \( \omega(\varepsilon) = \omega(a) = a, \ \omega(aa) = \emptyset \),
- \( \omega(\varepsilon) = \omega(a) = \omega(aa) = a \).

Given feasible sensor activation maps \( \omega, \omega' \), we use \( \omega \subseteq \omega' \) to denote that \( \omega' \) activates at least as many sensors (in terms of set containment) as \( \omega \). Formally, \( \omega \subseteq \omega' \) if and only if for all \( s \in \mathcal{L}(G) \), \( \omega(s) \subseteq \omega'(s) \). Given \( \omega \), we let \( T_{\text{conf}} \) denote the set of pairs of strings in \( \mathcal{L}(G) \) that the agent cannot distinguish between when sensor activation map \( \omega \) is used to determine which sensors to activate and when. Formally, \( T_{\text{conf}}(\omega) = \{(s, s') \in \mathcal{L}(G) \times \mathcal{L}(G) : \theta^\omega(s) = \theta^\omega(s')\} \).

For implementation purposes, a restricted domain of sensor activation maps is considered, not the general domain of language-based sensor activation maps. The finite domain chosen is the class of sensor activation maps which may be defined over the transitions of \( G \). That is, the set of sensor activation maps \textit{implementable} with respect to \( G \) are considered. Formally, \( \omega \) is implementable (with respect to \( G \)) if

\[
\forall s, s' \in \mathcal{L}(G), \delta(x_0, s) = \delta(x_0, s') \Rightarrow \omega(s) = \omega(s').
\] (3.2)

Let \( TR(G) \) denote the set of transitions of \( G \). Specifically, \( TR(G) = \{(x, \sigma) \in X \times \Sigma : \delta(x, \sigma)\} \). For implementable sensor activation maps, we can associate the activation of sensors with the transitions of \( G \): the event associated with each transition in
$TR(G)$ is either observed by the agent (i.e., the event sensor is turned on), or not (i.e., the event sensor is turned off). Given an implementable sensor activation map $\omega$, the set of transitions in $TR(G)$ observed by the agent through sensor activations using $\omega$ is defined as $\{(x, \sigma) \in TR(G) : \exists s \sigma \in L(G), \delta(x_0, s) = x \land \sigma \in \omega(s)\}$. This set is denoted by $\Omega \subseteq TR(G)$. Here $(x, \sigma) \in \Omega$ means that for all $s \in L(G)$, $\delta(x_0, s) = x \Rightarrow \sigma \in \omega(s)$. We call $\Omega$ a (transition-based sensor activation) policy.

For example, consider $L(\hat{G}) = \{\varepsilon, a, b, ab, ba, bab\}$. Consider sensor activation map $\hat{\omega}(\varepsilon) = \hat{\omega}(a) = \{b\}, \hat{\omega}(b) = \hat{\omega}(ba) = \hat{\omega}(ab) = \hat{\omega}(bab) = \emptyset$. Map $\hat{\omega}$ is a sensor activation map defined over domain $L(\hat{G})$. Two automata generators for $L(\hat{G})$ are provided in Figures 3.1 and 3.2. Map $\hat{\omega}$ is not implementable with respect to the generator of Figure 3.1 since strings $a$ and $ba$ both lead to the same state of the generator yet $\hat{\omega}(a) \neq \hat{\omega}(ba)$. However, $\hat{\omega}$ is implementable with respect to the generator of Figure 3.2. That is, for any two strings $s, s' \in L(\hat{G})$ leading to the same state of this generator, $\hat{\omega}(s) = \hat{\omega}(s')$. The policy $\hat{\Omega}$ corresponding to $\hat{\omega}$ can be defined over the transitions of the generator. Specifically, $\hat{\Omega} = \{(0, b), (3, b)\}$ and is represented by the set of transitions whose event labels are boxed in Figure 3.2.

---

**Figure 3.1:** Automaton generator of $L(\hat{G})$ for which sensor activation map $\hat{\omega}$ is not implementable.

**Figure 3.2:** Automaton generator of $L(\hat{G})$ for which sensor activation map $\hat{\omega}$ is implementable.
Given policy $\Omega$, we can obtain its corresponding sensor activation map $\omega$ as follows.

$$\omega(s) = \{\sigma \in \Sigma_o : (\delta(x_0, s), \sigma) \in \Omega\}. \quad (3.3)$$

Information map $\theta^\Omega$ is used to denote the information map $\theta^\omega$ when $\omega$ is derived from $\Omega$ by (3.3).

Policy $\Omega$ is feasible if the corresponding $\omega$ is feasible. Formally, $\Omega$ is feasible if:

$$\forall \sigma \in \Sigma, \forall s, s' \in \mathcal{L}(G), \theta^\Omega(s) = \theta^\Omega(s') \Rightarrow [(\delta(x_0, s), \sigma) \in \Omega \iff (\delta(x_0, s'), \sigma) \in \Omega]. \quad (3.4)$$

The set of confusable / indistinguishable state pairs of $G$ under $\Omega$, denoted by $\mathcal{T}_{conf}(\Omega)$, is defined as

$$\mathcal{T}_{conf}(\Omega) = \{(x, y) \in X \times X : \exists s, s' \in \mathcal{L}(G), \theta^\Omega(s) = \theta^\Omega(s'), \delta(x_0, s) = x, \delta(x_0, s') = y\}. \quad (3.5)$$

For example, for $G$ of Figure 3.2 and policy $\hat{\Omega} = \{(0, b), (3, b)\}$,

$$\mathcal{T}_{conf}(\hat{\Omega}) = \{(0, 3), (3, 0), (1, 4), (4, 1), (2, 4), (4, 2), (1, 2), (2, 1), (0, 0), (1, 1), \ldots\}.$$

Feasibility can be interpreted thus in terms of sensor activation policies. If $(x, y) \in \mathcal{T}_{conf}(\Omega)$ then, for all $\sigma \in \Sigma_o$, if $(x, \sigma), (y, \sigma) \in TR(G)$ then $(x, \sigma) \in \Omega \iff (y, \sigma) \in \Omega$.

It is the objective of the agent to minimize the use of its sensors while making sure that states in pairs of $T_{spec}$ do not become indistinguishable. When considering policies, the objective becomes to compute a policy $\Omega$ where the use of sensors is minimized (to be made precise) while ensuring that the specification $T_{spec}$ is satisfied. That is, $\mathcal{T}_{conf}(\Omega) \cap T_{spec} = \emptyset$. Formally, the following problem is considered.

**Problem 1.** Assume that if the agent activates all observable event sensors all the time then the specification $T_{spec}$ is satisfied.

Find a sensor activation policy $\Omega^* \subseteq TR(G)$ such that
C1 $\Omega^*$ is feasible;

C2 $T_{\text{spec}}$ is satisfied under $\Omega^*$ (i.e., $T_{\text{conf}}(\Omega^*) \cap T_{\text{spec}} = \emptyset$);

C3 $\Omega^*$ is minimal with respect to C1 - C2. That is, there does not exist an $\Omega' \subset \Omega^*$ such that $\Omega'$ satisfies C1 and C2.

Note that there may not exist a single unique solution to Problem 1. Multiple, incomparable solutions may exist.

Some important results regarding feasible sensor activation maps and policies are recalled.

Feasible sensor activation maps are monotonic in the sense that as more event sensors are turned on, more pairs of strings may be distinguished. That is, for feasible sensor activation maps, as more event sensors are activated fewer pairs of strings remain indistinguishable. Effectively, the more an agent sees, the more it knows. Formally, given feasible $\omega, \omega'$, if $\omega \subseteq \omega'$ then $T_{\text{conf}}(\omega) \supseteq T_{\text{conf}}(\omega')$ (Theorem 1 of Wang et al. [54]). This result also holds for feasible sensor activation policies (Corollary 1 of Wang et al. [54]). We note that these monotonicity results enable the solution approach to be reviewed next.

By Wang et al. [54], it is proven that feasible sensor activation maps are closed under union (Theorem 2 of Wang et al. [54]). That is, given feasible $\omega, \omega'$, for all $s \in \mathcal{L}(G)$ let $\omega''(s) = \omega(s) \cup \omega'(s)$. Then $\omega''$ is feasible. This result also holds for feasible policies (Theorem 3 of Wang et al. [54]). Thus, for a given policy $\Omega$, there exists a maximum feasible sub-policy $\Omega^{\uparrow F} \subseteq \Omega$ that contains all other feasible policies $\Omega' \subseteq \Omega$.

That the maximum feasible sub-policy of a given policy exists, and permits the
agent to distinguish more than if other feasible sub-policies are used (by the monotonicity result mentioned earlier), suggests an approach for computing solutions to Problem 1. An algorithm Max-Fea-Sub is provided for computing the maximum feasible sub-policy $\Omega^{\uparrow F}$ of a given policy $\Omega$ and its associated set of indistinguishable / confusable state pairs $T_{conf}(\Omega^{\uparrow F})$ [54]. The algorithm is described as follows.

1. Set $\hat{\Omega}$ to $\Omega$.
2. Compute the set $T_{conf}(\hat{\Omega})$ (see steps 1 - 2 of Wang et al. [54] Algorithm Max-Fea-Sub for details).
3. Remove all $(x, \sigma)$ from $\hat{\Omega}$ such that there exists pair $(x, y) \in T_{conf}(\hat{\Omega})$ where $(x, \sigma) \in \hat{\Omega}$ and $(y, \sigma) \notin \hat{\Omega}$.
4. Repeat steps 2 - 3 until there are no further changes to $\hat{\Omega}$.
5. Return $\Omega^{\uparrow F} = \hat{\Omega}$. Also return $T_{conf}(\Omega^{\uparrow F})$.

The correctness of Max-Fea-Sub follows from Theorem 4 of Wang et al. [54]. Algorithm Max-Fea-Sub is in $O(|X|^2 \cdot |\Sigma|)$ (Theorem 4 of Wang et al. [54]).

Algorithm Min-Sen-Act [54] is proposed for computing solutions to Problem 1. The algorithm is described as follows for a given $G$, $\Sigma_o$ and $T_{spec}$.

1. The initial sensor activation policy, $\Omega$, is set to $\Omega^{all} = \{(x, \sigma) \in TR(G) : \sigma \in \Sigma_o\}$ where all transitions in $G$ labelled with an observable event are observed. Let $D = \emptyset$. Set $D$ denotes the set of transitions in $\Omega$ which, when removed from $\Omega$, result in $T_{spec}$ not being satisfied.
2. Choose a transition $(x, \sigma) \in \Omega \setminus D$ (in a prespecified or arbitrary manner).
Invoke Max-Fea-Sub on $\Omega \setminus \{(x, \sigma)\}$ to compute $(\Omega \setminus \{(x, \sigma)\})^{\uparrow F}$ and $T_{conf}(\Omega \setminus \{(x, \sigma)\})^{\uparrow F}$.

3 If $T_{conf}(\Omega \setminus \{(x, \sigma)\})^{\uparrow F} \cap T_{spec} = \emptyset$ then set $\Omega = (\Omega \setminus \{(x, \sigma)\})^{\uparrow F}$. Otherwise, by Corollary 1 of Wang et al. [54], no feasible sub-policy of $\Omega \setminus \{(x, \sigma)\}$ permits distinguishing between states of some pair in $T_{spec}$. Consequently, add $(x, \sigma)$ to $D$.

4 Repeat steps 2 - 3 until $\Omega = D$ (i.e., we cannot remove any more transitions without producing an indistinguishable state pair in $T_{spec}$).

5 Return $\Omega$.

Algorithm Min-Sen-Act computes a solution to Problem 1 (Theorem 5 of Wang et al. [54]). Algorithm Min-Sen-Act is in $O(|X|^3 \cdot |\Sigma|^2)$ (Theorem 7 of Wang et al. [54]).

Furthermore, all solutions to Problem 1 can be computed by Min-Sen-Act. This is done by choosing different orders in which to remove transitions from $\Omega$ in step 2 described above (Theorem 6 of Wang et al. [54]). By refining the state-transition structure of $G$, finer sensor activation policies may be computed. See Section IV.B of Wang et al. [54] for a detailed example demonstrating computation of solutions to Problem 1.

Feasible policy $\Omega$ does not itself yield a map from observed event sequences to sensor activation decisions consistent with $\Omega$ (i.e., a dynamic observer in the sense of Cassez et al. [2]). From $G$ and $\Omega$ one can compute such an observer which also returns the current state estimate of $G$. This observer may be computed by first replacing the labels of all transitions of $G$ not in $\Omega$ (i.e., transitions in $TR(G) \setminus \Omega$) by the empty
string $\varepsilon$. Denote by $G_{\Omega}$. Second, apply the subset construction to $G_{\Omega}$. Denote by $O(G_{\Omega})$. This procedure is similar to the standard observer automaton construction. The current state $X'$ of $O(G_{\Omega})$ following an observed event sequence is the current state estimate of $G$. The determination of which sensors to activate is determined by the current state $X'$. Specifically, the event sensors to be activated correspond to those events labeling transitions leaving $X'$ in $O(G_{\Omega})$. If $\Omega$ is a solution to Problem 1 then no state in $O(G_{\Omega})$ will contain a pair of states in $T_{spec}$.

We note that, as determinization of NFA is required, constructing such an observer is in the worst-case exponential in $X$, the state set of $G$. Recent work \cite{29, 28, 34} has proposed conditions stronger than feasibility of sensor activation policies. When such conditions are satisfied by a policy $\Omega$, a DFA mapping observed event sequences to sensor activation decisions consistent with $\Omega$ can be computed which has state set cardinality no larger than $X$. This property holds even when computation of the observer from $G$ and $\Omega$ is exponential. However, this savings is achieved at the expense of the DFA not returning a precise state estimate of $G$. These results are also presented in Chapter 4.

Other conditions could be considered for achieving polynomial-time computation of an observer for a feasible policy $\Omega$. For instance, one could consider computation of a feasible policy $\Omega$ such that $\Omega$ satisfies the observer property \cite{55}. More generally, one could consider computation of a policy $\Omega$ such that $G_{\Omega}$ is a state-partition automaton \cite{11}. 
3.2.2 An online algorithm for minimal sensor activation in discrete event systems

The approach of Wang et al. [54] computes minimal transition-based sensor activation policies offline for purposes of satisfying a specification $T_{spec}$. In Wang et al. [53], computation of sensor activation decisions online following an observed event occurrence is considered for satisfying $T_{spec}$. A one-step lookahead algorithm is employed for computing such decisions where the only information that the agent needs to maintain to define its sensor activation decisions is its estimate of the current state of $G$.

An online approach to sensor activation had been considered previously in Thorsley et al. [45]. However, in Thorsley et al. [45], variable cost may be incurred when sensors are activated for different events. In Wang et al. [53] the cost incurred for activating a sensor is the same for all events as considered in Wang et al. [54].

The approach of Wang et al. [53] differs from their later work [54] in that sensor activation decisions are not assumed to be restricted to the transitions of $G$. Instead, sensor activation decisions are computed according to the current estimate of the state of $G$. We define such sensor activation policies next.

Given $X' \subseteq X$, $\Sigma_{ua} \subseteq \Sigma$, let $UR(X', \Sigma_{ua})$ denote the unobserved reach of $X'$ under unobserved event set $\Sigma_{ua}$. Formally, $UR(X', \Sigma_{ua}) = \{ x \in X : \exists x' \in X', \exists s \in \Sigma_{ua}, \delta(x', s) = x \}$.

Given $X' \subseteq X$, $\sigma \in \Sigma_o$, let $OR(X', \sigma)$ denote the observed reach of $X'$ under observed event $\sigma$. Formally, $OR(X', \sigma) = \{ x \in X : \exists x' \in X', \delta(x', \sigma) = x \}$.

Observations are defined by a (state-estimate-based sensor activation) policy $\Omega : 2^X \rightarrow 2^{\Sigma_o}$. We define $SE : \mathcal{L}(G) \rightarrow 2^X$, a map from strings of $\mathcal{L}(G)$ to the agent’s
base state estimate when sensor activation policy \( \Omega \) is used. Given \( \Omega \) and \( X' \in 2^X \), let \( UR(X') \) denote \( UR(X', \Sigma \setminus \Omega(X')) \). Formally, for the empty string \( \varepsilon \), \( SE(\varepsilon) = \{x_0\} \), and for all \( s\sigma \in L(G) \) with \( \sigma \in \Sigma \)

\[
SE(s\sigma) = \begin{cases} 
OR(UR(SE(s)), \sigma) & \text{if } \sigma \in \Omega(SE(s)) \\
SE(s) & \text{otherwise.}
\end{cases}
\]

Note that \( SE \) provides the base state estimate of the current state of \( G \) which the agent uses for deciding which sensors to turn on or off. It does not provide the actual state estimate of \( G \). The agent’s actual state estimate when string \( s \in L(G) \) is generated by \( G \) is more precisely \( UR(SE(s)) \).

Given \( SE \) we may define the agent’s information map \( \theta^\Omega : L(G) \to \Sigma^*_o \) corresponding to policy \( \Omega \) as follows. Formally, for the empty string \( \varepsilon \), \( \theta^\Omega(\varepsilon) = \varepsilon \), and for all \( s\sigma \in L(G) \) with \( \sigma \in \Sigma \)

\[
\theta^\Omega(s\sigma) = \begin{cases} 
\theta^\Omega(s)\sigma & \text{if } \sigma \in \Omega(SE(s)) \\
\theta^\Omega(s) & \text{otherwise.}
\end{cases}
\]

One can verify that the state-estimate-based sensor activation policies defined above are compatible with their information maps. That is, feasibility of the sensor activation policies considered here is satisfied. Specifically, given \( \Omega : 2^X \to 2^{\Sigma_o} \), given \( s, s' \in L(G) \), \( \theta^\Omega(s) = \theta^\Omega(s') \) implies \( \Omega(SE(s)) = \Omega(SE(s')) \) (i.e., \( SE(s) = SE(s') \)).

Given policies \( \Omega, \Omega' \), \( \Omega \subseteq \Omega' \) if and only if for all \( X' \in 2^X \), \( \Omega(X') \subseteq \Omega'(X') \). We can define the meaning for \( \Omega \subset \Omega' \) similarly.

Given policy \( \Omega \), we say that \( \Omega \) satisfies \( T_{\text{spec}} \) if, for all \( s \in L(G) \), \( (UR(SE(s)) \times UR(SE(s))) \cap T_{\text{spec}} = \emptyset \). That is, a state estimate is never produced using \( \Omega \) that contains two states in a pair of \( T_{\text{spec}} \).

The problem considered in Wang et al. [53] is equivalent to the following.
Problem 2. Suppose that if event sensors are activated for all observable events all the time then $T_{\text{spec}}$ is satisfied. That is, for all $s, s' \in \mathcal{L}(G)$, if $P(s) = P(s')$ then $(\delta(x_0, s), \delta(x_0, s')) \notin T_{\text{spec}}$ where $P$ is the natural projection from $\Sigma^*$ to $\Sigma_o^*$.

Implement online a (state-estimate-based sensor activation) policy $\Omega^* : 2^X \rightarrow 2^{\Sigma_o}$ such that

$C1$ $T_{\text{spec}}$ is satisfied under $\Omega^*$;

$C2$ $\Omega^*$ is minimal with respect to $C1$. That is, there does not exist an $\Omega' \subset \Omega^*$ such that $\Omega'$ satisfies $T_{\text{spec}}$.

The approach of Wang et al. [53] for computing solutions to Problem 2 is described next. Key to the approach of Wang et al. [53] is the use of the extended specification $T_{\text{spec}}^e$. The extended specification, $T_{\text{spec}}^e$ is defined from $T_{\text{spec}}$ as follows. Given state-based specification $T_{\text{spec}} \subseteq X \times X$, $T_{\text{spec}}^e = \{(x, x') : \exists s, s' \in \Sigma^*, P(s) = P(s'), (\delta(x, s), \delta(x', s')) \in T_{\text{spec}}\}$. Algorithm EXTEND-SPEC is proposed for computing $T_{\text{spec}}^e$ from $T_{\text{spec}}$ [53]. We also propose an alternative algorithm in Subsection 5.7.4 for computing the extended specification.

An important property of $T_{\text{spec}}^e$ is proposed. For all $(x, x') \notin T_{\text{spec}}^e$, for all $s, s' \in \Sigma^*$ if $P(s) = P(s') = \sigma \in \Sigma_o$ then $(\delta(x, s), \delta(x', s')) \notin T_{\text{spec}}^e$ (Lemma 1 of Wang et al. [53]). The implication of this is that if the agent’s current state estimate of $G$ does not contain pairs in $T_{\text{spec}}^e$ then, if all observable event sensors are activated, the state estimate produced after the next observed event occurrence contains no pairs in $T_{\text{spec}}^e$ (Theorem 2 of Wang et al. [53]).

This suggests a simple approach for computing sensor activation decisions online from a given state estimate. Turn off event sensors in $\Sigma_o$, one by one, until we are unable to turn off any remaining event sensor without producing a state estimate.
containing a pair in $T^e_{spec}$. By monotonicity of sensor activation maps (Theorem 1 of Wang et al. [54]), it is guaranteed that we cannot turn off any subset of the resulting set of activated sensors without producing a state pair in $T^e_{spec}$. Theorem 2 of Wang et al. [53] guarantees that, immediately after the next observed event occurrence, if we turn on all observable event sensors then the state estimate produced does not contain a pair in $T^e_{spec}$. We can then consider turning event sensors off, one by one as before while ensuring the state estimate does not contain a pair in $T^e_{spec}$, and so on.

By Theorem 2 of Wang et al. [53] and the fact that the estimate of the initial state of $G$ (i.e., $UR(SE(\varepsilon)))$ does not contain pairs in $T^e_{spec}$ when sensors for all events in $\Sigma_o$ are turned on (Theorem 3 of Wang et al. [53]), we are guaranteed that the described approach will work for reducing the use of event sensors while ensuring no pair in $T_{spec}$ is indistinguishable.

The described approach is formalized in Algorithm **Online-Min-Sen-Act** [53]. Assumed given is $G$, $T^e_{spec}$, $\Sigma_o$ and a total order $\leq$ over the events of $\Sigma_o$. The total order is used to specify the order in which the deactivation of event sensors is attempted.

The algorithm is described as follows.

1. Let $X_c$ denote the current base estimate (the actual state estimate of $G$ is $UR(X_c)$). Set $X_c = \{x_0\}$.

2. Set the activated sensor set $\Sigma_a$ to $\Sigma_o$. Set the deactivated sensor set $\Sigma_{ua}$ to $\Sigma \setminus \Sigma_o$.

3. For each event $\sigma \in \Sigma_o$ (selected using order $\leq$), do the following. Compute $UR(X_c, \Sigma_{ua} \cup \{\sigma\})$. If $UR(X_c, \Sigma_{ua} \cup \{\sigma\})$ contains no pairs in $T^e_{spec}$ then remove $\sigma$ from $\Sigma_a$ and add $\sigma$ to $\Sigma_{ua}$.
4 Activate (resp., deactivate) event sensors in \( \Sigma_a \) (resp., \( \Sigma_{ua} \)). On observation of an event \( \sigma \in \Sigma_a \), set \( X_c \) to \( OR(UR(X_c, \Sigma_{ua}), \sigma) \).

5 Repeat steps 2-4.

By Theorems 4 and 5 of Wang et al. [53], the sensor activation decisions computed by Online-Min-Sen-Act at any base estimate \( X_c \) is equivalent to \( \Omega^*(X_c) \) where \( \Omega^* \) is a state-estimate-based sensor activation policy satisfying C1 and C2 of Problem 2. That is, Online-Min-Sen-Act computes an online implementation of \( \Omega^* \) and hence a solution to Problem 2. The complexity of Online-Min-Sen-Act for computing the current sensor activation decision (i.e., steps 2 - 3 above) is polynomial in \(|X|\).

By considering different total orders on deactivating event sensors, different sensor activation decisions can be made online.

We provide the following example from Section V of Wang et al. [53] which illustrates how sensor activation decisions may be computed online using Online-Min-Sen-Act. Consider the plant \( G \) of Figure 3.3.

![Figure 3.3: Automaton used to demonstrate the approach of Wang et al. [53]](image)

![Figure 3.4: Automaton \( G_{rev} \) used to compute \( T^e_{spec} \) when \( \Sigma_o = \{a, c, f\} \).](image)

Suppose that \( \Sigma_o = \{a, c, f\} \) and that \( a > c > f \) in the total order. Suppose
$T_{\text{spec}} = \{(2, 6), (6, 2)\}$. One can verify that when all events in $\Sigma_o$ are observed all the time, state 2 is never indistinguishable from state 6. We can compute $T^e_{\text{spec}}$ by

1. replacing labels of transitions in $\Sigma \setminus \Sigma_o$ with $\varepsilon$ in Figure 3.3
2. reversing the transitions of $G$ resulting in $G_{\text{rev}}$ in Figure 3.4
3. computing the set of indistinguishable state pairs (e.g., by using the CLUSTER-TABLE algorithm of Wang et al. [50]) of $G_{\text{rev}}$ from each pair in $T_{\text{spec}}$.

For this example,

$T^e_{\text{spec}} = \{(2, 6), (6, 2), (4, 6), (2, 3), (0, 3), (4, 3), (1, 5), (1, 2), (1, 0), (1, 4), (5, 6), (5, 3)\}$

$\cup \{(6, 2), (6, 0), (6, 4), \ldots\}.$

Initially, $X_c = \{x_0\} = \{0\}$. Sets $\Sigma_a = \{a, c, f\}$ and $\Sigma_{ua} = \{b, d, e\}$. Using the total order, we see if the sensor for $a$ can be deactivated first. If the sensor for $a$ is deactivated, the current state estimate changes from $UR(X_c, \Sigma_{ua}) = UR(\{0\}, \{b, d, e\}) = \{0, 2, 5\}$ to $UR(\{0\}, \{a, b, d, e\}) = \{0, 2, 4, 5\}$. Since no pair of states of $T^e_{\text{spec}}$ is in $UR(X_c, \Sigma_{ua} \cup \{a\}) \times UR(X_c, \Sigma_{ua} \cup \{a\})$, it is safe to turn the sensor for $a$ off. That is, we add $a$ to $\Sigma_{ua}$ and remove $a$ from $\Sigma_a$. We then consider if the sensor for $c$ can be deactivated. Then the state estimate changes to $UR(X_c, \Sigma_{ua} \cup \{c\}) = UR(\{0\}, \{a, b, c, d, e\}) = \{0, 1, 2, 4, 5\}$. Since $1, 5 \in \{0, 1, 2, 4, 5\}$, $(1, 5) \in T^e_{\text{spec}}$, we cannot deactivate the sensor for event $c$. We then consider if the sensor for $f$ can be deactivated. Then the state estimate changes to $\{0, 1, 2, 4, 5\}$. We cannot deactivate the sensor for event $f$. Having considered all events in $\Sigma_o$, the agent decides to deactivate only the sensor for event $a$.

Suppose that the agent observes event $f$. Then its state estimate is $OR(UR(X_c, \Sigma_{ua}), f) = OR(\{0, 2, 4, 5\}, f) = \{1\}$. Set $X_c$ is set to $\{1\}$. When all event
sensors are turned on, the state estimate after having observed $f$ is $UR(X_c, \Sigma \setminus \Sigma_o) = \{1\}$, which we observe contains no two states of a pair in $T_{\text{spec}}^e$. Using the total order on events, one can verify that, after having considered all events in $\Sigma_o$, $\Sigma_a = \{c\}$. Using a different total order, one can verify that $\Sigma_a = \{a\}$ is also a valid choice.

### 3.2.3 Online sensor activation for detectability of discrete event systems

In Shu et al. [36] an online approach to computing sensor activation decisions for state disambiguation is considered. The online approach takes inspiration from the online approach of Wang et al. [53] as well as the problems of detectability Shu & Lin [37] which we consider later. However, the problem of Shu et al. [36] differs in that sensors must be turned on or off in order that states in pairs of a state-based specification $T_{\text{spec}}$ must eventually be distinguished. The problem of Wang et al. [53] require that states in pairs of $T_{\text{spec}}$ are always distinguished.

The plant automaton is $G = (X, \Sigma, \delta, X_0)$ where $X_0 \subseteq X$ denotes the set of possible initial states of $G$. It is assumed that $G$ is deadlock-free, i.e., a transition is defined from any state in $X$. This is to ensure that any string in $\mathcal{L}(G)$ can be extended to a string of infinite-length. It is assumed that there are no cycles of unobservable events in $G$. Such cycles would guarantee that there exists strings of infinite-length in $\mathcal{L}(G)$ whose occurrence may not enable the agent to distinguish between certain pairs of states of $G$.

Similar to [53], state-estimate-based sensor activation policies are considered. Observations are defined by a (state-estimate-based sensor activation) policy $\Omega : 2^X \rightarrow$
2^{\Sigma_o}. We refer the reader to the background provided in Wang et al. \[53\] for definitions of unobserved reach $UR$, observed reach $OR$, base state estimate function $SE$, information map $\theta^\Omega$ defined from state-estimate-based policy $\Omega$, and containment relations $\subseteq$, $\subset$ for state-estimate-based policies. The only difference here is in the definition of $SE$, where it is required that $SE(\varepsilon) = X_0$ since we are considering that $G$ has multiple initial states rather than a single initial state $x_0$.

We can define the state estimate of $G$ after sequence $t \in \Sigma_o^\ast$ has been observed by

$$R(t) = \{\delta(x_0, s) : x_0 \in X_0, s \in \Sigma^\ast, \theta^\Omega(s) = t \land \delta(x_0, s)!\}.$$ 

Let $L^\omega(G)$ denote the set of infinite-length strings in $L(G)$ \[44\]. Automaton $G$ is strongly D-detectable under $\Omega$ and $T_{\text{spec}}$ if we can distinguish pairs of states in $T_{\text{spec}}$ after a finite number of observed event occurrences for all infinite-length strings of $L(G)$. Formally,

$$\exists n \in \mathbb{N}, \forall s \in L^\omega(G), \forall t \in \bar{\theta}^\Omega(s), |t| > n \Rightarrow (R(t) \times R(t)) \cap T_{\text{spec}} = \emptyset.$$ 

If strong D-detectability holds under a fixed $k \in \mathbb{N}$ then we say more specifically that $G$ is $k$-step distinguishable under $\Omega$ and $T_{\text{spec}}$.

For all $X' \in 2^X$, let $\Omega_{\text{all}}(X') = \Sigma_o$. Policy $\Omega_{\text{all}}$ is the policy where all observable event sensors are turned on all the time (i.e., $\Omega_{\text{all}}$ corresponds to the natural projection $P : \Sigma^\ast \rightarrow \Sigma_o^\ast$).

The problem of minimizing the use of event sensors for satisfying $k$-step distinguishability considered in Shu et al. \[36\] is equivalent to the following.

**Problem 3.** It is assumed that $G$ is strongly D-detectable under $\Omega_{\text{all}}$ and $T_{\text{spec}}$. Let $k \in \mathbb{N}$ be the smallest possible value such that $G$ is $k$-step distinguishable under $\Omega_{\text{all}}$ and $T_{\text{spec}}$. Find a (state-estimate-based sensor activation) policy $\Omega^* : 2^X \rightarrow 2^{\Sigma_o}$ such
CHAPTER 3. LITERATURE REVIEW

that

\( C1 \) \( G \) is \( k \)-step distinguishable under \( \Omega^* \);

\( C2 \) \( \Omega^* \) is minimal. That is, there does not exist an \( \Omega' \) such that \( C1 \) is satisfied under \( \Omega' \) and \( \Omega' \subset \Omega^* \).

When strong D-detectability holds under \( \Omega^{all} \), Algorithm 1 of Shu et al. \[36\] is proposed for computing the smallest possible \( k \) such that \( G \) is \( k \)-step distinguishable under \( \Omega^{all} \) and \( T_{spec} \). This algorithm is in polynomial time Shu et al. \[36\]. Strong D-detectability can be verified in polynomial time using the detector automaton \( G_{det} \) constructed from \( G \) and natural projection \( P \) (Theorem 11 of Shu et al. \[39\]).

To compute solutions to Problem 3 an approach is provided which takes inspiration from the approach of ONLINE-MIN-SEN-ACT of Wang et al. \[53\]. The approach involves deactivating as many sensors as possible while guaranteeing that, after \( k \) observed event occurrences, the resulting state estimate does not contain any pair in the extended specification \( T_{spec}^e \) defined in Wang et al. \[53\]. After \( k \) observed event occurrences one may then apply ONLINE-MIN-SEN-ACT of Wang et al. \[53\] from the resulting state estimate to determine which sensors should be turned on or off for the duration.

The approach is formalized as Algorithm 2 of Shu et al. \[36\]. It is described as follows. Suppose that \( j \) events have been observed where \( 0 \leq j < k \). Denote the current base state estimate as \( X_c \). If \( j = 0 \) then \( X_c = X_0 \). Denote the activated sensor set by \( \Sigma_a \) (set to \( \Sigma_a \) initially) and deactivated sensor set by \( \Sigma_{ua} \) (set to \( \Sigma \setminus \Sigma_a \) initially). In order to determine if the sensor for event \( \sigma \in \Sigma_a \) can be turned off,
compute the following sets of state estimates

\[ X_j^c = \{ \text{UR}(X_c, \Sigma_{ua} \cup \{ \sigma \}) \} \]

\[ X_{j+1}^c = \{ \text{UR}(\text{OR}(x_j^c, \sigma'), \Sigma \setminus \Sigma_o) : x_j^c \in X_j^c, \sigma' \in \Sigma_o \setminus \{ \sigma \} \} \]

\[ X_{j+2}^c = \{ \text{UR}(\text{OR}(x_{j+1}^c, \sigma'), \Sigma \setminus \Sigma_o) : x_{j+1}^c \in X_{j+1}^c, \sigma' \in \Sigma_o \} \]

\[ \vdots \]

\[ X_k^c = \{ \text{UR}(\text{OR}(x_{k-1}^c, \sigma'), \Sigma \setminus \Sigma_o) : x_{k-1}^c \in X_{k-1}^c, \sigma' \in \Sigma_o \}. \]

For all \( X' \in X_k^c \) if \( T_{\text{spec}}^e \cap (X' \times X') = \emptyset \) then the sensor for \( \sigma \) can be turned off while guaranteeing that no pair in \( T_{\text{spec}}^e \) will become indistinguishable after a total of \( k \) event occurrences have been observed by the agent. Then, after having observed \( k \) event occurrences, turning on the sensors for all events in \( \Sigma_o \) guarantees that the resulting state estimate will not contain a pair in \( T_{\text{spec}}^e \) by Theorem 2 of Wang et al. [53]. On the other hand, if there exists an \( X' \in X_k^c \) such that \( T_{\text{spec}}^e \cap (X' \times X') \neq \emptyset \) then the sensor for \( \sigma \) must be turned on. Otherwise, a pair of \( T_{\text{spec}}^e \) may become indistinguishable to the agent.

The above is applied before each of the first \( k \) observed event occurrences. After \( k \) event occurrences have been observed, one may then apply ONLINE-MIN-SEN-ACT of Wang et al. [53] for computing sensor activation decisions which guarantee that no states of a pair in \( T_{\text{spec}}^e \) will ever be indistinguishable.

An assertion is made of the correctness of Shu et al. [36] Algorithm 2 for computing solutions to Problem [3] (Theorem 3 of Shu et al. [36]).
3.2.4 Detectability of discrete event systems with dynamic event observation

A problem of sensor activation for purposes of precisely estimating the state of $G$ is considered in Shu et al. [37]. Specifically, detectability of DES is considered. Detectability refers to the ability of an agent to determine the current and future state of $G$ based on the event sequence observed by the agent. In prior work Shu et al. [40] it is assumed that event observation is static (i.e., observation of event sequences is defined by the natural projection $P : \Sigma^* \rightarrow \Sigma_o^*$). In this work they consider that the observation of an event is dynamic.

A DFA $G$ is considered to have multiple possible initial states. Specifically, $G = (X, \Sigma, \delta, X_0)$ where $X_0 \subseteq X$ denotes the set of possible initial states of $G$. The actual initial state of $G$ is not known to the agent. It is assumed that $G$ is deadlock-free. This is to ensure that any string in $L(G)$ can be extended to a string of infinite-length. It is assumed that there are no cycles of unobservable events in $G$. Such cycles would guarantee that there exists strings of infinite-length in $L(G)$ whose occurrence would not enable the agent to unambiguously determine the current state of $G$.

State-based sensor activation policies are considered in Shu et al. [37] which are, for practical purposes, equivalent to the transition-based sensor activation policies of Wang et al. [54]. That is, observations are defined by a policy $\Omega : X \rightarrow 2^{\Sigma_o}$. For example, given transition $(x, \sigma) \in TR(G)$, $\sigma$ is observed on occurrence of transition $(x, \sigma)$ if $\sigma \in \Omega(x)$. As there exist multiple possible initial states in $G$, a string $s \in L(G)$ may appear differently, depending on the initial state from which $s$ is generated in $G$. So the information map $\theta^\Omega$, which specifies how a given string is observed in previous work on sensor activation, is specified as $\theta^\Omega : X \times \Sigma^* \rightarrow \Sigma_o^*$. Given $s \in L(G)$, $\theta^\Omega(x_0, s)$
specifies how \( s \) is observed when \( G \) starts at state \( x_0 \in X_0 \). Formally, given \( x \in X \), for the empty string \( \varepsilon \), \( \theta^\Omega(x, \varepsilon) = \varepsilon \), and for all \( s\sigma \in \mathcal{L}(G) \) with \( \sigma \in \Sigma \)

\[
\theta^\Omega(x, s\sigma) = \begin{cases} 
\theta^\Omega(x, s)\sigma & \text{if } \sigma \in \Omega(\delta(x, s)) \\
\theta^\Omega(x, s) & \text{otherwise.}
\end{cases}
\]

Given a set of possible states (i.e., a state estimate) \( X' \subseteq X \), we can define the next set of possible states of \( G \) after sequence \( t \in \Sigma^* \) has been observed by

\[
R(X', t) = \{ \delta(x', s) : x' \in X', s \in \Sigma^*, \theta^\Omega(x', s) = t \land \delta(x', s)! \}.
\]

The unobserved reach of \( X' \), denoted by \( UR(X') \), is defined as \( UR(X') = R(X, \varepsilon) \).

Given \( G \) and \( \Omega \), four detectability conditions are introduced:

- **Detectability**: \( G \) is detectable under \( \Omega \) if it can be determined, after a finite number of observed event occurrences, the current state and subsequent states of \( G \) for some infinite strings of \( G \). Formally,

\[
\exists n \in \mathbb{N}, \exists (x, s) \in X_0 \times \mathcal{L}^\omega(G), \forall t \in \overline{\theta^\Omega(x, s)}, |t| > n \Rightarrow |R(X_0, t)| = 1.
\]

- **Strong Detectability**: \( G \) is strongly detectable under \( \Omega \) if it can be determined, after a finite number of observed event occurrences, the current state and subsequent states of \( G \) for all infinite strings of \( G \). Formally,

\[
\exists n \in \mathbb{N}, \forall (x, s) \in X_0 \times \mathcal{L}^\omega(G), \forall t \in \overline{\theta^\Omega(x, s)}, |t| > n \Rightarrow |R(X_0, t)| = 1.
\]

- **Periodic Detectability**: \( G \) is periodically detectable under \( \Omega \) if the current state of the system can be determined during regular intervals of observed event occurrences for some infinite strings of \( G \). Formally,

\[
\exists n \in \mathbb{N}, \exists (x, s) \in X_0 \times \mathcal{L}^\omega(G), \forall t \in \overline{\theta^\Omega(x, s)}, \exists t' \in \Sigma^*, tt' \in \overline{\theta^\Omega(x, s)} \land |t'| < n \land |R(X_0, tt')| = 1.
\]
• Strong Periodic Detectability: $G$ is strongly periodically detectable under $\Omega$ if the current state of the system can be determined during regular intervals of observed event occurrences for all infinite strings of $G$. Formally,

$$\exists n \in \mathbb{N}, \forall (x, s) \in X_0 \times \mathcal{L}(G), \forall t \in \overline{\theta^\Omega(x, s)}, \exists t' \in \Sigma^*,

\quad tt' \in \overline{\theta^\Omega(x, s)} \land |t'| < n \land |R(X_0, tt')| = 1.$$
a state in a loop in $G_{det}$. Automaton $G$ is strongly detectable under $\Omega$ if and only if $Z_c \subseteq Z_m$ (Theorem 6 of Shu et al. [39]). Automaton $G$ is strongly periodically detectable under $\Omega$ if and only if no loop in $G_{det}$ is entirely within $Z \setminus Z_m$ (Theorem 7 of Shu et al. [39]).

The weak detectability conditions cannot be verified using detector automata, but can be verified using observer automata. The observer $G_{obs}$ is constructed from $G$ and policy $\Omega$ as follows.

1. For each $x, x' \in X$, $\sigma \in \Sigma$, if transition $(x, \sigma) \in TR(G)$, $\delta(x, \sigma) = x'$ and $\sigma \notin \Omega(x)$ then remove transition $\delta(x, \sigma)$ and add transition $\delta(x, \varepsilon) = x'$. Denote the resulting NFA by $G_\varepsilon$.

2. Apply the subset construction from $G_\varepsilon$ to obtain $G_{obs}$.

Denote the state set of $G_{obs}$ by $Y \subseteq 2^X$. Let $Y_m = \{y \in Y : |y| = 1\}$. Automaton $G$ is detectable under $\Omega$ if and only if there exists at least one loop in $G_{obs}$ which is entirely within $Y_m$ (Theorem 2 of Shu et al. [39]). Automaton $G$ is periodically detectable under $\Omega$ if and only if there exists at least one loop in $G_{obs}$ which includes at least one state in $Y_m$ (Theorem 4 of Shu et al. [39]).

Note that, for implementation purposes, an automaton which maps from observed event sequences to precise state estimates of $G$ is realized by $G_{obs}$. Due to nondeterminism, detector automata cannot be used for this purpose.

Then the problem of computing a minimal sensor activation policy for satisfying a given detectability condition is considered. The problem and approach are similar to the problem and approach of Wang et al. [54].

A notion of feasibility of sensor activation policies is considered in Shu et al. [37] which is stronger than the feasibility condition [3.5] of Wang et al. [54]. Specifically,
policy Ω is feasible if
\[ \forall x, x' \in X_0, \forall s, s' \in L(G), \theta^\Omega(x, s) = \theta^\Omega(x', s') \Rightarrow \Omega(\delta(x, s)) = \Omega(\delta(x', s')). \quad (3.6) \]

Feasibility condition (3.6) requires that if two states of G are indistinguishable then the sensor activation decisions at both states must be the same for all events in \( \Sigma_o \). However, feasibility condition (3.5) defined previously for transition-based sensor activation policies requires only that sensor activation decisions are the same for events that label transitions from both states. The following notion of feasibility defined in terms of state-based sensor activation policies is analogous to the feasibility condition (3.5) for transition-based policies.

\[ \forall x, x' \in X_0, \forall s\sigma, s'\sigma \in L(G), \]
\[ \theta^\Omega(x, s) = \theta^\Omega(x', s') \Rightarrow [\sigma \in \Omega(\delta(x, s)) \iff \sigma \in \Omega(\delta(x', s'))]. \]

It is the objective of the agent to minimize the use of its sensors while making sure that G satisfies one of the detectability conditions. Formally, the following problem is considered.

**Problem 4.** Suppose that if the agent activates all observable event sensors all the time then detectability (resp., strong, periodic, strongly periodic detectability) is satisfied.

Find a state-based sensor activation policy \( \Omega^* : X \to 2^{\Sigma_o} \) such that

\( \quad C1 \quad \Omega^* \) is feasible (3.6);

\( \quad C2 \quad G \) contains no cycles of unobserved event occurrences (i.e., there does not exist \( x \in X, s \in \Sigma^*, \delta(x, s) = x, \) and for all \( s'\sigma \in \pi, \sigma \notin \Omega^*(\delta(x, s')) \));

\( \quad C3 \quad G \) is detectable (resp., strongly, periodically, strongly periodically detectable)
under $\Omega^*$;

$C_4$ $\Omega^*$ is minimal with respect to $C1 - C3$. That is, there does not exist an $\Omega'$ such that $C1 - C3$ is satisfied under $\Omega'$, for all $x \in X$, $\Omega'(x) \subseteq \Omega^*(x)$ and there exists $x' \in X$ such that $\Omega'(x') \subset \Omega^*(x')$.

Note that there may not exist a single unique solution to Problem 4. Multiple, incomparable solutions may exist.

The solution approach to Problem 4 is grounded on results analogous to those in Wang et al. [54]. For policies satisfying feasibility (3.6), an assertion is made that monotonicity holds in the sense that as more event sensors are activated, more states become distinguished [37] (analogous to Wang et al. [54] Corollary 1). Analogous to Wang et al. [54], it is proven that feasible state-based policies are closed under union (Theorem 9 of Shu et al. [37]). This result can be regarded as a generalization of Wang et al. [54] Corollary 1 in some sense as the plant $G$ considered has multiple initial states, not just one. An algorithm similar to MAX-Fea-SUB of Wang et al. [54] is proposed for computing the maximum feasible sub-policy $\Omega^F$ of a given policy $\Omega$. The correctness of the algorithm is established (Theorem 10 of Shu et al. [37]). Also, by monotonicity of policies, it is proven that policies which satisfy a given detectability condition are closed under union (Theorem 11 of Shu et al. [37]).

Algorithm 2 of Shu et al. [37] is proposed for computing solutions to Problem 4. Algorithm 2 is identical in its approach to computing solutions as Algorithm MIN-SEN-ACT of Wang et al. [54]. It is proven that Algorithm 2 is correct (Theorem 12 of Shu et al. [37]). Though one can certainly compute different solutions to Problem 4 by considering different orders in which sensors at states are deactivated, it is open whether or not all possible solutions may be computed by considering different orders.
We illustrate Algorithm 2 of Shu et al. [37] on Example 4 of Shu et al. [37] for computing a solution to Problem 4 when strong detectability is considered. Consider the plant $G$ of Figure 3.5. Suppose that $\Sigma_0 = \Sigma = \{a, b, c\}$ and that, initially, all sensors are turned on all the time for events in $\Sigma_0$. That is, $\Omega(0) = \Omega(1) = \Omega(2) = \{a, b, c\}$.

Suppose that the sensor for $a$ is deactivated at state 0. Denote the resulting policy by $\Omega_1$ where $\Omega_1(0) = \{b, c\}$, $\Omega_1(1) = \Omega(1)$, $\Omega_1(2) = \Omega(2)$. Compute the maximum...
feasible sub-policy $\Omega_1^{\uparrow F}$ of $\Omega_1$. One can verify that $\Omega_1^{\uparrow F}(0) = \Omega_1^{\uparrow F}(1) = \{b, c\}$, $\Omega_1^{\uparrow F}(2) = \{a, b, c\}$. Compute the detector automaton for $G$ under $\Omega_1^{\uparrow F}$, illustrated in Figure 3.6. As all states in the detector reachable from a state in a loop in Figure 3.6 contain only a single state of $G$, strong detectability is satisfied.

Suppose that, from $\Omega_1^{\uparrow F}$, the sensor for $b$ is deactivated at state 0. Denote the resulting policy by $\Omega_2$. Compute the maximum feasible sub-policy $\Omega_2^{\uparrow F}$ of $\Omega_2$. One can verify that $\Omega_2^{\uparrow F}(0) = \Omega_2^{\uparrow F}(1) = \Omega_2^{\uparrow F}(2) = \{c\}$. Compute the detector automaton for $G$ under $\Omega_2^{\uparrow F}$, illustrated in Figure 3.7. As state $\{1, 2\}$ contains more than one state of $G$ and is in a cycle in Figure 3.7, strong detectability is not satisfied. So the sensor for event $b$ must remain on at state 0.

Continuing in this manner, a minimal sensor activation policy $\Omega^*$ can be computed where $\Omega^*(0) = \Omega^*(1) = \{b, c\}$, $\Omega^*(2) = \{b\}$.

### 3.2.5 Minimal sensor activation for co-detectability of multi-agent discrete event systems

Decentralized versions of the detectability conditions of Shu et al. [37] are introduced in Shu et al. [35]. Given a set of agents $I$ the problem considered is to compute offline a minimal state-based sensor activation policy for each agent such that a given decentralized detectability (i.e., codetectability) condition is satisfied.

Automaton $G = (X, \Sigma, \delta, X_0)$ where $X_0 \subseteq X$ denotes the set of possible initial states of $G$. It is assumed that $G$ is deadlock-free. It is assumed that there are no cycles of transitions in $G$ labeled by events in $\Sigma_u = \Sigma \setminus \bigcup_{i \in I} \Sigma_{i,o}$.

We refer the reader to the background provided for Shu et al. [37] for definitions of state-based sensor activation policies and their corresponding information maps.
Given sensor activation policy \( \Omega_i : X \rightarrow 2^{\Sigma_o} \) for each \( i \in I \), we denote the global observation policy by \( \Omega_g \) where for all \( x \in X \), \( \Omega_g(x) = \cup_{i \in I} \Omega_i(x) \).

Given state estimate \( X' \subseteq X \), we can define agent \( i \)'s state estimate of \( G \) after sequence \( t \in \Sigma^{\star}_{i,o} \) has been observed by

\[
R_i(X', t) = \{ \delta(x', s) : x' \in X', s \in \Sigma^*, \theta^{\Omega_i}(x', s) = t \land \delta(x', s) \}
\]

We use \( R_g(X', t) \) to denote such a state set when the global observation policy \( \Omega_g \) is used.

We provide a definition of codetectability, the decentralized analogue of detectability. Automaton \( G \) is codetectable under agent policies \( \Omega_1, \ldots, \Omega_{|I|} \) if it at least one agent \( i \in I \) can determine, after a finite number of observed event occurrences, the current state and subsequent states of \( G \) for some infinite strings of \( G \). Formally,

\[
\exists n \in \mathbb{N}, \exists (x, s) \in X_0 \times L(G), \forall t \in \theta^{\Omega_g}(x, s), |t| > n \Rightarrow \exists i \in I, |R_i(X_0, \theta^{\Omega_i}(s))| = 1.
\]

Strong codetectability, periodic codetectability and strong periodic codetectability are defined in a similar fashion from their centralized counterparts of Shu et al. [37].

The sensor activation problem considered is the following. This problem is stated in terms of codetectability but could also be phrased in terms of the other decentralized detectability conditions.

**Problem 5.** Suppose that \( L(G) \) is codetectable under the set of policies where all sensors for events in \( \Sigma_{i,o} \) are turned on all the time by agent \( i \).

Find state-based sensor activation policies \( \Omega^*_{i,1}, \ldots, \Omega^*_{i,|I|} \) such that

\( C1 \) \( \Omega^*_i \) is feasible \((3.6)\) for all \( i \in \{1, \ldots, |I|\} \);

\( C2 \) \( G \) contains no cycles of unobserved event occurrences under the global observation policy \( \Omega^*_g \) (i.e., there does not exist \( x \in X \), \( s \in \Sigma^* \), \( \delta(x, s) = x \), and for all
CHAPTER 3. LITERATURE REVIEW

\[ s' \sigma \in \bar{s}, \sigma \notin \Omega_g^*(\delta(x,s')) \];

\( C3 \) \( G \) is codetectable under \( \Omega_1^*, \ldots, \Omega_{|I|}^* \);

\( C4 \) \( \Omega_1^*, \ldots, \Omega_{|I|}^* \) is minimal with respect to \( C1 - C3 \). That is, there does not exist \( \Omega_1', \ldots, \Omega_{|I|}' \), \( j \in \{1, \ldots, |I|\} \) such that \( \Omega_1' \subseteq \Omega_1^*, \ldots, \Omega_{|I|}' \subseteq \Omega_{|I|}^* \), \( \Omega_j' \subset \Omega_j^* \) and \( C1 - C3 \) is satisfied under \( \Omega_1', \ldots, \Omega_{|I|}' \).

Note that it is not necessarily the case that there exists a single solution to Problem 5. In general, there exist multiple, incomparable solutions.

For verifying codetectability, the construction of the coobserver is proposed. The coobserver is an automaton which maps from strings in \( \mathcal{L}(G) \) to state estimates \( R_i(X_0, \theta^{\Omega_i}(s)) \) for all agents \( i \in I \) as well as \( R_g(X_0, \theta^{\Omega_g}(s)) \), the state estimate of the global observation policy. Codetectability conditions are characterized in terms of loops of the coobserver (Theorems 2 - 5 of Shu et al. \[35\]), similar to how detectability conditions are characterized in terms of loops of the observer automaton or detector automaton (Theorems 2, 4, 6, 7 of Shu et al. \[39\]). Note that in the case of natural projections a codetector automaton may be constructed Shu et al. \[38\], whose construction is exponential in the number of agents but polynomial in all other input parameters. It is open whether or not similar codetector automata may be constructed for verifying the codetectability conditions when state-based sensor activation policies are used.

Algorithm 2 of Shu et al. \[35\] is proposed for computing solutions to Problem 5. The algorithm is similar to algorithms Min-Sen-Codiag of Wang et al. \[49\] and Min-Sen-CO of Wang et al. \[54\]. Sensors are deactivated, one at a time, from states in an agent \( i \)'s policy. As usual, the choice of which agent's sensor should be deactivated at which state is either provided as input or is arbitrary. Then the
maximum feasible sub-policy of $i$’s policy $\Omega_i$ is computed and a test for the correctness criteria C2 - C3 is made. If neither C2 nor C3 are satisfied then the sensor must remain on (by a monotonicity result analogous to Theorem 1 of Wang et al. [54]). Otherwise, it is turned off. This procedure is repeated until no event sensor used by any agent can be turned off at any state of $G$ without resulting in C2 or C3 being violated.

Correctness of the algorithm is established (Theorem 6 of Shu et al. [35]). As usual, different solutions can be computed by different orders in which we choose to deactivate sensors in Algorithm 2 of Shu et al. [35].

3.3 Minimal Communication for Purposes of State Disambiguation

In this section we consider the computation of agent communication policies for purposes of state disambiguation. We restrict attention to the case where event observations and communications are defined over the transitions of a given automaton. That is, whether or not an event is observed (resp., communicated from one agent to another) depends on the transition that the event labels in the given automaton. The problems considered regard computing a minimal set of communication transitions for each agent that enable each agent to distinguish between certain states of an automaton ($G$ or otherwise). Multiple solutions may exist to the problems considered, similar to the sensor activation problems considered in Section 3.2.


3.3.1 Minimal communication in a distributed discrete-event system

One of the first works on transition-based communication in DES is Rudie et al. [23]. This work considers two agents, 1 and 2. Associated with each agent \( i \in \{1, 2\} \) is a DFA \( R_i = (X_i, \Sigma, \delta_i, x_{i,0}) \) where \( \delta_i \) is a total function defined over event set \( \Sigma = \Sigma_{1,o} \cup \Sigma_{2,o} \). Automaton \( R_i \) could be a supervisory controller, diagnoser, or other application-dependent DFA agent model, depending on the specific problem at hand. Automaton \( R_i \) can have transitions between two different states labelled by events in \( \Sigma \setminus \Sigma_{i,o} \). It is required that agent \( i \) observe all transitions between two different states in \( R_i \). Equivalently, agent \( i \) must always know the state of \( R_i \) following any string in \( \Sigma^* \). Events are communicated from 1 to 2 and vice versa for this purpose. It is assumed that events are communicated reliably, without delay and immediately upon their observation.

A language-based model of communication is proposed Rudie et al. [23]. Following the language-based model a transition-based model of communication is proposed for restricting computation of communication maps for each agent to a finite domain. We restrict attention to the transition-based model.

Communications are defined over the transitions of DFA

\[
R = R_1 || R_2 = (X, \Sigma, \delta_R, x_0)
\]

where \( X = X_1 \times X_2 \), \( x_0 = (x_{1,0}, x_{2,0}) \) and, for all \( x_1 \in R_2 \), for all \( x_2 \in R_2 \), for all \( \sigma \in \Sigma \), \( \delta_R((x_1, x_2), \sigma) = (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma)) \). Let \( TR(R) = \{(x, \sigma) : x \in X, \sigma \in \Sigma, \delta_R(x, \sigma)\} \). Agent 1’s (resp., 2’s) communication policy \( V_{12} \) (resp., \( V_{21} \)) is a subset of \( TR(R) \). Let \( i, j \in \{1, 2\} \) where \( i \neq j \). Specifically, \( V_{ij} \subseteq \{(x, \sigma) \in TR(R) : \sigma \in \Sigma \} \).
Σ_{i,o} \setminus Σ_{j,o}\}.

Given V_{12}, V_{21}, we can define how strings in Σ* appear to each agent. We use induction to define agent i’s information map θ_{V_{ji}} : Σ* → Σ* as follows. For the empty string ε, θ_{V_{ji}}(ε) = ε, and for all sσ ∈ Σ* with σ ∈ Σ

\[\theta_{V_{ji}}(sσ) = \begin{cases} 
\theta_{V_{ji}}(s)σ & \text{if } σ ∈ Σ_{i,o} \setminus Σ_{j,o}, \forall (δ_R(x_0,s),σ) \in V_{ji} \\
\theta_{V_{ji}}(s) & \text{otherwise}.
\end{cases}\]

Note that agent 2’s communication policy is used to define agent 1’s information map and vice versa. This is in contrast with the information maps defined in sensor activation settings where agent i’s observations are only based on i’s sensor activation decisions.

As in problems of sensor activation, it is required that communication policies V_{12}, V_{21} be feasible. That is, if any two strings s, s’ appear identical to agent i, then the decision to communicate an event to agent j should be the same following either s or s’. Given V_{12}, V_{21}, we let (V_{12}, V_{21}) denote a communication scheme. A communication scheme (V_{12}, V_{21}) means that agent 1 (resp., 2) is committed to communicating transitions in V_{12} (resp., V_{21}). Formally, communication scheme (V_{12}, V_{21}) is feasible if

\[∀s, s' ∈ Σ*, ∀σ ∈ Σ_{i,o} \setminus Σ_{j,o}, θ_{V_{ji}}(s) = θ_{V_{ji}}(s') \Rightarrow [\forall (δ_R(x_0,s),σ) ∈ V_{ij} \Leftrightarrow (δ_R(x_0,s'),σ) ∈ V_{ij}].\] (3.7)

In order that agent j can precisely determine the state of R_j following any sequence in Σ* it is required that communication policy V_{ij} (resp., V_{ji}) be valid with respect to R_j (resp., R_i). Formally, V_{ij} is valid with respect to R_j if

\[∀s, s' ∈ Σ*, θ_{V_{ij}}(s) = θ_{V_{ij}}(s') \Rightarrow δ_j(x_{j,0}, s) = δ_j(x_{j,0}, s').\] (3.8)
The problem considered in Rudie et al. [23] is equivalent to the following.

**Problem 6.** Given $R$ derived from $R_1, R_2$, observable event sets $\Sigma_{1,o}, \Sigma_{2,o}$, find communication policies $V^*_{12}, V^*_{21} \subseteq TR(R)$ such that

1. $V^*_{12}$ (resp., $V^*_{21}$) is valid (3.8) with respect to $R_2$ (resp., $R_1$);
2. $(V^*_{12}, V^*_{21})$ is feasible (3.7);
3. $V^*_{12}, V^*_{21}$ are minimal with respect to $C1$ - $C2$. That is, there does not exist $V'_{12}, V'_{21}$ where $V^*_{12}$ (resp., $V^*_{21}$) is valid with respect to $R_2$ (resp., $R_1$), $(V^*_{12}, V^*_{21})$ is feasible, $V'_{ij} \subseteq V^*_{ij}$ and either $V'_{12} \subset V^*_{12}$ or $V'_{21} \subset V^*_{21}$.

Note that there may exist multiple, incomparable solutions to Problem 6.

The solution approach of Rudie et al. [23] to Problem 6 is described next.

Given $V_{ji}$, we define the set of state pairs of $R$ that are indistinguishable to $i$. We denote this set by $\mathcal{T}^{V_{ji}}_{conf}$ which is defined as

$$\mathcal{T}^{V_{ji}}_{conf} = \{(x, y) \in X \times X : \exists s, s' \in \Sigma^*, \delta_R(x_0, s) = x$$
$$\quad \land \delta_R(x_0, s') = y \land \theta^i(s) = \theta^i(s')\}. $$

Set $\mathcal{T}^{V_{ji}}_{conf}$ can be computed using the CLUSTER-TABLE algorithm of Wang et al. [50] by relabeling transitions in $TR(R) \setminus \{(x, \sigma) \in TR(R) : \sigma \in \Sigma_{i,o} \cup V_{ji}\}$ with $\varepsilon$.

A characterization of feasibility (3.7) in terms of the states of $R$ may be established. Given $V_{ij}, V_{ji}$ we say agent $i$’s communication policy $V_{ij}$ is *consistent* if

$$\forall (x, y) \in \mathcal{T}^{V_{ji}}_{conf}, \forall \sigma \in \Sigma_{i,o} \setminus \Sigma_{j,o}, (x, \sigma) \in V_{ij} \iff (y, \sigma) \in V_{ij}. \quad (3.9)$$

If $V_{12}$ and $V_{21}$ are both consistent (3.9) then $(V_{12}, V_{21})$ is feasible (3.7) (Theorem 1 of Rudie et al. [23]).
A characterization of validity \textsuperscript{(3.8)} in terms of the states of $R$ may be established. We say agent 1’s communication policy $V_{12}$ is \textit{correct} if

$$\forall (x, y) \in T_{\text{conf}}^{V_{12}}, x = (x_1, x_2) \land y = (x'_1, x_2).$$

(3.10)

That is, for any two states $x, y$ of $R$ that agent 2 finds indistinguishable, the $R_2$ state component of $x$ and $y$ should be the same. This implies that agent 2 will always be able to determine the state of $R_2$ following any string in $\Sigma^*$. One can define correctness of agent 2’s communication policy $V_{21}$ symmetrically. If $V_{12}$ and $V_{21}$ are both consistent and $V_{12}$ (resp., $V_{21}$) is correct then $V_{12}$ (resp., $V_{21}$) is valid.

Using these characterizations, the approach of Rudie et al. \textsuperscript{[23]} proceeds by initially setting $V_{ij}$ to a set of transitions such that $V_{ij}$ is correct \textsuperscript{(3.10)}. Specifically, let $C_{12} = \{(x, \sigma) : \sigma \in \Sigma_{1,o} \setminus \Sigma_{2,o}, \exists y \in X_1 \times X_2, \delta_R(x, \sigma) = y \land x = (x_1, x_2) \land y = (y_1, y_2) \land x_2 \neq y_2\}$. One can define $C_{21}$ symmetrically. Any $V_{ij} \supseteq C_{ij}$ is correct (Lemma 1 of Rudie et al. \textsuperscript{[23]}).

In the approach, $V_{ij}$ is initially set to $C_{ij}$. It is not necessarily the case that $V_{ij}$ is consistent. So one can consider adding transitions to $V_{ij}$ from the set $\{(x, \sigma) \in TR(R) : \sigma \in \Sigma_{i,o} \setminus \Sigma_{j,o}\} \setminus V_{ij}$ in such a manner that $V_{ij}$ becomes consistent. Algorithm $\mathcal{N}_{ij}(V_{ij}, V_{ji})$ \textsuperscript{[23]} is proposed for computing a set of transitions $\mathcal{N}_{ij}$ which, if added to $V_{ij}$, would result in $V_{ij}$ being consistent. It proceeds by computing $T_{\text{conf}}^{V_{ij}}$. For all $(x, y) \in T_{\text{conf}}^{V_{ij}}$, for all $\sigma \in \Sigma_{i,o} \setminus \Sigma_{j,o}$, if $(x, \sigma) \in V_{ij}$ but $(y, \sigma) \notin V_{ij}$ then $(y, \sigma)$ is added to $\mathcal{N}_{ij}$. Repeat this step until there are no further changes to $\mathcal{N}_{ij}$.

There is an apparent difficulty in satisfying consistency due to the mutual dependency of observation and communication. In order to satisfy consistency, agent $i$ may have to communicate on occurrence of transitions not in $V_{ij}$ (i.e., transitions in set $\mathcal{N}_{ij}(V_{ij}, V_{ji})$). This, in turn, affects agent $j$’s observation (i.e., $\theta^{V_{ij}}$). The changes to
j’s observation may result in a violation of consistency with respect to policy \( V_{ji} \). So agent \( j \) may have to communicate on occurrence of transitions not in \( V_{ji} \) (i.e., transitions in set \( N_{ji}(V_{ji}, V_{ij} \cup N_{ij}(V_{ij}, V_{ji})) \)). This, in turn, affects agent \( i \)’s observation, and so on. So one may have to alternate between adding communication transitions from agent \( i \) and adding communication transitions from agent \( j \). However, the proposed approach only involves one such iteration.

First, agent \( i \) communicates \( C_{ij} \). Then, transitions are added to agent 1’s communication transition set to satisfy consistency. Specifically, the set of transitions to be added to agent 1’s communication transition set is \( N_{12}^{\text{max}} = N_{12}(C_{12}, C_{21}) \). Then, transitions are added to agent 2’s communication transition set to satisfy consistency. Specifically, agent 2’s communication transition set becomes and remains fixed as \( V_{21}^* = C_{21} \cup N_{21}(C_{21}, C_{12} \cup N_{12}^{\text{max}}) \). Afterwards we consider modifying only agent 1’s communication transition set. Compute \( N_{12}^{\text{min}} = N_{12}(C_{12}, V_{21}^*) \). One can verify that \( N_{12}^{\text{min}} \subseteq N_{12}^{\text{max}} \). The objective now is to find a set \( N_{12}^* \) such that

**S1** \( N_{12}^* \supseteq N_{12}^{\text{min}}; \)

**S2** \( N_{12}^* \subseteq N_{12}^{\text{max}}; \)

**S3** \( C_{12} \cup N_{12}^* \) is consistent when agent 2 communicates \( V_{21}^* \);

**S4** if \( C_{12} \cup N_{12}^* \) is communicated to agent 2 then \( V_{21}^* \) is communicated back to agent 1 (i.e., \( V_{21}^* = C_{21} \cup N_{21}(C_{21}, C_{12} \cup N_{12}^*) \));

**S5** there does not exist an \( N_{12}' \subset N_{12}^* \) under which S1 - S4 is satisfied in place of \( N_{12}^* \).

There are a number of ways in which one could compute \( N_{12}^* \). One method involves enumerating policies \( N_{12} \) where \( N_{12} \supseteq N_{12}^{\text{min}} \) and \( N_{12} \subseteq N_{12}^{\text{max}} \) in monotonically
increasing order of cardinality. This enumeration procedure is called the banker’s sequence \[18\]. This enumeration can be used to compute an \( N_{12}^* \) of minimum cardinality. Another, perhaps better method is suggested by Lemmas 6 - 8 in Section III D of Rudie et al. \[23\].

An \( N_{12} \) exists which satisfies S1 - S4. Specifically, \( N_{12}^{\text{max}} \) is one such policy (Proposition 2 of Rudie et al. \[23\]). Thus an \( N_{12}^* \) exists which satisfies S1 - S5.

It is conjectured that the communication scheme \((C_{12} \cup N_{12}^*, V_{21}^*)\) computed using the approach described above is a solution to Problem \[6\] \[23\].

However, note that the approach is exponential in the state and transition cardinalities of \( R \) since \( N_{12}^{\text{max}} \) may be the only solution, which requires us to test S3 - S4 for all policies \( N_{12} \) where \( N_{12} \supseteq N_{12}^{\text{min}} \) and \( N_{12} \subseteq N_{12}^{\text{max}} \).

We recall \[23\] Example IV A for illustrating the approach of Rudie et al. \[23\]. We suppose that agent 1 observes \( \Sigma_{1,o} = \{a_1, b_1\} \) and agent 2 observes \( \Sigma_{2,o} = \{a_2, b_2\} \).

Consider the \( R_1 \) of Figure 3.8 and \( R_2 \) of Figure 3.9. Note that in \( R_i \) there exist transitions between different states labeled by events in \( \Sigma_{j,o} \setminus \Sigma_{i,o} \). So events will need to be communicated from agent 1 to 2 and vice versa to satisfy the state disambiguation goal. Compute \( R_1 \| R_2 \), illustrated in Figure 3.10. We then identify the

\[
\begin{align*}
\text{Figure 3.8: Automaton } R_1 & \hspace{1cm} \text{Figure 3.9: Automaton } R_2
\end{align*}
\]

set of essential communications \( C_{12} \) and \( C_{21} \). Without communications in \( C_{ij} \), agent
j would not be able to precisely determine the state of \( R_j \) following some sequence in \( \Sigma^* \). Set \( C_{12} \) is the set of all transitions of \( R_1 \parallel R_2 \) labeled by events observed by agent 1, not observed by agent 2 and which cause a transition between two different states of \( R_2 \). Specifically,

\[
C_{12} = \{ ((1,1), a_1, (1,2)), ((2,1), a_1, (2,2)), ((3,1), a_1, (3,2)),
\]

\[
( (2,3), b_1, (2,2)), ((3,3), b_1, (3,2)) \}
\]

\[
C_{21} = \{ ((1,1), b_2, (2,1)), ((1,2), b_2, (2,3)) \}
\]

Next we determine which transitions of \( R_1 \parallel R_2 \) need to be communicated from agent 1 to agent 2 to satisfy consistency. In order to do this, we need to compute which pairs of states agent 1 finds indistinguishable in \( R_1 \parallel R_2 \) given that agent 2 communicates \( C_{21} \) to agent 1. This set is \( T_{conf}^{C_{21}} \). We can apply the CLUSTER-TABLE of algorithm of Wang et al. [50] for computing \( T_{conf}^{C_{21}} \) in polynomial time. Alternatively, we can compute agent 1’s observer DFA of \( R_1 \parallel R_2 \) by replacing the label of any transition in \( TR(R_1 \parallel R_2) \) that agent 1 does not observe by the empty string \( \varepsilon \) then applying the subset construction on the result. Specifically, we replace labels of transitions not in \( C_{21} \) or labeled by an event in \( \Sigma_{1,o} \) by \( \varepsilon \) then apply the subset construction. The resulting automaton is denoted by \( \tilde{R}_1(C_{21}) \) and is illustrated in Figure 3.11. Agent 1 finds states \( x, y \) of \( R_1 \parallel R_2 \) indistinguishable (i.e., in \( T_{conf}^{C_{21}} \)) if and only if there exists a state \( \tilde{x} \) in \( \tilde{R}_1(C_{21}) \) that contains \( x \) and \( y \). Next we scan \( T_{conf}^{C_{21}} \) for pairs \( (x, y) \) such that there exists a \( \sigma \in \Sigma_{1,o} \setminus \Sigma_{2,o} \) where \( (x, \sigma) \in C_{12} \) but \( (y, \sigma) \notin C_{12} \). When such a pair is identified, we add \( (y, \sigma) \) to set \( \mathcal{N}_{12}(C_{12}, C_{21}) \), the set of additional communications from agent 1 required to make agent 1’s communication set consistent. This procedure
is done iteratively over \( \mathcal{T}^{C_{21}}_{\text{conf}} \) until no such pairs \((x, y)\) are found. For this example, 
\[
\mathcal{N}_{12}(C_{12}, C_{21}) = \{((1, 2), a_1, (1, 2)), ((2, 2), a_1, (2, 2)), ((2, 3), a_1, (2, 3)), ((3, 2), a_1, (3, 2)),
((3, 3), a_1, (3, 3)), ((2, 1), b_1, (2, 1)), ((2, 2), b_1, (2, 2)), ((3, 1), b_1, (3, 1)),
((3, 2), b_1, (3, 2))\}.
\]
We set \( N^\text{max}_{12} \) to \( \mathcal{N}_{12}(C_{12}, C_{21}) \), which forms the upper bound for the later step where we search for the \( N^*_{12} \) used to define a solution to Problem 6. Next we determine which transitions of \( R_1 \| R_2 \) need to be communicated from agent 2 to satisfy consistency. We compute \( N^*_{21} = \mathcal{N}_{21}(C_{21}, C_{12} \cup N^\text{max}_{12}) \) similar to the way we computed \( \mathcal{N}_{12}(C_{12}, C_{21}) \).

One can verify that 
\[
N^*_{21} = \{((3, 1), b_2, (3, 1)), ((3, 2), b_2, (3, 3)), ((2, 1), b_2, (2, 1)), ((2, 2), b_2, (2, 3))\}.
\]
Now agent 2’s communications are fixed as \( V^*_{21} = C_{21} \cup N^*_{21} \). Given that agent 2 communicates \( V^*_{21} \) to agent 1, which is more than before (i.e., \( V^*_{21} \supseteq C_{21} \)), we consider
what agent 1 would have to communicate to agent 2 to satisfy consistency if agent 1 just communicates $C_{12}$. That is, we compute $N_{12}^{\text{min}} = \mathcal{N}_{12}(C_{12}, V_{12}^*)$. Set $N_{12}^{\text{min}}$ forms a lower bound for our search for $N_{12}^{*}$. However, for this example, $N_{12}^{\text{min}} = N_{12}^{\text{max}}$. Thus, for this example, there exists only one solution to Problem 6 that may be computed using the approach of Rudie et al. [23].

Given that agent 2 communicates $V_{21}^*$ to agent 1, we can compute $\tilde{R}_1(V_{21}^*)$ from $V_{21}$ in the same way that $\tilde{R}_1(C_{21})$ was computed from $C_{21}$. We use $\tilde{R}_1(V_{21}^*)$ to define a map from agent 1’s event observations to the set of events it communicates to agent 2. Let $\tilde{x}_{1,0}$ (resp., $\tilde{\delta}_1$) denote the initial state (resp., transition function) of $\tilde{R}_1(V_{21}^*)$. Specifically, given an observed string $s \in \Sigma^*$, the set of events that agent 1 would communicate to agent 2 is the set $\{\sigma \in \Sigma_1, o \setminus \Sigma_2, o : \exists x \in \tilde{\delta}_1(\tilde{x}_{1,0}, s), (x, \sigma) \in C_{12} \cup N_{12}^*\}$.

A different approach one may conjecture could be used is to begin with each agent communicating all event occurrences to the other then iteratively removing communications from one agent to another until removing any communication (besides transitions in $C_{12}$ and $C_{21}$) violates consistency or some other correctness criterion. A similar top-down approach is used to compute solutions to most sensor activation problems. However, this top-down approach will not work in general for computing solutions to Problem 6 and other problems where communication is transition-based. By further removing transitions from the sets of communication transitions, the resulting sets of communications may become consistent. This is due to an interesting property of transition-based communications: in some instances, with less communication, any two strings (resp., states) which were previously indistinguishable by an agent can become distinguishable. This counterintuitive phenomenon is referred to as
“lack of monotonicity” in Wang et al. [46] where a more detailed examination and example illustrating it is provided. Effectively, in the case of consistent transition-based communication policies, as more is communicated between agents it is not necessarily the case that more states / strings become distinguished by each agent respectively. However, we have observed in Section 3.2 that, in the case of feasible transition-based sensor activation policies, this property does hold (Corollary 1 of Wang et al. [54]).

We provide an example demonstrating lack of monotonicity of consistent communication policies which is similar to Example 3 of Wang et al. [54] and the lack of monotonicity example in Section 4 of Lafortune [13]. Consider the automaton of Figure 3.12. Suppose there exist two agents, 1 and 2. Agent 1 observes events $\alpha$ and $\beta$ whereas agent 2 observes event $\gamma$. That is, $\Sigma_{1,o} = \{\alpha, \beta\}$ and $\Sigma_{2,o} = \{\gamma\}$. It is required that agent 2 distinguish between states 4 and 6 in Figure 3.12 (equivalently, distinguish between strings $\alpha\beta\gamma$ and $\beta\gamma$). Without communication from agent 1, agent 2 cannot distinguish between strings $\alpha\beta\gamma$ and $\beta\gamma$. To remedy this, agent 1 could communicate $\beta$ on occurrence of transition $(1, \beta, 5)$ and not communicate on the occurrence of any other transition. Agent 2 is not required to communicate anything to agent 1. One can verify that, for the proposed communication scheme, agent 1’s communication policy is consistent and agent 2’s communication policy is
consistent. Let $\theta_2$ denote agent 2’s information map obtained from $\Sigma_{2,o}$ and agent 1’s communication policy. One can verify that $\theta_2(\alpha\beta\gamma) = \gamma$ whereas $\theta_2(\beta\gamma) = \beta\gamma$. Thus, agent 2 can distinguish between states 4 and 6 under the developed communication scheme.

Now, from the above described communication scheme, suppose that agent 1 communicates to agent 2 event $\beta$ on the occurrence of transition $(2, \beta, 3)$. One can verify that agent 1’s resulting communication policy is consistent. Let $\theta'_2$ denote agent 2’s information map obtained from $\Sigma_{2,o}$ and agent 1’s resulting communication policy. We have that $\theta'_2(\alpha\beta\gamma) = \theta'_2(\beta\gamma) = \beta\gamma$ and so agent 2 cannot distinguish between states 4 and 6.

To reaffirm, as more event occurrences are communicated between agents, it is not necessarily the case that each agent can distinguish between more pairs of strings or states. As observed from the above example, this is still true even when we restrict consideration to consistent communication policies.

### 3.3.2 Minimal communication for essential transitions in a distributed discrete-event system

A generalization of Problem 6 is considered in Rudie et al. [16]. The problem considered is to define communication policies for agents in such a manner that agent $i$ observes the occurrence of events labeling certain transitions in its DFA $R_i$. Such transitions are labelled by events in $\Sigma \setminus \Sigma_{i,o}$ that agent $i$ does not locally observe, so agent $i$ requires certain event occurrences to be communicated to it by the other agent $j$. Agent $i$ is required to observe certain transitions of $R_i$ in order that $i$ can distinguish certain pairs of states in $R_i$. This is a more general requirement than
Rudie et al. [23] where it is required that agent \( i \) always precisely know which state \( R_i \) is in.

The approach taken in Rudie et al. [16] can be used to compute all pairs of communication policies defined over automaton \( R_1||R_2 \) that result in each agent \( i \) observing the occurrence of certain transitions in \( R_i \). This contrasts with the approach of Rudie et al. [23] where only one solution to Problem 6 might be computed.

Formally, DFA \( R_i = (X_i, \Sigma, \delta_i, x_i, 0) \) where \( \delta_i \) is a total function defined over event set \( \Sigma = \Sigma_{1,o} \cup \Sigma_{2,o} \). Let \( TR(R_i) \) denote the transition set of \( R_i \). A set of essential transitions that agent \( i \) must observe, denoted by \( ETR_i \), is presumed given. Denote the set of transitions of \( R_i \) labelled by events not in \( \Sigma_{i,o} \) by \( UTR_i = \{ (x, \sigma) \in TR(R_i) : \sigma \notin \Sigma_{i,o} \} \). The set of \( i \)'s essential transitions \( ETR_i \subseteq UTR_i \).

Set \( ETR_i \) is defined depending on the state disambiguation problem at hand. Let \( OTR_i = TR(R_i) \setminus UTR_i \). For instance, if we let \( T^i_{spec} \subseteq X_i \times X_i \) denote the set of state pairs of \( R_i \) that are required to be distinguished by agent \( i \), then one can separately compute \( ETR_i \) in such a manner that observing transitions in \( OTR_i \cup ETR_i \) permits states of any pair in \( T^i_{spec} \) to be distinguished by \( i \). It is not necessary that \( OTR_i \cup ETR_i \) satisfies a notion of feasibility such as \( (3.4) \) since it is the responsibility of the other agent \( j \) to make sure that transitions in \( ETR_i \) are observed by \( i \). Computing such \( ETR_i \) given \( T^i_{spec} \) is unsolved.

As in Rudie et al. [23], communications are defined over the transitions of DFA \( R = R_1||R_2 = (X_1 \times X_2, \Sigma, \delta, x_0) \) where \( x_0 = (x_{1,0}, x_{2,0}) \). Agent 1’s (resp., 2’s) communication policy \( V_{12} \) (resp., \( V_{21} \)) is a subset of \( TR(R) \). See the background of Rudie et al. [23] for precise definitions as well as definitions of agent \( i \)'s information map \( \theta^{V_{ji}} \) when agent \( j \) communicates events to agent \( i \) on occurrence of transitions.
in $V_{ji}$.

In order that agent $j$ (resp., $i$) observes the occurrence of events labeling transitions in $ETR_j$ (resp., $ETR_i$) it is required that communication policy $V_{ij}$ (resp., $V_{ji}$) be \textit{legal} with respect to $R_j$ (resp., $R_i$). Formally, $V_{ij}$ is legal with respect to $R_j$ if

$$\forall s \in \Sigma^*, \forall \sigma \in \Sigma, (\delta_j(x_{j,0}, s), \sigma) \in OTR_j \cup ETR_j \Rightarrow \theta^{V_{ij}}(s\sigma) = \theta^{V_{ij}}(s)\sigma.$$  \hspace{1cm} (3.11)

One can verify that legality is more general than validity \hspace{1cm} (3.8).

The problem considered in Rudie et al. \cite{16} is equivalent to the following.

\textbf{Problem 7.} Given $R$ derived from $R_1, R_2$, observable event sets $\Sigma_{1, o}, \Sigma_{2, o}$, find communication policies $V^*_1, V^*_2 \subseteq TR(R)$ such that

$C1$ $V^*_2$ (resp., $V^*_1$) is legal \hspace{1cm} (3.11) with respect to $R_2$ (resp., $R_1$);

$C2$ $(V^*_1, V^*_2)$ is feasible \hspace{1cm} (3.7);

$C3$ $V^*_1, V^*_2$ are minimal with respect to $C1 - C2$. Minimality is defined in the same sense as in Problem 6.

Next we describe the approach of Rudie et al. \cite{16} for computing all solutions to Problem 7.

Given a set of transitions $T_i \subseteq TR(R_i)$, we define the extension of $T_i$ to $R$ as the set of transitions $EXT(T_i) = \{(x_1, x_2, \sigma) \in TR(R) : (x_i, \sigma) \in T_i\}$. For example, $EXT(ETR_i)$ is the set of transitions of $R$ whose event labels must be communicated to agent $i$.

The algorithm proposed in Rudie et al. \cite{16} for computing all solutions to Problem 7 is exhaustive. Set $EXT(ETR_j)$ is a lower bound for what agent $i$ needs to communicate to $j$ in order that $j$ observe all of its essential transitions $ETR_j$. Also,
$\text{EXT}(\text{UTR}_j)$ is an obvious upper bound for what agent $i$ needs to communicate to $j$. Any set $V_{ij}$ where $V_{ij} \supseteq \text{EXT}(\text{ETR}_j)$ and $V_{ij} \subseteq \text{EXT}(\text{UTR}_j)$ is legal with respect to $R_j$ (Theorem 2 of Rudie et al. [16]). The algorithm proposes to traverse all pairs $(V_{i2}, V_{21})$ where $V_{ij} \supseteq \text{EXT}(\text{ETR}_j)$ and $V_{ij} \subseteq \text{EXT}(\text{UTR}_j)$. A set MINSET is maintained of the possible solutions to Problem 7 found thus far. If there does not exist $(V_{i2}', V_{21}') \in \text{MINSET}$ such that $V_{i2}' \subseteq V_{i2}$ and $V_{21}' \subseteq V_{21}$ then pair $(V_{i2}, V_{21})$ is tested for consistency (3.9). This is done by testing if there exists a $\sigma \in \Sigma$, pair $(x, y) \in T_{\text{conf}}^{C_{12}}$ (resp., $T_{\text{conf}}^{C_{21}}$) such that $(x, \sigma) \in V_{i2}$ (resp., $V_{21}$) but $(y, \sigma) \notin V_{i2}$ (resp., $V_{21}$). If $(V_{i2}, V_{21})$ is consistent then pairs $(V_{i2}', V_{21}') \in \text{MINSET}$ are removed from MINSET where $V_{i2}' \supseteq V_{i2}$ and $V_{21}' \supseteq V_{21}$. Then $(V_{i2}, V_{21})$ it is added to MINSET.

An enumeration procedure for pairs $(V_{i2}, V_{21})$ is not specified. However, the banker’s sequence [18] could be used.

When the algorithm is complete, any element of MINSET is a solution to Problem 7 (Theorem 3 of Rudie et al. [16]). One can verify that MINSET contains all solutions to Problem 7.

However, the complexity of the algorithm is in the worst-case exponential in the state and transition set cardinalities of $R_i$.

### 3.3.3 On the minimization of communication in networked systems with a central station

The setting of Rudie et al. [23] and Rudie et al. [16] is used to study a problem of minimal communication in Wang et al. [52]. This work considers $|I| \geq 2$ agents and an explicit DFA representation of the plant, $G$, is provided. Agents observe the transitions of $G$ and are required to communicate observed event occurrences to
one another to disambiguate certain pairs of states in $G$. However, there are several restrictions on the problem setting:

- the agents can only communicate bidirectionally with a coordinator;
- the agents and coordinator collectively can observe all event occurrences generated by $G$;
- event occurrences are communicated by agents to the coordinator in such a manner that the coordinator can precisely determine the current state of $G$;
- $G$ is acyclic except for self-loops on its states.

Let $\Sigma_c$ denote the set of events that the coordinator can observe. It is assumed that any event in $\Sigma$ is observed by the coordinator or one of the agents. That is, $\Sigma = \Sigma_c \cup \bigcup_{i \in I} \Sigma_{i,o}$.

As in Rudie et al. [23], transition-based communication policies are considered for each agent and for the coordinator. Each agent $i$ can communicate to the coordinator event occurrences labeling transitions that $i$ observes. We denote such a set by $COM_{ic}$ where $COM_{ic} \subseteq \{(x, \sigma) \in TR(G) : \sigma \in \Sigma_{i,o} \setminus \Sigma_c\}$. Likewise, the coordinator can communicate to agent $i$ event occurrences labeling transitions that the coordinator observes or are communicated to the coordinator by some other agent. We denote such a set by $COM_{ci}$ where $COM_{ci} \subseteq \{(x, \sigma) \in TR(G) : \sigma \notin \Sigma_{i,o} \land [\sigma \in \Sigma_c \lor \exists j \in I, (x, \sigma) \in COM_{jc}]\}$.

Given $COM_{ic}, COM_{ci}$, we can define how strings in $\mathcal{L}(G)$ appear to each agent and the coordinator. We use induction to define agent $i$’s information map $\theta^i : \mathcal{L}(G) \to \Sigma^*$
as follows. For the empty string $\varepsilon$, $\theta^i(\varepsilon) = \varepsilon$, and for all $s\sigma \in \mathcal{L}(G)$ with $\sigma \in \Sigma$

$$
\theta^i(s\sigma) = \begin{cases} 
\theta^i(s)\sigma & \text{if } \sigma \in \Sigma_{i,o} \lor (\delta(x_0, s), \sigma) \in \text{COM}_{ci} \\
\theta^i(s) & \text{otherwise.}
\end{cases}
$$

Similarly, the coordinator’s information map $\theta^c : \mathcal{L}(G) \rightarrow \Sigma^*$ is defined for $s\sigma \in \mathcal{L}(G)$ as

$$
\theta^c(s\sigma) = \begin{cases} 
\theta^c(s)\sigma & \text{if } \sigma \in \Sigma_c \lor (\delta(x_0, s), \sigma) \in \bigcup_{i \in I} \text{COM}_{ic} \\
\theta^c(s) & \text{otherwise.}
\end{cases}
$$

Feasibility of agent communication policies is required. However, the notion of feasibility considered Wang et al. [52] for agents is weaker than (3.7) since the transition function of $G$ is not total. Formally, agent $i$’s communication policy $\text{COM}_{ic}$ is feasible if

$$
\forall s\sigma, s'\sigma \in \mathcal{L}(G), \theta^i(s) = \theta^i(s') \Rightarrow [(\delta(x_0, s), \sigma) \in \text{COM}_{ic} \Leftrightarrow (\delta(x_0, s'), \sigma) \in \text{COM}_{ic}].
$$

(3.12)

The coordinator and each agent have their own respective state disambiguation goals. The coordinator is required to know precisely the current state of $G$. Formally,

$$
\forall s, s' \in \mathcal{L}(G), \theta^c(s) = \theta^c(s') \Rightarrow \delta(x_0, s) = \delta(x_0, s').
$$

(3.13)

This condition is reminiscent of validity (3.8) of agent communication policies considered in Rudie et al. [23]. That (3.13) is satisfied implies that we do not have to be concerned about the feasibility of the coordinator’s communications since the coordinator can precisely determine the current state of $G$.

Each agent is required to distinguish between states in a pair of a given state-based specification $T^i_{\text{spec}} \subseteq X \times X$. Formally, given $T^i_{\text{spec}}$,

$$
\forall s, s' \in \mathcal{L}(G), \theta^i(s) = \theta^i(s') \Rightarrow (\delta(x_0, s), \delta(x_0, s')) \notin T^i_{\text{spec}}.
$$

(3.14)
CHAPTER 3. LITERATURE REVIEW

80

We can characterize feasibility of agent $i$’s communication policy in terms of a state-based specification $T_{feas}^i = \{(x, y) \in X \times X : \exists \sigma \in \Sigma_i \setminus \Sigma_c, (x, \sigma) \in COM_{ic} \land (y, \sigma) \notin COM_{ic}\}$. In order to satisfy feasibility, it is required that agent $i$ distinguish states of pairs in $T_{feas}^i$ (Lemma 2 of Wang et al. [52]). Formally,

$$\forall s, s' \in L(G), \theta^i(s) = \theta^i(s') \Rightarrow (\delta(x_0, s), \delta(x_0, s')) \notin T_{feas}^i.$$  \hspace{1cm} (3.15)

The problem considered in Wang et al. [52] is the following.

**Problem 8.** Given $G$, $T_{spec}^i$ find communication policies $COM_{ic^*}^i$, $COM_{ci^*}$ for all $i \in I$ such that

- $C1$ (3.13) is satisfied under $COM_{ic^*}^i$;
- $C2$ for all $i \in I$, (3.15) is satisfied under $COM_{ic^*}^i$, $COM_{ci^*}$;
- $C3$ for all $i \in I$, (3.14) is satisfied under $COM_{ic^*}^i$, $COM_{ci^*}$;
- $C4$ $COM_{ic^*}^i, COM_{ci}^*$ are minimal with respect to $C1 - C3$. That is, there does not exist $COM_{ic}^i$, $COM_{ci}^i$ which satisfies $C1 - C3$, $COM_{ic}^i \subseteq COM_{ic}^*$, $COM_{ci}^i \subseteq COM_{ci}^*$ and $COM_{ic}^i \subseteq COM_{ic}^j \subseteq COM_{ic}^*$ or $COM_{ci}^j \subseteq COM_{ci}^*$ for some $j \in I$.

Note that there may exist multiple, incomparable solutions to Problem 8.

Criterion $C1$ of Problem 8 is trivially satisfied. It is simply required that the coordinator observe any transition of $G$ between two different states of $G$. For all such transitions $(x, \sigma) \in TR(G)$ not locally observed by the coordinator (i.e., $\sigma \notin \Sigma_c$), we choose an agent $i \in I$ arbitrarily where $\sigma \in \Sigma_{i,o}$ and force agent $i$ to communicate $\sigma$ to the coordinator on occurrence of $(x, \sigma)$ (i.e., add $(x, \sigma)$ to agent $i$’s policy $COM_{ic}$). This method satisfies $C1$ and defines $COM_{ic}^*$, the agent communication policies in a solution to Problem 8 (Lemma 1 of Wang et al. [52]).
It remains to define the coordinator’s communications to each agent \( i \) (i.e., \( \text{COM}_{ci}^* \) for all \( i \in I \)) in order to satisfy C2 - C4 of Problem 8. This asymmetry in the solution approach is similar to the asymmetry in the solution approach of Rudie et al. [23] where we fix agent 2’s communication policy and it remains to compute agent 1’s communication policy.

One might conjecture that an approach for computing \( \text{COM}_{ci} \) is for the coordinator to first communicate to \( i \) transitions \( \text{COM}_{ci}^\text{all} = \{ (x, \sigma) \in TR(G) : \sigma \notin \Sigma_{i,o} \land \delta(x, \sigma) = y \land x \neq y \} \) then remove communication of transitions from \( \text{COM}_{ci}^\text{all} \), one by one, while C2 and C3 is satisfied. Unfortunately, due to the lack of monotonicity of transition-based communications, this procedure will not work in general for computing solutions to Problem 8. Also, it is impractical to search all subsets of \( \text{COM}_{ci}^\text{all} \) and select a minimal subset that satisfies C2 - C4 (see explanation at end of Section 2 of Wang et al. [52]). So a different approach is proposed in Wang et al. [52] where we search through all subsets of transitions in \( \text{COM}_{ci}^\text{all} \) from a given state \( x \in X \). By considering states of \( G \) in a specific order, policies \( \text{COM}_{ci}^* \) are produced which satisfy C2 - C4.

Algorithm Min-Com-CS of Wang et al. [52] is proposed as a general approach for computing \( \text{COM}_{ci}^* \). Initially, the coordinator communicates to agent \( i \) all transitions in \( \text{COM}_{ci} = \text{COM}_{ci}^\text{all} \). Under \( \text{COM}_{ci} \), conditions (3.14) and (3.15) are trivially satisfied. The approach taken to computing \( \text{COM}_{ci}^* \) is done on a state-by-state basis. From a given state \( x \in X \) we remove transitions from \( \text{COM}_{ci} \) which originate at state \( x \) while satisfying conditions (3.14) and (3.15). That is, we remove a set of transitions \( \text{RT}_{ci}^*(x) \) from set \( \text{COM}_{ci}(x) = \text{COM}_{ci} \cap \{ (x, \sigma) \in TR(G) \} \). Furthermore, \( \text{RT}_{ci}^*(x) \) is
as large as possible (with respect to set containment). We defer detailing the computation of $RT_{ci}^*(x)$. Then, the set of transitions from state $x$ to be communicated to agent $i$ is $COM_{ci}^*(x) = COM_{ci}(x) \setminus RT_{ci}^*(x)$. Communication policy $COM_{ci}^*$ is defined as $\bigcup_{x \in X} COM_{ci}^*(x)$.

In Algorithm Min-Com-CS we do not define communications from states in an arbitrary order. A specific partial order on $X$ is considered, corresponding to a topological sort on $X$. Recall that $G$ is acyclic with the exception of self-loops on states. We denote a leaf state of $G$ as any state $x$ which has no transitions to a state $y$ where $x \neq y$. First, we define communications $COM_{ci}^*(x)$ from leaf states $x$ of $G$. When the communications from a state $x \in X$ have been defined we add $x$ to a set $C$. Second, we recursively define communications from the remaining states $x \in X$ such that $x \notin C$ and, for all $(x, \sigma) \in TR(G)$, if $\delta(x, \sigma) = y$ and $y \neq x$ then $y \in C$.

Algorithm Min-Com-CS, as described above, computes a solution to Problem 8 (Theorem 1 of Wang et al. [52]).

While set $RT_{ci}^*(x)$ could be computed by testing all subsets of $COM_{ci}(x)$, a more effective approach is proposed. Set $RT_{ci}^*(x)$ can be computed using a partition of the set of strings originating from state $x$. Such a partition is detailed in Section 4 of Wang et al. [52]. Algorithm CS-3.2 is provided for computing $RT_{ci}^*(x)$ from state $x \in X$. Its correctness is established (Theorem 2 of Wang et al. [52]) and its complexity is $O(|\Sigma|^3 \cdot |X|^2)$ (Theorem 3 of Wang et al. [52]).

When using Algorithm CS-3.2 for computing $RT_{ci}^*(x)$, Algorithm Min-Com-CS is in $O(|I| \cdot |\Sigma|^3 \cdot |X|^3)$.

Note that, for a given assignment of communications from the agents to the coordinator, the computation of a minimal communication policy from the coordinator
to any agent is uniquely determined. The set of transitions of $G$ that the coordinator communicates to an agent can be changed by assigning different agents to communicate transitions to the coordinator.

### 3.3.4 Minimization of communication of event occurrences in acyclic discrete event systems

The approach of Wang et al. [52] is generalized in Wang et al. [51]. In Wang et al. [51] the central coordinator is removed from the problem setting and, instead, it is assumed that agents can communicate with one another directly. However, it is not necessarily the case that any two agents can communicate with each other directly. Communication between agents may occur indirectly via other agents. Specifically, it is assumed that the topology that governs which agents communicate directly is a strongly connected topology. The problem setting inherits many of the characteristics of the problem setting of Wang et al. [52]. Plant $G$ is assumed to be acyclic with the exception of self-loops on states. It is assumed that any event in $\Sigma$ is observed by one of the agents. That is, $\Sigma = \bigcup_{i \in I} \Sigma_{i,e}$. Events are communicated between agents without delay, even if communication must pass indirectly (due to topology limitations) through a set of intermediate agents.

As in previous works, transition-based communication policies are considered. Agent $i$’s communications to agent $j$ is specified by transition set $COM_{ij} \subseteq TR(G)$. It is redundant for agent $i$ to communicate anything to itself so $COM_{ii} = \emptyset$. We use induction to define agent $i$’s information map $\theta^i : L(G) \rightarrow \Sigma^*$ as follows. For the
empty string $\varepsilon$, $\theta_i(\varepsilon) = \varepsilon$, and for all $s\sigma \in \mathcal{L}(G)$ with $\sigma \in \Sigma$

$$\theta_i(s\sigma) = \begin{cases} 
\theta^i(s)\sigma & \text{if } \sigma \in \Sigma_{i,o} \lor (\delta(x_0, s), \sigma) \in \bigcup_{j \in I \setminus \{i\}} \text{COM}_{ji} \\
\theta^i(s) & \text{otherwise.}
\end{cases}$$

Given $\text{COM}_{ij}$ for all $i, j \in I$, we can define a communication matrix $\text{COM}$ where the $(i, j)$ entry of $\text{COM}$ contains $\text{COM}_{ij}$.

We mentioned that the communication topology is strongly connected. Specifically, let $\mathcal{G}(\text{COM})$ denote a directed graph whose vertices are agents and where edge $(i, j)$ is an edge in $\mathcal{G}(\text{COM})$ if $\text{COM}_{ij} \neq \emptyset$. Graph $\mathcal{G}(\text{COM})$ is the communication topology. It is required that $\text{COM}$ be defined such that, for all $i, j \in I$, $j$ is reachable from $i$ in $\mathcal{G}(\text{COM})$.

However, this does not mean that $i$ always communicates to $j$ the event occurrence labeling a given transition $(x, \sigma) \in \mathcal{TR}(G)$ when considering a given $\text{COM}$. Given $\text{COM}$ and transition $(x, \sigma) \in \mathcal{TR}(G)$, the restricted communication topology is a directed graph $\mathcal{G}((x, \sigma), \text{COM})$ whose vertices are agents and edge $(i, j)$ is in the graph if $(x, \sigma) \in \text{COM}_{ij}$. Graph $\mathcal{G}((x, \sigma), \text{COM})$ is a sub-graph of $\mathcal{G}(\text{COM})$. The set of chains of $\mathcal{G}((x, \sigma), \text{COM})$ is denoted by $\mathcal{C}((x, \sigma), \text{COM})$. For chain $c \in \mathcal{C}((x, \sigma), \text{COM})$, $c(i, j)$ means that $c$ starts at $i$ and ends at $j$.

Feasibility of agent communication policies are required. However, the notion of feasibility considered also takes into account the restricted communication topology. Formally, communication matrix $\text{COM}$ is feasible if

$$\forall s\sigma, s'\sigma \in \mathcal{L}(G), \theta_i(s) = \theta_i(s') \Rightarrow [(\delta(x_0, s), \sigma) \in \text{COM}_{ij} \Leftrightarrow (\delta(x_0, s'), \sigma) \in \text{COM}_{ij}].$$

$$\wedge \quad (3.16)$$

$$\forall (x, \sigma) \in \mathcal{TR}(G), (x, \sigma) \in \text{COM}_{ij} \Rightarrow [\sigma \in \Sigma_{i,o} \lor \exists c(k, i) \in \mathcal{C}((x, \sigma), \text{COM}), \sigma \in \Sigma_{k,o}]].$$

Each agent is required to distinguish between states in a pair of a given state-based
specification $T_{\text{spec}}^i \subseteq X \times X$. Formally, we require (3.14).

The problem considered in Wang et al. [51] is the following.

**Problem 9.** Given $G$, $T_{\text{spec}}^i$ find communication policies $COM^*_{ij}$ for all $i, j \in I$, with $COM^*$ denoting the matrix constructed from $COM^*_ij$, such that

1. $C1$ [3.16] is satisfied under $COM^*$;

2. $C2$ for all $i \in I$, (3.14) is satisfied under $COM^*$;

3. $COM^*$ is minimal with respect to $C1$ - $C2$. Minimality is defined as usual by set containment in $TR(G)$.

Note that there may exist multiple, incomparable solutions to Problem 9.

Suppose each agent communicates each event that it locally observes to its neighbours in $G(COM)$ and forwards all communications that it receives to its neighbours in $G(COM)$ (i.e., for all $(x, \sigma) \in TR(G)$, $G((x, \sigma), COM) = G(COM)$). We denote the corresponding communication matrix by $COM^{all}$. Under $COM^{all}$ each agent can precisely determine the current state of $G$. Then $C1$ and $C2$ of Problem 9 are satisfied under $COM^{all}$. Thus a solution to Problem 9 exists.

Algorithm **MIN-COM-GEN** of Wang et al. [51] is proposed for computing solutions to Problem 9. Algorithm **MIN-COM-GEN** is a generalization of Algorithm **MIN-COM-CS** of Wang et al. [52]. The initial communication matrix is $COM^{all}$. From leaf states to the initial state of $G$, states are considered one by one. For each state $x$, for each $i, j \in I$, compute $RT^*_ij(x)$, a subset of the transitions $COM_{ij}$ in $COM$ originating from $x$ (i.e., a subset of transitions in $COM_{ij} \cap \{(x, \sigma) \in TR(G)\}$). Set $RT^*_ij(x)$ is defined such that $C1$ - $C2$ of Problem 9 is satisfied when agent $i$ communicates transitions in $COM_{ij} \setminus RT^*_ij(x)$. Furthermore, there does not exist a set $RT^*_ij(x) \supset RT^*_ij(x)$,
$RT_{ij}(x) \subseteq COM_{ij} \cap \{ (x, \sigma) \in TR(G) \}$ such that C1 - C2 is satisfied when agent $i$ communicates transitions in $COM_{ij} \setminus RT_{ij}(x)$. Criteria C1 and C2 can be verified by computing the set of state pairs of $G$ indistinguishable to agent $j$ given the set of transitions that $j$ observes. Algorithm CLUSTER-TABLE of Wang et al. [50] can be used for this purpose.

Algorithm Min-Com-Gen computes a solution to Problem 9. However, Min-Com-Gen is not efficient in general, since it involves checking all possible subsets of $COM_{ij} \cap \{ (x, \sigma) \in TR(G) \}$ to find $RT^*_{ij}(x)$. For the particular case of a central coordinator, one avoids checking all possible subsets of $COM_{ij} \cap \{ (x, \sigma) \in TR(G) \}$ by exploiting partitions of the strings generated from state $x$ [52].
Chapter 4

Computing Deterministic Finite Automaton Representations of Sensor Activation Maps in Discrete-Event Systems

4.1 Introduction

Note that this chapter is derived from [34].

The behaviour of many critical dynamical systems require monitoring by external agents. The behaviour of such systems is monitored so that critical failures in the system may be diagnosed or anticipated prior to their occurrence. Control of such systems can be conducted in order to prevent failures from occurring in cases when failures can be anticipated in advance and when actuators for the system are available.
The behaviour of these systems is reported to agents via sensors embedded in the system. It may be the case that reading information from the sensors or keeping them active for a prolonged duration is costly. Using sensors may be costly in cases where each sensor only has a limited energy supply, or because limited bandwidth is available for transmission of readings from sensors to the agent or for security purposes in situations where the agent wishes to be undetected while taking measurements of the system. In all these cases it is important to minimize reading information from sensors or minimize the duration over which the sensors are turned on if the agent can turn sensors on and off.

Sensor activation problems can be modeled in many different ways, depending on the model of the system, the particular problem at hand (e.g., control, failure diagnosis, failure prognosis), the nature of the information reported by sensors, and the cost function to be minimized. In this chapter we consider that the system is specifically a discrete-event system (DES): a system with a discrete state-space and event-driven dynamics. In such systems, events occur sequentially and asynchronously. Sensor activation problems for discrete-event systems have been considered previously ([45, 2, 53, 49, 37, 12, 54, 48, 3]). A comprehensive overview of sensor activation problems can be found in our technical report [30] and literature review in Chapter 3.

We consider discrete-event systems which can be modeled by finite automata. For such systems the state-space is finite and the set of events which cause the system to transition from one discrete state to another is finite. As these systems can be modeled by finite automata, state-transition representations of such systems exist where events label the transitions of the system model. The individual sensors used by an agent
are associated with individual events in the system. Turning an event sensor on or off depends on the particular trajectory of the system. Specifically, given an automaton state-transition representation of the system, we consider that turning an event sensor on or off depends on the particular transition of the system representation: a sensor for an event labeling a particular transition of the system is always on whenever the transition is encountered or is always off whenever the transition is encountered. When the turning of event sensors on and off depends on the transitions of the system representation we say that the sensor activation decisions by the agent are dictated by a sensor activation policy ([54]) defined over the transitions of the system. Depending on the problem at hand (e.g., control, failure diagnosis, failure prognosis), one sensor activation policy may be preferable over another or two sensor activation policies may be incomparable, depending on the cost function for active sensors or the relative cost of sensor activation policies.

In this chapter, we do not consider the minimization of sensor activations. This topic has been explored in previous papers ([2, 12, 37, 45, 48, 49, 53, 54]). Instead, we consider the computation of maps from sequences of event observations to sensor activation decisions corresponding to given sensor activation policies. For a given sensor activation policy defined over the transitions of the system automaton model, such a computation typically involves the determinization of nondeterministic finite automata (NFA). It is well known that the determinization of an NFA is, in the worst-case, exponential in the state-space of the NFA. In this chapter, we consider specific classes of sensor activation policies and demonstrate procedures for computing these maps which are polynomial in the size of the representation of the system.
The procedures we provide do not involve determinization of NFA. We do not investigate determinization of NFA defined by the systems and sensor activation policies considered. This is an open topic.

We consider sensor activation policies which satisfy various notions of “feasibility”. Informally, a sensor activation policy is feasible if any two states of the system which are indistinguishable under the sensor activation policy (i.e., the two states are reached by strings that appear identical) are followed by the same sensor activation decisions for certain subsets of observable events. Feasibility was originally introduced in [23] in the context of communication policies between two agents in discrete-event systems. The most general notion of feasibility for sensor activation policies is introduced in [54]. The notions of feasibility that we consider are strictly stronger than the notion considered there. We introduce notions of feasibility of increasing generality.

The organization of the chapter is as follows. A preliminaries section introducing the requirements for understanding the remainder of the chapter is provided in Section 4.2. In Section 4.3 we consider sensor activation policies which satisfy a very strong notion of feasibility and, for such sensor activation policies, demonstrate how the automaton representation of the system itself can be used for determining the set of sensors to be activated following a sequence of observable events. This notion of feasibility is generalized in Section 4.4. For policies satisfying this more general notion of feasibility, we demonstrate that a very coarse estimate of the true state of the system may be used in the computation of a map from observed event sequences to sensor activation decisions and, furthermore, the computation is polynomial in the state-space and event set cardinalities of the system. We provide an example illustrating the computational benefit of our approach versus that of determinizing
NFA in computing such a map. A further generalization of sensor activation policies is considered in Section 4.5. For this more general class, it may not be the case that the coarse estimate of the true state of the system introduced in Section 4.4 can be used for computing maps from observed event sequences to sensor activation decisions. In fact, we demonstrate that determining if the coarse estimate can be used is PSPACE-complete. When it can be used, we demonstrate a procedure polynomial in the state-space and event set cardinalities of the system for computing maps from observed event sequences to sensor activation decisions. Finally, we summarize the results of the chapter in Section 4.6. Some of the results in this chapter originally appeared in earlier conference chapters, without accompanying proofs ([29, 28]).

4.2 Preliminaries

Here some preliminary concepts and notation specific to this chapter are introduced. We refer the reader to Chapter 2 for a summary of concepts and notation used in this chapter and other chapters of the dissertation.

Recall that an NFA, $G$, is denoted by a tuple $G = (X, \Sigma, \xi, x_0)$ where $X$ is the nonempty set of states, $\Sigma$ is the alphabet, $x_0$ is the initial state, and $\xi : X \times (\Sigma \cup \{\varepsilon\}) \to 2^X$ is the (state-)transition function. In this chapter, we consider a DES modeled by a DFA $G = (X, \Sigma, \xi, x_0)$. The set of transitions of $G$ is defined as $TR(G) := \{(x, e) \in X \times \Sigma : \xi(x, e)\!\}$. From the perspective of an agent which observes events generated by $G$, the set of events $\Sigma$ can be partitioned into $\Sigma_o$, the set of events whose occurrences can be observed by the agent, and $\Sigma_{uo}$, the set of events whose occurrences cannot be observed by the agent. Associated with each observable event is a sensor that can be used to detect occurrences of the event.
When the sensor is activated (i.e., the event’s sensor is on) any occurrence of the event is detected by the agent. Otherwise, when the sensor is deactivated (i.e., the event’s sensor is off) any occurrence of the event is not detected by the agent.

Next the sensor activation model of Wang et al. [54], which is used in this chapter, is recalled.

When to activate event sensors is described by a sensor activation map $\omega : \mathcal{L}(G) \to 2^{\Sigma_o}$. Specifically, for a string $s \in \mathcal{L}(G)$, $\omega(s)$ is the subset of observable events corresponding to the sensors that are active after $s$. Given sensor activation map $\omega$, we use induction to define the corresponding information map $\theta^\omega : \mathcal{L}(G) \to \Sigma_o^*$ as follows. For the empty string $\varepsilon$, $\theta^\omega(\varepsilon) = \varepsilon$, and for all $s, se \in \mathcal{L}(G)$ with $e \in \Sigma$

$$
\theta^\omega(se) = \begin{cases} 
\theta^\omega(s)e & \text{if } e \in \omega(s) \\
\theta^\omega(s) & \text{otherwise.}
\end{cases}
$$

In words, after the occurrence of $s$, the next event $e$ is seen or observed by the agent when it occurs after $s$ if and only if the sensor for $e$ is activated for the agent after the occurrence of $s$. The information map $\theta^\omega$ plays a comparable role to the projection map $P$ of standard partially-observed discrete-event systems. That is, $\theta^\omega(s)$ is the event sequence that is observed by the agent upon occurrence of $s$. The difference is that an event $e$ in $s$ is either always observed under $P$ or never observed whereas, with dynamic sensor activations, whether $e$ is observed in $\theta^\omega(s)$ will depend on where in $s$ it lies.

It is important to note that not all arbitrary sensor activation maps $\omega$ will be “feasible” based on the information available to the agent. To guarantee feasibility, it is required that any two strings of events that are indistinguishable to the agent must be followed by the same activation decision for every event. Namely, $\omega$ must be
“compatible” with the information map $\theta^\omega$ that is built from it. Formally, $\omega$ is said to be feasible if

$$\forall e \in \Sigma)(\forall s,e \in L(G)) \theta^\omega(s) = \theta^\omega(s') \quad (4.1)$$

$$\Rightarrow [e \in \omega(s) \iff e \in \omega(s')]$$

The above definitions of $\omega$ and $\theta^\omega$ are language-based. In this chapter we do not consider arbitrary language-based sensor activation maps. Instead, we consider sensor activation maps that are defined over the transitions of a given automaton $G$. For such maps, any two strings leading to the same state of $G$ are followed by the same sensor activation decision for every event. In Wang et al. [54] such sensor activation maps are called implementable. Formally, $\omega$ is implementable with respect to $G$ if

$$\forall s,s' \in L(G), \xi(x_0,s) = \xi(x_0,s') \Rightarrow \omega(s) = \omega(s').$$

In Wang et al. [54] implementable sensor activation maps were considered in order to restrict the solution space of the problem considered. By refining the state-transition structure of $G$, finer solutions to the problem they considered can be computed.

For implementable sensor activation maps, we can associate the activation of sensors with the transitions in $G$: the event associated with each transition in $TR(G)$ is either sensed (activated) by the agent, or not. Given an implementable sensor activation map $\omega$, the set of transitions sensed by the agent through sensor activations using $\omega$ can be defined. This set is denoted by $\Omega \subseteq TR(G)$. Here $(x,e) \in \Omega$ means that $\forall s \in L(G) \xi(x_0,s) = x \Rightarrow e \in \omega(s)$. We call $\Omega$ a (sensor activation) policy.

Note that $\omega$ can be obtained from $\Omega$ as follows:

$$\omega(s) = \{ e \in \Sigma_o : (\xi(x_0,s), e) \in \Omega \}. \quad (4.2)$$
It is not difficult to see that, when \( \omega \) is obtained from \( \Omega \) by (4.2), the following holds:

\[
\forall s \in L(G), \forall e \in \Sigma, e \in \omega(s) \Rightarrow se \in L(G) \tag{4.3}
\]

Information map \( \theta^\Omega \) is used to denote the information map \( \theta^\omega \) when such \( \omega \) is derived from \( \Omega \) by (4.2).

Policy \( \Omega \) is feasible if the corresponding \( \omega \) is feasible. It is not difficult to see that \( \Omega \) is feasible if

\[
(\forall e \in \Sigma)(\forall s, s' \in L(G)) \theta^\Omega(s) = \theta^\Omega(s') \tag{4.4}
\]

\[
\Rightarrow [(\xi(x_0, s), e) \in \Omega \iff (\xi(x_0, s'), e) \in \Omega].
\]

For a given sensor activation policy \( \Omega \), we define the unobserved reach of state \( x \in X \) under \( \Omega \) as the set of states that can be reached from \( x \) via “unobserved” transitions (i.e., transitions that do not exist in \( \Omega \)).

Given plant \( G \) and policy \( \Omega \), we can construct the observer automaton which maps sequences of observed events to sensor activation decisions by relabeling those transitions in \( G \) not in \( \Omega \) by \( \varepsilon \) and applying the subset construction. This results in the observer automaton, denoted by DFA \( DET_\Omega(G) \). Given an observed event sequence \( s \), \( DET_\Omega(G) \) can then be used to compute the sensor activation decisions following \( s \) by taking the union of all events labelling transitions in \( DET_\Omega(G) \) from the state in \( DET_\Omega(G) \) reached by \( s \).

Next we introduce the equivalence class of a state in \( G \) under a policy \( \Omega \). Given DFA \( G \) and policy \( \Omega \), we say that states \( x, x' \in X \) are indistinguishable if there exist \( s, s' \in L(G) \) such that \( \xi(x_0, s) = x, \xi(x_0, s') = x' \) and \( \theta^\Omega(s) = \theta^\Omega(s') \). Given plant \( G \) and policy \( \Omega \), we can compute the set of pairs of indistinguishable states of \( X \).
denoted by $T_\Omega$, using the product of Sears et al. \[31\] in $O(|X|^2 \cdot |\Sigma| + |X|^3)$. This computation is studied in Chapter 5.

We can interpret $T_\Omega \subseteq X \times X$ as a binary relation. It is not difficult to see that $T_\Omega$ is reflexive and symmetric but not necessarily transitive. For example, consider the plant in Figure 4.1. We consider $\Omega$ to consist of those transitions whose event label is boxed. Specifically, policy $\Omega = \{(2, \alpha), (3, \alpha), (3, \beta), (4, \beta)\}$. One can verify that $(5, 6) \in T_\Omega$ and $(6, 7) \in T_\Omega$, but $(5, 7) \notin T_\Omega$.

![Figure 4.1: Plant and policy demonstrating that $T_\Omega$ is not necessarily transitive.](image)

Given a set $X' \subseteq X$ and state $x \in X$, let $X'T_\Omega x$ (respectively, $xT_\Omega X'$) denote that $\forall x' \in X', x'T_\Omega x$ (resp., $\forall x' \in X', xT_\Omega x'$).

The transitive-closure of $T_\Omega$, denoted by $T_\Omega^*$, is an equivalence relation on $X$. We consider the equivalence class of a state $x \in X$, denoted by $[x]$, to be defined using this equivalence relation: $[x] = \{z \in X : xT_\Omega^* z\}$. We will find the following characterization of $z \in [x]$ using $T_\Omega$, rather than $T_\Omega^*$, useful in the proof details of results in the sequel. For state $z \in X, z \in [x]$ if and only if

$$\exists n \geq 0, \exists x^0, x^1, x^2, \ldots, x^n \in X, x^0 = x, \quad (4.5)$$

$$x^1T_\Omega x^0 \land x^2T_\Omega x^1 \land \ldots \land zT_\Omega x^n.$$  

If $z \in [x]$ then $x \in [z]$ by (4.5) and it follows that $[x] = [z]$.  

---

\[31\] Sears, et al.
Though the topic of computing policies for purposes of state disambiguation is considered in Wang et al. [54], it is not indicated how the computed policies can be used to construct a map from observed event sequences to sensor activation decisions. One could infer that, for policy $\Omega$ which satisfies (4.4), this is done by constructing observer automaton $DET_\Omega(G)$ then taking the union of all events labelling transitions in $DET_\Omega(G)$ from the state in $DET_\Omega(G)$ reached by the observed event sequence. However, constructing $DET_\Omega(G)$ using NFA-to-DFA conversion is, in the worst-case, exponential in $|X|$.

In this chapter we consider policies $\Omega$ where there exist procedures for computing a map from observed event sequences to sensor activation decisions that are polynomial in the size of $G$. In order to achieve this we consider policies that satisfy notions of feasibility that are stronger than (4.4). Specifically, we consider policies where, for all $(x, y)$ in $T_\Omega$, $x$ and $y$ are followed by exactly the same sensor activation decisions. For such policies, we show that the quotient of $G$ using relation $T^{*}_\Omega$ can be used to implement the aforementioned map. We also consider a more general class of policies which also permits using this quotient automaton for implementing the map.

4.3 Computing decisions for policies satisfying a strong notion of feasibility

In this section we consider policies that satisfy a stronger notion of feasibility than (4.4). The notion that we consider is the following.

$$(\forall e \in \Sigma_o)(\forall s, s' \in L(G)) \theta^\Omega(s) = \theta^\Omega(s') \Rightarrow [(\xi(x_0, s), e) \in \Omega \iff (\xi(x_0, s'), e) \in \Omega]$$

(4.6)
That (4.6) is satisfied by a policy implies that if two states are indistinguishable under the policy then an event sensor is activated following one state if and only if it is activated following the other state. For policies satisfying (4.4) this is only required of the intersection of events that follow both states, not their union as (4.6) requires.

We use the following example to illustrate the differences between policies satisfying (4.6) and those satisfying (4.4). Consider the plant in Figure 4.2. We consider \( \Omega \) to consist of those transitions whose event label is boxed. Specifically, policy \( \Omega = \{(x_1, e_1), (x_2, e_2), (x_4, e_1)\} \). One can verify that this policy satisfies (4.4). However, policy \( \Omega \) does not satisfy (4.6). One can verify that \( \theta^\Omega(e_1) = \theta^\Omega(e_2e_1) = e_1 \), resulting in states \( x_2 \) and \( x_5 \) being indistinguishable. An agent which observes string \( e_1 \) using policy \( \Omega \) knows that if the plant is in state \( x_2 \) then the sensor for \( e_2 \) needs to be activated. However, if the plant is in state \( x_5 \) then the sensor for \( e_2 \) does not necessarily have to be activated. This is permitted for purposes of satisfying (4.4). However, it is not permitted if the policy is to satisfy (4.6).

![Figure 4.2: Plant and policy which satisfies (4.4) but does not satisfy (4.6).](image)

It is not difficult to see that if policy \( \Omega \) satisfies (4.6) then map \( \omega \) derived from \( \Omega \) by (4.2) satisfies the following notion of feasibility for language-based sensor activation.
maps which is stronger than (4.1).

\[
(\forall e \in \Sigma_o)(\forall s, s' \in L(G)) \theta^\omega(s) = \theta^\omega(s') \quad (4.7)
\]

\[
\Rightarrow [e \in \omega(s) \Leftrightarrow e \in \omega(s')].
\]

For policies satisfying (4.6), we effectively demonstrate that if \( s \in L(G) \) then \( \theta^\omega(s) \in L(G) \). Furthermore, \( s \) and \( \theta^\omega(s) \) are indistinguishable and are both followed by the same sensor activation decisions. This allows one to compute sensor activation decisions following a sequence of observed events easily: simply transition from the initial state of \( G \) to the state reached by the observed event sequence then activate a sensor for an event from the reached state if and only if the transition labeled by the event is in \( \Omega \). In this way, \( G \) is used directly for mapping observations to sensor activation decisions, avoiding the need to compute the observer automaton for purposes of determining sensor activation decisions.

The following is dedicated to justifying the claims made above. The formal details focus on using a state’s equivalence class for determining sensor activation decisions. It is proven that for all \( s \in L(G) \), \([\xi(x_0, s)] = [\xi(x_0, \theta^\omega(s))] \). This allows for sensor activation decisions to be based on the state in \( G \) reached by \( \theta^\omega(s) \).

First we prove some results regarding state equivalence classes for policies satisfying (4.6). We have the following immediate result on sensor activation decisions following states in the same equivalence class for policies satisfying (4.6).

**Lemma 1.** Given plant \( G = (X, \Sigma, \xi, x_0) \) and policy \( \Omega \subseteq TR(G) \) which satisfies (4.6), \( \forall x, z \in X, \forall e \in \Sigma_o, \) if \( [x] = [z] \) then \( (x, e) \in \Omega \Leftrightarrow (z, e) \in \Omega \).

**Proof.** Follows directly by definition of state equivalence class in (4.5) and that \( \Omega \)

\footnote{By contrast, in standard partially-observed discrete-event systems, if a string \( s \) is in \( L(G) \), this does not imply that \( P(s) \) is in \( L(G) \)}
From this result we have the following lemma which demonstrates that it suffices for an agent to keep track of a state’s equivalence class for determining sensor activation decisions.

**Lemma 2.** Given plant $G = (X, \Sigma, \xi, x_0)$ and policy $\Omega \subseteq TR(G)$ which satisfies \((4.6)\), $\forall x, z \in X, \forall e \in \Sigma_o$, if $[x] = [z]$ and $(x, e) \in \Omega$ then $[\xi(x, e)] = [\xi(z, e)]$.

**Proof.** The following holds by $[x] = [z]$ and the definition of $[x]$:
\begin{equation}
\exists n \geq 0, \exists x^0, x^1, x^2, \ldots, x^n \in X, x^0 = x, \quad (4.8)
\end{equation}

$x^1T_\Omega x^0 \land x^2T_\Omega x^1 \land \ldots \land zT_\Omega x^n$.

For all $i \in \{1, \ldots, n\}$, $\xi(x^i, e)$! and $(x^i, e) \in \Omega$ by Lemma \([4.2] e \in \Sigma_o, [x] = [x']$ and $(x, e) \in \Omega$. Similarly, $(z, e) \in \Omega$. The fact that $x^iT_\Omega x^{i-1}$ and the definition of $T_\Omega$ imply that there exists $s, s' \in L(G)$ where $\xi(x^0, s) = x^{i-1}$, $\xi(x^0, s') = x^i$ and $\theta_\Omega(s) = \theta_\Omega(s')$. Since $\{(x^{i-1}, e), (x^i, e)\} \subseteq \Omega$, we have $\theta_\Omega(se) = \theta_\Omega(s)e$ and $\theta_\Omega(s'e) = \theta_\Omega(s')e$. Then $\theta_\Omega(se) = \theta_\Omega(s'e)$ by the previous argument and $\theta_\Omega(s) = \theta_\Omega(s')$. This fact and $\xi(x^0, s) = x^{i-1}$, $\xi(x^0, s') = x^i$ imply that $\xi(x^i, e)T_\Omega \xi(x^{i-1}, e)$. Similarly, $\xi(z, e)T_\Omega \xi(x^n, e)$. By these results and \((4.8)\) the following holds:
\[\xi(x^1, e)T_\Omega \xi(x^0, e) \land \xi(x^2, e)T_\Omega \xi(x^1, e) \land \ldots \land \xi(z, e)T_\Omega \xi(x^n, e).\]

Then $\xi(z, e) \in [\xi(x, e)]$ by this fact, $x = x^0$ and \((4.5)\). It follows that $[\xi(x, e)] = [\xi(z, e)]$. \[\square\]

We use this result to prove the main result of this section. A corollary of the following theorem is that, to determine an agent’s sensor activation decision following string $s \in L(G)$, it suffices to compute the sensor activation decision of the state reached by $\theta_\Omega(s)$ from $x_0$ in $G$. 

\[\square\]
Theorem 1. Given plant $G = (X, \Sigma, \xi, x_0)$ and policy $\Omega \subseteq TR(G)$ which satisfies (4.6), $\forall s \in \mathcal{L}(G), [\xi(x_0, s)] = [\xi(x_0, \theta^\Omega(s))]$.

Proof. This proof follows by induction on the length of $\theta^\Omega(s)$.

When $|\theta^\Omega(s)| = 0$, $\theta^\Omega(s) = \varepsilon$. The fact that $\theta^\Omega(\varepsilon) = \varepsilon$ and the previous fact imply $\theta^\Omega(\theta^\Omega(s)) = \varepsilon$. Then $\theta^\Omega(s) = \theta^\Omega(\theta^\Omega(s))$. It follows that $\xi(x_0, s)T\Omega \xi(x_0, \theta^\Omega(s))$ by the previous argument and the definition of $T\Omega$. So $[\xi(x_0, s)] = [\xi(x_0, \theta^\Omega(s))]$ by this fact and (4.5).

Now for the inductive step. Consider when $|\theta^\Omega(s)| = n + 1$. Let $s = s_1es_2$ where $s_1, s_2 \in \Sigma^*$, $e \in \Sigma_o$ and $s_1e$ is the shortest prefix of $s$ where $\theta^\Omega(s_1)e = \theta^\Omega(s_1)e = \theta^\Omega(s)$. Since $|\theta^\Omega(s_1)| = n$ and by the inductive hypothesis it follows that $[\xi(x_0, s_1)] = [\xi(x_0, \theta^\Omega(s_1))]$. Since $\theta^\Omega(s_1)e = \theta^\Omega(s_1)e$ it must be that $(x_0, s_1), e \in \Omega$. Then $[\xi(x_0, \theta^\Omega(s_1)e)] = [\xi(x_0, s_1e)]$ by this fact, $e \in \Sigma_o$, $[\xi(x_0, s_1)] = [\xi(x_0, \theta^\Omega(s_1))]$ and Lemma 2. Since $\theta^\Omega(s_1)e = \theta^\Omega(s_1)e$ it follows that $[\xi(x_0, \theta^\Omega(s_1)e)] = [\xi(x_0, s_1e)]$. The fact that $\theta^\Omega(s) = \theta^\Omega(s_1es_2) = \theta^\Omega(s_1)e$ and $\theta^\Omega(s_1e) = \theta^\Omega(s_1)e$ imply $\theta^\Omega(s_1es_2) = \theta^\Omega(s_1e)$. It follows that $\xi(x_0, s_1es_2)T\Omega \xi(x_0, s_1e)$ by the previous argument and the definition of $T\Omega$. So $[\xi(x_0, s_1es_2)] = [\xi(x_0, s_1e)]$. Also, since $\theta^\Omega(s_1es_2) = \theta^\Omega(s_1e)$ it follows that $[\xi(x_0, \theta^\Omega(s_1es_2))] = [\xi(x_0, \theta^\Omega(s_1e))]$. Thus $[\xi(x_0, s_1es_2)] = [\xi(x_0, \theta^\Omega(s_1es_2))]$ by the previous two facts and $[\xi(x_0, \theta^\Omega(s_1e))] = [\xi(x_0, s_1e)]$. □

To reaffirm, for automaton $G$ and policy $\Omega$ which satisfies (4.6), it suffices to use $G$ directly for computing sensor activation decisions. Initially, from state $x_0$, the agent activates the sensors for events labeling transitions in $\Omega$ that originate from $x_0$. When the agent receives its first observation of an event $e_1 \in \Sigma_o$ (corresponding to one of the activated sensors), the set of sensors to be activated is redetermined.
from state $\xi(x_0, e_1)$, which must be defined due to (4.3). The set of sensors to be activated corresponds to the events labeling transitions in $\Omega$ that originate from $\xi(x_0, e_1)$. When the agent observes another event $e_2 \in \Sigma_o$ the set of sensors to be activated is redetermined from state $\xi(x_0, e_1 e_2)$. The set of sensors to be activated corresponds to the events labeling transitions in $\Omega$ that originate from $\xi(x_0, e_1 e_2)$. The updating of the set of active sensors continues in this manner for subsequent events that the agent observes.

For example, consider the plant and policy of Figure 4.3. We consider $\Omega$ to consist of those transitions whose event label is boxed. Specifically, policy $\Omega = \{(x_1, e_1), (x_2, e_2), (x_4, e_1), (x_5, e_2)\}$. One can verify that this policy satisfies (4.6). Consider string $e_2 e_1$. This string appears as $\theta^\Omega(e_2 e_1) = e_1$ to an agent using policy $\Omega$. The event sensors to be activated following $e_2 e_1$ are $\{e_2\}$. That is, $\omega(e_2 e_1) = \{e_2\}$. Note also that $\theta^\Omega(e_2 e_1) = e_1$ is a string in $L(G)$. Furthermore, $\omega(\theta^\Omega(e_2 e_1)) = \{e_2\}$, the same set of sensor activations as $\omega(e_2 e_1)$. By Theorem 1 and the fact that $\Omega$ satisfies (4.6), for any string $s \in L(G)$, $\theta^\Omega(s) \in L(G)$ and $\omega(\theta^\Omega(s)) = \omega(s)$.

![Figure 4.3: Plant and policy which satisfies (4.6) used to demonstrate how sensor activation decisions are computed.](image)

We note that this approach does not allow one to compute a state estimate of $G$. To do this, computation of the observer automaton from $G$ and $\Omega$ is required.
However, maintaining a state estimate is not required so long as agent decisions following indistinguishable pairs of strings or states are the same. For instance, consider problems of centralized control when the controllable, observable specification automaton $K$ is a subautomaton of $G$. When this is the case the enable/disable event control actions can be defined over the transitions of $G$. Observability ([17]) can be characterized as a state disambiguation condition defined over the states of $G$ ([50]): no two states where the control action from both states is different should be indistinguishable. For $\Omega$ satisfying (4.6) under which the state disambiguation condition is satisfied, it suffices to compute sensor activation and control decisions from $\xi(x_0, \theta^\Omega(s))$ instead of from the state estimate following $s$ computed using an observer automaton constructed from $G$ and $\Omega$.

In Section 4.4 we consider a generalization of (4.6). For policies that satisfy this more general condition, it is not necessarily the case that $\theta^\Omega(s) \in \mathcal{L}(G)$ when $s \in \mathcal{L}(G)$, as was the case for policies satisfying (4.6) considered in this section. For such policies we prove that it suffices to base sensor activation decisions on the state equivalence class $[\xi(x_0, s)]$, once again avoiding the need to apply the subset construction.

### 4.4 Computing decisions when active event sensors will eventually detect event occurrences

In Section 4.3 we considered policies which satisfy a strong notion of feasibility: if two states of $G$ are indistinguishable under that policy then (i) they must be followed by exactly the same sensor activation decisions and (ii) if the sensor for an event is
activated then the event must label outgoing transitions from both states. In this section we consider policies which satisfy a weaker notion of feasibility: (i) must be satisfied and, furthermore, if the sensor for an event is activated then the event must label transitions following some unobserved sequences of transitions from both states.

Formally, for the sensor activation maps considered in Section 4.3, if \( e \in \omega(s) \) then \( se \in \mathcal{L}(G) \). In this section we consider a generalization where instead if \( e \in \omega(s) \) then there exists \( u \in \Sigma^* \) such that \( sue \in \mathcal{L}(G) \) and \( \theta_\omega(s) = \theta_\omega(su) \). We refer to this as **eventual feasibility**. Note that eventual feasibility is still stronger than the feasibility condition of (4.4). The remainder of this section introduces a formal definition of eventual feasibility and demonstrates that, when eventual feasibility is satisfied by policy \( \Omega \), the quotient of \( G \) using relation \( T_\Omega^* \) implements the map from event observations to sensor activation decisions consistent with \( \Omega \). We also demonstrate how this automaton may be computed, and provide an example motivating the use of this construction.

As before, when \( \omega \) is implementable it can be implemented by a policy \( \Omega \). However, in this section we consider that \( \omega \) is defined from \( \Omega \) differently than by (4.2). For any string \( s \in \mathcal{L}(G) \), the set of event sensors activated following \( s \) (denoted by \( \omega'_\Omega(s) \)) correspond to those events which are observed following unobserved extensions of \( s \).

Formally, map \( \omega'_\Omega(s) \) is defined as follows:

\[
\omega'_\Omega(s) = \{ e \in \Sigma_o : \exists n \geq 0, \exists \sigma_1, \ldots, \sigma_n \in \Sigma \text{ such that } (\xi(x_0, s), \sigma_1) \notin \Omega \\
\wedge \ldots \wedge (\xi(x_0, s\sigma_1 \ldots \sigma_{n-1}), \sigma_n) \notin \Omega \wedge (\xi(x_0, s\sigma_1 \ldots \sigma_n), e) \in \Omega \}. \tag{4.9}
\]

To distinguish this sensor activation map from \( \omega \) derived by (4.2), we use \( \omega'_\Omega \) to denote the sensor activation map defined in (4.2) from \( \Omega \).

We suppose that the sensor activation maps derived using (4.9) satisfy a notion
of feasibility more general than [4.7]. We characterize this notion next and also
determine an equivalent characterization in terms of the policies from which these
maps are derived. By definition of [4.7], sensor activation map \( \omega'_{\Omega} \) is feasible if

\[
(\forall e \in \Sigma_o)(\forall s, s' \in \mathcal{L}(G)) \theta^{\omega'_{\Omega}}(s) = \theta^{\omega'_{\Omega}}(s')
\]

\[
\Rightarrow [e \in \omega'_{\Omega}(s) \Leftrightarrow e \in \omega'_{\Omega}(s')].
\]

This holds if and only if the following holds by definition of \( \omega'_{\Omega} \)

\[
(\forall e \in \Sigma_o)(\forall s, s' \in \mathcal{L}(G)) \theta^{\omega'_{\Omega}}(s) = \theta^{\omega'_{\Omega}}(s') \Rightarrow
\]

\[
[(\exists \sigma_1, \ldots, \sigma_n \in \Sigma \text{ such that } (\xi(x_0, s), \sigma_1) \notin \Omega \land \ldots \land
\]

\[
(\xi(x_0, s\sigma_1 \ldots s_{n-1}), \sigma_n) \notin \Omega \land (\xi(x_0, s\sigma_1 \ldots s_n), e) \in \Omega)
\]

\[
\Leftrightarrow
\]

\[
(\exists \sigma'_1, \ldots, \sigma'_m \in \Sigma \text{ such that } (\xi(x_0, s'), \sigma'_1) \notin \Omega \land \ldots \land
\]

\[
(\xi(x_0, s'\sigma'_1 \ldots s'_{m-1}), \sigma'_m) \notin \Omega \land (\xi(x_0, s'\sigma'_1 \ldots s'_m), e) \in \Omega)].
\]

For brevity we can express the above equivalently by the following by definition of
\( \theta^{\omega_{\Omega}} \) and by selecting \( u = \sigma_1 \ldots \sigma_n \) and \( u' = \sigma'_1 \ldots \sigma'_m \) appropriately for each \( e \in \Sigma_o \)
and pair of strings \( s, s' \in \mathcal{L}(G) \) where \( \theta^{\omega'_{\Omega}}(s) = \theta^{\omega'_{\Omega}}(s') \): We refer to this condition as

\[\text{eventual feasibility}.\]

\[
(\forall e \in \Sigma_o)(\forall s, s' \in \mathcal{L}(G)) \theta^{\omega'_{\Omega}}(s) = \theta^{\omega'_{\Omega}}(s') \Rightarrow
\]

\[
[(\exists u \in \Sigma^* \text{ such that } sue \in \mathcal{L}(G) \land \theta^{\omega_{\Omega}}(su) = \theta^{\omega_{\Omega}}(s) \land (\xi(x_0, su), e) \in \Omega)
\]

\[
\Leftrightarrow
\]

\[
(\exists u' \in \Sigma^* \text{ such that } s'u'e \in \mathcal{L}(G) \land \theta^{\omega_{\Omega}}(s'u') = \theta^{\omega_{\Omega}}(s') \land (\xi(x_0, s'u'), e) \in \Omega)].
\]

As the information maps \( \theta^{\omega_{\Omega}}, \theta^{\omega'_{\Omega}} \) are derived from \( \omega_{\Omega}, \omega'_{\Omega} \) which are, in turn, derived
from \( \Omega \), we consider [4.10] to be a feasibility condition satisfiable by \( \Omega \).
In the remainder of this section, when we say that $\Omega$ satisfies (4.4) (resp., (4.6)) we intend that the $\theta^\Omega$ used in the definition of (4.4) (resp., (4.6)) be equal to $\theta^\omega_\Omega$.

It can be proven that if $\Omega$ satisfies (4.6) then $\Omega$ satisfies (4.10). More specifically, it is not difficult to see that if a given policy $\Omega$ satisfies (4.6) then $\omega^\Omega = \omega'^\Omega$. However, if $\Omega$ does not satisfy (4.6) then it is not necessarily the case that $\omega^\Omega = \omega'^\Omega$. Even though $\Omega$ may not satisfy (4.6), $\Omega$ may satisfy (4.10). That is, the set of policies satisfying (4.10) contains the set of policies satisfying (4.6) (i.e., (4.6) is stronger than (4.10)).

Though a policy $\Omega$ may satisfy (4.10), policy $\Omega$ is not guaranteed to be practical in the sense that an agent using $\Omega$ can make sensor activation decisions without ambiguity. More precisely, it is not necessarily the case that if a sensor for an event $e$ is active at a state $x \in X$ then all occurrences of $e$ following unobserved transition sequences from $x$ shall be observed. For instance, consider the plant and policy of Figure 4.4. We consider $\Omega$ to consist of those transitions whose event label is boxed. Specifically, policy $\Omega = \{(x_2, e_2), (x_4, e_2)\}$. One can verify that this policy satisfies (4.10). The set of sensors activated following $\varepsilon$ is $\omega'^\Omega(\varepsilon) = \{e_2\}$. So if $e_2$ follows $\varepsilon$ then it should be observed and the transition it labels should be in $\Omega$. However, $(x_1, e_2)$ is not in $\Omega$.

So an additional requirement on policy $\Omega$ is needed to ensure that if a sensor for an
observable event $e$ is active then any subsequent occurrence of $e$ is observed following any unobserved sequence of events. This requirement is exactly (4.4), the feasibility condition on policies introduced in Wang et al. [54]. Consequently, in the remainder of this section we require that policy $\Omega$ satisfy both (4.4) and (4.10).

The following result allows us to simplify the definition of (4.10) when $\Omega$ satisfies (4.4).

**Lemma 3.** Given plant $G = (X, \Sigma, \xi, x_0)$ and policy $\Omega \subseteq TR(G)$ which satisfies (4.4), $\forall e \in \Sigma, \forall se \in L(G)$, $e \in \omega_\Omega(s) \iff e \in \omega'_\Omega(s)$.

**Proof.** Consider any $e \in \Sigma$, $se \in L(G)$.

(i) Suppose $e \in \omega_\Omega(s)$. Since $e \in \omega_\Omega(s)$ and by definition of $\omega_\Omega$ (4.2), it follows that $(\xi(x_0, s), e) \in \Omega$. Then $e \in \omega'_\Omega(s)$ by definition of $\omega'_\Omega$.

(ii) Suppose $e \notin \omega_\Omega(s)$. Then $(\xi(x_0, s), e) \notin \Omega$. Assume $e \in \omega'_\Omega(s)$. Then $\exists \sigma_1 \ldots \sigma_n \in \Sigma, (\xi(x_0, s), \sigma_1) \notin \Omega \land \ldots \land (\xi(x_0, s\sigma_1 \ldots \sigma_{n-1}), \sigma_n) \notin \Omega \land (\xi(x_0, s\sigma_1 \ldots \sigma_n), e) \in \Omega$. It follows that $\theta^{\omega\Omega}(s) = \theta^{\omega\Omega}(s\sigma_1 \ldots \sigma_n)$ by definition of $\theta^{\omega\Omega}$. Since $se \in L(G)$ it follows that $(\xi(x_0, s), e) \in TR(G)$. That $(\xi(x_0, s), e) \in TR(G)$, $(\xi(x_0, s), e) \notin \Omega$, $(\xi(x_0, s\sigma_1 \ldots \sigma_n), e) \in \Omega$ and $\theta^{\omega\Omega}(s) = \theta^{\omega\Omega}(s\sigma_1 \ldots \sigma_n)$ holds implies that $\Omega$ does not satisfy (4.4). A contradiction is reached. Thus, it must be that $e \notin \omega'_\Omega(s)$.

A corollary of Lemma 3 is the following which can be used to simplify (4.10).

**Corollary 1.** If $\Omega \subseteq TR(G)$ satisfies (4.4) then $\forall s \in L(G)$, $\theta^{\omega\Omega}(s) = \theta^{\omega'_\Omega}(s)$.

**Proof.** Follows by Lemma 3 and by definition of $\theta^{\omega\Omega}, \theta^{\omega'_\Omega}$. 


By Corollary 1, when \( \Omega \) satisfies (4.4), we can reason about the appearance of a string using \( \theta^{\omega_{\Omega}} \) rather than \( \theta^{\omega'_{\Omega}} \) when the sensor activation map under consideration, \( \omega'_{\Omega} \), is derived from \( \Omega \) by (4.9). In light of this, (4.10) can be simplified to the following where \( \theta^{\omega_{\Omega}} \) has been replaced by \( \theta^{\omega_{\Omega}} \):

\[
(\forall e \in \Sigma_o)(\forall s, s' \in L(G)) \theta^{\omega_{\Omega}}(s) = \theta^{\omega_{\Omega}}(s') \Rightarrow (4.11)
\]

\[
[(\exists u \in \Sigma^* \text{ such that } su \in L(G) \land \theta^{\omega_{\Omega}}(su) = \theta^{\omega_{\Omega}}(s) \land (\xi(x_0, su), e) \in \Omega) \Leftrightarrow (\exists u' \in \Sigma^* \text{ such that } s'u'e \in L(G) \land \theta^{\omega_{\Omega}}(s'u') = \theta^{\omega_{\Omega}}(s') \land (\xi(x_0, s'u'), e) \in \Omega)].
\]

An example is provided demonstrating a policy which satisfies (4.4) and (4.11) (equivalently, (4.10)) but not (4.6). Consider the plant in Figure 4.5. We consider \( \Omega \) to consist of those transitions whose event label is boxed. Condition (4.6) is not satisfied by \( \Omega \) as \( \theta^{\omega_{\Omega}}(e_1) = \theta^{\omega_{\Omega}}(e_2e_1), (\xi(x_0, e_1), e_2) \in \Omega \) and \( (\xi(x_0, e_2e_1), e_2) \notin \Omega \) (and in fact this transition is not defined). One can easily verify that (4.4) and (4.11) are satisfied. For any two states that are indistinguishable from each other, if an observable event \( e \) labels transitions from both states then the same sensor activation decision is made for \( e \) from both states (i.e., (4.4) is satisfied). Also, if an event \( e \) is observed after some sequence of unobserved transitions from a given state \( x \) then

\[
(x_1, e_2, x_2) \xrightarrow{e_1} (x_4, e_2, x_6) \xrightarrow{e_1} (x_5, e_1, x_7) \xrightarrow{e_2} (x_3, e_2, x_3)
\]

Figure 4.5: Plant and policy defined which satisfies (4.4) and (4.11) but not (4.6).
it is observed after sequences of unobserved transitions from any state $x'$ which is indistinguishable from $x$ (i.e., (4.11) is satisfied).

In the remainder of this section we prove that, for policies satisfying (4.4) and (4.11), sensor activation decisions can be made based on the sensor activation decisions of an arbitrary state in the state equivalence class of the true state of $G$. We describe an algorithm for computing an automaton, denoted by $[G]$ (with a slight abuse of notation), whose states are the state equivalence classes of $X$. This automaton maps an observed event sequence to the state equivalence class of the true state of $G$ and hence to the sensor activation decisions following the true state of $G$. Our algorithm is in $O(|X|^2 \cdot |\Sigma|)$. For policies satisfying (4.4) and (4.11), the observer automaton need not be computed for purposes of defining a map from observed event sequences to sensor activation decisions.

Before we proceed, some notation is introduced for convenience of presentation. For given $\Omega$ and $x \in X$, we denote the set of sequences of events which label transitions not in $\Omega$ from state $x$ by $S^\Omega_x = \{ s \in \Sigma^* : \exists n \geq 0, s = \sigma_1 \ldots \sigma_n \land (x, \sigma_1) /\in \Omega \land \ldots \land (\xi(x, \sigma_1 \ldots \sigma_{n-1}), \sigma_n) /\in \Omega \}$.

The following proceeds in the same manner as from Lemma 1 to the end of Section 4.3. We first provide a result analogous to Lemma 1:

**Lemma 4.** Given plant $G = (X, \Sigma, \xi, x_0)$, policy $\Omega \subseteq TR(G)$ which satisfies (4.4) and (4.11), $\forall x, z \in X, \forall e \in \Sigma_o$, if $[x] = [z]$ then $\exists s \in S^{\Omega}_x, (\xi(x, s), e) \in \Omega \Leftrightarrow \exists s' \in S^{\Omega}_z, (\xi(z, s'), e) \in \Omega$.
Proof. Since \( z \in [x] \) and by definition of \([x]\) in (4.5) we have the following:

\[
\exists n \geq 0, \exists x^0, x^1, x^2, \ldots, x^n \in X, x^0 = x,
\]
\[
x^1T_\Omega x^0 \land x^2T_\Omega x^1 \land \ldots \land zT_\Omega x^n.
\]

Since \( x^{i+1}T_\Omega x^i \) it follows that \( \exists t, t' \in \mathcal{L}(G) \) such that \( \xi(x_0, t) = x^{i+1}, \xi(x_0, t') = x^i \) and \( \theta^{e\alpha}(t) = \theta^{e\alpha}(t') \). Then, since \( \Omega \) satisfies (4.11), it follows that

\[
(\forall e \in \Sigma_0)
\]
\[
[(\exists u \in \Sigma^* \text{ such that } tue \in \mathcal{L}(G) \land \theta^{e\alpha}(tue) = \theta^{e\alpha}(t) \land (\xi(x^{i+1}, u), e) \in \Omega)
\]
\[
\Leftrightarrow
\]
\[
(\exists u' \in \Sigma^* \text{ such that } t'u'e \in \mathcal{L}(G) \land \theta^{e\alpha}(t'u') = \theta^{e\alpha}(t') \land (\xi(x^i, u'), e) \in \Omega)].
\]

Let \( e \in \Sigma_0 \). Suppose \( \exists u' \in S^G_{x^i}, (\xi(x^i, u'), e) \in \Omega \). That \( (\xi(x^i, u'), e) \in \Omega \) implies \( \xi(x^i, u'e) \). That \( \xi(x_0, t') = x^i \) and \( \xi(x^i, u'e) \) implies \( \xi(x_0, t'u'e) \) and so \( t'u'e \in \mathcal{L}(G) \).

Also, \( u' = u'_1 \ldots u'_k, (x^i, u'_l) \notin \Omega, \ldots (\xi(x^i, u'_1 \ldots u_{k-1}), u'_k) \notin \Omega \) by \( u' \in S^G_{x^i} \) and the definition of \( S^G_{x^i} \). It follows that \( u'_{l+1} \notin \omega(t'u_0 \ldots u_l) \) for \( l \in \{0 \ldots k-1\} \) where \( u_0' = \varepsilon \) by (4.2) and \( \xi(x_0, t') = x^i \). Then, by definition of \( \theta^{e\alpha}, \theta^{e\alpha}(t'u') = \theta^{e\alpha}(t') \).

From (4.12) and the three facts above, namely, \( t'u'e \in \mathcal{L}(G), \theta^{e\alpha}(t'u') = \theta^{e\alpha}(t') \) and \( (\xi(x^i, u'), e) \in \Omega, \) we have that \( \exists u \in \Sigma^* \text{ such that } tue \in \mathcal{L}(G) \land \theta^{e\alpha}(tue) = \theta^{e\alpha}(t) \land (\xi(x^{i+1}, u), e) \in \Omega) \). Let \( u = u_1 \ldots u_m \). Since \( \theta^{e\alpha}(tue) = \theta^{e\alpha}(t) \) it follows that \( u_{j+1} \notin \omega(tu_1 \ldots u_j) \). By definition of \( \omega \), it follows that \( (\xi(x^{i+1}, u_1 \ldots u_j), u_{j+1}) \notin \Omega \). Then \( u \in S^G_{x^{i+1}} \).

Using an argument symmetric to the above, one can show that if \( \exists u \in S^G_{x^{i+1}}, (\xi(x^{i+1}, u), e) \in \Omega \) then \( \exists u' \in S^G_{x^i}, (\xi(x^i, u'), e) \in \Omega \). Thus the following holds

\[
\exists u \in S^G_{x^{i+1}}, (\xi(x^{i+1}, u), e) \in \Omega \Leftrightarrow \exists u' \in S^G_{x^i}, (\xi(x^i, u'), e) \in \Omega.
\]

From iterative application of the above fact, the lemma statement holds. \( \square \)
CHAPTER 4. DFA REPRESENTATIONS OF SENS. ACT. MAPS

From Lemma 4 we have the following:

**Lemma 5.** Given plant $G$, policy $\Omega \subseteq TR(G)$ which satisfies (4.4) and (4.11), $\forall x, z \in X, \forall e \in \Sigma_o, \forall s \in S_{x}^G, \forall s' \in S_{x}^G$, if $[x] = [z], (\xi(x, s), e) \in \Omega$ and $(\xi(z, s'), e) \in TR(G)$ then $[\xi(\xi(x, s), e)] = [\xi(\xi(z, s'), e)]$.

**Proof.** The following holds by $[x] = [z]$ and the definition of $[x]$:

$$\exists n \geq 0, \exists x^0, x^1, x^2, \ldots, x^n \in X, x^0 = x, \quad (4.13)$$

$$x^1T_{\Omega}x^0 \land x^2T_{\Omega}x^1 \land \ldots \land zT_{\Omega}x^n.$$  

For all $i \in \{1, \ldots, n\}, \exists s_{x^i} \in S_{x_i}^G, (\xi(x^i, s_{x^i}), e)$ is defined and $(\xi(x^i, s_{x^i}), e) \in \Omega$ by Lemma 4, $e \in \Sigma_o, [x] = [x^i]$ and $(\xi(x, s), e) \in \Omega$. Similarly, $\exists s_z \in S_z^G, (\xi(z, s_z), e) \in \Omega$. As $s_z, s' \in S_z^G$, it follows that $\exists t, t' \in \mathcal{L}(G), \xi(x_0, t) = \xi(z, s_z), \xi(x_0, t') = \xi(z, s')$ and $\theta^{\omega}(t) = \theta^{\omega}(t')$. It follows that $(\xi(z, s'), e) \in \Omega$ by this fact, $t, t' \in \mathcal{L}(G)$ (follows from $(\xi(x_0, t) = \xi(z, s_z)$ and $(\xi(z, s_z), e) \in \Omega)$, $t', t' \in \mathcal{L}(G)$ (follows from $(\xi(x_0, t') = \xi(z, s')$ and $(\xi(z, s'), e) \in TR(G))$, $(\xi(z, s_z), e) \in \Omega$ and $\Omega$ satisfies (4.4).

That $x^iT_{\Omega}x^i-1$ and the definition of $T_{\Omega}$ implies that there exists $w, w' \in \mathcal{L}(G)$ where $\xi(x_0, w) = x^i-1, \xi(x_0, w') = x^i$ and $\theta^{\omega}(w) = \theta^{\omega}(w')$.

Since $(\xi(x^i-1, s_{x^i-1}), e), (\xi(x^i, s_{x^i}), e) \in \Omega, \theta^{\omega}(ws_{x^i-1}e) = \theta^{\omega}(ws_{x^i-1}e)$ and $\theta^{\omega}(w's_{x^i}e) = \theta^{\omega}(w's_{x^i}e)$. Further, $\theta^{\omega}(ws_{x^i-1}e) = \theta^{\omega}(w)$ since $s_{x^i-1} \in S_{x^i-1}^G$ and $\theta^{\omega}(w's_{x^i}) = \theta^{\omega}(w')$ since $s_{x^i} \in S_{x^i}^G$. By these five facts it follows that $\theta^{\omega}(ws_{x^i-1}e) = \theta^{\omega}(w's_{x^i}e)$. It follows that

$$\xi(\xi(x^i, s_{x^i}), e)T_{\Omega}(\xi(x^i-1, s_{x^i-1}), e)$$

by this fact, $\xi(x_0, w) = x^i-1$ and $\xi(x_0, w') = x^i$. Similarly,

$$\xi(\xi(z, s_z), e)T_{\Omega} \xi(\xi(x^n, s_n), e).$$
Recall that \( \exists t, t' \in \mathcal{L}(G), \xi(x_0, t) = \xi(z, s_z), \xi(x_0, t') = \xi(z, s') \) and \( \theta^{\omega}(t) = \theta^{\omega}(t') \). Recall that \((\xi(z, s_z), e), (\xi(z, s'), e) \in \Omega\). These two facts imply \( \theta^{\omega}(te) = \theta^{\omega}(t'e) \). It follows that \( \xi(\xi(z, s'), e)T_{\Omega}\xi(\xi(z, s_z), e) \).

By the above results and (4.13), the following holds:

\[
\xi(\xi(x^1, s_{x^1}), e)T_{\Omega}\xi(\xi(x^0, s), e) \land \xi(\xi(x^2, s_{x^2}), e)T_{\Omega}\xi(\xi(x^1, s_{x^1}), e) \land \ldots \land \xi(\xi(z, s_z), e)T_{\Omega}\xi(\xi(z, s_z), e) .
\]

Then \( \xi(\xi(z, s'), e) \in [\xi(\xi(x, s), e)] \) by this fact, \( x = x^0 \) and (4.5). It follows that \( [\xi(\xi(x, s), e)] = [\xi(\xi(z, s'), e)] \).

Before we present the main result of this section, we require the following. Given \( s \in \mathcal{L}(G) \), we let \( [\theta^{\omega}(s)] \) denote (with a slight abuse of notation) the agent’s estimate of \([\xi(x_0, s)]\) (i.e., the class containing the true state of \( G \)) when the agent observes \( \theta^{\omega}(s) \). Formally, \( [\theta^{\omega}(s)] \) is defined as follows where \( \theta^{\omega}(s) = \sigma_1\sigma_2\ldots\sigma_n \):

\[
[\theta^{\omega}(s)] = \{ [x] : \exists q_0, q_1, q_2, \ldots, q_n \in X, x_0 \in [q_0], [x] = [q_n], \forall i \in \{0, \ldots, n-1\}, (q_i, \sigma_{i+1}) \in \Omega \land \xi(q_i, \sigma_{i+1}) \in [q_{i+1}] \} .
\]

We prove that, for any \( s \in \mathcal{L}(G) \), \([\theta^{\omega}(s)]\) is a singleton set when \( \Omega \) satisfies (4.4) and (4.11). When policies do not satisfy (4.11), \([\theta^{\omega}(s)]\) is not necessarily a singleton set. This is explored further in Section 4.5.

**Theorem 2.** Given plant \( G = (X, \Sigma, \xi, x_0) \), policy \( \Omega \subseteq TR(G) \) which satisfies (4.4) and (4.11), \( \forall s \in \mathcal{L}(G), \forall [x] \in [\theta^{\omega}(s)], [x] = [\xi(x_0, s)] \).

**Proof.** This proof proceeds by induction on the length of \( \theta^{\omega}(s) \).

When \( |\theta^{\omega}(s)| = 0, \theta^{\omega}(s) = \varepsilon \). It holds that \([\varepsilon] = \{ [x] : \exists q_0 \in X, x_0 \in [q_0], [x] = [q_0] \} \) by (4.14). So, \( \forall [x] \in [\varepsilon], x_0 \in [x] \). It follows that \([\varepsilon] = \{[x_0]\} \) and so \( [\theta^{\omega}(s)] = \{[x_0]\} \). The fact that \( \theta^{\omega}(\varepsilon) = \varepsilon \) and \( \theta^{\omega}(s) = \varepsilon \) implies \( \theta^{\omega}(\varepsilon) = \theta^{\omega}(s) \). Then
\( \xi(x_0, s)T_\Omega x_0 \) by this fact, \( \xi(x_0, \varepsilon) = x_0 \) and the definition of \( T_\Omega \). Then \( \xi(x_0, s) \in \langle x_0 \rangle \) by this fact and (4.5). Thus, \( \forall [x] \in [\theta^{\omega_n}(s)], [x] = [\xi(x_0, s)] \).

Now for the inductive step, consider when \( |\theta^{\omega_n}(s)| = n + 1 \). Let \( s = s_1 e s_2 \) where \( s_1, s_2 \in \Sigma^* \), \( e \in \Sigma_o \), and \( s_1 e \) is the shortest prefix of \( s \) where \( \theta^{\omega_n}(s_1 e) = \theta^{\omega_n}(s_1) e = \theta^{\omega_n}(s) \). Since \( |\theta^{\omega_n}(s_1)| = n \) and by the inductive hypothesis it follows that \( \forall [x] \in [\theta^{\omega_n}(s_1)], [x] = [\xi(x_0, s_1)] \). Since \( \theta^{\omega_n}(s_1 e) = \theta^{\omega_n}(s_1) e \) it must be that \( (\xi(x_0, s_1), e) \in \Omega \). It follows that \( \forall [x] \in [\theta^{\omega_n}(s_1)], \forall q \in [x], \forall s_q \in S^*_q \), if \( (\xi(q, s_q), e) \in TR(G) \) then \( [\xi(\xi(q, s_q), e)] = [\xi(x_0, s_1), e)] \) by this fact, \( \Omega \) satisfies (4.4) and (4.11), \( e \in \Sigma_o \), \( [q] = [x] = [\xi(x_0, s_1)] \), and Lemma 5. Then \( \forall [x] \in [\theta^{\omega_n}(s_1 e)], [x] = [\xi(x_0, s_1 e)] \).

It holds that \( \xi(x_0, s_1 e s_2)T_\Omega \xi(x_0, s_1 e) \) since \( \theta^{\omega_n}(s_1 e s_2) = \theta^{\omega_n}(s_1 e) \) and by definition of \( T_\Omega \). So \( [\xi(x_0, s_1 e s_2)] = [\xi(x_0, s_1 e)] \). Then \( \forall [x] \in [\theta^{\omega_n}(s_1 e s_2)], [x] = [\xi(x_0, s_1 e)] \) by this fact, \( \theta^{\omega_n}(s_1 e s_2) = \theta^{\omega_n}(s_1 e) \) and the previous fact that \( \forall [x] \in [\theta^{\omega_n}(s_1 e)], [x] = [\xi(x_0, s_1 e)] \).

For the policies of Section 4.3 it sufficed to base sensor activation decisions on the state reached by the sequence of events observed by the agent from the initial state of \( G \). However, policies satisfying (4.4) and (4.11), it is not difficult to see that the sequence of events observed may not be a string in \( L(G) \) (e.g., consider plant and policy of Figure 4.4 where instead an unobserved event \( e_{u_0} \) labels the transition from \( x_1 \) to \( x_4 \) instead of \( e_2 \)). Instead, according to Theorem 2 it suffices for an agent to keep track of the equivalence class of the true state of \( G \) for purposes of determining sensor activation decisions.

Tracking the state equivalence class of the true state of \( G \) can be done using an automaton \( [G] \) which is defined next. It will be demonstrated that the set of state equivalence classes of \( X \), denoted by \( [X] \) (with a slight abuse of notation), and a
map, $\xi_{[G]} : [X] \times \Sigma_o \rightarrow [X]$ from a state equivalence class and observed event to the state equivalence class encountered on observation of the event can be computed polynomially in $|X|$ and $|\Sigma|$. This allows the following procedure to be used for computing sensor activation decisions. The initial sensor activation decisions are based on any state in $[x_0]$. From $[x_0]$, if an event $e_1$ is observed (due to detection by $e_1$’s sensor) then the state equivalence class from which sensor activation decisions are based is changed to $\xi_{[G]}([x_0],e_1)$. Sensors are updated according to any state in $\xi_{[G]}([x_0],e_1)$. Afterward, if an event $e_2$ is observed then the state equivalence class from which sensor activation decisions are based is changed to $\xi_{[G]}([x_0],e_1e_2)$. The state equivalence class from which sensor activation decisions are based is updated in this manner as further event occurrences are observed.

Next we describe how $[G]$ may be computed. First we describe a procedure for computing $[X]$. From $X$ and $T_\Omega$ we construct an undirected graph $(X,T_\Omega)$. Associated with each vertex in $X$ is a flag indicating whether or not the vertex has been visited. Initially, for all $x \in X$, the flag of $x$ is assigned the value 0, denoting that $x$ has not yet been visited. For each $x \in X$ whose flag is equal to 0 we conduct a depth-first search. When a vertex is visited we assign its flag to the value 1. It is not difficult to see that any vertex visited in the search belongs to $[x]$ and, furthermore, when the search terminates the set of vertices traversed is equal to $[x]$. This procedure is in $\Theta(|X| + |T_\Omega|)$ due to the well known tight asymptotic bounds on depth-first search. In the worst-case, $T_\Omega$ contains an edge between any two vertices in $X$ (i.e., every state in $X$ is indistinguishable from every other state in $X$). In this case $|T_\Omega| = \frac{|X||X+1|}{2}$ and so the procedure is in $\Theta(|X|^2)$ for this case.

After computing $[X]$ we can construct DFA $[G] = ([X],\Sigma_o,\xi_{[G]},[x_0])$. Transition
function $\xi_{[G]}$ is defined as follows:

$$\forall [x] \in [X], \forall e \in \Sigma_o, \xi_{[G]}([x], e) = [\xi(x, e)] \text{ if } \exists y \in [x] \text{ where } (y, e) \in \Omega.$$ (4.15)

To compute $\xi_{[G]}$ we require a map from each state $x \in X$ to its corresponding state class. Given $[X]$ this can be computed in $O(|X|)$. Then computing $\xi_{[G]}$ is in $O(|[X]| \cdot |\Sigma_o|) \subseteq O(|X| \cdot |\Sigma_o|)$ as any two states in the same state equivalence class are followed by the same sensor activation decisions.

To summarize, given plant $G$ and policy $\Omega \subseteq TR(G)$, which satisfies (4.4) and (4.11), if event sequence $s$ is generated by $G$ then the set of events whose corresponding sensors need to be turned on following $s$ (i.e., $\omega_\Omega(s)$) is the set of events labeling transitions in $[G]$ originating from state $\xi_{[G]}([x_0], \theta^{\omega_\Omega}(s))$.

Next we produce a plant $G$ and sensor activation policy $\Omega$ satisfying (4.4) and (4.11) where, for purposes of obtaining a map from observed strings to sensor activation decisions, computing the observer automaton is in exponential-time whereas computing $[G]$ is in polynomial-time.

Consider the plant $G$ in Fig. 4.6 where sensor activation policy $\Omega$ consists of those transitions whose event labels are boxed. In computing the observer automaton we first replace transition labels that are not boxed (i.e., not observed) with $\varepsilon$. This produces an NFA with $\varepsilon$-transitions denoted by $G'$. Automaton $G'$ accepts the language $L_m(G') = \{\alpha, \beta\}^* \alpha \{\alpha, \beta\}^{n-3} \{\alpha, \beta\}^*$. It is well known that the minimum-state DFA accepting $\{\alpha, \beta\}^* \alpha \{\alpha, \beta\}^{n-1}$ has $2^{n-1}$ states. Using this fact one can easily show that the minimum-state DFA accepting $L_m(G')$ has $2^{n-3}$ states. Thus, the observer automaton computed from $G'$ requires $2^{n-3}$ states. It follows that computing the observer automaton requires at least $2^{n-3}$ computational steps.

It is not difficult to see that, given $\Omega$, the set of indistinguishable state pairs of
$G$, denoted by $T_{\Omega}$, consists of all pairs of states in $G$. By definition of $T_{\Omega}$ it follows that $[X]$, the set of equivalence classes induced by $T_{\Omega}^*$, is a singleton set. Thus $[G]$ is a single-state automaton with a self-loop labeled by events $\alpha$ and $\beta$. This automaton is computed in polynomial-time as per our previous analysis. One can verify that $\Omega$ satisfies (4.4) and (4.11). Then it follows from Theorem 2 that $[G]$ provides the same map from observed event sequences to sensor activation decisions as the observer automaton computed from $G$. Automaton $[G]$ only maps observed event sequences to sensor activation decisions. Precise state estimates of $G$ and the distinction between marked and unmarked observed strings are forfeited.

![Figure 4.6: Plant $G$ and policy $\Omega$ which satisfies (4.4) and (4.11). Computing observer automaton is exponential whereas computing $[G]$ is polynomial.](image)

4.5 Computing decisions using state equivalence classes

4.5.1 Introduction

In this section we introduce a class of sensor activation policies that is larger than the class of policies studied in Section 4.4. That is, the policies studied in this section
satisfy conditions more general than \((4.11)\). Similar to the policies studied in Section 4.4, it suffices to use the coarse estimate of the true state of \(G\) specified by \((4.14)\) (i.e., the estimate of the true state’s equivalence class) to determine the sensor activation decisions that are consistent with a given policy. One need not construct the observer automaton to determine sensor activation decisions in this case.

In Subsection 4.5.2 we describe and define the class of sensor activation policies that we consider and contrast this class with the class studied in Section 4.4. Policies belonging to the class we consider must satisfy three conditions. In Subsection 4.5.3 we provide an algorithm for verifying the first and simplest of these conditions. In Subsection 4.5.4 we prove that verifying the second condition is PSPACE-complete, so it is difficult to verify if an arbitrary policy belongs to the class we consider. Finally, in Subsection 4.5.5 we describe an algorithm for verifying the third condition.

### 4.5.2 A general class of policies whose decisions may be computed using state equivalence classes

In order for a sensor activation policy to be usable by an agent, the agent should not encounter ambiguity in deciding if an event’s sensor should be activated or not. Condition \((4.4)\) is the most general condition one could consider that, when satisfied, implies there is no ambiguity in decision-making. So the policies we consider satisfy \((4.4)\) as a preliminary requirement.

In Section 4.4 we considered that policies satisfy \((4.4)\) and \((4.11)\). That these two conditions are satisfied implies that any two states in a state equivalence class must have exactly the same sensor activation decisions. It is not required that condition \((4.11)\) be satisfied by the policies considered in this section. Though the policies we
consider satisfy (4.4), this alone is not sufficient to guarantee that two states in a state equivalence class have exactly the same sensor activations decisions. Even when (4.4) is satisfied, it may be that two states in a given state equivalence class are followed by conflicting decisions for a given event’s sensor. Note that two such states are not indistinguishable, for otherwise condition (4.4) would not hold.

In Section 4.4 we considered that if a sensor for an event $e$ is active then there exists some unobserved trace following activation of $e$’s sensor after which $e$ occurs. This resulted in the true state’s equivalence class estimate (i.e., $[\theta^{\omega_n}(s)]$ where $s$ is the string generated by $G$) being a singleton set. In this section we relax this restriction. That is, though an event $e$’s sensor is active, there may not exist any unobserved trace following its activation after which $e$ occurs. Instead, for instance, the occurrence of some other event whose sensor is active may occur before $e$ is generated by $G$. For these types of sensor activation maps, the true state’s equivalence class estimate may not be a singleton set. It may be that two state equivalence classes belonging to the estimate are followed by conflicting decisions. That is, there exists a state $x$ from one class and state $y$ from another such that $x$ and $y$ are followed by conflicting decisions for a given event’s sensor.

So removing the requirements that (4.11) be satisfied and that events must eventually be detected if their corresponding sensors are activated can introduce problems that prevent the use of state equivalence class estimates for determining sensor activation decisions. In total, there are three scenarios that may be encountered which prevent basing sensor activation decisions on state equivalence class estimates. In addressing these three scenarios, we define the class of sensor activation policies that we consider in this section.
In the first scenario, sensor activation conflicts may exist within a state equivalence class, as described previously. That is, one of the state equivalence classes $[x] \in [X]$ contains two states $x_1, x_2 \in [x]$ such that, for some event $e \in \Sigma_o$, $(x_1, e) \in TR(G)$, $(x_2, e) \in TR(G)$, $(x_1, e) \in \Omega$ but $(x_2, e) \notin \Omega$ where $\Omega$ is the policy used. The negation of this scenario is a condition to be satisfied by policies in this section to ensure, in part, that sensor activation decisions can be made using state equivalence class estimates. This condition is formalized in the following:

$$(\forall e \in \Sigma_o), (\forall [x] \in [X]), (\forall z, y \in [x]), (z, e) \in TR(G) \land (y, e) \in TR(G) \quad (4.16)$$

$$\Rightarrow [(z, e) \in \Omega \iff (y, e) \in \Omega].$$

In Subsection 4.5.3 we describe an algorithm for verifying (4.16) in polynomial-time.

When the first scenario does not occur (i.e., (4.16) holds), sensor activation conflicts do not exist in a state equivalence class. However, it is possible that the set of sensor activation decisions computed from a given state equivalence class might result in sensors being activated that would not otherwise be activated if sensor activation decisions were computed from the observer automaton. This is the second scenario. Specifically, it may be that there exists $e \in \Sigma_o$, $x \in X$ such that $(x, e) \notin TR(G)$.

Further, state $x$ may be indistinguishable from other states $Q$ where, for all $q \in Q$, $(q, e) \in \Omega$. If there exists a string $s \in \mathcal{L}(G)$ that leads to $x$, and $s$ is not observed the same as any string leading to a state in $Q$, then the sensor for $e$ does not have to be activated following $s$. If sensor activation decisions are based on the observer automaton $DET_{\Omega}(G)$, then the sensor for $e$ will not be activated following $s$. However, if the sensor activation decision is based on $[x]$ (or any other state class in $[\theta^\Omega(s)]$ which contains a state of $X$ from which $e$ occurs and is observed), then the sensor for $e$ will be activated following $s$. 
For $x \in X$, let $L(G, x) = \{ s \in L(G) : \xi(x_0, s) = x \}$. The following condition captures the negation of the second scenario, which the policies that we consider must satisfy. Informally, the condition states that, for any $e \in \Sigma_o$, if a state $x$ is indistinguishable from a state $q$ where $(x, e) \notin TR(G)$ and $(q, e) \in \Omega$ then any string leading to $x$ is indistinguishable from some string leading to a state from which $e$ is observed:

\[
(\forall x, q \in X)(\forall e \in \Sigma_o)xTq \land (x, e) \notin TR(G) \land (q, e) \in \Omega \\
\Rightarrow [(\forall s \in L(G, x))(\exists t \in L(G))\theta^\Omega(s) = \theta^\Omega(t) \land (\xi(x_0, t), e) \in \Omega].
\]

When condition (4.17) holds, even if we use the observer automaton $DET_\Omega(G)$ for determining sensor activation decisions, the sensor for $e$ will always be turned on for any state estimate reached in $DET_\Omega(G)$ that contains $x$. This too permits using $[x]$ for determining sensor activation decisions, without activating more sensors than when using $DET_\Omega(G)$.

However, the problem of verifying whether or not (4.17) holds is PSPACE-complete. Proof of this is provided in Subsection 4.5.4.

When neither the first nor second scenarios occur (i.e., conditions (4.16) and (4.17) hold) it may still be the case that a third scenario occurs: sensor activation conflicts may exist between state equivalence classes in a state equivalence class estimate. That is, for policy $\Omega$, it may be the case that, for some string $s \in L(G)$, $[\theta^\Omega(s)]$ contains two state equivalence classes $[\hat{x}_1], [\hat{x}_2]$ where $[\hat{x}_1]$ contains a state $x_1$ such that, for some event $e \in \Sigma_o$, $(x_1, e) \in \Omega$ but there does not exist a state $x_2 \in [\hat{x}_2]$ where $(x_2, e) \in \Omega$.

The following condition captures the negation of this scenario, which the policies that we consider must satisfy. Informally, the condition states that $\forall s \in L(G), [\theta^\Omega(s)]$ contains only state equivalence classes which are followed by exactly the same sensor
activation decisions for any given observable event. Formally, this is expressed as the following:

\[(\forall s \in \mathcal{L}(G))(\forall e \in \Sigma_o)(\forall [\hat{q}_1], [\hat{q}_2] \in [\vartheta^\Omega(s))],\]

\[\exists q_1 \in [\hat{q}_1], (q_1, e) \in \Omega \iff \exists q_2 \in [\hat{q}_2], (q_2, e) \in \Omega)\]

In Subsection 4.5.5 we describe an algorithm for verifying (4.18) in polynomial-time.

When a sensor activation policy \(\Omega\) satisfies (4.16)–(4.18), the sensor activation decisions corresponding to \(\Omega\) following observation of an event sequence can be determined using the NFA \([G]\) which was introduced in Section 4.4. That is, the sensor activation decisions made using \([G]\) are equivalent to those made using the observer automaton \(DET_\Omega(G)\).

Note that the policies considered in Section 4.4 satisfy (4.16)–(4.18). In Section 4.4 we considered policies that satisfy (4.4) and (4.11). For these policies, it was proven that no sensor activation conflicts exist between any two states in a state equivalence class (see proof of Lemma 5). Thus (4.16) is satisfied. Also, by definition of (4.11), the second scenario described above does not occur. Thus (4.17) is satisfied. Condition (4.18) is satisfied by the fact that there is only ever one state equivalence class in the true state equivalence class’ estimate (cf. Theorem 2).

For simplicity, in this section we consider that the sensor activations following a string \(s \in \mathcal{L}(G)\) are defined by (4.2), not by (4.9). We could consider that sensor activations are defined by (4.9) instead but, when computing the sensor activation decisions of a state equivalence class, which is the union of all sensor activations following all states in the state equivalence class, the same set of activations will be computed as if sensor activations are defined by (4.2). The reason for this is that any state \(x'\) in the unobserved reach of a state \(x\) belongs to \([x]\) and so, in addition to
sensors active at state $x$ being included in the set of sensor activations of $[x]$, sensors active at state $x'$ will also be included in the set of sensor activations of $[x]$.

### 4.5.3 Verifying if sensor activation conflicts exist in a state equivalence class

In this subsection we consider verification of whether or not sensor activation conflicts exist within state equivalence classes. This corresponds to the first scenario described in Subsection 4.5.2. Formally, we consider the problem of verifying whether or not condition (4.16) holds. We demonstrate that condition (4.16) can be verified in polynomial-time.

Verification of whether or not (4.16) holds is simple. For each state equivalence class, compute the set of events whose sensors are active following any state in the equivalence class and compute the set of events which occur after states in the equivalence class but whose sensors are deactivated after those states. For a given state equivalence class, if the intersection of these two sets is nonempty then a sensor activation conflict exists for that state equivalence class. An algorithm is described for conducting this verification in $O(|X| \cdot |\Sigma|)$.

**Lemma 6.** Condition (4.16) can be verified in $O(|X| \cdot |\Sigma|)$.

**Proof.** An algorithm for verifying (4.16) is described next. We suppose a representation is provided for $\xi$ which permits $\xi(x, e)$ to be computed in $O(1)$ for a given $x \in X$, $e \in \Sigma$. This is a typical requirement of transition functions of automata when regarding the computational complexity of operations on automata. Given a set $S$, we say $S$ can be *effectively enumerated* if a representation is provided for $S$ that permits
enumerating all elements of $S$ in $O(|S|)$. We suppose one can effectively enumerate sets $X$, $\Sigma$, $\Omega$, $[X]$ and $[x]$ for all $[x] \in [X]$. This is not a difficult requirement to be satisfied. The state set (e.g., $X$) and event set (e.g., $\Sigma$) of automata are typically represented using lists. If $[X]$ is computed using the procedure described in Section 4.4 for computing DFA $[G]$ then a list representation of $[X]$ and list representations of all $[x] \in [X]$ can be computed without adding overhead to the algorithm for computing DFA $[G]$ described in Section 4.4.

First, from $\xi$, we construct transition function $\xi'$ as follows. Copy $\xi$ to $\xi'$. I.e., for all $x \in X$, for all $e \in \Sigma$, set $\xi'(x, e)$ to $\xi(x, e)$. This operation is proportional to the size of $\xi$, i.e., in $O(|TR(G)|) \subseteq O(|X| \cdot |\Sigma|)$. Second, for all $(x, e) \in \Omega$, set $\xi'(x, e)$ to state $f$ where $f$ is a new state not in $X$. This operation can be done in $O(|\Omega|) \subseteq O(|X| \cdot |\Sigma|)$ since $\Omega$ can be effectively enumerated, $\Omega \subseteq TR(G)$ and $O(TR(G)) \subseteq O(|X| \cdot |\Sigma|)$.

Third, for all $[\hat{x}] \in [X]$, for all $e \in \Sigma$, test to see if there exist $x, y \in [\hat{x}]$ such that $\xi'(x, e) \in X$ and $\xi'(y, e) = f$. If so, there is a sensor activation conflict at class $[\hat{x}]$ for event $e$ (i.e., (4.16) does not hold) and no conflict otherwise (i.e., (4.16) holds). For all $[\hat{x}] \in [X]$, for all $e \in \Sigma$, the described test can be implemented using a variable $C_{[\hat{x}], e}$ that may be assigned a value from the set $\{doesNotOccur, active, inactive\}$. Initially, before enumerating $[\hat{x}]$, set $C_{[\hat{x}], e} = doesNotOccur$, denoting the fact that event $e$ labels no transition from a state in $[\hat{x}]$ visited yet. Next (enumerating $[\hat{x}]$), for all $x \in [\hat{x}]$, if $\xi'(x, e)$ is defined then

- if $\xi'(x, e) = f$
  - if $C_{[\hat{x}], e} = doesNotOccur$ then set $C_{[\hat{x}], e} = active$;
  - otherwise, if $C_{[\hat{x}], e} = inactive$ then report that (4.16) does not hold;
  - otherwise (i.e., $C_{[\hat{x}], e} = active$) do nothing;
• otherwise, if $\xi'(x,e) \neq f$ then
  
  - if $C_{[\hat{x}],e} = doesNotOccur$ then set $C_{[\hat{x}],e} = inactive$;
  
  - otherwise, if $C_{[\hat{x}],e} = inactive$ then do nothing;

  - otherwise (i.e., $C_{[\hat{x}],e} = active$) report that (4.16) does not hold.

This operation can be done in $O(|X| \cdot |\Sigma|)$ since $\Sigma$, $[X]$ and $[\hat{x}]$ can be effectively enumerated for all $[\hat{x}] \in [X]$. Each of the three operations mentioned above is in $O(|X| \cdot |\Sigma|)$, so it follows that this algorithm is in $O(|X| \cdot |\Sigma|)$. \qed

4.5.4 Verifying if sensors are only activated when necessary using state equivalence classes

In this section we consider, for a given automaton $G$ and policy $\Omega$, verification of whether or not activating sensors using a state’s equivalence class results in more sensors being active than is necessary. This corresponds to the second scenario described in Subsection 4.5.1. Formally, we consider the problem of verifying whether or not condition (4.17) holds.

First, an example is provided to illustrate the second scenario. Consider the plant in Fig. 4.7. We consider $\Omega$ to consist of those transitions whose event label is boxed. Specifically, policy $\Omega = \{(x_1, e_1), (x_2, e)\}$. One can verify that this policy satisfies (4.4). The set of indistinguishable state pairs for the given plant under the given sensor activation policy can be computed by replacing the labels of those transitions not in $\Omega$ by $\epsilon$ and applying the product of Sears et al. [31] (which is also studied in Chapter 5). This results in the set $T_\Omega = \{(x_1, x_1), (x_2, x_2), (x_3, x_3), (x_4, x_4), (x_1, x_2), (x_1, x_3), (x_2, x_3)\}$. In computing $DET_\Omega(G)$, we obtain the DFA in Fig. 4.8. As we can see, $x_3$
can be distinguished from states $x_1$ and $x_2$ on string $e_1 \in \mathcal{L}(G)$ where $\theta^\Omega(e_1) = e_1$.

However, we cannot obtain this information from $[x_3] = \{x_1, x_2, x_3\}$. When sensor activation decisions are computed using equivalence classes (i.e., using NFA $[G]$ which will be defined in Section 4.5.5), the set of sensors activated will be $\{e_1, e\}$ on observation of $e_1$. However, the state estimate returned by the observer automaton on observed sequence $\theta^\Omega(e_1) = e_1$ is $\{x_3\}$. Since no sensors are active from state $x_3$, no sensors need to be activated after $e_1$ is observed.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{plant_and_policy}
\caption{Plant and policy which satisfies (4.4) and where scenario 2 occurs.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{dfa}
\caption{DFA $DET^\Omega(G)$ constructed from applying subset construction on plant of Fig. 4.7 when transitions not in $\Omega$ are relabeled with $\varepsilon$}
\end{figure}

So, given plant $G$ and policy $\Omega$, it is necessary to verify if the above situation occurs to determine if equivalence classes may be used for informing the same sensor activation decisions as would be informed using the observer automaton. That is, we would like to be able to verify whether or not (4.17) holds. Let us consider...
the negation of (4.17) (corresponding to the second scenario described in Subsection 4.5.1):

\[
\exists x, q \in X \exists e \in \Sigma_o \exists T_\Omega q \land (x, e) \notin TR(G) \land (q, e) \in \Omega \land [\exists s \in L(G, x) \forall t \in L(G) (\theta^\Omega(s) \neq \theta^\Omega(t) \lor (\xi(x_0, t), e) \notin \Omega].
\]

Given \(x \in X, e \in \Sigma_o\), we let \(X^\Omega_{x,e} = \{q \in X : qT_\Omega x \land (q, e) \in \Omega\}\). Then the following is equivalent to (4.19):

\[
\exists x \in X \exists e \in \Sigma_o \exists (x, e) \notin TR(G) \land X^\Omega_{x,e} \neq \emptyset \land [\exists s \in L(G, x) \forall t \in L(G) (\theta^\Omega(s) \neq \theta^\Omega(t) \lor (\xi(x_0, t), e) \notin X^\Omega_{x,e}].
\]

Stating (4.20) differently, we obtain:

\[
\exists x \in X \exists e \in \Sigma_o \exists (x, e) \notin TR(G) \land X^\Omega_{x,e} \neq \emptyset \land [\exists s \in L(G, x) \forall t \in L(G) (\xi(x_0, t), e) \in X^\Omega_{x,e} \Rightarrow (\theta^\Omega(s) \neq \theta^\Omega(t)].
\]

We consider the problem of verifying (4.21).

**Problem 10.** Given a DFA \(G = (X, \Sigma, \xi, x_0)\), observable event set \(\Sigma_o\) and policy \(\Omega \subseteq TR(G)\), determine if (4.21) holds.

Our objective is to determine the computational complexity of Problem 10 (and hence (4.17)). To do so, we consider the complexity of a closely-related problem:

**Problem 11.** Given a DFA \(G = (X, \Sigma, \xi, x_0)\), observable event set \(\Sigma_o\), policy \(\Omega \subseteq TR(G)\), state \(x \in X\) and a set of states \(Q\) where \(QT_\Omega x\), determine if there exists a string \(s' \in L(G)\) such that \(x = \xi(x_0, s')\) and \(\forall s \in L(G), \xi(x_0, s) \in Q \Rightarrow (\theta^\omega(s) \neq \theta^\omega(s')).\)

We find that Problems 10 and 11 are polynomial-time reducible to one another.

**Lemma 7.** Problem 10 is polynomial-time reducible to Problem 11.
Proof. A simple algorithm is described for reducing Problem 10 to Problem 11. We are given \( \langle G, \Sigma_o, \Omega \rangle \), a tuple denoting an instance of Problem 10. For each \( x \in X \), \( e \in \Sigma_o \), determine if \((x, e) \notin TR(G)\). If so, then compute \( X^\Omega_{x,e} \). If \( X^\Omega_{x,e} \neq \emptyset \), then tuple \( \langle G, \Sigma_o, \Omega, x, X^\Omega_{x,e} \rangle \) is an instance of Problem 11. If a decider for Problem 11 returns \( \text{accept} \) on instance \( \langle G, \Sigma_o, \Omega, x, X^\Omega_{x,e} \rangle \) then there exists a string \( s' \in L(G) \) such that \( x = \xi(x_0, s') \) and \( \forall s \in L(G), \xi(x_0, s) \in X^\Omega_{x,e} \Rightarrow \theta^\Omega(s) \neq \theta^\Omega(s') \). In this case, (4.21) holds, so return \( \text{accept} \). After trying all \( x \in X \), \( e \in \Sigma_o \), if the decider for Problem 11 did not return \( \text{accept} \), then (4.21) does not hold, so return \( \text{reject} \).

For a given \( x \in X \), \( e \in \Sigma_o \), verifying if \((x, e) \notin TR(G)\) can be done in \( O(1) \) by querying if \( \xi(x, e)! \). Computing \( X^\Omega_{x,e} \) can be done by computing set \( Y \) where \( Y = \{ y \in X : yT_{\Omega} x \} \) then computing the set \( Z = \{ y \in Y : (y, e) \in \Omega \} \), where \( Z = X^\Omega_{x,e} \). Set \( Y \) may be computed by enumerating the set of indistinguishable state pairs \( T_{\Omega} \) and selecting the second element of a pair if the first element is \( x \). This is in \( O(|T_{\Omega}|) \subseteq O(|X|^2) \). Set \( X^\Omega_{x,e} \) may be computed by enumerating \( Y \) and \( \Omega \) and selecting each state \( y \in Y \) where \( (y, e) \in \Omega \) (though more efficient ways to compute \( X^\Omega_{x,e} \) certainly do exist). So computing set \( X^\Omega_{x,e} \) can be done in \( O(|Y| \cdot |\Omega|) \subseteq O(|X|^2 \cdot |X| \cdot |\Sigma_o|) = O(|X|^3 \cdot |\Sigma_o|) \). Testing for emptiness of \( X^\Omega_{x,e} \) can be done in \( O(1) \) if we use a flag to denote if, during computation of \( X^\Omega_{x,e} \), an element was added to \( X^\Omega_{x,e} \). Computing \( T_{\Omega} \) need only be done once using the product of Sears et al. \[31\] in \( O(|X|^2 \cdot |\Sigma| + |X|^3) \). So the algorithm described requires a number of operations in \( |X| \cdot |\Sigma| \cdot (O(1) + O(|X|^2) + O(|X|^3 \cdot |\Sigma_o|)) + O(|X|^2 \cdot |\Sigma| + |X|^3) \) which is in \( O(|X|^4 \cdot |\Sigma_o| \cdot |\Sigma|) \).

Lemma 8. Problem 11 is polynomial-time reducible to Problem 10.

Proof. An algorithm is described for reducing Problem 11 to Problem 10.
We are given \(\langle G, \Sigma_o, \Omega, x, Q\rangle\), a tuple denoting an instance of Problem \([11]\). First, conduct a test to determine if \(x \in Q\). If \(x \in Q\), then there does not exist a string \(s' \in \mathcal{L}(G)\) that satisfies the specification of Problem \([11]\). So return \text{reject}.

Otherwise, \(x \notin Q\). Let \(x'\) be a new state not in \(X\). Let \(Y = X \cup \{x'\}\). Let \(e\) denote a new event not in \(\Sigma\). Let \(\xi'\) be a copy of \(\xi\). The following additions are made to \(\xi'\). For all \(y \in Y\), for all \(\sigma \in \Sigma\), if \(\xi'(y, \sigma)\) is undefined then let \(\xi'(y, \sigma) = x'\). For all \(y \in Y \setminus \{x\}\), let \(\xi'(y, e) = x'\). So \(\xi'\) would be a total function were it not for \(\xi'(x, e)\) being undefined. Let \(\Omega' = \Omega \cup \{(q, e) : q \in Q\}\). Let \(G' = (Y, \Sigma \cup \{e\}, \xi', x_0)\). Tuple \(\langle G', \Sigma_o \cup \{e\}, \Omega'\rangle\) is an instance of Problem \([10]\), where \(\Sigma_o \cup \{e\}\) is taken to be the observable event set for this instance of Problem \([10]\).

By definition of (4.21), a decider for Problem \([10]\), when input with \(\langle G', \Sigma_o \cup \{e\}, \Omega'\rangle\), would return \text{accept} if and only if

\[
(\exists y \in Y)(\exists \alpha \in \Sigma_o \cup \{e\})(y, \alpha) \notin TR(G') \land Y^\Omega_{y,\alpha} \neq \emptyset \tag{4.22}
\]

\[
\land [([\exists s \in \mathcal{L}(G', y)](\forall t \in \mathcal{L}(G'))(\xi'(x_0, t) \in Y^\Omega_{y,\alpha} \Rightarrow \theta^\Omega(s) \neq \theta^\Omega(t))].
\]

By definition of \(\xi'\), \((Y \times (\Sigma \cup \{e\})) \setminus TR(G') = \{(x, e)\}\). Then, since \(e\) is in the observable event set for Problem \([10]\) instance \(\langle G', \Sigma_o \cup \{e\}, \Omega'\rangle\), (4.22) can be simplified to the following:

\[
Y^\Omega_{x, e} \neq \emptyset \tag{4.23}
\]

\[
\land ([\exists s \in \mathcal{L}(G', x)](\forall t \in \mathcal{L}(G'))(\xi'(x_0, t) \in Y^\Omega_{x, e} \Rightarrow \theta^\Omega(s) \neq \theta^\Omega(t))].
\]

By definition of \(\xi'\), we have \(\mathcal{L}(G) \subseteq \mathcal{L}(G')\). By definition of \(\Omega'\), we have, for all \(s \in \mathcal{L}(G)\), \(\theta^\Omega(s) = \theta^\Omega(s)\). Then \(QT_{\Omega'}x\) since \(QT_{\Omega}x\). Further, by definition of \(\Omega'\), the set of states in \(G'\) from which the new event \(e\) is observed is \(Q\). Thus \(Q = Y^\Omega_{x, e}\). Since
\[ Q \neq \emptyset, \] can be simplified to the following:

\[(\exists s \in \mathcal{L}(G', x))(\forall t \in \mathcal{L}(G'))\xi'(x_0, t) \in Q \Rightarrow \theta^\Omega'(s) \neq \theta^\Omega'(t). \] \hspace{1cm} (4.24)

By definition of \( \xi' \), for all \( y \in \{x\} \cup Q, \mathcal{L}(G', y) = \mathcal{L}(G, y) \). Equivalently, for all \( y \in \{x\} \cup Q, \{w \in \mathcal{L}(G) : \xi(x_0, w) = y\} = \{w \in \mathcal{L}(G') : \xi'(x_0, w) = y\} \) by definition of \( \mathcal{L}(G, y), \mathcal{L}(G', y) \). Then for all \( t \in \mathcal{L}(G'), \xi'(x_0, t) \in Q \) if and only if \( \xi(x_0, t) \in Q \). Recall that, for all \( s \in \mathcal{L}(G), \theta^\Omega(s) = \theta^\Omega'(s) \). From these four facts we can simplify (4.24) as follows:

\[(\exists s \in \mathcal{L}(G, x))(\forall t \in \mathcal{L}(G))\xi(x_0, t) \in Q \Rightarrow \theta^\Omega(s) \neq \theta^\Omega(t). \] \hspace{1cm} (4.25)

A decider for Problem 10, when input with \( \langle G', \Sigma_o \cup \{e\}, \Omega' \rangle \), would return accept if and only if (4.25) holds. When accept is returned, the decider for Problem 11 returns accept. Otherwise, reject is returned.

The reduction made here from Problem 11 to Problem 10 is in polynomial-time. Testing if \( x \in Q \) can trivially be implemented in a number of steps polynomial in \( |Q| \).

Constructing \( G' \) from \( G \) requires adding one state and at most \((|X| + 1) \cdot (|\Sigma| + 1) - 1\) transitions to \( G \). Constructing \( \Omega' \) requires adding \(|Q|\) transitions to \( \Omega \).

Next we prove the complexity of Problem 11, which is used to prove the complexity of Problem 10 (and hence the problem of verifying (4.17)). However, deciding Problem 11 is not trivial. We find that deciding Problem 11 is \textit{PSPACE-complete}. The proof of this follows in the remainder of this subsection.

In order to prove PSPACE-completeness we provide an algorithm in PSPACE for deciding Problem 11 and show that a related problem, Problem 12, which we prove to be PSPACE-complete, is polynomial-time reducible to Problem 11. Problem 12 is defined below.
Problem 12. Given an NFA $N = (X, \Sigma, \xi, x_0)$, state $x \in X$ and a set of states $Q \subseteq X \setminus \{x\}$, where for any $q \in Q$ there exists a string $s \in \mathcal{L}(N)$ such that $\{x, q\} \subseteq \xi(x_0, s)$, determine if there exists a string $s' \in \mathcal{L}(N)$ such that $x \in \xi(x_0, s')$ and $Q \cap \xi(x_0, s') = \emptyset$.

Next we provide an algorithm in PSPACE for solving Problem 12 and provide a polynomial-time reduction to Problem 12 from the problem of determining whether two NFA recognize the same language, which is known to be PSPACE-complete ([41]).

Lemma 9. Problem 12 is PSPACE-complete.

Proof. Problem 12 is in PSPACE: We provide Algorithm 1 which solves Problem 12 in linear space. Specifically, this algorithm solves a generalization of Problem 12 where we do not require that for any $q \in Q$ there exists a string $s \in \mathcal{L}(N)$ such that $\{x, q\} \in \xi(x_0, s)$.

We describe informally the operation of the algorithm. First we start with the $\varepsilon$-reach of the initial state, $x_0$. Denote this by $X'$. We determine if $x \in X'$ and if $Q \cap X' = \emptyset$. If so then there exists a string leading to $x$ that does not lead to any state in $Q$ and the algorithm terminates with accept. Otherwise, we nondeterministically choose a letter $e \in \Sigma \setminus \{\varepsilon\}$ labelling a transition from a state in $X'$, compute the set $X''$ of states reached by $e$ followed $\varepsilon$ transitions from a state in $X'$, set $X'$ to $X''$, then repeat the test mentioned. This is done until a certain bound is reached at which point the algorithm terminates with reject. We argue in the following why this bound is used.

When determinizing an NFA $N = (X, \Sigma, \xi, x_0)$ using the subset construction, in the worst-case the DFA constructed, denoted by $D$, will have $2^{|X|} - 1$ states. Consider the shortest string $s$ leading to a state in $D$ containing $x$ and not containing
any state in \( Q \). In the worst-case, for each nonempty subset \( \hat{X} \) of \( X \) that does not contain \( x \) or contains \( x \) and a state of \( Q \), there exists a prefix of \( s \) that leads to \( \hat{X} \). Then one can establish a bound on the length of \( s \) as follows. The number of nonempty subsets of \( X \) containing \( x \) (resp., not containing \( x \)) is \( 2^{(|X| - 1)} \) (resp., \( 2^{(|X| - 1)} - 1 \)). The number of nonempty subsets of \( X \) containing \( x \) and no element of \( Q \) is \( 2^{(|X| \setminus (Q \cup \{x\})|} = 2^{(|X| - 1)} \). Thus the number of subsets of \( X \) containing \( x \) and an element of \( Q \) is \( 2^{(|X| - 1)} - 2^{(|X| \setminus Q| - 1)} \). Thus a bound on the length of \( s \) is 
\[
\left[ 2^{(|X| - 1)} - 1 \right] + \left[ 2^{(|X| - 1)} - 2^{(|X| \setminus Q| - 1)} \right] - 1 + 1 = 2^{|X|} - (2^{(|X| \setminus Q| - 1)} + 1).
\]

**Algorithm 1** A decider for Problem 12

\[
\text{Require: } N = (X, \Sigma, \xi, x_0) \\
\text{Require: } x \in X \\
\text{Require: } Q \subseteq X
\]

1. \( X' \leftarrow \varepsilon\text{-reach}_\xi(x_0) \)
2. \( i \leftarrow 0 \)
3. \( \text{while } (x \notin X' \lor Q \cap X' \neq \emptyset) \land i < 2^{|X|} - (2^{|X| \setminus Q| - 1} + 1) \) do
4. \( \text{Nondeterministically select } e \text{ from } \Sigma \setminus \{\varepsilon\} \text{ where } \exists x' \in X', \xi(x', e)! \)
5. \( X'' \leftarrow \emptyset \)
6. \( \text{for all } x' \in X' \text{ do} \)
7. \( \text{if } \xi(x', e)! \text{ then} \)
8. \( \quad X'' \leftarrow X'' \cup \varepsilon\text{-reach}_\xi(\xi(x', e)) \)
9. \( \text{end if} \)
10. \( \text{end for} \)
11. \( X' \leftarrow X'' \)
12. \( i \leftarrow i + 1 \)
13. \( \text{end while} \)
14. \( \text{if } x \in X' \land Q \cap X' == \emptyset \text{ then} \)
15. \( \quad \text{accept} \)
16. \( \text{else} \)
17. \( \quad \text{reject} \)
18. \( \text{end if} \)

Next we characterize the space complexity of Algorithm 1. Note that the bounds we provide are not necessarily tight.
First, we evaluate the storage requirements for variables used in the algorithm. Each of \(x', X', X''\) requires \(O(|X|)\) tape cells for storage. Variable \(i\) requires \(\lceil \log(2^{|X|} - (2^{|X|} - Q^{-1}) + 1) \rceil \leq \lceil \log(2^{|X|}) \rceil = c \cdot |X| \in O(|X|)\) tape cells for storage where \(c > 0\). Variable \(e\) requires \(O(|\Sigma|)\) tape cells for storage.

Second, we evaluate the storage requirements for operations involving variables used in the algorithm. Computing \(X'\) in line 1 requires \(O(|X|)\) tape cells since computing the \(\varepsilon\)-reach of a state in \(X\) requires \(O(|X|)\) tape cells. The assignment to \(i\) on line 2 requires \(O(1)\) tape cells. The comparisons in the first clause of the loop invariant on line 3 requires \(O(|X|)\) tape cells. The comparison in the second clause of the loop invariant on line 3 requires \(O(|X|)\) tape cells. The assignment to \(e\) on line 4 requires \(O(|X| + |\Sigma|)\) tape cells. Iterating over \(X'\) on line 6 requires \(O(|X|)\) tape cells. Evaluating the condition of line 7 requires \(O(1)\) tape cells. The assignment on line 8 requires \(O(|X|)\) tape cells. The assignment on line 11 requires \(O(|X|)\) tape cells. The incrementing of \(i\) on line 12 can be done in place on \(i\) if \(i\) is represented appropriately and hence requires \(O(1)\) tape cells. The comparisons on line 14 require \(O(|X|)\) tape cells.

All of the storage requirements mentioned are further upper bounded by a constant multiple of the size of the input to Algorithm 1. Thus, Problem 12 is in NPSPACE. By Savitch’s Theorem ([27]), Problem 12 is also in PSPACE.

Every problem in PSPACE is polynomial-time reducible to Problem 12.

Next we provide a polynomial-time (many-to-one) reduction to Problem 12 from the problem of determining if two NFA accept the same language, which is known to be PSPACE-complete ([41]).
Consider two NFA $M$ and $N$ where

$$M = (X^M, \Sigma^M, x_0^M, \xi^M, F^M)$$

$$N = (X^N, \Sigma^N, x_0^N, \xi^N, F^N).$$

We would like to verify if $L_m(M) = L_m(N)$. The following characterization is important in reducing this verification to an instance of Problem 12:

$$L_m(M) = L_m(N)$$

if and only if

$$(L_m(M) \cup L_m(N)) \setminus (L_m(M) \cap L_m(N)) = \emptyset.$$

An automaton $M \cup N$ which accepts $L_m(M) \cup L_m(N)$ can be constructed in polynomial-time. Automaton $M \cup N$ has two new states, $x_{0}^{M \cup N}$ and $x_{F}^{M \cup N}$, denoting a new initial state and a new final state respectively. The new final state may seem superfluous at first, but it is necessary for the reduction to Problem 12 as will be seen. Specifically, $M \cup N = (X^M \cup X^N \cup \{x_0^{M \cup N}, x_F^{M \cup N}\}, \Sigma^M \cup \Sigma^N, x_0^{M \cup N}, \xi^{M \cup N}, \{x_F^{M \cup N}\})$ where

- $\xi^{M \cup N}$ contains all transitions in $\xi^M$ and $\xi^N$;
- $\xi^{M \cup N}(x_0^{M \cup N}, \varepsilon) = \{x_0^M, x_0^N\}$;
- for all $f \in F^M$, $\xi^{M \cup N}(f, \varepsilon) = \xi^M(f, \varepsilon) \cup \{x_F^{M \cup N}\}$;
- for all $f \in F^N$, $\xi^{M \cup N}(f, \varepsilon) = \xi^N(f, \varepsilon) \cup \{x_F^{M \cup N}\}$.

An automaton $M \cap N$ which accepts $L_m(M) \cap L_m(N)$ can be constructed in polynomial-time using the standard cross-product construction of $\varepsilon$-NFA. This cross-product is the same as $\otimes$ discussed later in Chapter 5 where it is shown that the product of two NFA using $\otimes$ can be computed in polynomial-time (Theorem 6). The
state set of \( M \cap N \), denoted by \( X^{M \cap N} \), is a subset of \( X^M \times X^N \). The set of accepting states of \( M \cap N \) is denoted by \( F^{M \cap N} \).

From automata \( M \cup N \) and \( M \cap N \) we construct an automaton \( V \), which is used to construct the instance of Problem 12 used for deciding if \( L_m(M) = L_m(N) \).

The construction of \( V \) is similar to the standard construction for the union of NFA (i.e., similar to the construction of \( M \cup N \)). However, we add a new symbol \( \alpha \) to the language generated by \( V \), such that string \( \alpha \) transitions to states \( x_{F}^{M \cup N} \) and \( F^{M \cap N} \) in \( V \). This too is done to construct the instance of Problem 12. Without loss of generality, we assume the state sets of \( M \cup N \) and \( M \cap N \) are pairwise-disjoint. Let \( x_0^{M \cap N} \) denote the initial state of \( M \cap N \). Let \( \xi^V \) denote the transition function of \( V \). We add a new state \( x_0^V \) to \( V \), where \( x_0^V \) is the initial state of \( V \) and \( \xi^V(x_0^V, \varepsilon) = \{x_0^{M \cup N}, x_0^{M \cap N}\} \). We also add a new symbol \( \alpha \), such that \( \alpha \notin \Sigma^M \cup \Sigma^N \).

We let \( \xi^V(x_0^V, \alpha) = \{x_F^{M \cup N}\} \cup F^{M \cap N} \). Also, function \( \xi^V \) contains all transitions of \( M \cup N \) and \( M \cap N \). Altogether, \( V \) has

- state set \( X^M \cup X^N \cup X^{M \cap N} \cup \{x_0^{M \cup N}, x_F^{M \cup N}, x_0^V\} \);

- event set \( \Sigma^M \cup \Sigma^N \cup \{\alpha\} \);

- initial state \( x_0^V \);

- transition function \( \xi^V \).

We do not consider \( V \) to have a set of accepting states, as this is not required for an instance of Problem 12.

Tuple \( \langle V, x_F^{M \cup N}, F^{M \cap N} \rangle \) is a valid input for a decider of Problem 12 by construction of \( V \), \( x_F^{M \cup N} \notin F^{M \cap N} \) and \( \{x_F^{M \cup N}\} \cup F^{M \cap N} \subseteq \xi(x_0^V, \alpha) \). From the construction of \( V \),
we have that \((L_m(M) \cup L_m(N)) \cup \{\alpha\} = \{s \in \mathcal{L}(V) : x_F^{\text{MUN}} \in \xi^V(x_0^V, s)\}\). Also, we have that \((L_m(M) \cap L_m(N)) \cup \{\alpha\} = \{s \in \mathcal{L}(V) : F^{\text{M\&N}} \cap \xi^V(x_0^V, s) \neq \emptyset\}\).

We use the facts in the above paragraph to justify the following two directions.

**Only if:**

Suppose the decider for Problem 12 returns accept on input \((V, x_F^{\text{MUN}}, F^{\text{M\&N}})\). Then there exists a string \(s \in \mathcal{L}(V)\) such that \(x_F^{\text{MUN}} \in \xi^V(x_0^V, s)\) and \(F^{\text{M\&N}} \cap \xi^V(x_0^V, s) = \emptyset\). It cannot be that \(s = \alpha\), for \(\alpha\) leads to \(x_F^{\text{MUN}}\) and states in \(F^{\text{M\&N}}\). Thus \((L_m(M) \cup L_m(N)) \setminus (L_m(M) \cap L_m(N)) \neq \emptyset\), so \(L_m(M) \neq L_m(N)\). Suppose instead that the decider for Problem 12 returns reject on input \((V, x_F^{\text{MUN}}, F^{\text{M\&N}})\). Then there does not exist a string \(s \in \mathcal{L}(V)\) such that \(x_F^{\text{MUN}} \in \xi^V(x_0^V, s)\) and \(F^{\text{M\&N}} \cap \xi^V(x_0^V, s) = \emptyset\). Thus \((L_m(M) \cup L_m(N)) \cup \{\alpha\} \subseteq (L_m(M) \cap L_m(N)) \cup \{\alpha\}\). Then \((L_m(M) \cup L_m(N)) \setminus (L_m(M) \cap L_m(N)) = \emptyset\), resulting in \(L_m(M) = L_m(N)\).

**If:**

Suppose that \(L_m(M) \neq L_m(N)\). Then \((L_m(M) \cup L_m(N)) \setminus (L_m(M) \cap L_m(N)) \neq \emptyset\). Then there exists \(s \in \mathcal{L}(V) \setminus \{\alpha\}\) such that \(x_F^{\text{MUN}} \in \xi^V(x_0^V, s)\) and \(F^{\text{M\&N}} \cap \xi^V(x_0^V, s) = \emptyset\). The decider for Problem 12 will return accept on input \((V, x_F^{\text{MUN}}, F^{\text{M\&N}})\) for this case. Suppose instead that \(L_m(M) = L_m(N)\). Then \((L_m(M) \cup L_m(N)) \setminus (L_m(M) \cap L_m(N)) = \emptyset\). Then there does not exist \(s \in \mathcal{L}(V) \setminus \{\alpha\}\) such that \(x_F^{\text{MUN}} \in \xi^V(x_0^V, s)\) and \(F^{\text{M\&N}} \cap \xi^V(x_0^V, s) = \emptyset\). The decider for Problem 12 will return reject on input \((V, x_F^{\text{MUN}}, F^{\text{M\&N}})\) for this case.

From the above two directions, we have that \(L_m(M) = L_m(N)\) if and only if the decider for Problem 12 returns reject on input \((V, x_F^{\text{MUN}}, F^{\text{M\&N}})\).

From the previous result we achieve the following.

**Theorem 3.** Problem 11 is PSPACE-complete.
Proof. Problem 11 is in PSPACE: Algorithm 1 can be adapted to solve Problem 11 in linear space with few modifications. The modifications are described:

1. In place of $\varepsilon$-reach we use the unobserved reach using policy $\Omega$, which is denoted as $UR_{\varepsilon}$-reach and defined as follows:
   \[
   UR_{\varepsilon}\text{-reach}(\hat{x}) = \{\hat{x}\} \cup \{\tilde{x} \in UR_{\varepsilon}\text{-reach}(\bar{x}) : \exists e \in \Sigma, \bar{x} = \xi(\hat{x}, e) \land (\hat{x}, e) \notin \Omega\}.
   \]

2. In line 4, instead of nondeterministically selecting $e$ from $\Sigma \setminus \{\varepsilon\}$ where $\exists x' \in X', \xi(x', e)!$, we nondeterministically select $e$ from $\Sigma$ where $\exists x' \in X', \xi(x', e)! \land (x', e) \in \Omega$.

3. In line 7, instead of using $\xi(x', e)!$ as the condition we use $\xi(x', e)! \land (x', e) \in \Omega$.

Note that the bounds we provide next are not necessarily tight. A depth-first search algorithm can be used to compute $UR_{\varepsilon}\text{-reach}(x)$ for a given $x \in X$ using $O(|X|)$ tape cells. The truth of the modified conditions of lines 4 and 7 can be tested using $O(|X| \cdot |\Sigma|)$ tape cells.

The storage requirements of this algorithm (including those of Algorithm 1) are further bounded above by a constant multiple of the size of the input to Problem 11. Thus, Problem 11 is in NPSPACE. By Savitch’s Theorem (27), Problem 11 is also in PSPACE.

Every problem in PSPACE is polynomial-time reducible to Problem 11. We prove that Problem 12 is polynomial-time reducible to Problem 11. By Lemma 9 it will then follow that every problem in PSPACE is polynomial-time reducible to Problem 11.
CHAPTER 4. DFA REPRESENTATIONS OF SENS. ACT. MAPS

Consider NFA $N = (X_N, \Sigma, \xi_N, x_0)$, $x \in X$ and $Q \subseteq X_N \setminus \{x\}$ where for any $q \in Q$ there exists a string $s \in \mathcal{L}(N)$ such that $\{x, q\} \subseteq \xi_N(x_0, s)$. Tuple $\langle N, x, Q \rangle$ denotes an instance of Problem 12.

From $N$ we construct NFA $N' = (X_N \cup X', \Sigma \cup \{\varepsilon\}, \xi', x_0)$. The NFA $N'$ is to be defined such that it utilizes the state-space of $N$ in addition to some auxiliary states and $\varepsilon$-transitions to these states which are defined in such a way that $\mathcal{L}(N') = \mathcal{L}(N)$ and nondeterminism from a state in $N'$ only occurs on $\varepsilon$-transitions.

Specifically, transition function $\xi'$ is defined in the same way as $\xi_N$ except for the following cases. For every $e \in \Sigma \setminus \{\varepsilon\}$, for every state $x_N \in X_N$, for every state $x_N' \in X_N$, if $x_N' \in \xi_N(x_N, e)$ and $|\xi_N(x_N, e)| > 1$ then a new state $x_{x_1 \rightarrow \varepsilon x_2}$ is included in $X'$, $x_N'$ is not included in $\xi'(x_N, e)$, $x_{x_1 \rightarrow \varepsilon x_2}$ is included in $\xi'(x_N, \varepsilon)$ and $x_N'$ is included in $\xi'(x_{x_1 \rightarrow \varepsilon x_2}, e)$.

It is easy to see that $\mathcal{L}(N') = \mathcal{L}(N)$.

From $N'$ we construct DFA $G = (X_N \cup X', \Sigma \cup \Sigma', \xi^G, x_0)$. The DFA $G$ is defined such that any transition labelled by $\varepsilon$ in $N'$ is relabelled with a unique symbol $\alpha \in \Sigma'$ where $\alpha \notin \Sigma \cup \{\varepsilon\}$. Furthermore, no two $\varepsilon$-transitions from $N'$ are labelled with the same symbol in $\Sigma'$. As a result, it is easy to see that $G$ is deterministic.

Specifically, for every $x_1 \in X_N \cup X'$, $\xi^G(x_1, \varepsilon) = \{x_1\}$. Also, let $\Sigma'$ be defined such that $\Sigma' \cap \Sigma = \emptyset$ and $|\Sigma'| = \sum_{x' \in X_N \cup X'} |\xi'(x', \varepsilon) \setminus \{x'\}|$. For every $x_1 \in X_N \cup X'$, for every $x_2 \in X_N \cup X'$, if $x_2 \in \xi'(x_1, \varepsilon) \setminus \{x_1\}$ then $x_2 \in \xi^G(x_1, \alpha)$ where $\alpha \in \Sigma'$.

Furthermore, for every $\alpha \in \Sigma'$, for every $x_1 \in X_N \cup X'$, for every $x_2 \in X_N \cup X'$, $\xi^G(x_1, \alpha)! \cap \xi^G(x_2, \alpha)! \Rightarrow x_1 = x_2$.

Take $\Sigma$ to be a set of observable events. Let $TR(G)$ denote the transitions of $G$. That is, for any event $e \in \Sigma \cup \Sigma'$, for any state $x' \in X_N \cup X'$, if $\xi^G(x', e)!$ then
(x', e) ∈ TR(G). Let policy Ω consist of all transitions in TR(G) labeled by events in Σ. That is, Ω = TR(G) \ \{(x', e) : x' ∈ X_N ∪ X' ∧ e ∈ Σ'\}.

By the previous two facts, we have that for all

We are given that for every q ∈ Q, there exists an s ∈ L(N) such that \{x, q\} ⊆ ξ(x_0, s). Consider projection \(P_\Sigma : (\Sigma ∪ \Sigma')^* → \Sigma^*\). The previous fact and the definition of \(L(G)\) imply \(P_\Sigma^{-1}(s) \cap \{s' ∈ L(G) : ξ^G(x_0, s') = x\} \neq \emptyset\) and \(P_\Sigma^{-1}(s) \cap \{s' ∈ L(G) : ξ^G(x_0, s') = q\} \neq \emptyset\). Then there exists \(s', s'' ∈ L(G), ξ^G(x_0, s') = x, ξ^G(x_0, s'') = q\) and \(P(s') = P(s'') = s\). By definition of Ω, for all \(w ∈ L(G), P_\Sigma(w) = θ_Ω(w)\). Then, by the previous two facts, there exists \(s', s'' ∈ L(G), ξ^G(x_0, s') = x, ξ^G(x_0, s'') = q\) and \(θ_Ω(s') = θ_Ω(s'') = s\). It follows by definition of \(T_Ω\) that \(QT_Ωx\).

Thus, \(⟨G, Σ, Ω, x, Q⟩\) is an instance of Problem 11. Next we show that there exists \(s ∈ L(N)\) such that \(x ∈ ξ_N(x_0, s)\) and \(Q ∩ ξ_N(x_0, s) = \emptyset\) if and only if a decider for Problem 11 returns accept on input \(⟨G, Σ, Ω, x, Q⟩\).

Only if:

Suppose a decider for Problem 11 returns accept on input \(⟨G, Σ, Ω, x, Q⟩\). Then there exists \(s' ∈ L(G)\) such that \(x = ξ^G(x_0, s')\) and for all \(s ∈ L(G), ξ^G(x_0, s) ∈ Q\) implies \(θ_Ω(s) ≠ θ_Ω(s')\). By definition of Ω, for all \(w ∈ L(G), P_\Sigma(w) = θ_Ω(w)\). Then there exists \(s' ∈ L(G)\) such that \(x = ξ^G(x_0, s')\) and for all \(s ∈ L(G), ξ^G(x_0, s) ∈ Q\) implies \(P_Σ(s) ≠ P_Σ(s')\). By definition of G, for all \(w ∈ L(G), ξ^G(x_0, w) ∈ ξ_N(x_0, P_Σ(w))\).

By the previous two facts, we have that for all \(s ∈ L(G), ξ^G(x_0, s) ∈ Q\) implies \(P_Σ(s) ≠ P_Σ(s')\) which implies \(Q ∩ ξ_N(x_0, P_Σ(s')) = \emptyset\). We also have that \(x ∈ ξ_N(x_0, P(s'))\) since \(x = ξ^G(x_0, s')\). Thus there exists a \(t ∈ L(N)\) such that \(x ∈ ξ_N(x_0, t)\) and \(Q ∩ ξ_N(x_0, t) = \emptyset\).

If:

Suppose there exists \(s' ∈ L(N)\) such that \(x ∈ ξ_N(x_0, s')\) and \(Q ∩ ξ_N(x_0, s') = \emptyset\).
Then, by definition of $G$, 

$$P_{\Sigma}^{-1}(s') \cap \{ s \in L(G) : \xi^G(x_0, s) = x \} \neq \emptyset$$

(4.26)

and for all $q \in Q$, 

$$P_{\Sigma}^{-1}(s') \cap \{ s \in L(G) : \xi^G(x_0, s) = q \} = \emptyset.$$ (4.27)

By (4.26), there exists $t \in L(G)$ such that $\xi^G(x_0, t) = x$ and $P_{\Sigma}(t) = s'$. By (4.27), for all $t' \in L(G)$, if $\xi^G(x_0, t') \in Q$ then $P_{\Sigma}(t') \neq s'$. By these two facts, there exists $t \in L(G)$ such that $\xi^G(x_0, t) = x$ and, for all $t' \in L(G)$, if $\xi^G(x_0, t') \in Q$ then $P_{\Sigma}(t') \neq P_{\Sigma}(t)$. By definition of $\Omega$, for all $w \in L(G)$, $P_{\Sigma}(w) = \theta^\omega \omega (w)$. Thus there exists $t \in L(G)$ such that $\xi^G(x_0, t) = x$ and, for all $t' \in L(G)$, if $\xi^G(x_0, t') \in Q$ then $\theta^\omega \omega (t') \neq \theta^\omega \omega (t)$. So the decider for Problem 11 returns accept on input $\langle G, \Sigma, \Omega, x, Q \rangle$.

Reduction is in polynomial-time:

Constructing $N'$ from $N$ can be done in polynomial time in $|X_N|$ and $|\Sigma|$. A simple algorithm for computing $N'$ would proceed by iterating through states in $X_N$. If an event $e \in \Sigma \setminus \{ \varepsilon \}$ labels multiple transitions from the current state $x_N^1 \in X_N$ then we replace every transition to a state $x_N^2$ on $e$ from $x_N^1$ by a transition from $x_N^1$ to a new state $x_{x_1 \rightarrow \varepsilon x_2}$ labelled by $\varepsilon$ followed by a transition from $x_{x_1 \rightarrow \varepsilon x_2}$ to $x_N^2$ on $e$. This algorithm is in $O(|X_N|^2 \cdot |\Sigma|)$ which generates $N'$ containing $O(|X_N|^2 \cdot |\Sigma|)$ states.

Constructing $G$ and $\Omega$ from $N'$ can be done in polynomial time in $|X_N \cup X'|$ and $|\Sigma|$. A simple algorithm for computing $G$ and $\Omega$ iterates over the transitions of $N'$. If a transition is encountered that is labeled with $\varepsilon$ then it is relabeled with a new symbol $\alpha$. Otherwise, the transition is added to $\Omega$. An upper bound for the algorithm is $O(|X_N \cup X'|^2 \cdot |\Sigma| \cup \{ \varepsilon \}|)$.

As $|X_N \cup X'|$ is in $O(|X_N|^2 \cdot |\Sigma|)$, it follows that the reduction from Problem 12
to Problem 11 is in $O(|X_N|^2 \cdot |\Sigma| + (|X_N|^2 \cdot |\Sigma|)^2 \cdot |\Sigma \cup \{\varepsilon\}|) \subseteq O(|X_N|^2 \cdot |\Sigma| + |X_N|^4 \cdot |\Sigma|^2 \cdot (|\Sigma| + 1)) = O(|X_N|^2 \cdot |\Sigma| + |X_N|^4 \cdot |\Sigma|^3 + |X_N|^4 \cdot |\Sigma|) = O(|X_N|^4 \cdot |\Sigma|^3)$. Thus the reduction from Problem 12 to Problem 11 is in polynomial time. Since Problem 12 is PSPACE-complete, every problem in PSPACE is polynomial-time reducible to Problem 11.

From the above we arrive at the main result of this subsection.

\textbf{Corollary 2.} The problem of verifying (4.17) is PSPACE-complete.

\textit{Proof.} By Theorem 3, Lemmata 7 and 8, we have that Problem 10 is PSPACE-complete. By (4.19)–(4.21) and the definition of Problem 10, we have that the problem of verifying (4.17) is PSPACE-complete.

4.5.5 Verifying if sensor activation conflicts exist between state equivalence classes

In this subsection we consider verification of whether or not sensor activation conflicts exist between state equivalence classes in a state equivalence class estimate (i.e., if sensor activation conflicts result due to, roughly speaking, differences between the elements of $[\theta^o(s)]$ for some $s \in \mathcal{L}(G)$). This corresponds to the third scenario described in Subsection 4.5.2. Formally, we consider the problem of verifying whether or not condition (4.18) holds. We demonstrate that condition (4.18) can be verified in polynomial-time. For verification purposes, we presume that the first and second scenarios described in Subsection 4.5.2 do not occur (i.e., (4.16) and (4.17) hold).

We consider the following procedure for verifying if (4.18) is satisfied for a given plant $G = (X, \Sigma_o \cup \Sigma_u, \xi, x_0)$ and policy $\Omega$. Verification is conducted using the
indistinguishable state pairs of the automaton \([G]\) defined in Section 4.4. However, the transition function of \([G]\) is generalized here since a state equivalence class estimate (i.e., \([\theta^\Omega(s)]\)) is not necessarily a singleton set for an arbitrary \(G\) and \(\Omega\), so \([G]\) is not necessarily deterministic.

First, compute NFA \([G]\). This is analogous to the computation of DFA \([G]\) (4.15). Compute the set of state equivalence classes \([X]\) of \([G]\). This can be done using a depth-first search algorithm in \(\Theta(|X| + |T_\Omega|)\) which is described at the end of Section 4.4. After computing \([X]\) construct NFA \([G] = ([X], \Sigma_o, \xi[G], [x_0])\). Transition function \(\xi[G]\) is defined as follows:

\[
\forall \hat{x} \in [X], \forall e \in \Sigma_o, \xi[G](\hat{x}, e) = \{[\xi(x, e)] : \exists x \in [\hat{x}], (x, e) \in \Omega\}.
\]

Constructing \(\xi[G]\) also requires a map from each state \(x \in X\) to its corresponding state class. Given \([X]\) this can be computed in \(O(|X|)\). Then computing \(\xi[G]\) is in \(O(|X| \cdot |\Sigma_o|)\).

To demonstrate that the indistinguishable state pairs of \([G]\) may be used to verify (4.18) we first demonstrate a relation between state equivalence class estimates and the states reached by strings in \([G]\). Also, we demonstrate that the language generated by the observer automaton \(DET_{\Omega}(G)\) is a subset of the language generated by \([G]\) and, when (4.16)–(4.18) hold, these two languages are equivalent.

**Lemma 10.** Given plant \(G\) and \(\Omega\), for all \(s \in \mathcal{L}(G)\), \([\theta^\Omega(s)] = \xi[G]([x_0], \theta^\Omega(s))\).

*Proof.* Given \(s \in \mathcal{L}(G)\), in the following we prove \([\theta^\Omega(s)] = \xi[G]([x_0], \theta^\Omega(s))\) by induction on the length of \(\theta^\Omega(s)\).

Suppose \(|\theta^\Omega(s)| = 0\). Then \(\theta^\Omega(s) = \varepsilon\). By definition of (4.14), \([\theta^\Omega(\varepsilon)] = \{[x_0]\}\).

By definition of \(\xi[G]\), \(\xi[G]([x_0], \varepsilon) = \{[x_0]\}\). The base case holds.
Suppose $|θ^Ω(s)| = n + 1$. Then $θ^Ω(s) = σ_1 ... σ_nσ_{n+1}$ where $σ_i ∈ Σ_o$. By the inductive hypothesis, $[σ_1 ... σ_n] = ξ_G([x_0], σ_1 ... σ_n)$. The following holds by (4.14),

$$[σ_1 ... σ_nσ_{n+1}] = \{[x] : ∃q_n, q_{n+1} ∈ X, [q_n] ∈ [σ_1 ... σ_n], [x] = [q_{n+1}], (q_n, σ_{n+1}) ∈ Ω \land ξ(q_n, σ_{n+1}) ∈ [q_{n+1}]\}.$$ 

Then, by $[σ_1 ... σ_n] = ξ_G([x_0], σ_1 ... σ_n),$ 

$$[σ_1 ... σ_nσ_{n+1}] = \{[x] : ∃q_n, q_{n+1} ∈ X, [q_n] ∈ ξ_G([x_0], σ_1 ... σ_n), [x] = [q_{n+1}], (q_n, σ_{n+1}) ∈ Ω \land ξ(q_n, σ_{n+1}) ∈ [q_{n+1}]\}.$$ 

Then, by definition of $ξ_G,$

$$[σ_1 ... σ_nσ_{n+1}] = \{[x] : ∃q_n, q_{n+1} ∈ X, [q_n] ∈ ξ_G([x_0], σ_1 ... σ_n), [x] = [q_{n+1}], [q_{n+1}] ∈ ξ_G([q_n], σ_{n+1})\}.$$ 

The set on the right side is $ξ_G([x_0], σ_1 ... σ_nσ_{n+1}).$ 

Then $[θ^Ω(s)] = ξ_G([x_0], θ^Ω(s)).$ \hfill \Box

With abuse of notation, let $θ^Ω(ℒ(G)) = \{θ^Ω(s) : s ∈ ℒ(G)\}$. Note that $θ^Ω(ℒ(G))$ is the language generated by $DET_Ω(G)$. In general, it is not necessarily the case that $θ^Ω(ℒ(G)) = ℒ([G])$. However, this is the case when (4.16) - (4.18) hold.

**Lemma 11.** Given plant $G$ and $Ω$, if (4.16) - (4.18) hold then $θ^Ω(ℒ(G)) = ℒ([G]).$

**Proof.** ($⊆$) : From Lemma 10 we have, for all $s ∈ ℒ(G)$, $ξ_G([x_0], θ^Ω(s))!$. It follows that $θ^Ω(ℒ(G)) ⊆ ℒ([G]).$

($⊇$) :

Suppose (4.16) - (4.18) hold for the given plant $G$, policy $Ω$. Assume there exists $s ∈ ℒ([G])$ where $s ∉ θ^Ω(ℒ(G))$. Let $s'$ denote the shortest prefix of $s$ such that $s' ∈ ℒ([G])$ and $s' ∉ θ^Ω(ℒ(G))$. String $s'$ must be nonempty since $ε$ is a prefix of all
strings, \( \varepsilon \in \mathcal{L}([G]) \) and \( \varepsilon \in \theta^\Omega(\mathcal{L}(G)) \). Then \( s' = te \) where \( e \in \Sigma_o \). By definition of \( s' \) and the fact that \( \mathcal{L}([G]) \) is prefix-closed, \( t \in \mathcal{L}([G]) \) and \( t \in \theta^\Omega(\mathcal{L}(G)) \). Since \( t \in \theta^\Omega(\mathcal{L}(G)) \), there exists a \( w \in \mathcal{L}(G) \) such that \( \theta^\Omega(w) = t \). Let \( x_w = \xi(x_0, w) \). By definition of \( (4.14) \), \( [x_w] \in \theta^\Omega(w) \).

Since \( te \in \mathcal{L}([G]) \), there exists \( [x] \in \xi([G])([x_0], t) \) such that \( \xi([G])([x], e) \). State \( [x] \in \theta^\Omega(w) \) by Lemma 10 and \( [x] \in \xi([G])([x_0], t) \).

We have that there exists \( q_w \in [x_w], (q_w, e) \in \Omega \) if and only if there exists \( q \in [x], (q, e) \in \Omega \) by (4.18), \( [x] \in \theta^\Omega(w) \), \( [x_w] \in \theta^\Omega(w) \) and \( e \in \Sigma_o \). There exists \( q \in [x] \) such that \( (q, e) \in \Omega \) by \( \xi([G])([x], e) \) and the definition of \( \xi([G]) \). By the previous two facts, there exists \( q_w \in [x_w], (q_w, e) \in \Omega \).

Suppose \( (x_w, e) \in TR(G) \). Then, by \( q_w \in [x_w], (q_w, e) \in \Omega \), and \( (4.16) \), \( (x_w, e) \in \Omega \). Hence \( \theta^\Omega(we) = te = s' \) and so \( s' \in \theta^\Omega(\mathcal{L}(G)) \), a contradiction.

Suppose \( (x_w, e) \notin TR(G) \). We have that \( q_w \neq x_w \) since \( (q_w, e) \in TR(G) \) and \( (x_w, e) \notin TR(G) \). The following holds by \( \{q_w, x_w\} \subseteq [x_w] \), by \( q_w \neq x_w \) and by definition of \( [x_w] \):

\[
\exists n > 0, \exists x^0, x^1, x^2, \ldots, x^n \in X, x^0 = x_w, x^n = q_w, x^1 T_\Omega x^0 \land x^2 T_\Omega x^1 \land \ldots \land x^n T_\Omega x^{n-1}.
\]

Let \( m \leq n \) be defined such that \( (x^m, e) \in TR(G) \) and, for all \( 0 \leq i < m \), \( (x^i, e) \notin TR(G) \). Such an \( m \) must exist since \( (q_w, e) \in TR(G) \). Then \( (x^m, e) \in \Omega \) by \( (q_w, e) \in \Omega \) and \( (4.16) \).

Let \( z_i = x^{m-i} \) for \( 0 \leq i \leq m \). For all \( u_1 \in \mathcal{L}(G, z_1) \), there exists \( v_1 \in \mathcal{L}(G) \) such that \( \theta^\Omega(u_1) = \theta^\Omega(v_1) \) and \( (\xi(x_0, v_1), e) \in \Omega \) by \( (z_1, e) \notin TR(G) \), \( (z_0, e) \in \Omega \), \( z_1 T_\Omega z_0 \) and \( (4.17) \).

Let \( 1 \leq i < m \). That \( z_i T_\Omega z_{i+1} \) implies there exists \( s_i, s_{i+1} \in \mathcal{L}(G) \) such that

Let \( (4.16) \):
\[ \xi(x_0, s_i) = z_i, \xi(x_0, s_{i+1}) = z_{i+1} \] and \( \theta^\Omega(s_i) = \theta^\Omega(s_{i+1}) \). Now suppose for all \( u_i \in \mathcal{L}(G, z_i) \), there exists \( v_i \in \mathcal{L}(G) \) such that \( \theta^\Omega(u_i) = \theta^\Omega(v_i) \) and \( (\xi(x_0, v_i), e) \in \Omega \). Then, since \( \xi(x_0, s_i) = z_i \), there exists \( s'_i \in \mathcal{L}(G) \) such that \( \theta^\Omega(s_i) = \theta^\Omega(s'_i) \) and \( (\xi(x_0, s'_i), e) \in \Omega \). Then \( \theta^\Omega(s_{i+1}) = \theta^\Omega(s'_i) \). From this discussion, we have that \( z_{i+1} \in \text{TR}(G) \) and \( (\xi(x_0, v_{i+1}), e) \in \Omega \). Then, by (4.17), for all \( u_{i+1} \in \mathcal{L}(G, z_{i+1}) \), there exists \( v_{i+1} \in \mathcal{L}(G) \) such that \( \theta^\Omega(u_{i+1}) = \theta^\Omega(v_{i+1}) \) and \( (\xi(x_0, v_{i+1}), e) \in \Omega \).

From the above discussion and the fact that

- for all \( u_1 \in \mathcal{L}(G, z_1) \), there exists \( v_1 \in \mathcal{L}(G) \) such that \( \theta^\Omega(u_1) = \theta^\Omega(v_1) \) and \( (\xi(x_0, v_1), e) \in \Omega \)
- for all \( u_i \in \mathcal{L}(G, z_i) \), there exists \( v_i \in \mathcal{L}(G) \) such that \( \theta^\Omega(u_i) = \theta^\Omega(v_i) \) and \( (\xi(x_0, v_i), e) \in \Omega \).

Thus, for all \( u \in \mathcal{L}(G, x_w) \), there exists \( v \in \mathcal{L}(G) \) such that \( \theta^\Omega(u) = \theta^\Omega(v) \) and \( (\xi(x_0, v), e) \in \Omega \). It follows that, since \( \xi(x_0, w) = x_w \), there exists \( w' \in \mathcal{L}(G) \) such that \( \theta^\Omega(w) = \theta^\Omega(w') \) and \( (\xi(x_0, w'), e) \in \Omega \). Hence \( \theta^\Omega(w'e) = te = s' \) and so \( s' \in \theta^\Omega(\mathcal{L}(G)) \), a contradiction. \( \Box \)

Given plant \( G \) and policy \( \Omega \), suppose (4.16) - (4.18) hold. Then, by Lemmata 10 and 11 (4.18) is equivalent to the following:

\[ (\forall s \in \mathcal{L}([G]))(\forall e \in \Sigma_o)(\forall q_1, q_2 \in \xi_G([x_0], s)), \]
\[ (\exists \tilde{q}_1 \in [q_1], (\tilde{q}_1, e) \in \Omega \iff \exists \tilde{q}_2 \in [q_2], (\tilde{q}_2, e) \in \Omega). \]
By definition of $\xi_{[G]}$, (4.28) is equivalent to the following:

\[(\forall s \in \mathcal{L}([G]))(\forall e \in \Sigma_o)(\forall [q_1], [q_2] \in \xi_{[G]}([x_0], s)),\]

\[(((q_1, e) \in TR([G]) \iff (q_2, e) \in TR([G]))).
\]

Let $P$ denote the set of pairs of states in $[G]$ that are reached by a common string (i.e., $P$ is the set of indistinguishable state pairs of $[G]$). That is, $P = \{([x], [y]) \in [X] \times [X] : \exists w \in \mathcal{L}([G]), \{[x], [y]\} \subseteq \xi_{[G]}([x_0], w)\}$. Then (4.29) is equivalent to the following:

\[(\forall e \in \Sigma_o)(\forall ([q_1], [q_2]) \in P), (([q_1], e) \in TR([G]) \iff ([q_2], e) \in TR([G])).
\]

Conversely, given plant $G$ and policy $\Omega$, suppose (4.18) does not hold. That is,

\[(\exists s \in \mathcal{L}(G))(\exists e \in \Sigma_o)(\exists [q_1], [q_2] \in [\theta^\Omega(s)]),
\]

\[(\exists \tilde{q}_1 \in [q_1], (\tilde{q}_1, e) \in \Omega \land \forall \tilde{q}_2 \in [q_2], (\tilde{q}_2, e) \notin \Omega).
\]

By Lemma 10, (4.31) is equivalent to the following:

\[(\exists s \in \mathcal{L}([G]))(\exists e \in \Sigma_o)(\exists [q_1], [q_2] \in \xi_{[G]}([x_0], s)),
\]

\[(\exists \tilde{q}_1 \in [q_1], (\tilde{q}_1, e) \in \Omega \land \forall \tilde{q}_2 \in [q_2], (\tilde{q}_2, e) \notin \Omega).
\]

By definition of $\xi_{[G]}$, (4.32) is equivalent to the following:

\[(\exists s \in \mathcal{L}([G]))(\exists e \in \Sigma_o)(\exists [q_1], [q_2] \in \xi_{[G]}([x_0], s)),
\]

\[\xi_{[G]}([q_1], e) \in TR([G]) \land \xi_{[G]}([q_2], e) \notin TR([G]).
\]

By definition of $P$, (4.33) is equivalent to the following:

\[(\exists e \in \Sigma_o)(\exists ([q_1], [q_2]) \in P), ([q_1], e) \in TR([G] \land ([q_2], e) \notin TR([G]).
\]

Thus, by (4.30) and (4.34), we can verify if (4.18) holds (supposing (4.16), (4.17) hold) or does not hold by comparing the set of events labeling transitions from the
states of pairs in $P$. Specifically, for each pair $([x],[y]) \in P$, test if $\{e : ([x],e) \in TR([G])\} = \{e : ([y],e) \in TR([G])\}$. If this holds, then (4.18) holds. Otherwise, (4.18) does not hold. Set $P$ can be computed using the product of Sears et al. [31] in $O(|X|^4 \cdot |\Sigma|)$ where $\Sigma = \Sigma_o \cup \Sigma_uo$ (given that $[G]$ is nondeterministic in general).

Specifically, set $P$ is the state set of automaton $[G] \otimes [G]$ where $\otimes$ is defined in Sears et al. [31] (and in Chapter 5). Comparing the set of events labeling transitions from states in pairs of $P$ is in $O(|X|^2 \cdot |\Sigma_o|) \subseteq O(|X|^2 \cdot |\Sigma_o|)$ if $\xi_{[G]}$ is used to query if a transition labeled by an event exists from a given state in $[G]$.

We recall briefly how $[G]$ may be used to compute sensor activation decisions consistent with $DET_\Omega(G)$. Given plant $G$ and policy $\Omega$ which satisfy (4.16) - (4.18), if event sequence $s$ is generated by $G$ then $s$ is observed as $\theta^\Omega(s)$ and the set of event sensors to be turned on (i.e., $\omega(s)$ where $\omega$ is derived from $\Omega$ by (4.2)) is the set of events labeling transitions in $[G]$ from any single state in $\xi_{[G]}([x_0], \theta^\Omega(s))$. Computing $\xi_{[G]}([x_0], \theta^\Omega(s))$ in its entirety is not required for purposes of determining which event sensors to turn on and off. One only needs to track a single state $[x]$ in $\xi_{[G]}([x_0], \theta^\Omega(s))$ and activate event sensors according to events labeling transitions from $[x]$. Afterward, when an event $e$ is observed, one can arbitrarily choose a state in $\xi_{[G]}([x_0], \theta^\Omega(s))$ (which is in $\xi_{[G]}([x_0], \theta^\Omega(s)e))$ and activate event sensors according to events labeling transitions from the chosen state.

We note that, for a given $s \in L(G)$, when $|\theta^\Omega(s)| = 1$, it must be the case that the agent’s other decisions (e.g., control, communication) can be based on the single state equivalence class in $[\theta^\Omega(s)]$ (i.e., the single state reached in $\xi_{[G]}([x_0], \theta^\Omega(s)))$. However, when $|\theta^\Omega(s)| > 1$, though (4.16) - (4.18) may hold it is not necessarily the case that an agent can base its other decisions on an arbitrary state equivalence class.
in \([\theta^R(s)]\). Verification of whether or not other decisions can be made using \([G]\) can be done by checking whether or not the decisions following two state equivalence classes \([x_1], [x_2]\) are equivalent for all \(([x_1], [x_2]) \in P\). This verification is in \(O(|X|^2 \cdot |\Sigma_0|) \subseteq O(|X|^2 \cdot |\Sigma_0|)\).

### 4.6 Conclusions

In this chapter we considered sensor activation policies which satisfy various notions of feasibility. In Section 4.3 we considered sensor activation policies which satisfy a very strong notion of feasibility. In Section 4.4 we considered sensor activation policies which are more general. Finally, we considered when the coarse estimate of the true state of the system can be used for computing a map from observed event sequences to sensor activation decisions in Section 4.5. For the classes of sensor activation policies considered in each section we demonstrated how a map from observed event sequences to sensor activation decisions can be computed in polynomial time. However, determining if an arbitrary sensor activation policy belongs to the policies considered in Section 4.5 is PSPACE-complete.

Investigating other classes of sensor activation policies from which computation of a deterministic map from observed event sequences to sensor activation decisions can be done efficiently remains as future work.
Chapter 5

Computation and Applications of Indistinguishable State Pairs in Discrete-Event Systems

Note that this chapter is derived from [33].

In this chapter we consider the computation of indistinguishable state pairs of non-deterministic finite automata where some transitions of the automata are observable whereas other transitions are not observable.

5.1 Introduction

We consider nondeterministic finite-state automata whose transitions are partitioned into two sets: those transitions which are observable and those which are unobservable. If an observable transition occurs from one state to another in a given automaton then the label of the transition is observed, not the particular transition itself.
Conversely, if an unobservable transition (i.e., $\varepsilon$-transition) occurs then nothing is observed.

The finite automata that we consider in this chapter contain a unique initial state. As not all transitions are observable, it may be the case that two different transition sequences from the initial state of a given automaton result in the same observed event sequence. The states reached by such a pair of transition sequences are said to be indistinguishable.

In this chapter we consider the problem of computing the indistinguishable state pairs of a given automaton with partially observable transitions. We demonstrate how the indistinguishable state pairs of a given automaton may be computed by application of the product of nondeterministic finite automata.

The study of indistinguishable state pairs in finite automata is important in discrete-event systems. For instance, several discrete-event system-theoretic properties such as observability [17], diagnosability [26], detectability [40] and feasibility (of sensor activation and communication policies) [51, 23, 29] may be verified in a straightforward manner by querying which pairs of states of a given automaton are indistinguishable.

In Section 5.2 we provide the concepts and notation used throughout the chapter.

The problem of computing indistinguishable state pairs has been considered previously [50]. In Section 5.3 we review the CLUSTER-TABLE algorithm for computing indistinguishable state pairs [50]. We describe CLUSTER-TABLE and demonstrate for a specific parameterized example that CLUSTER-TABLE is in $O(|X|^4 \cdot |\Sigma|)$ where $X$ denotes the state set and $\Sigma$ denotes the event set of the input automaton.
In Section 5.4 we recall the product of nondeterministic finite automata, denoted throughout by $\otimes$. We demonstrate an equivalence between the indistinguishable state pairs of an automaton $G$ and the states of $G \otimes G$. We propose an algorithm for the computation of $G_1 \otimes G_2$ for given automata $G_1, G_2$. This algorithm follows in a straightforward manner from the definition of the product. Given automaton $G$, let $X$ denote the state set of $G$ and $\Sigma$ the event set of $G$. The proposed algorithm is in $O(|X|^2 \cdot |\Sigma| + |X|^3)$ when the observable transitions of $G$ are deterministic. Otherwise, the proposed algorithm is in $O(|X|^4 \cdot |\Sigma|)$.

In Section 5.5 we define the construction of a quotient automaton when automaton $G$ contains cycles of unobservable transitions. We demonstrate an equivalence between the indistinguishable state pairs of automaton $G$ and the states of the product of the quotient automaton with itself. In certain cases, it is more efficient to compute the indistinguishable state pairs of $G$ from the quotient automaton rather than from $G$ directly.

In Section 5.6 we provide an example demonstrating computation of the quotient automaton and the product.

In Section 5.7 we demonstrate applications of using $\otimes$ for computing indistinguishable state pairs. We propose unique tests for verifying observability [17], coobservability [24] and eventual feasibility [29]. We demonstrate how $\otimes$ may be used to compute the extended specification used in problems of sensor activation [53].

In Section 5.8 we contrast the problems of determining if two states are indistinguishable versus the problem of verifying if there exists a string that leads to one state, but not another. While the former problem may be verified in polynomial-time, we show that the latter problem is PSPACE-complete.
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

5.2 Preliminaries

Here some preliminary concepts and notation specific to this chapter are introduced. We refer the reader to Chapter 2 for a summary of concepts and notation used in this chapter and other chapters of the dissertation.

Recall that an NFA, $G$, is denoted by a tuple $G = (X, \Sigma, \delta, x_0)$ where $X$ is the nonempty set of states, $\Sigma$ is the alphabet, $x_0$ is the initial state, and $\delta : X \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^X$ is the (state-)transition function. Given $\delta$, we can define the set of observable transitions, $\delta_o$, and set of unobservable transitions (commonly called $\varepsilon$-transitions), $\delta_\varepsilon$. Specifically, $\delta_\varepsilon = \{ (x, \varepsilon, y) \in \delta : x, y \in X \}$ and $\delta_o = \delta \setminus \delta_\varepsilon$.

Automaton $G$ can be considered as a model of an untimed discrete-event system (DES). Typical in problems considered in the DES literature is the presence of one or more agents which observe events generated by a DES. We consider that a single agent observes events generated by a given DES $G$. Given $\delta$, transition set $\delta_o$ (resp., $\delta_\varepsilon$) can be regarded as the set of transitions of $G$ that the agent observes (resp., does not observe). Specifically, if transition $(x, \sigma, y) \in \delta_o$ (resp., $(x, \varepsilon, y) \in \delta_\varepsilon$) occurs then event $\sigma$ (resp., nothing) is observed by the agent.

Given $x \in X$, let $\delta_\varepsilon^*(x)$ denote the $\varepsilon$-closure of $x$ in $G$. Formally, $\delta_\varepsilon^*(x) = \{ x \} \cup \{ x' \in X : \exists x_1, \ldots, x_n \in X, x_1 = x \land x_n = x' \land (x_i, \varepsilon, x_{i+1}) \in \delta_\varepsilon \}$.

A pair of states $(x, y) \in X \times X$ is indistinguishable if $x$ and $y$ are both contained in the same state of $G_{obs}$. See Chapter 3.1 for the definition of $G_{obs}$. Formally, the set of indistinguishable state pairs of $G$ is set

$$\Pi = \{ (x, y) : \exists q \in Q, x \in q \land y \in q \}. \quad (5.1)$$

We recall the product of $\varepsilon$-NFA ([10] Exercise 4.2.14), which is used throughout
this chapter. We denote the product by \( \otimes \). For purposes of defining product \( \otimes \), we suppose that NFA \( G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}) \) is given where \( i \in \{1, 2\} \). Denote the observable (resp., unobservable) transition set of \( G_i \) by \( \delta_{i,o} \) (resp., \( \delta_{i,\varepsilon} \)). Let \( \Gamma_i : X_i \to 2^{\Sigma_i} \) denote a map from states \( x \in X_i \) to events in \( \Sigma_i \) labeling transitions from \( x \) in \( \delta_{i,o} \). That is, \( \Gamma_i(x) = \{ \sigma \in \Sigma_i : \exists x' \in X_i, (x, \sigma, x') \in \delta_{i,o} \} \).

We define NFA \( G_1 \otimes G_2 = (\mathcal{X}, \Sigma_1 \cap \Sigma_2, \Delta, (x_{0,1}, x_{0,2})) \) where \( \mathcal{X} \subseteq X_1 \times X_2 \) and \( \Delta : \mathcal{X} \times ((\Sigma_1 \cap \Sigma_2) \cup \{ \varepsilon \}) \to 2^\mathcal{X} \). Transition function \( \Delta \) is defined such that, for a given state pair \( (x, y) \in \mathcal{X} \), synchronization is required between transitions from \( x \) in \( \delta_{1,o} \) and transitions from \( y \) in \( \delta_{2,o} \). However, synchronization is not required between transitions in \( \delta_{1,\varepsilon} \) and \( \delta_{2,\varepsilon} \). Specifically, \( \Delta \) is defined as follows. For all \( (x, y) \in \mathcal{X} \),

\[
\Delta((x, y), \varepsilon) = \{(x', y) : (x, \varepsilon, x') \in \delta_{1,\varepsilon}\} \cup \{(x, y') : (y, \varepsilon, y') \in \delta_{2,\varepsilon}\}
\]

and for all \( \sigma \in \Gamma_1(x) \cap \Gamma_2(y) \),

\[
\Delta((x, y), \sigma) = \{(x', y') : (x, \sigma, x') \in \delta_{1,o} \land (y, \sigma, y') \in \delta_{2,o}\}\]

(5.2)

The product \( \otimes \) is similar to the product automaton technique of Wang et al. [51]. The product automaton technique [51] is used to compute an NFA whose state set is the set of indistinguishable state pairs (i.e., \( \Pi \)) of a given automaton \( G \) when \( G \) is deterministic with respect to its set of observable transitions (i.e., \( \delta_o \) is deterministic). However, the product \( \otimes \) is applicable when \( G \) is, more generally, nondeterministic with respect to its observable transitions (i.e., \( \delta_o \) may be nondeterministic). Furthermore, it is a binary operation on automata, not a unary operation.
5.3 A known algorithm for computing indistinguishable clusters

Here we review the CLUSTER-TABLE algorithm of Wang et al. [50] for computing the set of indistinguishable cluster pairs of a given NFA $G = (X, \Sigma, \delta, x_0)$. Informally, a cluster denotes the $\varepsilon$-closure (i.e., unobservable reach) of a state in $G$ of a specific type (to be defined precisely later). An indistinguishable cluster pair corresponds to a pair whose elements denote the $\varepsilon$-closure of two states of this specific type, and also these two states are indistinguishable. From the set of indistinguishable cluster pairs of $G$ one may compute the set of indistinguishable state pairs of $G$. We introduce the concepts necessary for understanding CLUSTER-TABLE. We demonstrate that, for a specific parameterized example, CLUSTER-TABLE is in $O(|X|^4 \cdot |\Sigma|)$. We conjecture that this is the worst-case complexity of CLUSTER-TABLE.

We recall some definitions from Wang et al. [50] in order to precisely describe CLUSTER-TABLE. The CLUSTER-TABLE algorithm makes use of the notion of a state’s cluster. Given $x \in X$, the cluster of state $x$ (if defined) is the $\varepsilon$-closure of $x$ (formalized equivalently as the unobservable reach of $x$ in Wang et al. [50]). The cluster of state $x$ is defined if $x$ is the initial state of $G$ or if $x$ has an observable transition pointing to it in $G$. Otherwise, the cluster of $x$ is undefined. Formally, for all $x \in X$, the cluster of $x$ is defined as

$$c(x) = \begin{cases} 
\delta^*_\varepsilon(x) & \text{if } x = x_0 \lor \exists x' \in X, \exists \sigma \in \Sigma, \\
(x', \sigma, x) \in \delta_o & \text{undefined otherwise.}
\end{cases}$$

When the cluster of $x$ is defined we refer to $x$ as being a cluster head.
The set of clusters of $G$ is denoted by $\Phi = \{ c(x) : x \in X \land c(x) \text{ is defined}\}$.

The set of states reached from a cluster $c(x)$ on observation of $\sigma \in \Sigma$ is denoted by $R(c(x), \sigma)$. Formally, $R : 2^X \times \Sigma \to 2^X$ and is defined as follows. For all $X' \subseteq X$, for all $\sigma \in \Sigma$, $R(X', \sigma) = \{ x \in X : \exists x' \in X', (x', \sigma, x) \in \delta \}$.

The set of indistinguishable cluster pairs of $G$ is denoted by $T$. Formally, $T \subseteq \Phi \times \Phi$ and is defined recursively as follows:

1. $(c(x_0), c(x_0)) \in T$

2. if $(\phi_i, \phi_j) \in T$ then for all $\sigma \in \Sigma$, for all $x_i \in R(\phi_i, \sigma)$, for all $x_j \in R(\phi_j, \sigma)$, $(c(x_i), c(x_j)) \in T$.

We have the following characterization of the set of indistinguishable state pairs of $G$ (i.e., $\Pi$) expressed in terms of $T$. For all $x_i, x_j \in X$, $(x_i, x_j) \in \Pi \iff \exists (\phi_i, \phi_j) \in T, x_i \in \phi_i \land x_j \in \phi_j$ (Wang et al. [50] Theorem 1).

We describe the CLUSTER-TABLE algorithm for computing $T$. For practical purposes, CLUSTER-TABLE is analagous to converting $G$ from an $\varepsilon$-NFA to an $\varepsilon$-free NFA $G'$ (whose state set is equivalent to $\Phi$, the set of clusters of $G$), which can be done using a well-known transformation studied in Automata Theory, then computing $G' \otimes G'$ where $\otimes$ denotes the product of automata. The state set of $G' \otimes G'$ is $T$.

A more precise description is offered next. CLUSTER-TABLE proceeds by identifying $\Phi$, the set of clusters of $G$. For all $\phi \in \Phi$, pair $(\phi, \phi)$ is marked indistinguishable (i.e., $(\phi, \phi)$ is added to $T$) and is marked as a pair to be processed. Then the following is conducted until all pairs in $T$ have been marked processed. A pair $(\phi_i, \phi_j) \in T$ is chosen which is marked as a pair to be processed. For each $\sigma \in \Sigma$, the set of states $R(\phi_i, \sigma)$ (resp., $R(\phi_j, \sigma)$) reached from states in $\phi_i$ (resp., $\phi_j$) on
observable transitions labeled by event $\sigma$ are computed. For each $x_i \in R(\phi_i, \sigma)$, for each $x_j \in R(\phi_j, \sigma)$, $(c(x_i), c(x_j))$ is marked indistinguishable and is marked as a pair to be processed if not already. Then $(\phi_i, \phi_j)$ is marked as processed.

Next we provide an example for which CLUSTER-TABLE is in $\Theta(|X|^4 \cdot |\Sigma|)$. Consider the automaton $G = (X, \Sigma, \delta, \{n\})$ in Figure 5.1 where $X = \{n, n-1, \ldots, 1\}$ and $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$. The $\varepsilon$-transitions in Figure 5.1 are members of the set $\delta_{\varepsilon}$. Transitions labeled by events $\sigma_1, \ldots, \sigma_k$ in Figure 5.1 are members of the set $\delta_o$.

For each $m \in X$, $c(m) = \delta_\varepsilon^*(m) = \{m, m-1, \ldots, 1\}$ and for each $m \in X$, for each $\sigma \in \Sigma$, $R(c(m), \sigma) = c(m)$, by the foregoing, by definition of $\delta_\varepsilon$, $\delta_o$, $c(\cdot)$ and $R(\cdot, \cdot)$. Pair $(c(n), c(n)) \in T$ as $n$ is the initial state of $G$ and by definition of $T$. For all $p, q \in X$, $(c(p), c(q)) \in T$ since $(c(n), c(n)) \in T$, $R(c(n), \sigma) = X$ for all $\sigma \in \Sigma$, and by definition of $T$.

For all $p, q \in X$, for each $\sigma \in \Sigma$, for each $(p', q') \in R(c(p), \sigma) \times R(c(q), \sigma)$, CLUSTER-TABLE marks $(c(p'), c(q'))$ as indistinguishable and, if necessary, marks $(c(p'), c(q'))$ as a pair to be processed since $(c(p), c(q)) \in T$. Set $R(c(p), \sigma) \times R(c(q), \sigma) = c(p) \times c(q)$. Then, for all $p, q \in X$, for each $\sigma \in \Sigma$, there are $d \cdot |c(p)| \cdot |c(q)| = d \cdot p \cdot q$ primitive operations conducted by CLUSTER-TABLE when processing pair $(c(p), c(q))$ where $d \geq 1$. Then, outside of computing cluster heads and clusters, the number of primitive operations conducted by CLUSTER-TABLE
is the following.

\[
|\Sigma| \cdot d \cdot \sum_{i=1}^{n} i \cdot \sum_{j=1}^{n} j = |\Sigma| \cdot d \cdot \sum_{i=1}^{n} i \cdot \frac{n \cdot (n + 1)}{2} \\
= |\Sigma| \cdot d \cdot \frac{n \cdot (n + 1)}{2} \cdot \sum_{i=1}^{n} i = |\Sigma| \cdot d \cdot \frac{n \cdot (n + 1)}{2} \cdot \frac{n \cdot (n + 1)}{2} \\
\in \Theta(|\Sigma| \cdot n^4) = \Theta(|\Sigma| \cdot |X|^4).
\]

The cluster heads of \(G\) can be computed by scanning \(\delta_o\). As \(|\delta_o| = |X| \cdot |\Sigma|\), this operation is in \(O(|X| \cdot |\Sigma|)\). The cluster of a given cluster head is the set of states reached by sequences of \(\varepsilon\)-transitions from the cluster head. One may compute the set of clusters by computing the transitive-closure of directed graph whose vertex set is \(X\) and edge set \(\delta_\varepsilon\). The best-known algorithm for computing transitive-closure (using matrix multiplication) is in \(O(|X|^2.3728639)\) \[6\].

So processing cluster pairs in \(T\) is the most expensive operation for this example. We conjecture that this example is the worst-case for CLUSTER-TABLE.

It must be noted that, if \((c(p), c(q)) \in T\) for \(p, q \in X\), then \((c(q), c(p)) \in T\).

For purposes of computing indistinguishable cluster pairs, it suffices for CLUSTER-TABLE to process only one of \((c(p), c(q))\) or \((c(q), c(p))\). This would nearly halve the number of primitive operations conducted by CLUSTER-TABLE, though the total number of primitive operations is still in \(\Theta(|\Sigma| \cdot |X|^4)\).

### 5.4 Computing Indistinguishable State Pairs using the Product of NFA

Recall \(\otimes\) \[5.2\], the product of \(\varepsilon\)-NFA. In this section, we demonstrate that, for a given NFA \(G\), the state set of \(G \otimes G\) is precisely the set of indistinguishable state pairs of
We propose Algorithm 2 for computing $G_1 \otimes G_2$ for two NFA $G_1, G_2$. This algorithm follows in a straightforward manner from the definition of the product. We demonstrate the worst-case complexity of Algorithm 2. To our knowledge, the algorithm for computing the product has not been formalized and its formal definition has not been studied in either the discrete-event systems literature or automata theory literature.

First, we provide a brief comparison between our approach and CLUSTER-TABLE for purposes of computing the indistinguishable state pairs of an NFA $G$. Our approach is to compute $G \otimes G$ directly. CLUSTER-TABLE is analogous to converting $G$ from an $\varepsilon$-NFA to an $\varepsilon$-free NFA $G'$ then computing $G' \otimes G'$. In this section we find that the worst-case of our approach is not worse than the conjectured worst-case for CLUSTER-TABLE ($O(|X|^4 \cdot |\Sigma|)$), which was demonstrated in Section 5.3). Thus, when $\otimes$ is used for computing indistinguishable state pairs, it is not necessary to first conduct an $\varepsilon$-NFA to $\varepsilon$-free NFA conversion.

The following is trivial but necessary for proving that the state space of $G \otimes G$ is the set of indistinguishable state pairs of $G$.

**Lemma 12.** For all $(x, y) \in X$, if $x' \in \delta_\varepsilon^*(x)$ (resp., $y' \in \delta_\varepsilon^*(y)$) then $(x', y) \in X$ (resp., $(x, y') \in X$).

**Proof.** Let $TS(x)$ denote the set of all sequences of $\varepsilon$-transitions from $x$ in $G$. Formally, $TS(x) = \{\varepsilon\} \cup \{(x_1, \varepsilon, x_2) \cdot (x_2, \varepsilon, x_3) \cdot \cdots \cdot (x_{n-1}, \varepsilon, x_n) : x_1 = x \wedge (x_i, \varepsilon, x_{i+1}) \in \delta_\varepsilon\}$. Given $t \in TS(x)$, let $FS_x(t)$ denote the final state of transition sequence $t$. Formally, $FS_x : TS(x) \rightarrow X$ where $FS_x(\varepsilon) = x$ and for all $t \cdot (u, \varepsilon, v) \in TS(x)$, $FS_x(t \cdot (u, \varepsilon, v)) = v$.

We prove that, for all $t \in TS(x)$, $(FS_x(t), y) \in X$. We prove this by induction on
the length of transition sequences in $TS(x)$.

Let $t \in TS(x)$.

Suppose $|t| = 0$. Then $t = \varepsilon$ and so $FS_x(t) = FS_x(\varepsilon) = x$. It follows that $(FS_x(t), y) \in \mathcal{X}$ since $(x, y) \in \mathcal{X}$.

Assume that $(FS_x(s), y) \in \mathcal{X}$ where $s \in TS(x)$ and $|s| = n$.

Suppose $|t| = n + 1$. Then $t = t' \cdot (u, \varepsilon, v)$ where $t' \in TS(x)$, $(u, \varepsilon, v) \in \delta_\varepsilon$ and $FS_x(t') = u$ since $t \in TS(x)$. Then $(FS_x(t'), y) \in \mathcal{X}$ by the inductive hypothesis and $|t'| = n$. Thus $(u, y) \in \mathcal{X}$. It follows that $(v, y) \in \mathcal{X}$ by definition of $\Delta$, $(u, y) \in \mathcal{X}$, $(u, \varepsilon, v) \in \delta_\varepsilon$. Then $(FS_x(t), y) \in \mathcal{X}$.

If $x' \in \delta^*_\varepsilon(x)$ then either $x' = x$ or there exists $x_1, \ldots, x_n \in X$ where $x_1 = x$, $x_n = x'$ and $(x_i, \varepsilon, x_{i+1}) \in \delta_\varepsilon$ by definition of $\delta^*_\varepsilon$. In the first case $(x', y) \in \mathcal{X}$ since $x' = x$ and $(x, y) \in \mathcal{X}$. Consider the second case. Let $t = (x_1, \varepsilon, x_2) \cdots (x_n, \varepsilon, x_n) \in TS(x)$. String $t \in TS(x)$ by definition of $TS(x)$. Pair $(x', y) \in \mathcal{X}$ since $FS_x(t) = x_n = x'$, $t \in TS(X)$ and for all $t' \in TS(x)$, $(FS_x(t'), y) \in \mathcal{X}$.

Proof. Let $G_{\text{obs}} = (Q, \Sigma, \xi, q_0)$.

$(\Rightarrow)$: We demonstrate by induction that, for all $s \in \mathcal{L}(G_{\text{obs}})$, for all $x, y \in \xi(q_0, s)$, $(x, y) \in \mathcal{X}$. 

The following posits that the set of indistinguishable state pairs of $G$ is the state set of $G \otimes G$. Each direction of the equivalence is proven by induction on the length of transition sequences from the initial state of automaton $G \otimes G$ and the initial state of automaton $G_{\text{obs}}$ defined in Chapter \ref{ch2}.

**Theorem 4.** Given NFA $G = (X, \Sigma, \delta, x_0)$. Let $G \otimes G = (X, \Sigma, \Delta, (x_0, x_0))$. Let $\Pi$ be defined as in \ref{eq:5.1}:

$$(x, y) \in \Pi \iff (x, y) \in \mathcal{X}.$$ 

**Proof.** Let $G_{\text{obs}} = (Q, \Sigma, \xi, q_0)$.

$(\Rightarrow)$: We demonstrate by induction that, for all $s \in \mathcal{L}(G_{\text{obs}})$, for all $x, y \in \xi(q_0, s)$, $(x, y) \in \mathcal{X}$.
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

Let \( s \in \mathcal{L}(G_{\text{obs}}) \).

Suppose \(|s| = 0\). String \( s = \varepsilon \). By definition of \( \xi \), \( \xi(q_0, \varepsilon) = q_0 \). By definition of \( G_{\text{obs}} \), \( q_0 = \delta^*_\varepsilon(x_0) \). Let \( x \in \delta^*_\varepsilon(x_0) \). Then \( (x, x_0) \in \mathcal{X} \) by Lemma \([12]\) and \( (x_0, x_0) \in \mathcal{X} \).

Let \( y \in \delta^*_\varepsilon(x_0) \). Then \( (x, y) \in \mathcal{X} \) by Lemma \([12]\) and \( (x, x_0) \in \mathcal{X} \). It follows that, for all \( x, y \in \xi(q_0, s) \), \( (x, y) \in \mathcal{X} \).

Assume that, for all \( t \in \mathcal{L}(G_{\text{obs}}) \), for all \( x, y \in \xi(q_0, t) \), if \(|t| = n\) then \( (x, y) \in \mathcal{X} \).

Suppose \(|s| = n + 1\). String \( s = s'\sigma \) where \( s' \in \mathcal{L}(G_{\text{obs}}) \), \( \sigma \in \Sigma \). Let \( q' = \xi(q_0, s') \). Let \( x, y \in \xi(q', \sigma) \). That \( x \in \xi(q', \sigma) \) implies \( \exists x'' \in q' \), \( \exists x' \in \delta_o(x'', \sigma) \), \( x \in \delta^*_\varepsilon(x') \) by definition of \( \xi \). That \( y \in \xi(q', \sigma) \) implies \( \exists y'' \in q' \), \( \exists y' \in \delta_o(y'', \sigma) \), \( y \in \delta^*_\varepsilon(y') \) by definition of \( \xi \). For all \( u, v \in q' \), \( (u, v) \in \mathcal{X} \) by the inductive hypothesis, \( s' \in \mathcal{L}(G_{\text{obs}}) \), \( |s'| = n \). Then \( (x'', y'') \in \mathcal{X} \). Pair \( (x'', y'') \in \mathcal{X} \) by definition of \( \Delta \), \( (x'', y'') \in \mathcal{X} \), \( x' \in \delta_o(x'', \sigma) \) and \( y' \in \delta_o(y'', \sigma) \). Pair \( (x, y) \in \mathcal{X} \) by Lemma \([12]\), \( (x', y') \in \mathcal{X} \) and \( x \in \delta^*_\varepsilon(x') \). Pair \( (x, y) \in \mathcal{X} \) by Lemma \([12]\), \( (x, y') \in \mathcal{X} \) and \( y \in \delta^*_\varepsilon(y') \).

For all \( q \in Q \), there exists \( s \in \mathcal{L}(G_{\text{obs}}) \), \( \xi(q_0, s) = q \) since \( G_{\text{obs}} \) is reachable. Then, by the previous result, for all \( q \in Q \), for all \( x, y \in q \), \( (x, y) \in \mathcal{X} \). It follows by definition of \( \Pi \) \([5.1]\) that for all \( x, y \in X \), \( (x, y) \in \Pi \Rightarrow (x, y) \in \mathcal{X} \).

\((\Leftarrow): \) Let \( TS = \{ \varepsilon \} \cup \{ (x_1, \sigma_1, x_2) \cdot (x_2, \sigma_2, x_3) \cdots (x_{n-1}, \sigma_{n-1}, x_n) : x_1, \ldots, x_n \in \mathcal{X}, x_1 = (x_0, x_0), \sigma_1, \ldots, \sigma_{n-1} \in \Sigma \cup \{ \varepsilon \}, (x_i, \sigma_i, x_{i+1}) \in \Delta \} \). Let \( FS : TS \rightarrow \mathcal{X} \) where \( FS(\varepsilon) = (x_0, x_0) \) and for all \( t \cdot (u, \sigma, v) \in TS \), \( FS(t \cdot (u, \sigma, v)) = v \).

We demonstrate by induction that, for all \( t \in TS \), \( \exists q \in Q \), \( FS(t) \in q \times q \).

Let \( t \in TS \).

Suppose \(|t| = 0 \). Then \( t = \varepsilon \). State \( FS(t) = FS(\varepsilon) = (x_0, x_0) \). State \( q_0 = \delta^*_\varepsilon(x_0) \) by definition of \( G_{\text{obs}} \). State \( x_0 \in \delta^*_\varepsilon(x_0) \) by definition of \( \delta^*_\varepsilon \). Then \( (x_0, x_0) \in q_0 \times q_0 \).

It follows that there exists a \( q \in Q \) where \( FS(t) \in q \times q \).
Assume that if $s \in TS$ and $|s| = n$ then there exists a $q \in Q$ such that $FS(s) \in q \times q$.

Suppose $|t| = n + 1$. Then $t = t' \cdot (u, \sigma, v)$ where $t' \in TS$, $(u, \sigma, v) \in \Delta$ and $FS(t') = u$ since $t \in TS$. There exists $q' \in Q$, $FS(t') \in q' \times q'$ by the inductive hypothesis, $t' \in TS$ and $|t'| = n$.

Let $u = (x, y)$, $v = (x', y')$. That $(u, \sigma, v) \in \Delta$ implies one of the following by definition of $\Delta$.

1. $\sigma = \varepsilon$, $(x, \sigma, x') \in \delta_\varepsilon$ and $y' = y$;

2. $\sigma = \varepsilon$, $x' = x$ and $(y, \sigma, y') \in \delta_\varepsilon$;

3. $\sigma \in \Sigma$, $(x, \sigma, x') \in \delta_o$ and $(y, \sigma, y') \in \delta_o$.

States $x, y \in q'$ since $u \in q' \times q'$. Consider the first case listed above. Set $q' = \cup_{z \in q'} \delta^*_\varepsilon(z)$ by definition of $\xi$ and $q_0 = \delta^*_\varepsilon(x_0)$. Then $x' \in q'$ by definition of $\delta^*_\varepsilon$ and $(x, \varepsilon, x') \in \delta_\varepsilon$. Then $(x', y') \in q' \times q'$ since $y' = y$. Thus $FS(t) \in q' \times q'$ since $FS(t) = v = (x', y')$. The second case listed above follows symmetrically to the first case.

Consider the third case listed above. Set $\xi(q', \sigma) = \{z \in X : \exists z'' \in q', \exists z' \in \delta_o(z'', \sigma), z \in \delta^*_\varepsilon(z')\}$ by definition of $\xi$. State $x' \in \delta^*_\varepsilon(x')$ by definition of $\delta^*_\varepsilon$. State $x' \in \xi(q', \sigma)$ by definition of $\xi(q', \sigma)$, $x \in q'$, $(x, \sigma, x') \in \delta_o$ and $x' \in \delta^*_\varepsilon(x')$. Symmetrically, $y' \in \xi(q', \sigma)$. Then $(x', y') \in \xi(q', \sigma) \times \xi(q', \sigma)$. Thus $FS(t) \in \xi(q', \sigma) \times \xi(q', \sigma)$ since $FS(t) = v = (x', y')$.

If $x \in X$ then either $x = (x_0, x_0)$ or there exists $x_1, \ldots, x_n \in X$, $\sigma_1, \ldots, \sigma_{n-1} \in \Sigma \cup \{\varepsilon\}$, $x_1 = (x_0, x_0)$, $x_n = x$ and $(x_i, \sigma_i, x_{i+1}) \in \Delta$ by definition of $\Delta$. In the first case $\exists q \in Q, x \in q \times q$ since $x = (x_0, x_0)$, $\varepsilon \in TS$, $FS(\varepsilon) = (x_0, x_0)$ and for all $t \in TS$, $\exists q' \in$
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

Let \( t = (x_1, \sigma_1, x_2) \cdots (x_{n-1}, \sigma_{n-1}, x_n) \). Consider the second case. Let \( t = (x_1, \sigma_1, x_2) \cdots (x_{n-1}, \sigma_{n-1}, x_n) \).

String \( t \in TS \) by definition of \( TS \), \( x_1 \cdots x_n \) and \( \sigma_1 \cdots \sigma_{n-1} \). Then \( \exists q \in Q, x \in q \times q \) since \( FS(t) = x_n = x, t \in TS \) and for all \( t' \in TS, \exists q' \in Q, FS(t') \in q' \times q' \).

Thus, for all \( x \in \mathcal{X}, \exists q \in Q, x \in q \times q \). Then, for all \( x \in \mathcal{X}, x \in \Pi \) by definition of \( \Pi \) (5.1).

We propose Algorithm 2 for computing \( G_1 \otimes G_2 \) and demonstrate its worst-case asymptotic complexity.

Next we establish the complexity of Algorithm 2.

From the definition of \( \otimes \) one easily obtains upper bounds on the number of transitions of \( G_1 \otimes G_2 \), characterized in Theorem 5.

**Theorem 5.** Given NFA \( G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}) \) where \( \delta_i = \delta_i,o \cup \delta_i,\varepsilon \) for \( i \in \{1, 2\} \),

1. the number of unobservable transitions (i.e., \( \varepsilon \)-transitions) of \( G_1 \otimes G_2 \) is bounded by \( |X_1| \cdot |\delta_{2,\varepsilon}| + |X_2| \cdot |\delta_{1,\varepsilon}| \);

2. (a) if \( \delta_{1,o} \) and \( \delta_{2,o} \) are deterministic then the number of observable transitions of \( G_1 \otimes G_2 \) is bounded by \( |X_1| \cdot |\delta_{2,o}| \) (also \( |X_2| \cdot |\delta_{1,o}| \));

(b) otherwise, the number of observable transitions of \( G_1 \otimes G_2 \) is bounded by \( |X_1|^2 \cdot |\delta_{2,o}| \) (also \( O|X_2|^2 \cdot |\delta_{1,o}| \)).

**Proof.** 1. For each \( (x, y) \in \mathcal{X}, \) there exists \( |\delta_{1,\varepsilon}(x, \varepsilon)| + |\delta_{2,\varepsilon}(y, \varepsilon)| \) \( \varepsilon \)-transitions from
Algorithm 2 Compute $G_1 \otimes G_2$

Input:

- NFA $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2})$.
- Maps $\Gamma_1 : X_1 \rightarrow 2^{\Sigma_1}$, $\Gamma_2 : X_2 \rightarrow 2^{\Sigma_2}$.

1: Let $\mathcal{X} \leftarrow \text{Pending} \leftarrow \{(x_{0,1}, x_{0,2})\}$
2: while Pending is not empty do
3: Remove an element $(x, y)$ from Pending
4: for all $x'$ such that $(x, \varepsilon, x') \in \delta_1, \varepsilon$ do
5: Add $((x, y), \varepsilon, (x', y))$ to $\Delta$
6: if $(x', y) \notin \mathcal{X}$ then
7: Add $(x', y)$ to $\mathcal{X}$, Pending
8: end if
9: end for
10: for all $y'$ such that $(y, \varepsilon, y') \in \delta_2, \varepsilon$ do
11: Add $((x, y), \varepsilon, (x, y'))$ to $\Delta$
12: if $(x, y') \notin \mathcal{X}$ then
13: Add $(x, y')$ to $\mathcal{X}$, Pending
14: end if
15: end for
16: Compute $\Gamma_1(x) \cap \Gamma_2(y)$
17: for all $\sigma \in \Gamma_1(x) \cap \Gamma_2(y)$ do
18: for all $x'$ such that $(x, \sigma, x') \in \delta_1, \sigma$ do
19: for all $y'$ such that $(y, \sigma, y') \in \delta_2, \sigma$ do
20: Add $((x, y), \sigma, (x', y'))$ to $\Delta$
21: if $(x', y') \notin \mathcal{X}$ then
22: Add $(x', y')$ to $\mathcal{X}$, Pending
23: end if
24: end for
25: end for
26: end for
27: end while
(x, y) in $G_1 \otimes G_2$ by definition of $\Delta$. Then there are the following number of $\varepsilon$-transitions in $\Delta$:

$$\sum_{(x,y) \in \mathcal{X}} |\delta_{1,\varepsilon}(x, \varepsilon)| + |\delta_{2,\varepsilon}(y, \varepsilon)|$$

$$= \sum_{(x,y) \in \mathcal{X}} |\delta_{1,\varepsilon}(x, \varepsilon)| + \sum_{(x,y) \in \mathcal{X}} |\delta_{2,\varepsilon}(y, \varepsilon)|$$

$$\leq \sum_{(x,y) \in X_1 \times X_2} |\delta_{1,\varepsilon}(x, \varepsilon)| + \sum_{(x,y) \in X_1 \times X_2} |\delta_{2,\varepsilon}(y, \varepsilon)|$$

$$= |X_2| \cdot \sum_{x \in X_1} |\delta_{1,\varepsilon}(x, \varepsilon)| + |X_1| \cdot \sum_{y \in X_2} |\delta_{2,\varepsilon}(y, \varepsilon)|$$

$$= |X_2| \cdot |\delta_{1,\varepsilon}| + |X_1| \cdot |\delta_{2,\varepsilon}|.$$

2. (a) For each $(x, y) \in \mathcal{X}$ there exists $\sum_{\sigma \in \Gamma_1(x) \cap \Gamma_2(y)} |\delta_{1,o}(x, \sigma)| \cdot |\delta_{2,o}(y, \sigma)|$ observable transitions from $(x, y)$ in $G_1 \otimes G_2$ by definition of $\Delta$. Since $\delta_{1,o}$ and $\delta_{2,o}$ are deterministic, there exists $|\Gamma_1(x) \cap \Gamma_2(y)|$ observable transitions from $(x, y)$. Given $x \in X_1$, let $\delta_{1,o}(x) = \{(x, \sigma, x') \in \delta_{1,o}\}$. There are the following number of observable transitions in $\Delta$:

$$\sum_{(x,y) \in \mathcal{X}} |\Gamma_1(x) \cap \Gamma_2(y)| \leq \sum_{(x,y) \in \mathcal{X}} |\delta_{1,o}(x)|$$

$$\leq \sum_{(x,y) \in X_1 \times X_2} |\delta_{1,o}(x)| = |X_2| \cdot \sum_{x \in X_1} |\delta_{1,o}(x)|$$

$$= |X_2| \cdot |\delta_{1,o}|.$$

2. (b) For each $(x, y) \in \mathcal{X}$ there exists $\sum_{\sigma \in \Gamma_1(x) \cap \Gamma_2(y)} |\delta_{1,o}(x, \sigma)| \cdot |\delta_{2,o}(y, \sigma)|$ observable transitions from $(x, y)$ in $G_1 \otimes G_2$ by definition of $\Delta$. There are the following number
of observable transitions in $\Delta$.

$$\sum_{(x,y) \in X_1 \times X_2} \sum_{\sigma \in \Gamma_1(x) \cap \Gamma_2(y)} |\delta_{1,o}(x,\sigma)| \cdot |\delta_{2,o}(y,\sigma)|$$

$$\leq \sum_{(x,y) \in X_1 \times X_2} \sum_{\sigma \in \Gamma_1(x) \cap \Gamma_2(y)} |\delta_{1,o}(x,\sigma)| \cdot |\delta_{2,o}(y,\sigma)|$$

$$\leq \sum_{(x,y) \in X_1 \times X_2} \sum_{\sigma \in \Gamma_1(x)} |\delta_{1,o}(x,\sigma)| \cdot |X_2|$$

$$= |X_2| \cdot \sum_{x \in X_1} \sum_{\sigma \in \Gamma_1(x)} |\delta_{1,o}(x,\sigma)| \cdot |X_2|$$

$$= |X_2|^2 \cdot \sum_{x \in X_1} \sum_{\sigma \in \Gamma_1(x)} |\delta_{1,o}(x,\sigma)|$$

$$\leq |X_2|^2 \cdot \sum_{x \in X_1} \sum_{\sigma \in \Sigma_1} |\delta_{1,o}(x,\sigma)|$$

$$= |X_2|^2 \cdot |\delta_{1,o}|.$$

The size of $G_1 \otimes G_2$ provides a lower bound for the complexity of computing $G_1 \otimes G_2$. Also, we find that the worst-case asymptotic complexity of Algorithm 2 is bounded above by the bounds on the size of $G_1 \otimes G_2$ provided in Theorem 5.

**Theorem 6.** Given NFA $G_i = (X_i, \Sigma_i, \delta_i, x_{0,i})$ where $\delta_i = \delta_{i,o} \cup \delta_{i,\varepsilon}$ for $i \in \{1,2\}$, Algorithm 2 is in $O(|\Delta| + |X_2| \cdot |\delta_{1,o}|)$ (also in $O(|\Delta| + |X_1| \cdot |\delta_{2,o}|)$) where $\Delta$ is the transition function of output $G_1 \otimes G_2$.

Expressed in terms of the input and regarding the type of $\delta_{i,o}$, Algorithm 2 is in

- $O(|X_1| \cdot |\delta_{2,\varepsilon}| + |X_2| \cdot |\delta_{1,\varepsilon}| + |X_2| \cdot |\delta_{1,o}|)$ when $\delta_{1,o}$ and $\delta_{2,o}$ are both deterministic (also in $O(|X_1| \cdot |\delta_{2,\varepsilon}| + |X_2| \cdot |\delta_{1,\varepsilon}| + |X_1| \cdot |\delta_{2,o}|)$);

- $O(|X_1| \cdot |\delta_{2,\varepsilon}| + |X_2| \cdot |\delta_{1,\varepsilon}| + |X_2|^2 \cdot |\delta_{1,o}|)$ otherwise (also in $O(|X_1| \cdot |\delta_{2,\varepsilon}| + |X_2| \cdot |\delta_{1,\varepsilon}| + |X_1|^2 \cdot |\delta_{2,o}|)$).
Proof. Suppose a total order on $\Sigma_1 \cup \Sigma_2$ is given and elements of $\Sigma_1$, $\Sigma_2$ may be effectively enumerated in this order. Then we can compute $\Sigma_1 \cap \Sigma_2$ in $O(|\Sigma_1| + |\Sigma_2|)$. One could construct a direct-address hash table over domain $\Sigma_1 \cup \Sigma_2$ to test for membership in $\Sigma_1 \cap \Sigma_2$. Then testing membership of an element of $\Sigma_1 \cup \Sigma_2$ in $\Sigma_1 \cap \Sigma_2$ would be in $O(1)$. Consequently, computing $\Gamma_i$ may be done in $O(|\delta_{i,o}|)$. With an appropriate choice of data structures for variables $\mathcal{X}$, $\text{Pending}$, $\Delta$, $\Gamma_1$, $\Gamma_2$, $\delta_{i,\varepsilon}$, $\delta_{i,o}$ (e.g., direct-address table, FIFO queue, linked list, adjacency lists relating states to events labeling outgoing transitions, adjacency matrix relating states to events labeling outgoing transitions, adjacency lists and adjacency lists, respectively), one can guarantee the following asymptotic bounds on the operations in Algorithm 2:

- testing membership of an element in $\mathcal{X}$ in $O(1)$;
- inserting an element into $\mathcal{X}$ in $O(1)$;
- inserting an element into $\text{Pending}$ in $O(1)$;
- testing emptiness of $\text{Pending}$ in $O(1)$;
- removing an arbitrary element from $\text{Pending}$ in $O(1)$;
- inserting an element into $\Delta$ in $O(1)$;
- for all $x \in X_1$, for all $y \in X_2$, computation of $\Gamma_1(x) \cap \Gamma_2(y)$ in $O(|\Gamma_1(x)|)$;
- for all $x \in X_1$, for all $y \in X_2$, enumeration of elements of $\Gamma_1(x) \cap \Gamma_2(y)$ in $O(|\Gamma_1(x) \cap \Gamma_2(y)|)$;
- for all $x_i \in X_i$, enumeration of elements of $\delta_{i,\varepsilon}(x_i, \varepsilon)$ in $O(|\delta_{i,\varepsilon}(x_i, \varepsilon)|)$;
- for all $x_i \in X_i$, $\sigma \in \Sigma_1 \cap \Sigma_2$, enumeration of elements of $\delta_{i,o}(x_i, \sigma)$ in $O(|\delta_{i,o}(x_i, \sigma)|)$. 


These bounds yield the following total costs for evaluating each line of Algorithm 2.

Line 1 is evaluated once at a cost of $O(1)$ per evaluation for a total cost of $O(1)$.

With the exception of line 1, which is evaluated only once, one can verify that for each element added to $\text{Pending}$ an element is added to $\Delta$. Lines 2, 3, 27 are evaluated once for each element added to $\text{Pending}$. Then lines 2, 3, 27 are each evaluated at most $|\Delta|$ times. Each of lines 2, 3, 27 incur a cost of $O(1)$ per evaluation. Thus the total cost of evaluating lines 2, 3, 27 is in $O(|\Delta|)$.

The collective sum of the total number of iterations of the loops on lines 4–9 and 10–15 is equal to the number of unobservable transitions of $G_1 \otimes G_2$. The cost of evaluating each of lines 4–9, 10–15 is in $O(1)$. Thus the total cost of evaluating lines 4–9, 10–15 is asymptotically upper-bounded by the number of unobservable transitions of $G_1 \otimes G_2$.

Line 16 is evaluated for each $(x, y)$ added to $\text{Pending}$. Equivalently, line 16 is evaluated for each $(x, y) \in \mathcal{X}$ by the fact that an element is added to $\text{Pending}$ if and only if the element is added to $\mathcal{X}$. For each $(x, y)$ added to $\mathcal{X}$ line 16 is evaluated at a cost of $O(|\Gamma_1(x)|)$. Then the total cost of evaluating line 16 is in $O(|X_2| \cdot |\delta_{1,o}|)$ as demonstrated in the following.

\[
\sum_{(x,y) \in \mathcal{X}} O(|\Gamma_1(x)|) = \sum_{(x,y) \in \mathcal{X}} c \cdot |\Gamma_1(x)| \quad \text{for a positive constant } c
\]

\[
\leq c \cdot \sum_{(x,y) \in \mathcal{X}} |\delta_{1,o}(x)| \leq c \cdot \sum_{(x,y) \in \mathcal{X}_1 \times X_2} |\delta_{1,o}(x)|
\]

\[
= c \cdot \sum_{x \in X_1} |X_2| \cdot |\delta_{1,o}(x)| = c \cdot |X_2| \cdot |\delta_{1,o}|.
\]

The total number of iterations of the loop on lines 19–24 is equal to the number of observable transitions of $G_1 \otimes G_2$. This number is a bound on the number of times

\footnote{One may also demonstrate that the cost is in $O(|X_1| \cdot |X_2| \cdot |\Sigma_1|)$.}
lines 17, 18, 25 and 26 are evaluated. The cost of evaluating each of lines 17–26 is in \(O(1)\). Thus the total cost of evaluating lines 17–26 is asymptotically upper bounded by the number of observable transitions of \(G_1 \otimes G_2\).

From the above analysis, it follows that Algorithm 2 is in \(O(|\Delta| + |X_2| \cdot |\delta_{1,o}|)\). Symmetrically, an alternative choice of data structures for \(\Gamma_1\) and \(\Gamma_2\) (i.e., representing \(\Gamma_1\) by an adjacency matrix relating states to events labeling outgoing transitions and representing \(\Gamma_2\) by a set of adjacency lists relating states to events labeling outgoing transitions) would produce a bound of \(O(|\Delta| + |X_1| \cdot |\delta_{2,o}|)\) The theorem statement follows directly from this fact and Theorem 5. \(\square\)

The following corollary is immediate.

**Corollary 3.** Given \(G = (X, \Sigma, \delta, x_0)\) where \(\delta = \delta_o \cup \delta_\varepsilon\), computing \(G \otimes G\) by Algorithm 2 is in

- \(O(|X| \cdot |\delta|) \subseteq O(|X|^3 + |X|^2 \cdot |\Sigma|)\) when \(\delta_o\) is deterministic;
- \(O(|X| \cdot |\delta_\varepsilon| + |X|^2 \cdot |\delta_o|) \subseteq O(|X|^4 \cdot |\Sigma|)\) otherwise.

Note that, as demonstrated in the example of Section 5.3 when \(\delta_o\) is deterministic CLUSTER-TABLE may be in \(\Theta(|X|^4 \cdot |\Sigma|)\). So, when \(\delta_o\) is deterministic, computing indistinguishable state pairs using \(G \otimes G\) is preferable in the worst-case (by Corollary 3).

We note that, when automaton \(G\) is nondeterministic with respect to its observable transitions (i.e., \(\delta_o\) is nondeterministic), an alternate procedure for computing the indistinguishable state pairs of \(G\) is to convert \(G\) to an automaton that is nondeterministic only on its \(\varepsilon\)-transitions, denoted by \(G'\), then compute \(G' \otimes G'\). The indistinguishable state pairs of \(G\) will be a subset of the state set of \(G' \otimes G'\). This
procedure is not necessarily more efficient than computing $G \otimes G$ directly, however
the asymptotic complexity of the procedure is different. We describe this procedure
next, and use the bounds provided in Corollary 3 to demonstrate its complexity.

Given NFA $G = (X, \Sigma, \delta, x_0)$, we construct a new NFA $G' = (X \cup X', \Sigma, \delta', x_0)$
which is defined such that $\delta'_o$ is deterministic. Set $X'$ and transition function $\delta'$ are
derived from $X$ and $\delta$ in the following manner. A new state $(\sigma, y) \in \delta'_o$.
Function $\delta'$ is the same as $\delta$ except for the following modifications. For each transition $(x, \sigma, y) \in \delta'_o$, the following
adjustments are made to $\delta'$:

- transition $(x, \varepsilon, (\sigma, y))$ is added to $\delta'$;
- transition $((\sigma, y), \sigma, y)$ is added to $\delta'$;
- transition $(x, \sigma, y)$ is removed from $\delta'$.

It is simple to verify that $L(G) = L(G')$ and $x \in \delta(x_0, s)$ if and only if $x \in \delta'(x_0, s)$
where $x \in X$ and $s \in L(G)$. The only difference is that $G'$ is nondeterministic only
on occurrence of $\varepsilon$-transitions (i.e., $\delta'_x$ is nondeterministic while $\delta'_o$ is deterministic).

One can easily verify that, for states $x, y \in X$, $(x, y)$ is a state in $G \otimes G$ if and only
if $(x, y)$ is a state in $G' \otimes G'$. That is, the state set of $G \otimes G$ is a subset of the state set
of $G' \otimes G'$. The remaining states of $G' \otimes G'$ are a subset of $X \times X' \cup X' \times X \cup X' \times X'$,
which can be ignored for purposes of computing the indistinguishable state pairs of $G$.

Automaton $G'$ has the following number of states and transitions:

- at most $|X| + |X| \cdot |\Sigma|$ states;
- at most $|X| \cdot |\Sigma|$ observable transitions (i.e., transitions in $\delta'_o$);
• $|\delta|$ unobservable transitions (i.e., transitions in $\delta'_e$).

By Corollary 3, computing $G' \otimes G'$ is in

$$O((|X| + |X| \cdot |\Sigma|) \cdot (|X| \cdot |\Sigma| + |\delta|))$$

$$= O((|X| + |X| \cdot |\Sigma|) \cdot (|X| \cdot |\Sigma| + |\delta_e| + |\delta_o|))$$

$$= O((|X| + |X| \cdot |\Sigma|) \cdot (|X| \cdot |\Sigma| + |X|^2 + |X|^2 \cdot |\Sigma|))$$

$$\subseteq O(|X|^3 \cdot |\Sigma|^2).$$

For purposes of computing the indistinguishable state pairs of $G$, this procedure is not necessarily better than computing $G \otimes G$ directly (in $O(|X|^4 \cdot |\Sigma|)$ for this case). Which approach should be chosen depends on $|X|$ and $|\Sigma|$.

5.5 Computing Indistinguishable State Pairs in the Presence of Unobservable Cycles

In Section 5.4 we introduced the application of the product of NFA, $\otimes$, for computing the indistinguishable state pairs of an NFA $G = (X, \Sigma, \delta, x_0)$. In this section we consider more specifically that $G$ contains cycles of unobservable transitions (i.e., cycles in transition function $\delta_e$). Under this constraint, we define an equivalence relation $\rho$ on the states of $G$. For states $x, y \in X$, $x \rho y$ if and only if $x$ is reachable from $y$ in $\delta_e$ and vice versa. From $\rho$ we define the quotient automaton $G/\rho$. We demonstrate an equivalence between the state set of automaton $G/\rho \otimes G/\rho$ and $\Pi$, the indistinguishable state pairs of $G$. We demonstrate that the complexity of computing $G/\rho$ from $G$ is in $O(|X| + |\delta|)$. We demonstrate how $\Pi$ can be computed from the state set of $G/\rho \otimes G/\rho$ in $O(|\Pi|)$. When cycles in $\delta_e$ are present, $G/\rho$ contains strictly
fewer states and transitions than $G$. In such cases, $G/\rho \otimes G/\rho$ yields a more compact representation of the indistinguishable state pairs of $G$ than $G \otimes G$.

Given NFA $G = (X, \Sigma, \delta, x_0)$, we define relation $\rho$:

$$\forall x, y \in X, x \rho y \iff x \in \delta^*_\varepsilon(y) \land y \in \delta^*_\varepsilon(x).$$

One can verify that $\rho$ is an equivalence relation on $X$. Given $x \in X$ let $[x]_\rho$ denote the equivalence class of $x$ with respect to equivalence relation $\rho$:

$$[x]_\rho = \{x' \in X : x' \rho x\}.$$

We let $X/\rho$ denote $\bigcup_{x \in X} [x]_\rho$, the set of all equivalence classes of $X$ with respect to $\rho$. Set $X/\rho$ forms a partition of $X$. We refer to the elements of $X/\rho$ as cells. Given $X$ and $\rho$, we define the canonical projection $g' : X \to X/\rho$

$$\forall x \in X, g'(x) = [x]_\rho.$$

Renaming the cells of $X/\rho$, we get a set $Y$ and a bijection $r : X/\rho \leftrightarrow Y$. The composition of $g'$ and $r$ is a function

$$g = r \circ g' : X \to Y.$$

One can verify that, for all $x, y \in X$, $x \rho y$ if and only if $g(x) = g(y)$. The inverse of $g$, denoted by $g^{-1}$, is defined

$$\forall y \in Y, g^{-1}(y) = \{x \in X : g(x) = y\}.$$ 

The quotient automaton $G/\rho$ roughly corresponds to automaton $G$ but is defined over the cells of $X/\rho$ rather than states in $X$. Quotient automaton $G/\rho = (Y, \Sigma, \eta, y_0)$ where $Y = \{g(x) : x \in X\}$, $y_0 = g(x_0)$. The transition function is $\eta : Y \times (\Sigma \cup \{\varepsilon\}) \to 2^Y$. For all $y \in Y$, for all $\sigma \in \Sigma \cup \{\varepsilon\}$,

$$\eta(y, \sigma) = \{g(x') : \exists x \in g^{-1}(y), x' \in \delta(x, \sigma)\}.$$
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

Given NFA $G_1, G_2$, we have the following relation between the states of $G_1 \otimes G_2$ and the product of the quotient automata derived from $G_1, G_2$.

**Theorem 7.** Given NFA $G_i = (X_i, \Sigma, \delta_i, x_{0,i})$ for $i \in \{1, 2\}$. Let $\rho_i$ be defined from $G_i$ in the same way that $\rho$ is defined from $G$ in \([5.3]\). Let quotient automaton $G_1/\rho_1 = (Y_1, \Sigma_i, \eta_i, y_{0,i})$. Let $g_i : X_i \to Y_1$ be defined from $X_i, Y_1$ and $\rho_i$ in the same way that $g$ is defined from $X, Y$ and $\rho$ in \([5.4]\). Let $\mathcal{X}$ denote the state set of $G_1 \otimes G_2$ and $\mathcal{Y}$ denote the state set of $G_1/\rho_1 \otimes G_2/\rho_2$.

$$(x, y) \in \mathcal{X} \iff (g_1(x), g_2(y)) \in \mathcal{Y}.$$ 

**Proof.** Let $\Delta$ denote the transition function of automaton $G_1 \otimes G_2$. Let $\Lambda$ denote the transition function of $G_1/\rho_1 \otimes G_2/\rho_2$.

$(\Rightarrow)$ Let $TS = \{\varepsilon\} \cup \{(x_1, \sigma_1, x_2) : (x_2, \sigma_2, x_3) \cdots (x_{n-1}, \sigma_{n-1}, x_n) : x_1, \ldots, x_n \in \mathcal{X}, x_1 = (x_{0,1}, x_{0,2}), \sigma_1, \ldots, \sigma_{n-1} \in (\Sigma_1 \cap \Sigma_2) \cup \{\varepsilon\}, (x_i, \sigma_i, x_{i+1}) \in \Delta\}$. Let $FS : TS \to \mathcal{X}$ where $FS(\varepsilon) = (x_{0,1}, x_{0,2})$ and for all $t \cdot (u, \sigma, v) \in TS$, $FS(t \cdot (u, \sigma, v)) = v$. We demonstrate by induction that, for all $t \in TS$, $(g_1(x), g_2(y)) \in \mathcal{Y}$ where $(x, y) = FS(t)$.

Let $t \in TS$.

Suppose $|t| = 0$. Then $t = \varepsilon$. State $FS(t) = FS(\varepsilon) = (x_{0,1}, x_{0,2})$. State $y_{0,i} = g_i(x_{0,i})$ by definition of $G_i/\rho_i$ and $x_{0,i}$ is the initial state of $G_i$. Pair $(y_{0,1}, y_{0,2})$ is the initial state of $G_1/\rho_1 \otimes G_2/\rho_2$ by definition of $\otimes$, $y_{0,1}$ is the initial state of $G_1/\rho_1$ and $y_{0,2}$ is the initial state of $G_2/\rho_2$. Thus, $(g_1(x_{0,1}), g_2(x_{0,2})) \in \mathcal{Y}$.

Assume that if $s \in TS$ and $|s| = n$ then $(g_1(x), g_2(y)) \in \mathcal{Y}$ where $(x, y) = FS(s)$.

Suppose $|t| = n + 1$. Then $t = t' \cdot (u, \sigma, v)$ where $t' \in TS$, $(u, \sigma, v) \in \Delta$ and $FS(t') = u$ since $t \in TS$. Pair $(g_1(x), g_2(y)) \in \mathcal{Y}$ where $FS(t') = (x, y)$ by the inductive hypothesis, $t' \in TS$ and $|t'| = n$.

Let $v = (x', y')$. That $(u, \sigma, v) \in \Delta$ implies one of the following by definition of $\Delta$. 


1. \( \sigma = \varepsilon, (x, \sigma, x') \in \delta_{1, \varepsilon} \) and \( y' = y \);

2. \( \sigma = \varepsilon, x' = x \) and \( (y, \sigma, y') \in \delta_{2, \varepsilon} \);

3. \( \sigma \in \Sigma, (x, \sigma, x') \in \delta_{1, o} \) and \( (y, \sigma, y') \in \delta_{2, o} \).

In all of the above cases, if \((z, \sigma, z') \in \delta_i\) then \(g_i(z') \in \eta_i(g_i(z), \sigma)\) by definition of \(\eta_i\).

Consider the first case listed above. State \(g_1(x') \in \eta_1(g_1(x), \varepsilon)\) since \((x, \varepsilon, x') \in \delta_1\). Transition \(((g_1(x), g_2(y)), \varepsilon, (g_1(x'), g_2(y))) \in \Lambda\) by definition of \(\otimes\), \((g_1(x), g_2(y)) \in \mathcal{Y}\) and \((g_1(x), \varepsilon, g_1(x')) \in \eta_1, \varepsilon\). Thus \((g_1(x'), g_2(y')) \in \mathcal{Y}\) where \((x', y') = v = FS(t)\) since \(y' = y\).

The second case listed above follows symmetrically to the first case.

Consider the third case listed above. State \(g_1(x') \in \eta_1(g_1(x), \sigma)\) since \((x, \sigma, x') \in \delta_1\). State \(g_2(y') \in \eta_2(g_2(y), \sigma)\) since \((y, \sigma, y') \in \delta_2\). Transition

\[ ((g_1(x), g_2(y)), \sigma, (g_1(x'), g_2(y'))) \in \Lambda \]

by definition of \(\otimes\), \((g_1(x), g_2(y)) \in \mathcal{Y}, (g_1(x), \sigma, g_1(x')) \in \eta_{1, o}\) and \((g_2(y), \sigma, g_2(y')) \in \eta_{2, o}\). Thus \((g_1(x'), g_2(y')) \in \mathcal{Y}\) where \((x', y') = v = FS(t)\).

As in the proof of Theorem 4 one can easily demonstrate that if \(x \in \mathcal{X}\) then there exists a \(t \in TS\) such that \(FS(t) = x\). Then, by the previous result, for all \((x, y) \in \mathcal{X}, (g_1(x), g_2(y)) \in \mathcal{Y}\).

\[ (\Leftarrow) \]

Let \(TS = \{ \varepsilon \} \cup \{(x_1, \sigma_1, x_2) \cdot (x_2, \sigma_2, x_3) \cdots (x_{n-1}, \sigma_{n-1}, x_n) : x_1, \ldots, x_n \in \mathcal{Y}, x_1 = (y_{0,1}, y_{0,2}), \sigma_1, \ldots, \sigma_{n-1} \in (\Sigma_1 \cap \Sigma_2) \cup \{ \varepsilon \}, (x_i, \sigma_i, x_{i+1}) \in \Lambda\}\). Let \(FS : TS \rightarrow \mathcal{Y}\) where \(FS(\varepsilon) = (y_{0,1}, y_{0,2})\) and for all \(t \cdot (j, \sigma, k) \in TS, FS(t \cdot (j, \sigma, k)) = k\). We demonstrate by induction that, for all \(t \in TS\), for all \(x \in g_1^{-1}(u)\), for all \(y \in g_2^{-1}(v)\), \((x, y) \in \mathcal{X}\) where \((u, v) = FS(t)\).
Let $t \in TS$.

Suppose $|t| = 0$. Then $t = \varepsilon$. State $FS(t) = FS(\varepsilon) = (y_{0,1}, y_{0,2})$. State $x_{0,i} \in g_{i}^{-1}(y_{0,i})$ since $y_{0,i} = g_i(x_{0,i})$ and by definition of $g_i$, $g_{i}^{-1}$. State $(x_{0,1}, x_{0,2}) \in \mathcal{X}$ by definition of $G_1 \otimes G_2$. Let $x_i \in g_{i}^{-1}(y_{0,i})$. State $x_i \in \delta_{i,\varepsilon}^\ast(x_{0,i})$ since $g_i(x_i) = g_i(x_{0,i})$ and by definition of $g_i$. Then $(x_1, x_{0,2}) \in \mathcal{X}$ by Lemma 12, $(x_{0,1}, x_{0,2}) \in \mathcal{X}$ and $x_1 \in \delta_{1,\varepsilon}^\ast(x_{0,1})$. State $(x_1, x_2) \in \mathcal{X}$ by Lemma 12, $(x_{0,1}, x_{0,2}) \in \mathcal{X}$ and $x_2 \in \delta_{2,\varepsilon}^\ast(x_{0,2})$.

Assume that if $s \in TS$ and $|s| = n$ then, for all $x \in g_{1}^{-1}(u)$, for all $y \in g_{2}^{-1}(v)$, $(x, y) \in \mathcal{X}$ where $(u, v) = FS(s)$.

Suppose $|t| = n + 1$. Then $t = t' \cdot (j, \sigma, k)$ where $t' \in TS$, $(j, \sigma, k) \in \Lambda$ and $FS(t') = j$ since $t \in TS$. Let $j = (p, q)$. Let $k = (p', q')$. That $(j, \sigma, k) \in \Lambda$ implies one of the following by definition of $\Lambda$.

1. $\sigma = \varepsilon$, $(p, \sigma, p') \in \eta_{1,\varepsilon}$ and $q' = q$;

2. $\sigma = \varepsilon$, $p' = p$ and $(q, \sigma, q') \in \eta_{2,\varepsilon}$;

3. $\sigma \in \Sigma_1 \cap \Sigma_2$, $(p, \sigma, p') \in \eta_{1,0}$ and $(q, \sigma, q') \in \eta_{2,0}$.

Consider the first case listed above. That $(p, \varepsilon, p') \in \eta_{1,\varepsilon}$ implies there exists $x \in g_{1}^{-1}(p)$, $x' \in g_{1}^{-1}(p')$, $x' \in \delta_{1}(x, \varepsilon)$ by definition of $\eta_1$. For all $y \in g_{2}^{-1}(q)$, $(x, y) \in \mathcal{X}$ by the inductive hypothesis, $t' \in TS$, $|t'| = n$, $FS(t') = (p, q)$ and $x \in g_{1}^{-1}(p)$. For all $x'' \in \delta_{1,\varepsilon}^\ast(x)$, for all $y \in g_{2}^{-1}(q)$, $(x'', y) \in \mathcal{X}$ by Lemma 12 and $(x, y) \in \mathcal{X}$. For all $z \in g_{1}^{-1}(p')$, $z \in \delta_{1,\varepsilon}^\ast(x')$ since $g_1(z) = g_1(x')$ and by definition of $g_1$. Set $\delta_{1,\varepsilon}^\ast(x') \subseteq \delta_{1,\varepsilon}^\ast(x)$ since $(x, \varepsilon, x') \in \delta_{1,\varepsilon}$ and by definition of $\delta_{1,\varepsilon}^\ast$. Then, for all $z \in g_{1}^{-1}(p')$, for all $y \in g_{2}^{-1}(q)$, $(z, y) \in \mathcal{X}$ by the previous three facts. Thus for all $z \in g_{1}^{-1}(p')$, for all $y \in g_{2}^{-1}(q')$, $(z, y) \in \mathcal{X}$ since $q' = q$.

The second case listed above follows symmetrically to the first case.
Consider the third case listed above. That \((p, \sigma, p') \in \eta_{1,o}\) implies there exists \(x \in g_1^{-1}(p)\), \(x' \in g_1^{-1}(p')\), \(x' \in \delta_1(x, \sigma)\) by definition of \(\eta_1\). That \((q, \sigma, q') \in \eta_{2,o}\) implies there exists \(y \in g_2^{-1}(q)\), \(y' \in g_2^{-1}(q')\), \(y' \in \delta_2(y, \sigma)\) by definition of \(\eta_2\). State \((x, y) \in \mathcal{X}\) by the inductive hypothesis, \(t' \in TS\), \(|t'| = n\), \(FS(t') = (p, q)\), \(x \in g_1^{-1}(p)\) and \(y \in g_2^{-1}(q)\). Then \((x', y') \in \mathcal{X}\) by the previous three facts and by definition of \(\Delta\).

For all \(x'' \in \delta_1^*(x')\), \((x'', y') \in \mathcal{X}\) by Lemma 12 and \((x', y') \in \mathcal{X}\). For all \(x'' \in \delta_1^*(x')\), for all \(y'' \in \delta_2^*(y')\), \((x'', y'') \in \mathcal{X}\) by Lemma 12 and the previous fact. As in the first case, one can demonstrate that \(g_1^{-1}(p') \subseteq \delta_1^*(x')\). Symmetrically, \(g_2^{-1}(q') \subseteq \delta_2^*(y')\). Thus for all \(x''' \in g_1^{-1}(p')\), for all \(y''' \in g_2^{-1}(q')\), \((x''', y''') \in \mathcal{X}\) by the previous three facts.

One can easily demonstrate that for all \((u, v) \in \mathcal{Y}\), there exists \(t \in TS\), \(FS(t) = (u, v)\). Then, by the previous fact and result proven inductively, for all \((u, v) \in \mathcal{Y}\), for all \(x \in g_1^{-1}(u)\), for all \(y \in g_1^{-1}(v)\), \((x, y) \in \mathcal{X}\).

The following is immediate from the previous result and Theorem 4. It states that two states of \(G\) are indistinguishable if and only if their corresponding states in the quotient automaton are indistinguishable.

**Corollary 4.** Given NFA \(G = (X, \Sigma, \delta, x_0)\). Let \(\Pi\) be defined as in (5.1). Let \(\rho\) be defined as in (5.3). Let \(g\) be defined as in (5.4). Let \(\mathcal{Y}\) denote the state set of \(G/\rho \otimes G/\rho\).

\[(x, y) \in \Pi \iff (g(x), g(y)) \in \mathcal{Y}\]

The quotient automaton can be computed in linear time. We provide a proof of this result.

**Theorem 8.** Given NFA \(G = (X, \Sigma, \delta, x_0)\). Let \(\rho\) be defined as in (5.3).
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

Quotient automaton $G/\rho$ is computable in $O(|X| + |\delta|)$.

Proof. The cells of $G/\rho$ and function $g$ (5.4) can be computed by applying Tarjan’s (strongly-connected components) algorithm [42]. Apply Tarjan’s algorithm on directed graph with state set $X$ and directed edge set $V$ where $V = \{(x, y) : (x, \varepsilon, y) \in \delta_x\}$. The cells of $G/\rho$ are computed. Tarjan’s algorithm operates in $O(|X| + |V|)$ [42]. From $G/\rho$ one can easily compute map $g$ (5.4) (resp., $g^{-1}$ (5.5)) in $O(|X|)$.

The transitions of $G/\rho$ may be computed by scanning $\delta$. For each $(x, \sigma, y) \in \delta$ where $\sigma \in \Sigma \cup \{\varepsilon\}$, add $(g(x), \sigma, g(y))$ to $\eta$ (if the transition is not already present in $\eta$). For each $\sigma \in \Sigma \cup \{\varepsilon\}$, one may use an adjacency matrix representation over the state set of $G/\rho$ for representing transitions labeled by $\sigma$ between states in $G/\rho$. This guarantees that computation of the transitions of $G/\rho$ is in $O(|\delta|)$.

Theorem 9. Given NFA $G = (X, \Sigma, \delta, x_0)$. Let $\Pi$ be defined as in (5.1). Let $\rho$ be defined as in (5.3). Let $Y$ denote the state set of $G/\rho \otimes G/\rho$.

Set $\Pi$ is computable from set $Y$ in $O(|\Pi|)$.

Proof. Compute $g^{-1}$ (5.5), the inverse of $g$ (5.4). This can be done in $O(|X|) \subseteq O(|\Pi|)$, for example, by enumerating states $x \in X$, computing $g(x)$ then adding $x$ to $g^{-1}(g(x))$. By Corollary 4, $\Pi = \bigcup_{(y_1, y_2) \in Y} (g^{-1}(y_1) \times g^{-1}(y_2))$.

One can easily verify that for any two $y_1, y_2 \in Y$, if $y_1 \neq y_2$ then $g^{-1}(y_1) \cap g^{-1}(y_2) = \emptyset$. Let $y_1, y_2 \in Y$ such that $y_1 \neq y_2$. Assume that there exists $x \in g^{-1}(y_1) \cap g^{-1}(y_2)$. That $x \in g^{-1}(y_1)$ implies $g(x) = y_1$ by (5.5). That $x \in g^{-1}(y_2)$ implies $g(x) = y_2$ by (5.5). Then $y_1 = y_2$, a contradiction.

It follows that for any two $(y_1, y_2), (y'_1, y'_2) \in Y$, if $(y_1, y_2) \neq (y'_1, y'_2)$ then $(g^{-1}(y_1) \times g^{-1}(y_2)) \cap (g^{-1}(y'_1) \times g^{-1}(y'_2)) = \emptyset$. By this property, computation of $\Pi$ from $Y$ is in $O(|\Pi|)$.

\qed
5.6 Example Demonstrating Computation of Indistinguishable State Pairs

We provide an example which, given NFA $G$, demonstrates construction of the quotient automaton $G/\rho$ of Section 5.5 then demonstrates construction of automaton $G/\rho \otimes G/\rho$ where $\otimes$ denotes the product of Section 5.4.

Consider the NFA $G = (X, \Sigma, \delta, x_0)$ of Figure 5.2 with state set $\{1, \ldots, 11\}$ and event set $\{\alpha, \beta\}$. Unobservable transitions are labeled with $\varepsilon$. We define equivalence relation $\rho$ over the states of $G$. For states $x, y \in X$, $x \rho y$ if and only if $x \in \delta_\varepsilon^*(y)$ and $y \in \delta_\varepsilon^*(x)$. It follows that the cells of $X/\rho$ are $\{\{1, 2, 3\}, \{4, 5\}, \{6, 7\}, \{8\}, \{9, 10, 11\}\}$, as denoted by the boxes in Figure 5.2. The cells of $X/\rho$ may be computed by applying Tarjan’s algorithm [42] on directed graph $(X, V)$ where $V = \{(x, y) : (x, \varepsilon, y) \in \delta_\varepsilon\}$. Tarjan’s algorithm provides map $g'$ from states in $X$ to their corresponding cell in $X/\rho$ as well as a distinct label for each cell of $X/\rho$ (i.e., a cell’s position in a topological order defined over the cells). The label of a cell is depicted next to the cell in Figure 5.2. We let $g$ denote a map from a state in $G$ to the label of its corresponding cell (e.g., $g(1) = g(2) = g(3) = A, g(4) = g(5) = B$).

Using map $g$ and the transitions in $\delta$, the quotient automaton $G/\rho$ in Figure 5.3 is computed. If $(x, \sigma, y)$ is a transition in $G$ then $(g(x), \sigma, g(y))$ is a transition in $G/\rho$. For example, since $(3, \varepsilon, 4)$ is a transition in $\delta$, $(A, \varepsilon, B)$ is a transition in $G/\rho$. We omit self-loops on $\varepsilon$ in Figure 5.3.
From $G/\rho$ we compute $G/\rho \otimes G/\rho$ depicted in Figure 5.4. Computation of $G/\rho \otimes G/\rho$ is straightforward from the definition of $\otimes$. For brevity, we depict only one of $(y_1, y_2)$ or $(y_2, y_1)$ in Figure 5.4 for $y_1, y_2$ in the state set of $G/\rho$. For purposes of computing $\Pi$, it is not necessary to compute both $(y_1, y_2)$ and $(y_2, y_1)$. One can easily adapt Algorithm 2 to do this. Automaton $G/\rho \otimes G/\rho$ computed using this slight
modification has 14 states, which contrasts with the 57 states required for $G \otimes G$.

To decide if a pair $(x, y) \in X \times X$ is in $\Pi$, one need only decide if $(g(x), g(y))$ or $(g(y), g(x))$ is a state in $G/\rho \otimes G/\rho$.

5.7 Applications of Computing Indistinguishable State Pairs

In this section we demonstrate some applications where computing indistinguishable state pairs of NFA using the product $\otimes$ of Section 5.4 may be used. We demonstrate how observability \[17\], coobservability \[24\] and eventual feasibility, studied in a problem of sensor activation \[29\] in Section 4.4, may be verified by verifying which pairs of states are indistinguishable between two automata. We demonstrate how the extended specification, used in a problem of sensor activation \[53\] may be computed using an automaton computed using $\otimes$.

5.7.1 Testing Observability

A simple approach for verifying observability using indistinguishable state pairs is provided in Wang et al. \[50\]. We recall this approach in the following, and present an alternative approach for using indistinguishable state pairs to verify observability.

A plant language $\mathcal{L}(G)$, modeled using DFA $G$, and specification language $\mathcal{L}(K)$, modeled using DFA $K$ are given. Further, $\mathcal{L}(K) \subseteq \mathcal{L}(G)$, $\mathcal{L}(K)$ is prefix-closed, $\mathcal{L}(G)$ is prefix-closed and $K$ is presumed to be given as a sub-automaton of $G$ (i.e., the states and transitions of $K$ are contained in $G$). If $K$ is not modeled as a sub-automaton of $G$, then a transformation can be applied to $K$ and $G$ such that the
resulting generator of $\mathcal{L}(K)$ is a sub-automaton of the resulting generator of $\mathcal{L}(G)$ (see, e.g., Lafortune & Chen [14]). Then $G = (X, \Sigma, \delta_G, x_0)$ and $K = (Y, \Sigma, \delta_K, x_0)$ where $Y \subseteq X$ and $\delta_K \subseteq \delta_G$.

Suppose $\Sigma_c$ denotes the controllable alphabet where $\Sigma_c \subseteq \Sigma$. Further, suppose that the transitions of $K$ are partitioned into observable transitions and unobservable transitions. That is, if a transition $(x, \sigma, y) \in \delta_K$ is observable (resp., unobservable), then $\sigma$ is observed (resp., nothing is observed) when transition $(x, \sigma, y)$ is triggered in $K$. We denote the set of observable transitions by $OT$ where $OT \subseteq \delta_K$.

We can erase the labels of the unobservable transitions in $K$ to denote that nothing is observed on their occurrence. That is, from $K$ we can construct $K' = (Y, \Sigma, \delta_{K'}, x_0)$ such that if a transition $(x, \sigma, y) \in \delta_K$ is observable (resp., unobservable) then $(x, \sigma, y) \in \delta_{K'}$ (resp., $(x, \varepsilon, y) \in \delta_{K'}$).

By Theorem 4, Wang et al. [50] Section 5 equation (23) and Wang et al. [50] Theorem 1, language $\mathcal{L}(K)$ is observable with respect to language $\mathcal{L}(G)$, controllable events $\Sigma_c$ and observable transitions $OT$ if, for all $\sigma \in \Sigma_c$, for all $x, x', y \in Y$, for all $y' \in X$

$$(x, \sigma, x') \in \delta_K \land (y, \sigma, y') \in \delta_G \land (y, \sigma, y') \notin \delta_K$$

$\Rightarrow (x, y) \text{ is not a state in } K' \otimes K'$.

An alternative approach for verifying observability is proposed next. We are given $\mathcal{L}(G), G, \mathcal{L}(K)$ and $K$ as before. Also, we are given DFA $K_{\text{min}} = (Z, \Sigma, \delta_{K_{\text{min}}}, z_0)$ for generating $\mathcal{L}(K)$, such that $K_{\text{min}}$ has a minimum number of states. Automaton $K_{\text{min}}$ could be obtained by minimizing $K$ using the well-known algorithm of Hopcroft [9].

Initially, observations are not modeled using the transitions of an automaton. Instead, a type of finite-state transducer is used to model the observation of event
occurrences, which we define next.

An erasing (finite-state) transducer, $T$, is defined by a tuple $T = (Q, \Psi, \Lambda, \xi, q_0)$ where $Q$ is the set of states, $\Psi$ the input alphabet, $\Lambda$ the output alphabet where $\Lambda = \Psi \cup \{\varepsilon\}$, $q_0$ the initial state, and $\xi$ the transition function where $\xi : Q \times \Psi \to \Lambda \times Q$.

Transition function $\xi$ is defined such that either the same event that is input to a transition is also output by the transition, or nothing is output by the transition. That is, if $(q, \psi, \lambda, q') \in \xi$, then either $\lambda = \psi$ or $\lambda = \varepsilon$. Further, $\xi$ is a total function, i.e., for all $q \in Q$, for all $\psi \in \Psi$, there exist $\lambda \in \Lambda$, $q' \in Q$ such that $(q, \psi, \lambda, q') \in \xi$.

We define operations involving automata and erasing transducers. First, given an erasing transducer $T = (Q, \Sigma, \Sigma \cup \{\varepsilon\}, \xi, q_0)$, we let $\lceil T \rceil$ denote the automaton obtained by replacing the input symbol to transitions in $\xi$ by their output symbol. That is, $(q, \lambda, q')$ is a transition in $\lceil T \rceil$ only if $(q, \sigma, \lambda, q')$ is a transition in $T$.

Second, given an automaton $A = (S, \Sigma, f, s_0)$ and erasing transducer $T = (Q, \Sigma, \Sigma \cup \{\varepsilon\}, \xi, q_0)$ which share the same input alphabet $\Sigma$, $A \times T$ denotes an erasing transducer $(S \times Q, \Sigma, \Sigma \cup \{\varepsilon\}, f \times \xi, (s_0, q_0))$, which is defined in the same manner as the product of automata, but where $((s, q), \sigma, \lambda, (s', q'))$ is a transition in $f \times \xi$ only if $(s, q) \in S \times Q$, $(s, \sigma, s') \in f$ and $(q, \sigma, \lambda, q') \in \xi$. Automaton $\lceil A \times T \rceil$ simulates the operation of $A$ but where the observation of event occurrences (or lack thereof) is determined by $T$.

Without loss of generality, we can assume that the erasing transducer used for determining which event occurrences are observed has a minimum number of states, since it is possible to minimize any finite-state transducer \cite{19}. Let $T_{\min}$ denote this erasing transducer with state set $Q$. From $K, K_{\min}$ and $T_{\min}$, we construct an automaton that may be used for verifying observability. We compare this approach
with the approach of Wang et al. [50] for verifying observability.

Observability can be verified using the state set of $[K_{\text{min}} \times T_{\text{min}}] \otimes [K \times T_{\text{min}}]$ as follows. Language $\mathcal{L}(K)$ is observable with respect to language $\mathcal{L}(G)$, controllable events $\Sigma_c$ and erasing transducer $T_{\text{min}}$ if, for all $\sigma \in \Sigma_c$, for all $z, z' \in Z$, for all $y \in Y$, for all $y' \in X$

$$(z, \sigma, z') \in \delta_{K_{\text{min}}} \land (y, \sigma, y') \in \delta_G \land (y, \sigma, y') \notin \delta_K$$

$$\Rightarrow \forall q_1, q_2 \in Q, ((z, q_1), (y, q_2))$$ is not a state in $[K_{\text{min}} \times T_{\text{min}}] \otimes [K \times T_{\text{min}}]$.

Automaton $[K_{\text{min}} \times T_{\text{min}}] \otimes [K \times T_{\text{min}}]$ is no larger (and may be smaller) than $K' \otimes K'$, as we demonstrate next. Suppose the transitions of $K$ are partitioned into observable transitions and unobservable transitions. We can extract an erasing transducer $T$ from $K$ such that $T$ yields the same observations of event occurrences as would be produced using the observable and unobservable transitions of $K$. Let $T = (Y, \Sigma, \Sigma \cup \{\varepsilon\}, \xi, x_0)$. Transition function $\xi$ is defined as follows. First, if $(x, \sigma, y) \in \delta_K$ is an observable transition (resp., an unobservable transition) then transition $(x, \sigma, y)$ (resp., $(x, \sigma, \varepsilon, y)$) is added to $\xi$. Second, $\xi$ is made into a total function. For all $x \in X, \sigma \in \Sigma$, if there does not exist a $y \in X$ such that $(x, \sigma, y) \in \delta_G$, then $(x, \sigma, \sigma, x)$ is added to $\xi$.

One can verify that $[K \times T]$ produces an automaton equivalent to $K'$. Note that transducer $T$ may not be minimal. From $T$, a transducer $T_{\text{min}}$, which outputs the same string as $T$ on the same input string, but has a minimum number of states, can be derived [19]. Let $K_{\text{min}}$ denote a minimum-state generator for $\mathcal{L}(K)$. Automaton $[K_{\text{min}} \times T_{\text{min}}]$ has no more states than in $[K \times T]$. Also, automaton $[K \times T_{\text{min}}]$ has no more states than in $[K \times T]$. Then, since $K' = [K \times T]$, $[K_{\text{min}} \times T_{\text{min}}] \otimes [K \times T_{\text{min}}]$ has no more states than in $K' \otimes K'$. 
5.7.2 Testing Coobservability

Here we demonstrate how coobservability may be verified by computing indistinguishable state pairs between automata. We demonstrate the construction for two agents, though it may be generalized for any number of agents. Note that we consider only prefix-closed languages here, and not marked languages as considered in the initial characterization of coobservability [24].

The setting is similar to Subsection 5.7.1. A plant language \( L(\mathcal{G}) \), modeled using DFA \( \mathcal{G} \), and specification language \( L(\mathcal{K}) \), modeled using DFA \( \mathcal{K} \) are given. Further, \( L(\mathcal{K}) \subseteq L(\mathcal{G}) \), \( L(\mathcal{K}) \) is prefix-closed, \( L(\mathcal{G}) \) is prefix-closed and \( \mathcal{K} \) is presumed to be given as a sub-automaton of \( \mathcal{G} \). Also, \( K_{\text{min}} \) is given as a minimal-state generator for \( L(\mathcal{K}) \).

Suppose controllable alphabets \( \Sigma_{1,c}, \Sigma_{2,c} \) are given where \( \Sigma_{1,c} \subseteq \Sigma \) and \( \Sigma_{2,c} \subseteq \Sigma \). Erasing transducers \( T_1 \) and \( T_2 \) are given for modeling the observations of agent 1 and 2, respectively. See subsection 5.7.1 for the definition of erasing transducer. Without loss of generality, we may assume that \( T_i \) has a minimum number of states.

We define a second product on automata and transducers for purposes of verifying coobservability. This product is analogous to \( \otimes \), but transitions in the resulting automaton are labeled by the input to the transducer when a corresponding transition is made in the transducer. Formally, given an NFA \( \mathcal{A} = (S, \Sigma, f, s_0) \) and erasing transducer \( T = (Q, \Sigma, \Sigma \cup \{\varepsilon\}, \xi, q_0) \), \( \mathcal{A} \otimes T \) denotes an automaton \( (\mathcal{A}, \Sigma, \Delta, (s_0, q_0)) \) where \( \mathcal{A} \subseteq S \times Q \) and \( \Delta : \mathcal{A} \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^\mathcal{A} \). If \( (s, q) \in \mathcal{A} \) then

- if \( (s, \varepsilon, s') \in f \) then \( ((s, q), \varepsilon, (s', q)) \in \Delta \);
- if \( (q, \sigma, \varepsilon, q') \in \xi \) then \( ((s, q), \sigma, (s, q')) \in \Delta \);
if \((s,\sigma,s')\) \(\in f\) and \((q,\sigma,q')\) \(\in \xi\) then \(((s,q),\sigma,(s',q'))\) \(\in \Delta\).

One can verify that \(A \otimes [T]\) has the same state set as \(A \odot T\). Automaton \(A \otimes T\) may be computed by applying a few modifications to Algorithm 2. On line 10, the transition considered would be of the form \((y,\sigma,\varepsilon,y')\) instead of \((y,\varepsilon,y')\) to account for both input and output events labeling transitions in the transducer. On line 11, transition \(((x,y),\sigma,(x,y'))\) would be added to \(\Delta\) instead of \(((x,y),\varepsilon,(x,y'))\). On line 19, the transition considered would be of the form \((y,\sigma,\sigma,y')\).

Automaton \([K_{min} \times T_i] \otimes [K \times T_i]\) may be used to verify observability of \(L(K)\) with respect to \(L(G)\), \(\Sigma_{i,c}\), and \(T_i\) as detailed in Subsection 5.7.1. Automaton \([K_{min} \times T_i] \odot (K \times T_i)\) has the same state set as \([K_{min} \times T_i] \otimes [K \times T_i]\), so it too may be used for verifying observability. Automaton \([K_{min} \times T_i] \odot (K \times T_i)\) is different in that all transitions made in the underlying automaton \(K\) are observed in \([K_{min} \times T_i] \odot (K \times T_i)\).

Let \(OV_i = [K_{min} \times T_i] \odot (K \times T_i)\). Automaton \(OV_1 \otimes OV_2\) may be used to verify coobservability.

First, we provide a simplifying characterization of the state space of \(OV_1 \otimes OV_2\). Let \(Y\) (resp., \(Z, Q_i\)) denote the state set of \(K\) (resp., \(K_{min}, T_i\)). States in \(OV_1 \otimes OV_2\) are in the set \((Z \times Q_1) \times (Y \times Q_1) \times (Z \times Q_2) \times (Y \times Q_2)\). By definition of \(OV_i\), any transition made in \(OV_i\) due, in part, to a transition in \(K\) is labeled by the event labeling the corresponding transition in \(K\). Then, by definition of \(\otimes\), the \(Y\) components of states in \(OV_1 \otimes OV_2\) are the same. That is, states in \(OV_1 \otimes OV_2\) are specifically of the form \(((z_1,q_1),(y,q'_1),(z_2,q_2),(y,q'_2))\) where \(y \in Y\), \(z_1,z_2 \in Z\), \(q_1,q'_1 \in Q_1\) and \(q_2,q'_2 \in Q_2\).

Language \(L(K)\) is coobservable with respect to \(L(G)\), controllable events \(\Sigma_{1,c}\), \(\Sigma_{2,c}\) and erasing transducers \(T_1,T_2\) if
(i) $\mathcal{L}(K)$ is observable with respect to $\mathcal{L}(G)$, $\Sigma_{1,c} \setminus \Sigma_{2,c}$ and $T_1$ (verified, e.g., using $OV_1$ as detailed in Subsection 5.7.1);

(ii) $\mathcal{L}(K)$ is observable with respect to $\mathcal{L}(G)$, $\Sigma_{2,c} \setminus \Sigma_{1,c}$ and $T_2$;

(iii) for all $\sigma \in \Sigma_{1,c} \cap \Sigma_{2,c}$, for all $z_1, z_1', z_2, z_2' \in Z$, for all $y \in Y$, for all $y' \in X$

$$(z_1, \sigma, z_1') \in \delta_{K_{\min}} \land (z_2, \sigma, z_2') \in \delta_{K_{\min}} \land (y, \sigma, y') \in \delta_G \land (y, \sigma, y') \notin \delta_K$$

$$\Rightarrow \forall q_1, q_1' \in Q_1, \forall q_2, q_2' \in Q_2, ((z_1, q_1), (y, q_1'), (z_2, q_2), (y, q_2'))$$

is not a state in $OV_1 \otimes OV_2$.

We note a potential optimization that may be used for verifying (iii) in the above when the state set of $OV_1 \otimes OV_2$ is large. Suppose (i) and (ii) in the above hold but we have not yet constructed $OV_1 \otimes OV_2$ for testing (iii). Then the following approach may be used for testing (iii) using an automaton whose state set is no larger than the state set of $OV_1 \otimes OV_2$. Automaton $OV_i$ may be used to test observability of $\mathcal{L}(K)$ is respect to $\mathcal{L}(G)$, $\Sigma_{1,c} \cap \Sigma_{2,c}$ and $T_i$. If this test fails, then we may mark the states of $OV_i$ that resulted in the failure and trim the automaton. We denote the result by automaton $OV_i'$. Then (iii) may be verified using the state set of $OV_1' \otimes OV_2'$, which is no larger than than the state set of $OV_1 \otimes OV_2$. Also, for purposes of verifying (iii), consideration will only need to be given to states in $OV_1' \otimes OV_2'$ that contain a marked state from $OV_1'$ and a marked state from $OV_2'$.

### 5.7.3 Testing Eventual Feasibility

In this section we describe and formalize a procedure for verifying the eventual feasibility condition introduced in earlier work [29] and in Section 4.4. Informally, this feasibility condition is satisfied if and only if any two indistinguishable states of an
automaton $H$ are eventually followed by the same observations of events. This condition finds its application in the construction of an automaton that maps from what an agent sees to the set of possible events that the agent could observe subsequently (corresponding to an agent’s sensor activation decisions). Such a map is also referred to as an observer of $H$\textsuperscript{[55].}

We recall the setting and the eventual feasibility condition\textsuperscript{[29].} We are given DFA $H = (X, \Sigma, \xi, x_0)$ and a set of observable events $\Sigma_o$ where $\Sigma_o \subseteq \Sigma$. We are also given $\Omega$, a set of observable transitions in $\xi$. Specifically, $\Omega \subseteq \{(x, \sigma) \in \xi : \sigma \in \Sigma_o\}$. If a transition $(x, \sigma) \in \Omega$ is triggered in $\xi$, then $\sigma$ is observed. No event observations are made when any transitions in $\xi$ that are not also in $\Omega$ are triggered.

To formalize eventual feasibility, we can define a map from the event sequence that an agent sees according to $\Omega$ to the set of events that the agent may observe immediately following the observed event sequence. This map is denoted by $\omega$ and is formally defined from $\Omega$ as follows:

$$\omega(s) = \{e \in \Sigma_o : (\xi(x_0, s), e) \in \Omega\}. \quad (5.6)$$

From $\omega$ we may compute a map, $\theta^\omega : \mathcal{L}(H) \to \Sigma_o^*$, from strings in $\mathcal{L}(H)$ to how they appear to the agent. Formally, for the empty string $\epsilon$, $\theta^\omega(\epsilon) = \epsilon$, and for all $s, se \in \mathcal{L}(H)$ with $e \in \Sigma$,

$$\theta^\omega(se) = \begin{cases} 
\theta^\omega(s)e & \text{if } e \in \omega(s) \\
\theta^\omega(s) & \text{otherwise.}
\end{cases} \quad (5.7)$$

Eventual feasibility is formalized below, and expresses that if two strings in $\mathcal{L}(H)$ look the same, according to $\theta^\omega$, then they should eventually be followed by the same
event observations, according to $\Omega$.

\[(\forall e \in \Sigma_o)(\forall s, s' \in L(H)) \theta^\omega(s) = \theta^\omega(s') \Rightarrow (5.8)\]

\[\left(\exists u \in \Sigma^*, sue \in L(H) \wedge \theta^\omega(sue) = \theta^\omega(s) \wedge (\xi(x_0, su), e) \in \Omega\right)\]

\[\Leftrightarrow (\exists u' \in \Sigma^*, s'ue \in L(H) \wedge \theta^\omega(s'ue) = \theta^\omega(s') \wedge (\xi(x_0, s'u'e), e) \in \Omega)\]

For purposes of verifying (5.8), we dedicate this subsection to reducing (5.8) to a condition defined over the indistinguishable state pairs and transitions of an automaton $G$.

From $H$ we construct NFA $G$ by erasing the labels of transitions not in $\Omega$. Specifically, $G = (X, \Sigma_o, \delta, x_0)$ such that if $(x, \sigma, y) \in \Omega$ (resp., $(x, \sigma, y) \in \xi \setminus \Omega$) then $(x, \sigma, y) \in \delta$ (resp., $(x, \varepsilon, y) \in \delta$).

The following Lemmata are necessary for reducing (5.8) to a condition defined over the indistinguishable state pairs of $G$. The following states that, for any state $x$ reached by a string $s$ in $H$, $x$ is reached in $G$ by the string denoting observation of $s$ using sensor activation map $\omega$ (i.e., $\theta^\omega(s)$).

**Lemma 13.** For all $s \in L(H)$, $\xi(x_0, s) \in \delta(x_0, \theta^\omega(s))$.

**Proof.** Proven by induction on the length of $\theta^\omega(s)$. Let $s = \sigma_1 \ldots \sigma_m$.

Suppose $|\theta^\omega(s)| = 0$. Then $\theta^\omega(s) = \varepsilon$ by (5.7). For all $i \in \{1, \ldots, m - 1\}$, event $\sigma_{i+1} \notin \omega(\sigma_1 \ldots \sigma_i)$ by $|\theta^\omega(s)| = 0$ and (5.7). Then $(\xi(x_0, \sigma_1 \ldots \sigma_i), \sigma_{i+1}) \notin \Omega$ by (5.6). Then $\xi(x_0, \sigma_1 \ldots \sigma_i) \in \delta(x_0, \varepsilon)$ by definition of $\delta$. It follows that $\xi(x_0, s) \in \delta(x_0, \theta^\omega(s))$.

Assume that, for all $t \in L(H)$, if $|\theta^\omega(t)| \leq n$ then $\xi(x_0, t) \in \delta(x_0, \theta^\omega(t))$.

Suppose $|\theta^\omega(s)| = n+1$. Let $s'$ denote the longest prefix of $s$ such that $|\theta^\omega(s')| = n$. String $s' = \sigma_1 \ldots \sigma_k$ where $k < m$. By definition of $s'$ and (5.7), $\theta^\omega(\sigma_1 \ldots \sigma_k \sigma_{k+1}) =$
\(\theta^\omega(\sigma_1 \ldots \sigma_k)\sigma_{k+1}\). Then \((\xi(x_0, s'), \sigma_{k+1}) \in \Omega\) by (5.6). Then \(\xi(x_0, s'\sigma_{k+1}) \in \delta(\xi(x_0, s'), \sigma_{k+1})\) by definition of \(\delta\).

By the inductive hypothesis and \(|\theta^\omega(s')| = n\), \(\xi(x_0, s') \in \delta(x_0, \theta^\omega(s'))\). By \(\xi(x_0, s'\sigma_{k+1}) \in \delta(\xi(x_0, s'), \sigma_{k+1})\) and \(\xi(x_0, s') \in \delta(x_0, \theta^\omega(s'))\), we have \(\xi(x_0, s'\sigma_{k+1}) \in \delta(\delta(x_0, \theta^\omega(s')), \sigma_{k+1})\). Equivalently, \(\xi(x_0, s'\sigma_{k+1}) \in \delta(\delta(x_0, \theta^\omega(s'))\sigma_{k+1})\). Then \(\xi(x_0, s'\sigma_{k+1}) \in \delta(x_0, \theta^\omega(s')\sigma_{k+1})\) since \(\theta^\omega(s'\sigma_{k+1}) = \theta^\omega(s')\sigma_{k+1}\).

String \(\theta^\omega(s'\sigma_{k+1} \ldots \sigma_{k+i}) = \theta^\omega(s'\sigma_{k+1})\) for \(2 \leq i \leq m - (k + 1)\) since \(\theta^\omega(s'\sigma_{k+1}) = \theta^\omega(s'\sigma_{k+1} \ldots \sigma_{m})\). Then event \(\sigma_{k+i} \notin \omega(s'\sigma_{k+1} \ldots \sigma_{k+i-1})\) by (5.7). Then \((\xi(x_0, s'\sigma_{k+1} \ldots \sigma_{k+i-1}), s_{k+i}) \notin \Omega\) by (5.6). Then \(\xi(x_0, s'\sigma_{k+1} \ldots \sigma_{k+i}) \in \delta(\xi(x_0, s'\sigma_{k+1}), \varepsilon)\) by definition of \(\delta\). Thus \(\xi(x_0, s) \in \delta(\xi(x_0, s'\sigma_{k+1}), \varepsilon)\). It follows that \(\xi(x_0, s) \in \delta(x_0, \theta^\omega(s'\sigma_{k+1}))\) since \(\xi(x_0, s'\sigma_{k+1}) \in \delta(x_0, \theta^\omega(s'\sigma_{k+1}))\). Finally, \(\xi(x_0, s) \in \delta(x_0, \theta^\omega(s))\) since \(\theta^\omega(s'\sigma_{k+1}) = \theta^\omega(s)\).

**Lemma 14.** For all \(x \in X\), if there exists \(t \in \mathcal{L}(G)\) such that \(x \in \delta(x_0, t)\) then there exists \(s \in \mathcal{L}(H)\) such that \(\xi(x_0, s) = x\) and \(\theta^\omega(s) = t\).

**Proof.** Let \((x_0, \alpha_1, x_1) \ldots (x_{n-1}, \alpha_n, x_n)\) denote a transition sequence in \(\delta\) such that \(\alpha_1 \ldots \alpha_n = t\) and \(x_n = x\). Such a transition sequence exists since \(x \in \delta(x_0, t)\). Let \(|t| = k\). There exist \(l_1, \ldots, l_k \in \{1, \ldots, n\}\) where \(l_i < l_{i+1}\) and \(\alpha_{l_1} \ldots \alpha_{l_k} = t\) since \(\alpha_1 \ldots \alpha_n = t\). Then, for all \(j \in \{1, \ldots, n\} \setminus \{l_1, \ldots, l_k\}\), \(\alpha_{l_j} = \varepsilon\). It follows that, by definition of \(\delta\), for all \(i \in \{l_1, \ldots, l_k\}\), \((x_{i-1}, \beta_i, x_i) \in \xi\) and \((x_{i-1}, \beta_i) \in \Omega\) where \(\beta_i = \alpha_i\). Also, by definition of \(\delta\), for all \(j \in \{1, \ldots, n\} \setminus \{l_1, \ldots, l_k\}\), there exists \(\beta_j \in \Sigma\) such that \((x_{j-1}, \beta_j, x_j) \in \xi\) and \((x_{j-1}, \beta_j) \notin \Omega\). Then \(\theta^\omega(\beta_1 \ldots \beta_n) = t\) by (5.6) and (5.7). Also, since \((x_0, \beta_1, x_1) \ldots (x_{n-1}, \beta_n, x_n)\) is a transition sequence in \(\xi\) and \(x_n = x\), we have that \(\xi(x_0, \beta_1 \ldots \beta_n) = x\).
Lemma 15. Let $\Pi$ denote the set of indistinguishable state pairs of $G$ (as defined by (5.1)).

1. For all $s, s' \in \mathcal{L}(H)$, if $\theta^\omega(s) = \theta^\omega(s')$ then $(\xi(x_0, s), \xi(x_0, s')) \in \Pi$.

2. For all $x, y \in X$, if $(x, y) \in \Pi$ then there exist $s, s' \in \mathcal{L}(H)$ such that $\xi(x_0, s) = x$, $\xi(x_0, s') = y$ and $\theta^\omega(s) = \theta^\omega(s')$.

Proof.

1. For strings $s, s' \in \mathcal{L}(H)$, if $\theta^\omega(s) = \theta^\omega(s')$ then $\{\xi(x_0, s), \xi(x_0, s')\} \subseteq \delta(x_0, \theta^\omega(s))$ by Lemma 13. For all $x, y \in X$, there exists $t \in \mathcal{L}(G)$ such that $\{x, y\} \subseteq \delta(x_0, t)$ if and only if $(x, y) \in \Pi$ by definition of the subset construction and $\Pi$ (5.1). Thus, if $\theta^\omega(s) = \theta^\omega(s')$ then $(\xi(x_0, s), \xi(x_0, s')) \in \Pi$.

2. For all $x, y \in X$, if there exists $t \in \mathcal{L}(G)$ such that $\{x, y\} \subseteq \delta(x_0, t)$ then there exist $s, s' \in \mathcal{L}(H)$ such that $\xi(x_0, s) = x$, $\xi(x_0, s') = y$ and $\theta^\omega(s) = \theta^\omega(s')$ by Lemma 14. For all $x, y \in X$, there exists $t \in \mathcal{L}(G)$ such that $\{x, y\} \subseteq \delta(x_0, t)$ if and only if $(x, y) \in \Pi$ by definition of the subset construction and $\Pi$ (5.1). Thus, for $x, y \in X$, if $(x, y) \in \Pi$ then there exist $s, s' \in \mathcal{L}(H)$ such that $\xi(x_0, s) = x$, $\xi(x_0, s') = y$ and $\theta^\omega(s) = \theta^\omega(s')$.

Eventual feasibility (5.8) is equivalent to the following by Lemma 15, (5.7) and
(5.6):

\[
(\forall e \in \Sigma_o)(\forall x, x' \in X) (x, x') \in \Pi \Rightarrow \\
[(\exists \sigma_1, \ldots, \sigma_n \in \Sigma \text{ such that } (x, \sigma_1) \notin \Omega \land \ldots \land (\xi(x, \sigma_1 \ldots \sigma_{n-1}), \sigma_n) \notin \Omega \\
\land (\xi(x, \sigma_1 \ldots \sigma_n), e) \in \Omega) \tag{5.9}]
\]

\[
\Leftrightarrow \\
(\exists \sigma'_1, \ldots, \sigma'_m \in \Sigma \text{ such that } (x', \sigma'_1) \notin \Omega \land \ldots \land (\xi(x', \sigma'_1 \ldots \sigma'_{m-1}), \sigma'_m) \notin \Omega \\
\land (\xi(x', \sigma'_1 \ldots \sigma'_m), e) \in \Omega)].
\]

We reduce (5.9) to a condition defined over the transitions of \(G\) in the following. For all \(x \in X\), for all \(\sigma_1, \ldots, \sigma_n, e \in \Sigma\), \(\xi(x, \sigma_1 \ldots \sigma_n e)\) is defined, \((x, \sigma_1) \notin \Omega \ldots (\xi(x, \sigma_1 \ldots \sigma_{n-1}), \sigma_n) \notin \Omega \) and \((\xi(x, \sigma_1 \ldots \sigma_n), e) \in \Omega\) if and only if \(\delta(x, e) \neq \emptyset\) by definition of \(\delta\). Then condition (5.9) is equivalent to the following:

\[
(\forall e \in \Sigma_o)(\forall x, x' \in X) (x, x') \in \Pi \Rightarrow [\delta(x, e) \neq \emptyset \Leftrightarrow \delta(x', e) \neq \emptyset]. \tag{5.10}
\]

Let \(\mathcal{X}\) denote the state set of \(G \otimes G\). By Theorem 4, (5.10) is equivalent to the following:

\[
(\forall e \in \Sigma_o)(\forall x, x' \in X) (x, x') \in \mathcal{X} \Rightarrow [\delta(x, e) \neq \emptyset \Leftrightarrow \delta(x', e) \neq \emptyset]. \tag{5.11}
\]

One could verify (5.11) in order to verify eventual feasibility holds. However, our test for eventual feasibility uses quotient automaton \(G/\rho\) (as defined in Section 5.5) instead of \(G\). Let \(G/\rho = (Y, \Sigma_o, \eta, y_0)\). Let \(\mathcal{Y}\) denote the state set of \(G/\rho \otimes G/\rho\). Given \(e \in \Sigma_o\), we define a predicate \(P_e : Y \times Y \rightarrow \{true, false\}\) such that, for states \(y, y' \in Y\),

\[
P_e(y, y') \Leftrightarrow [\eta(y, e) \neq \emptyset \Leftrightarrow \eta(y', e) \neq \emptyset].
\]
By Theorem 7, (5.11) is equivalent to the following:

\[(\forall e \in \Sigma_o)(\forall y, y' \in Y) \ (y, y') \in \mathcal{Y} \Rightarrow P_e(y, y').\]  
(5.12)

We define another predicate for purposes of conveniently characterizing (5.12), which is used in the remainder of this subsection. Let \( F : Y \times Y \to \{true, false\} \) such that, for \( V \subseteq Y \times Y \),

\[ F(V) \Leftrightarrow [\forall e \in \Sigma_o, \forall (y, y') \in V, P_e(y, y')] .\]

Then (5.12) is equivalent to the following:

\[ F(\mathcal{Y}). \]  
(5.13)

One could exhaustively test pairs in \( \mathcal{Y} \) to verify (5.13). However, in the following, we demonstrate that it is possible to verify (5.13) using a subset of \( \mathcal{Y} \). To verify eventual feasibility using a subset of \( \mathcal{Y} \), we first introduce some basic properties of \( P_e \), then we provide some results on subsets of \( \mathcal{Y} \) that could be used.

For \( y \in Y \), let \( R(y) = \eta(y, \varepsilon) \). That is, \( R(y) \) is used to denote the \( \varepsilon \)-reach (i.e., \( \varepsilon \)-closure) of \( y \). The following is a trivial property of \( P_e \) that we find useful for verifying (5.13) using a subset of \( \mathcal{Y} \):

\[ \forall y_1, y_2, y_3 \in Y, y_2, y_3 \in R(y_1) \land y_3 \in R(y_2) \Rightarrow \]

\[ \forall e \in \Sigma_o, P_e(y_1, y_3) \Leftrightarrow [P_e(y_1, y_2) \land P_e(y_2, y_3)] .\]  
(5.14)

In the following, we use (5.14) and other properties of \( P_e \) to define a subset of \( \mathcal{Y} \) that can be used to verify eventual feasibility.

Consider the following partition of \( \mathcal{Y} \):

\[ \mathcal{Y}_1 = \{(y, y') \in \mathcal{Y} : y \in R(y') \lor y' \in R(y)\} , \]

\[ \mathcal{Y}_2 = \mathcal{Y} \setminus \mathcal{Y}_1 . \]
We define \( \mathcal{Y}_1 \subseteq \mathcal{Y} \) such that \( F(\mathcal{Y}) \) if and only if \( F(\mathcal{Y}_1 \cup \mathcal{Y}_2) \). Let \( \eta_r \) denote the transition function obtained by reversing transitions in \( \eta \). That is, for all \( \sigma \in \Sigma_o \cup \{ \varepsilon \} \), \((y', \sigma, y) \in \eta_r \) if and only if \((y, \sigma, y') \in \eta \). Let \( Y^*_{\varepsilon} \) denote the set of states in \( Y \) that have no incoming \( \varepsilon \)-transition in \( \eta \). That is, \( Y^*_{\varepsilon} = \{ y \in Y : \eta_r(y, \varepsilon) = \{ y \} \} \). Let \( Y^\dagger_{\varepsilon} \) denote the set of states in \( Y \) that have no outgoing \( \varepsilon \)-transition in \( \eta \). That is, \( Y^\dagger_{\varepsilon} = \{ y \in Y : \eta(y, \varepsilon) = \{ y \} \} \). Set \( \mathcal{Y}_1 \) denotes the subset of pairs of \( \mathcal{Y} \) involving a state \( y \in Y^\dagger_{\varepsilon} \) and a state \( y' \in R(y) \cap Y^\dagger_{\varepsilon} \). Formally,

\[
\mathcal{Y}_1 = \{(y, y') \in \mathcal{Y} : y \in Y^\dagger_{\varepsilon} \land y' \in R(y) \cap Y^\dagger_{\varepsilon} \lor y' \in Y^\dagger_{\varepsilon} \land y \in R(y') \cap Y^\dagger_{\varepsilon}\}.
\]

One can verify that, for any \((y_2, y_3) \in \mathcal{Y}_1 \), there exists a \((y_1, y_4) \in \mathcal{Y}_1 \) such that \( y_2 \in R(y_1), y_3 \in R(y_2), y_4 \in R(y_3) \). Then, by (5.14), \( F(\{(y_1, y_4)\}) \) implies \( F(\{(y_2, y_4)\}) \), which implies \( F(\{(y_2, y_3)\}) \). It follows that \( F(\mathcal{Y}_1) \) if and only if \( F(\mathcal{Y}_1') \). Consequently, \( F(\mathcal{Y}_1 \cup \mathcal{Y}_2) \) if and only if \( F(\mathcal{Y}_1 \cup \mathcal{Y}_2) \).

We define another subset of \( \mathcal{Y} \) that may be used in conjunction with \( \mathcal{Y}_1' \) to verify \( F(\mathcal{Y}) \). We define set \( \mathcal{Y}_2' \), which denotes the smallest subset of \( \mathcal{Y} \) such that \( \mathcal{Y} \) is the \( \varepsilon \)-closure of \( \mathcal{Y}_2' \). Let \( \Lambda \) denote the transition function of \( G/\rho \otimes G/\rho \). Let \( \Lambda_r \) denote the reversal of \( \Lambda \), i.e., \((y'_1, y'_2), \sigma, (y_1, y_2) \in \Lambda_r \) if and only if \((y_1, y_2), \sigma, (y'_1, y'_2) \in \Lambda \) for \( \sigma \in \Sigma_o \cup \{ \varepsilon \} \). Set \( \mathcal{Y}_2' \) is the set of states in \( \mathcal{Y} \) that have no incoming \( \varepsilon \)-transition in \( \Lambda \). That is, \( \mathcal{Y}_2' = \{(y_1, y_2) \in \mathcal{Y} : \Lambda_r((y_1, y_2), \varepsilon) = \{(y_1, y_2)\}\} \). One can verify that \( \mathcal{Y} = \bigcup_{(y_1, y_2) \in \mathcal{Y}_2'} \Lambda_*(y_1, y_2, \varepsilon) \).

We will show in Theorem 10 that \( F(\mathcal{Y}) \) if and only if \( F(\mathcal{Y}_1' \cup \mathcal{Y}_2') \). First, we require some more properties of \( P_e \).

Another property of \( P_e \) that is useful for purposes of simplifying the number of
test for verifying $F(Y)$ is that $P_e$ is transitive:

$$
\forall y_1, y_2, y_3 \in Y, \forall e \in \Sigma_o, P_e(y_1, y_2) \wedge P_e(y_2, y_3) \Rightarrow P_e(y_1, y_3).
$$

(5.15)

We also have the following property of $P_e$, which follows from (5.14) and (5.15):

**Lemma 16.**

$$
\forall y_1, y_2 \in Y, \forall y'_1 \in R(y_1), \forall y'_2 \in R(y_2), \forall y''_1 \in R(y'_1), \forall y''_2 \in R(y'_2), \forall e \in \Sigma_o,

P_e(y_1, y_2) \wedge P_e(y_1, y''_1) \wedge P_e(y_2, y''_2) \Rightarrow P_e(y'_1, y'_2).
$$

(5.16)

**Proof.** We have $P_e(y_1, y'_1)$ by $P_e(y_1, y''_1)$, $y'_1 \in R(y_1)$, $y''_1 \in R(y'_1)$ and (5.14). Symmetrically, we have $P_e(y_2, y'_2)$. Then $P_e(y'_1, y'_2)$ by $P_e(y_1, y'_1)$, $P_e(y_1, y_2)$, $P_e(y_2, y'_2)$ and (5.15).

We use (5.14)–(5.16) to demonstrate that $F(Y'_1 \cup Y'_2)$ if and only if $F(Y'_1 \cup Y'_3)$.

**Theorem 10.** $F(Y'_1 \cup Y'_2)$ if and only if $F(Y'_1 \cup Y'_3)$.

**Proof.** Let $(y_3, z_3) \in Y'_2$. There exists $(y_2, z_2) \in Y'_1$ such that $y_3 \in R(y_2)$ and $z_3 \in R(z_2)$ since $Y = \bigcup_{(y', y'') \in Y'_1} \Lambda((y', y''), \varepsilon)$. For all $y \in Y$, $R(y) \cap Y'_1$ is not empty. So there exist $y_1 \in R(y_3) \cap Y'_1$ and $z_4 \in R(z_3) \cap Y'_1$. For all $y \in Y$, there exists $y' \in Y'_1$ such that $y \in R(y')$. So there exist $y_1, z_1 \in Y'_1$ such that $y_2 \in R(y_1)$ and $z_2 \in R(z_1)$.

Pairs $(y_1, y_4), (z_1, z_4) \in Y'_1$ since $y_1, z_1 \in Y'_1$, $y_4, z_4 \in Y'_1$, $y_4 \in R(y_1)$, $z_4 \in R(z_1)$ and by definition of $Y'_1$. The situation described here is depicted in Figure 5.5.

Suppose that $P_e(y_1, y_4), P_e(z_1, z_4)$ and $P_e(y_3, z_3)$ where $e \in \Sigma_o$. We have $P_e(y_1, y_3)$ by $P_e(y_1, y_4)$, $y_3 \in R(y_1)$, $y_4 \in R(y_3)$ and (5.14). We have $P_e(y_1, z_3)$ by $P_e(y_1, y_3)$, $P_e(y_3, z_3)$ and (5.15). We have $P_e(z_1, z_3)$ by $P_e(z_1, z_4)$, $z_3 \in R(z_1)$, $z_4 \in R(z_3)$ and (5.14). Then $P_e(y_1, z_1)$ by $P_e(y_1, z_3)$, $P_e(z_1, z_3)$, and (5.15). Then $P_e(y_2, z_2)$ by $y_2 \in R(y_1), z_2 \in R(z_1), y_4 \in R(y_2), z_4 \in R(z_2), P_e(y_1, y_4), P_e(z_1, z_4), P_e(y_1, z_1)$ and (5.16).
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

192

\[ \begin{align*}
&y_1 & y_2 & y_3 & y_4 \\
&\varepsilon^* & \varepsilon^* & \varepsilon^* & \varepsilon^* \\
&y_2 & y_3 & & \\
&\varepsilon^* & & \varepsilon^* & \\
&z_2 & z_3 & & \\
&\varepsilon^* & \varepsilon^* & & \\
&z_1 & & & z_4 \\
\end{align*} \]

Figure 5.5: A visual aid to help demonstrate that \( F(Y'_1 \cup Y'_2) \) if and only if \( F(Y'_1 \cup Y'_f) \).

A dashed, undirected edge labeled by a set denotes that a pair formed by the two states with which the edge is incident is a member of the set, e.g., \((y_1, y_4) \in Y'_1\). A directed edge labeled by \( \varepsilon^* \) denotes that the sink vertex is in the \( \varepsilon \)-closure of the source vertex in \( G/\rho \).

\((\Rightarrow)\) Suppose \( F(Y'_1 \cup Y'_2) \) holds. Let \((y_2, z_2) \in Y'_f\).

Suppose there exists a \((y_3, z_3) \in Y_2\) such that \( y_3 \in R(y_2) \) and \( z_3 \in R(z_2) \). Then we have the same situation as is described above. It follows that \( F(\{(y_2, z_2)\}) \) holds, since \( F(\{(y_3, z_3)\}) \) by our reasoning about \( P_e \) for the above situation.

Suppose there does not exist a \((y_3, z_3) \in Y_2\) such that \( y_3 \in R(y_2) \) and \( z_3 \in R(z_2) \). Then either \( y_2 \in R(z_2) \) or \( z_2 \in R(y_2) \). Suppose \( y_2 \in R(z_2) \). Let \( p = y_2, q = z_2 \).

For all \( y \in Y \), \( R(y) \cap Y^+_f \) is not empty. So there exists a \( p' \in R(p) \cap Y^+_f \). For all \( y \in Y \), there exists \( y' \in Y^+_f \) such that \( y \in R(y') \). Then there exists a \( q' \in Y^+_f \) such that \( q \in R(q') \). Pair \((q', p') \in Y'_1\), since \( q' \in Y^+_f \), \( p' \in Y^+_f \) and \( p' \in R(q') \).

Thus \( F(\{(q', p')\}) \) since \( p \in R(q'), p' \in R(p) \) and (5.14). Therefore, \( F(\{(q, p)\}) \) since \( q \in R(q'), p \in R(q) \) and (5.14). Thus \( F(\{(z_2, y_2)\}) \) holds. Symmetrically, \( F(\{(z_2, y_2)\}) \) holds when \( z_2 \in R(y_2) \).

By the above two cases, if \( F(Y'_1 \cup Y'_2) \) holds then \( F(Y'_1 \cup Y'_f) \) holds.
Suppose $F(Y'_1 \cup Y'^1_f)$ holds. Let $(y_3, z_3) \in Y_2$. Then we have the situation described above and depicted in Figure 5.5. The only difference is that, initially, we have $P_e(y_1, y_4), P_e(z_1, z_4)$ and $P_e(y_2, z_2)$ for all $e \in \Sigma$. We have $P_e(y_1, y_2)$ by $P_e(y_1, y_4), y_2 \in R(y_1), y_4 \in R(y_2)$ and (5.14). We have $P_e(y_1, y_2)$ by $P_e(y_1, y_2), P_e(y_2, z_2)$ and (5.15). We have $P_e(z_1, z_4)$ by $P_e(z_1, z_4), z_2 \in R(z_1), z_4 \in R(z_2)$ and (5.14). Then $P_e(y_1, z_1)$ by $P_e(y_1, z_2), P_e(z_1, z_2)$, and (5.15). Then $P_e(y_3, z_3)$ by $y_3 \in R(y_1), z_3 \in R(z_1), y_4 \in R(y_3), z_4 \in R(z_3), P_e(y_1, y_4), P_e(z_1, z_4), P_e(y_1, z_1)$ and (5.16).

It follows that if $F(Y'_1 \cup Y'^1_f)$ holds then $F(Y'_1 \cup Y_2)$ holds.

By Theorem 10, one could exhaustively test pairs in $Y'_1 \cup Y'^1_f$ in order to verify $F(Y)$. However, noting that $P_e$ is transitive (5.15), there is a potentially better way which we investigate next.

We provide an example where a set of state pairs derived from pairs in $Y'^1_f$ may be used instead of $Y'^1_f$ for verifying eventual feasibility. Consider the automaton $G/\rho$ in Figure 5.6. Dashed, undirected edges between two states denote that the states are indistinguishable. One can verify that $Y'_1 = \{(1, 5), (2, 3), (2, 4), (6, 7), (6, 8)\}$ and $Y'^1_f = \{(3, 7), (4, 8), (1, 1), (2, 2), (6, 6)\}$. One could verify eventual feasibility by testing $F(Y'_1 \cup Y'^1_f)$. However, noting that $P_e$ is transitive, we may be able to omit testing some pairs. Suppose $F(Y'_1)$ holds. We could omit testing pair $(3, 7)$. This is discussed in the following.

We have $F(\{(2, 3), (2, 4)\})$ since $F(Y'_1)$ and $\{(2, 3), (2, 4)\} \subseteq Y'_1$. Then $F(\{(3, 4)\})$ by (5.15). Symmetrically, we have $F(\{(6, 7), (6, 8)\})$ and $F(\{(7, 8)\})$. Then $F(\{(3, 7)\})$ if and only if $F(\{(4, 8)\})$ by (5.15). So we could omit testing $F(\{(3, 7)\})$, and instead just test $F(\{(4, 8)\})$ (or vice versa). Also, observe that $F(\{(3, 7), (4, 8)\})$ if and only if
Figure 5.6: Example used to demonstrate that one could omit testing some pairs in $Y^\uparrow_f$.

$F(\{(2,6)\})$ by (5.15), though $(2,6) \notin Y^\uparrow_f$. So there are multiple ways that we could omit testing of pairs in $Y^\uparrow_f$, or substitute testing pairs in $Y^\uparrow_f$ by the testing of pairs outside of $Y^\uparrow_f$.

We formalize some of the ways in which one could verify eventual feasibility by omitting pairs from $Y^\uparrow_f$ or by substituting pairs from $Y^\uparrow_f$ with other pairs next. Given $u \in Y$, let $R^\uparrow(u)$ denote the union of the set of states in the $\varepsilon$-closure of $u$ and the set of states which have $u$ in their $\varepsilon$-closure. Formally, $R^\uparrow(u) = \{ v \in Y : u \in R(v) \lor v \in R(u) \}$. We let $R^{\uparrow*}(u)$ denote the transitive-closure of $R^\uparrow(u)$. Formally, state $v \in R^{\uparrow*}(u)$ if and only if $\exists y_1, y_2, \ldots, y_n \in Y, y_1 \in R^\uparrow(u) \land y_2 \in R^\uparrow(y_1) \land \ldots \land v \in R^\uparrow(y_n)$.

We demonstrate in Theorem 11 that, when $F(Y^\uparrow_f)$ holds, an effective way to test if $F(Y^\uparrow_f \cup Y^\uparrow_f)$ holds is to test if $F(Y^\uparrow_f \cup S)$ holds (i.e., $F(Y^\uparrow_f \cup Y^\uparrow_f) \iff F(Y^\uparrow_f \cup S)$) where $S$ is a set containing pairs that are reachable from pairs in $Y^\uparrow_f$ using $R^{\uparrow*}$. Specifically,
$S$ is specified as follows:

$$\forall u', v' \in Y, (u', v') \in S \Rightarrow u' \in Y_{↑} \cup Y_{↓} \land v' \in Y_{↑} \cup Y_{↓}$$ (5.17)

\[
\land \\
\forall u, v \in Y, (u, v) \in Y_{↑} \Rightarrow \\
\exists u', v' \in Y, (u', v') \in S \land u' \in R^{\geq}(u) \land v' \in R^{\leq}(v)
\]

\[
\land \\
\forall u', v' \in Y, (u', v') \in S \Rightarrow \\
\exists u, v \in Y, (u, v) \in Y_{↑} \land u \in R^{\geq}(u') \land v \in R^{\leq}(v').
\]

By using $Y'$ and $S$ to verify $F(Y)$, we are reducing verification of $F(Y)$ to verification of pairs in a subset of $(Y_{↑} \cup Y_{↓}) \times (Y_{↑} \cup Y_{↓})$. In some cases, $S$ may be chosen in a manner such that $S \cap Y'_{↑} \neq \emptyset$, hence reducing the number of pairs that needed to be tested for verifying $F(Y)$. As $S$ is only specified in (5.17) and not precisely defined, the problem that arises is to determine how $S$ should be chosen for purposes of verifying $F(Y)$. This is left as an open problem. Instead, in the following, we define a set of state pairs $FOREST_{Y'_{↑} \cup S}$ that is derived from $Y'_{↑} \cup S$. The cardinality of $FOREST_{Y'_{↑} \cup S}$ is no greater than $Y'_{↑} \cup S$, which is to be demonstrated in Theorem 13. We demonstrate in Corollary 5 that $FOREST_{Y'_{↑} \cup S}$ may be used to verify $F(Y)$. For any two choices of $S, S'$ that satisfy (5.17), $|FOREST_{Y'_{↑} \cup S}| = |FOREST_{Y'_{↑} \cup S'}|$, which is demonstrated in Theorem 12. Thus, any choice of $S$ may be used when verifying $F(Y)$ using the proposed set $FOREST_{Y'_{↑} \cup S}$.

First, the following results are necessary. The following is necessary in order to justify using sets satisfying (5.17) for verifying $F(Y' \cup Y'_{↓})$. The following states that, if pairs of states in $Y'_{↑}$ are eventually followed by the same event observations (i.e.,
sensor activation decisions), then any two states which are reachable via sequences of ε-transitions and reversed ε transitions are also eventually followed by the same event observations.

**Lemma 17.** \( F(Y'_1) \Rightarrow [\forall u, v, u', v' \in Y, u' \in R^{\uparrow \ast}(u) \land v' \in R^{\downarrow \ast}(v) \Rightarrow (F(\{(u, v)\}) \Leftrightarrow F(\{(u', v')\}))]. \)

**Proof.** For all \( a, b \in Y, b \in R(a) \Rightarrow F(\{(a, b)\}) \) by \( F(Y'_1) \) and \( (5.14) \). Then the Lemma statement holds by \( (5.15) \) and definition of \( R^{\downarrow \ast} \).

The next result states that we can use \( Y'_1 \) and \( S \) to verify \( F(Y) \). The next result follows directly from Lemma 17 by \( (5.17) \) and \( R^{\downarrow \ast} \) is symmetric.

**Theorem 11.** Given \( S \) satisfying \( (5.17) \), \( F(Y'_1) \Rightarrow [F(Y'_1 \cup Y'_1) \Leftrightarrow F(Y'_1 \cup S)]. \)

Let \((V, E)\) denote an undirected graph, with \( V \) denoting the set of vertices and \( E \) denoting the set of edges such that \( E \subseteq V \times V \). Let \( \phi : V \times V \) denote the reachability relation of an undirected graph \((V, E)\). That is, for \( u, v \in V \), \( u \phi v \) if and only if \( u \) is reachable from \( v \) by a (possibly empty) sequence of edges in \( E \) (and vice versa). One can verify that \( \phi \) is an equivalence relation on \( V \). For \( v \in V \), let \([v]_{\phi} \) denote the equivalence class of \( v \) with respect to relation \( \phi \):

\[ [v]_{\phi} = \{ u \in V : u \phi v \}. \]

We let \( V/\phi \) denote \( \cup_{v \in V} [v]_{\phi} \), the set of all equivalence classes of \( V \) with respect to \( \phi \). Set \( V/\phi \) forms a partition of \( V \). Specifically, the elements of \( V/\phi \) denote the connected components of \((V, E)\). Given a connected component \( C \in V/\phi \), any two vertices in \( C \) are reachable from one another. A spanning tree of \( C \) is a graph \((C, E')\) such that any two vertices in \( C \) are reachable from one another by edges in \( E' \). Furthermore, no edge in \( E' \) is incident on a vertex not in \( C \), and \( E' \) is acyclic.
Given undirected graph \((V, E)\), let \(\text{FOREST}_E\) denote an undirected graph where every connected component \(C \in V/\phi\) is replaced with a spanning tree of \(C\).

The following Lemma is trivial.

**Lemma 18.** \(\forall V \subseteq Y \times Y, F(V) \Leftrightarrow F(\text{FOREST}_V)\)

**Proof.** It is not difficult to see that, given a set \(V \subseteq Y \times Y, F(V) \Rightarrow F(\text{FOREST}_V)\) since \(\text{FOREST}_V \subseteq V\). Furthermore, \(F(\text{FOREST}_V) \Rightarrow F(V)\) by (5.15). The Lemma statement follows.

We have the following final result concerning verification of (5.8).

**Corollary 5.** Given \(S\) satisfying (5.17), \(F(Y'_1) \Rightarrow [5.8] \Leftrightarrow F(\text{FOREST}_{Y'_1\cup S})\).

**Proof.** When \(F(Y'_1)\) holds, we have that \(F(Y'_1 \cup Y'_2) \Leftrightarrow F(Y'_1 \cup S) \Leftrightarrow F(\text{FOREST}_{Y'_1\cup S})\) for \(S\) satisfying (5.17) by Theorem 11 and Lemma 18. We have already established that \(F(Y'_1 \cup Y'_2) \Leftrightarrow F(Y'_1 \cup Y'_2) \Leftrightarrow F(Y') \Leftrightarrow (5.13) \Leftrightarrow \ldots \Leftrightarrow (5.8)\). The Corollary statement follows.

Next we prove that, given \(S, S'\) satisfying (5.17), \(|\text{FOREST}_{Y'_1\cup S}| = |\text{FOREST}_{Y'_1\cup S'}|\). That is, it does not matter which \(S\) satisfying (5.17) is chosen for purposes of reducing the number of tests required for verifying if \(F(Y'_1 \cup Y'_2)\) holds.

**Theorem 12.** Given \(S, S'\) satisfying (5.17), \(F(Y'_1) \Rightarrow |\text{FOREST}_{Y'_1\cup S}| = |\text{FOREST}_{Y'_1\cup S'}|\).

**Proof.** We prove this by demonstrating that \(\text{FOREST}_{Y'_1\cup S}\) and \(\text{FOREST}_{Y'_1\cup S'}\) contain the same components. Specifically, we demonstrate that a state \(v \in Y'_{\uparrow, f} \cup Y'_{\downarrow, f}\) is reachable from state \(u \in Y'_{\uparrow, f} \cup Y'_{\downarrow, f}\) in \(\text{FOREST}_{Y'_1\cup S}\) if and only if \(v\) is reachable from
u in FOREST_{Y_1 \cup S'}$. The Theorem statement follows directly from this fact and by definition of FOREST_{Y_1 \cup S} and FOREST_{Y_1 \cup S'}.

Assume there exists \((u, v) \in \text{FOREST}_{Y_1 \cup S}\) such that \(v\) is not reachable from \(u\) in FOREST_{Y_1 \cup S}. It cannot be that \((u, v) \in Y_1\) for otherwise \(v\) is reachable from \(u\) in FOREST_{Y_1 \cup S} by definition of FOREST_{Y_1 \cup S}. So \((u, v) \in S\).

Since \(S\) satisfies (5.17) there exists \(u_1, v_1 \in Y_1\), \((u_1, v_1) \in Y_1^\uparrow \cup Y_1^\downarrow\), \(u_1 \in R^\uparrow v(u)\) and \(v_1 \in R^\downarrow u(v)\). Since \(S'\) satisfies (5.17) and \((u_1, v_1) \in Y_1^\uparrow \cup Y_1^\downarrow\), \(u_1 \in R^\uparrow v(u_1)\) and \(v_1 \in R^\downarrow u(v_1)\). Then, by definition, \(v_1\) is reachable from \(u_1\) in \(\text{FOREST}_{Y_1 \cup S'}\). Furthermore, by transitivity of \(R^\uparrow u\) and \(R^\downarrow v\).

Given \(a, b \in Y_1^\uparrow \cup Y_1^\downarrow\), let \(a \leftrightarrow b\) hold if and only if \(a \in R^\uparrow b\). If \(a \leftrightarrow b\) then \((a, b) \in Y_1^\uparrow \cup (b, a) \in Y_1^\downarrow\). It follows that \(a\) is reachable from \(b\) and vice versa in \(\text{FOREST}_{Y_1 \cup S'}\).

That \(u' \in R^\uparrow v(u)\) implies that there exists \(b_1, b_2, \ldots, b_n \in Y_1^\uparrow \cup Y_1^\downarrow\), \(u' \leftrightarrow b_1 \leftrightarrow b_2 \leftrightarrow \ldots \leftrightarrow b_n \leftrightarrow u\). It follows that \(u\) is reachable from \(u'\) in \(\text{FOREST}_{Y_1 \cup S'}\).

Symmetrically, it follows that \(v'\) is reachable from \(v\) in \(\text{FOREST}_{Y_1 \cup S'}\). By these two facts and \(u'\) is reachable from \(v'\), it follows that \(u\) is reachable from \(v\) in \(\text{FOREST}_{Y_1 \cup S'}\). A contradiction is reached.

Symmetrically, one can prove that \((u, v) \in \text{FOREST}_{Y_1 \cup S'}\) implies \(v\) is reachable from \(u\) in \(\text{FOREST}_{Y_1 \cup S}\).

It follows directly from the previous two facts that a state \(v \in Y_1^\uparrow \cup Y_1^\downarrow\) is reachable from state \(u \in Y_1^\uparrow \cup Y_1^\downarrow\) in \(\text{FOREST}_{Y_1 \cup S}\) if and only if \(v\) is reachable from \(u\) in \(\text{FOREST}_{Y_1 \cup S'}\). 

We argue that the cardinality of \(\text{FOREST}_{Y_1 \cup S}\) is not greater than the cardinality of other sets defined in this subsection that may be used to test (5.8). This justifies
the use of $FOREST_{\mathcal{Y}_1 \cup S}$ for verifying \((5.8)\).

**Theorem 13.** Given $S$ satisfying \((5.17)\), the cardinality of $FOREST_{\mathcal{Y}_1 \cup S}$ is not greater than the cardinality of sets $\mathcal{Y}_1 \cup S$, $\mathcal{Y}_1 \cup \mathcal{Y}_2$, and $\mathcal{Y}_1 \cup \mathcal{Y}_2$.

**Proof.** Cardinality $|FOREST_{\mathcal{Y}_1 \cup S}| \leq |\mathcal{Y}_1 \cup S|$ by definition of $FOREST_{\mathcal{Y}_1 \cup S}$. One can select an $S'$ satisfying \((5.17)\) such that $|S'| = |\mathcal{Y}_1'|$, in which case $|\mathcal{Y}_1' \cup S'| = |\mathcal{Y}_1' \cup \mathcal{Y}_2|$. Cardinality $|FOREST_{\mathcal{Y}_1 \cup S'}| \leq |\mathcal{Y}_1 \cup S'|$ by definition of $FOREST_{\mathcal{Y}_1 \cup S}$. Then $|FOREST_{\mathcal{Y}_1 \cup S'}| \leq |\mathcal{Y}_1' \cup \mathcal{Y}_2'|$. It follows that $|FOREST_{\mathcal{Y}_1 \cup S}| \leq |\mathcal{Y}_1' \cup \mathcal{Y}_2'|$ by Theorem 12.

Let $(u, v) \in \mathcal{Y}_1' \cup \mathcal{Y}_2$.

For all $y \in \mathcal{Y}$, there exists a $y' \in \mathcal{Y}_1'^\uparrow$ such that $y \in R(y')$. Let $\{(u_1, v_1), \ldots, (u_n, v_n)\} \subseteq \mathcal{Y}_1'^\uparrow$ denote the set of all such elements for $(u, v)$ where $n > 0$. Let $u' \in R^{\uparrow \star}(u) \cap (\mathcal{Y}_1'^\uparrow \cup \mathcal{Y}_2'^\uparrow)$, $v' \in R^{\downarrow \star}(v) \cap (\mathcal{Y}_1'^\downarrow \cup \mathcal{Y}_2'^\downarrow)$. Since $u \in R(u_i)$, $u' \in R^{\uparrow \star}(u_i) \cap (\mathcal{Y}_1'^\uparrow \cup \mathcal{Y}_2'^\uparrow)$. Also, since $v \in R(v_i)$, $v' \in R^{\downarrow \star}(v_i) \cap (\mathcal{Y}_1'^\downarrow \cup \mathcal{Y}_2'^\downarrow)$. We add $(u', v')$ to set $S''$.

Consider any $(p, q) \in \mathcal{Y}_1'^\uparrow$ such that there does not exist a $(u, v) \in \mathcal{Y}_2$ where $u \in R(p)$ and $v \in R(q)$. Then either $p \in R(q)$ or $q \in R(p)$. In either case, there exists $(y, y') \in \mathcal{Y}_1'$ such that $\{p, q\} \in R(y)$, $y' \in R(p)$ and $y' \in R(q)$. We add $(y, y')$ to set $S'''$.

From the above two cases and by definition of $S''$ and $S'''$, it follows that $S'' \cup S'''$ satisfies \((5.17)\). We have that $S''' \subseteq \mathcal{Y}_1'$. Then $\mathcal{Y}_1' \cup S'' \cup S''' = \mathcal{Y}_1' \cup S''$. We have that $|S'''| \leq |\mathcal{Y}_2|$. Then $|\mathcal{Y}_1' \cup S'''| = |\mathcal{Y}_1' \cup \mathcal{Y}_2|$. Cardinality $|FOREST_{\mathcal{Y}_1 \cup S'''}| \leq |\mathcal{Y}_1' \cup S'''|$. Then $|FOREST_{\mathcal{Y}_1 \cup S'''}| \leq |\mathcal{Y}_1' \cup \mathcal{Y}_2|$. \(\square\)

To summarize, Corollary 5 and Theorem 12 permit the following test for verifying \((5.8)\). First, we verify that the same set of events are eventually observed following any two states where one state is in the $\varepsilon$-reach of the other state. Formally, we verify
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

200

\( F(\mathcal{Y}_1') \), i.e., for all states \( u \in Y_1^+ \), for all \( v \in R(u) \cap Y_1^+ \), we verify that \( F(\{(u, v)\}) \) holds. If this holds, then we may compute \( \mathcal{Y}_1^+ \) from the set of states of \( G/\rho \otimes G/\rho \). We then compute a set \( S \) satisfying (5.17) from \( \mathcal{Y}_1^+ \). That is, we compute states \( u', v' \in Y_1^+ \cup Y_1^- \) from pairs \((u, v) \in \mathcal{Y}_1^+\) where \( u' \in R^3(u) \), \( v' \in R^3(v) \) then add \((u', v')\) to a set \( S \). From \( \mathcal{Y}_1' \) and \( S \) we can compute \( FOREST_{\mathcal{Y}_1' \cup S} \) and test \( F(FOREST_{\mathcal{Y}_1' \cup S}) \) in order to determine if (5.8) holds. From Theorem 13, we are guaranteed that the number of tests conducted using \( FOREST_{\mathcal{Y}_1' \cup S} \) is not greater than when testing using other sets considered in this subsection. Further, since we have already verified that \( F(\mathcal{Y}_1') \) holds, it suffices to test that \( F(FOREST_{\mathcal{Y}_1' \cup S \setminus \mathcal{Y}_1'}) \) holds in order to determine if (5.8) holds. Depending on which pairs are added to \( FOREST_{\mathcal{Y}_1' \cup S} \), \( FOREST_{\mathcal{Y}_1' \cup S \setminus \mathcal{Y}_1'} \) may have cardinality less than the cardinality of \( S \).

5.7.4 Computing the Extended Specification in Problems of Sensor Activation

In Wang et. al. [53] an algorithm for computing sensor activation decisions for purposes of distinguishing between certain pairs of states of a given automaton is demonstrated. The solution used relies on computing the extended specification of a given set of state pairs. We demonstrate how the extended specification may be computed using a slight generalization of the product \( \otimes \).

Let \( G \) be a DFA where \( G = (X, \Sigma, \delta, x_0) \). We are given a set of state pairs, \( T_{spec} \), where \( T_{spec} \subseteq X \times X \). Set \( \Sigma \) is partitioned into an observable event set \( \Sigma_o \), and unobservable event set \( \Sigma_{uo} \). We let \( P : \Sigma^* \rightarrow \Sigma_o^* \) denote the natural projection. The extended specification denotes those pairs of states in \( G \) that are followed by strings that appear identical (according to \( P \)) and which lead to a pair of states in \( T_{spec} \). The
extended specification, $T_{\text{spec}}$, is defined as follows

$$T_{\text{spec}}^e = \{(x, x') \in X \times X : \exists s, s' \in \Sigma^*, P(s) = P(s') \land (\delta(x, s), \delta(x', s')) \in T_{\text{spec}}\}.$$

The extended specification is used in subsequent works [4, 58, 47].

We let $G'$ denote an automaton obtained from $G$ by replacing the labels of transitions labeled by events in $\Sigma_{\text{uo}}$ with $\varepsilon$, and reversing all transitions in $\delta$. Further, $G'$ contains no initial state. Formally, $G' = (X, \Sigma, \delta')$ where $\delta'$ is defined as follows

- if $(x, \sigma, x') \in \delta$ and $\sigma \in \Sigma_o$ then $(x', \sigma, x) \in \delta'$;
- if $(x, \sigma, x') \in \delta$ and $\sigma \in \Sigma_{\text{uo}}$ then $(x', \varepsilon, x) \in \delta'$.

We generalize $\otimes$ for NFA with no specific initial state, but where pairs consisting of a single state from each of the respective input NFA are given. Given an NFA $A_i = (S_i, \Sigma_i, f_i)$ and a set of states $S_0 \subseteq S_1 \times S_2$, $A_1 \otimes_{S_0} A_2$ denotes an automaton $(\mathcal{S}, \Sigma_1 \cap \Sigma_2, \Delta, S_0)$ where $\mathcal{S} \subseteq S_1 \times S_2$, $\mathcal{S} \supseteq S_0$ and $\Delta : \mathcal{S} \times ((\Sigma_1 \cap \Sigma_2) \cup \{\varepsilon\}) \to 2^\mathcal{S}$. If $(s_1, s_2) \in \mathcal{S}$ then

- if $(s_1, \varepsilon, s_1') \in f_1$ then $((s_1, s_2), \varepsilon, (s_1', s_2)) \in \Delta$;
- if $(s_2, \varepsilon, s_2') \in f_2$ then $((s_1, s_2), \varepsilon, (s_1, s_2')) \in \Delta$;
- if $(s_1, \sigma, s_1') \in f_1$ and $(s_2, \sigma, s_2') \in f_2$ then $((s_1, s_2), \sigma, (s_1', s_2')) \in \Delta$.

Note that Algorithm 2 may be easily modified for computing $A_1 \otimes_{S_0} A_2$. Set $S_0$ would be provided as an additional input parameter to Algorithm 2. Then, on line 1, assign $\mathcal{S} \leftarrow \text{Pending} \leftarrow S_0$ instead of $\mathcal{S} \leftarrow \text{Pending} \leftarrow \{(x_{0,1}, x_{0,2})\}$.

One can verify that the extended specification, $T_{\text{spec}}^e$, is precisely the state set of automaton $G' \otimes_{T_{\text{spec}}} G'$. 


5.8 Verifying if one state is distinguished from another

One problem that is solved by the results of Section 5.4 is the problem of determining if two states of a given NFA are indistinguishable. That is, given NFA \( G = (X, \Sigma, \delta, x_0) \) and states \( x, y \in X \), determine if \( (x, y) \notin \Pi \) (cf. (5.1)). This problem is equivalent to the following problem.

**Problem 13.** Given NFA \( G = (X, \Sigma, \delta, x_0) \) and states \( x, y \in X \), determine if there exists a string \( s \in L(G) \) such that \( x \in \delta(x_0, s) \) and \( y \notin \delta(x_0, s) \).

Problem 13 is solvable in polynomial-time by Theorem 4 and Corollary 3. Also, the complement of Problem 13 is easily solvable in polynomial-time. That is, for states \( x, y \in X \), the problem of verifying if \( (x, y) \notin \Pi \) can be decided using the same approach.

In this section, we consider a much more difficult problem than verifying if two states are indistinguishable. We consider the problem of determining if one state \( x \) may ever be distinguished from another state \( y \), i.e., the problem of deciding if there exists a string \( s \) that leads to state \( x \), but not to state \( y \). One can verify that if \( x \) and \( y \) are distinguishable (i.e., \( (x, y) \notin \Pi \)) then for all strings \( s \) that lead to \( x \), \( s \) does not lead to \( y \). That is, the problem of verifying if \( x \) and \( y \) are distinguishable is stronger than the problem we consider in this section.

Also, if \( x \) and \( y \) are indistinguishable (i.e., \( (x, y) \in \Pi \)), then it is not necessarily the case that the answer is negative. Even when two states are indistinguishable, there may still exist a string that leads to one state, but not the other.

Formally, the problem we consider is the following:
Problem 14. Given NFA $G = (X, \Sigma, \delta, x_0)$ and states $x, y \in X$, determine if there exists a string $s \in L(G)$ such that $x \in \delta(x_0, s)$ and $y \notin \delta(x_0, s)$.

Unlike Problem 13, Problem 14 is likely not solvable in polynomial-time. Specifically, Problem 14 is PSPACE-complete. We prove this claim in the remainder of this section.

The proof follows by trivial polynomial-time reductions between Problem 14 and Problem 12 of Chapter 4, and the fact that Problem 12 is PSPACE-complete (Lemma 9).

Theorem 14. Problem 14 is PSPACE-complete.

Proof. Problem 12 to Problem 14

Let $\langle G, x, Q \rangle$ denote an instance of Problem 12 where $G = (X, \Sigma, \delta, x_0)$. We construct automaton $G'$ from $G$ by introducing a new state $y$ that is not in $G$, and add an $\varepsilon$-transition from $q$ to $y$ for all $q \in Q$. Let $\delta'$ denote the transition function of $G'$. Tuple $\langle G', x, y \rangle$ is an instance of Problem 14

Only If:

Suppose a decider for Problem 14 returns accept on input $\langle G', x, y \rangle$. Then there exists an $s \in L(G)$ such that $x \in \delta'(x_0, s)$ and $y \notin \delta'(x_0, s)$. Then $Q \cap \delta'(x_0, s) = \emptyset$ by definition of $G'$ and $y \notin \delta'(x_0, s)$. By definition of $G'$, for all $q \in Q$, for all $t \in L(G)$, $q \in \delta(x_0, t)$ if and only if $q \in \delta'(x_0, t)$. Similarly, for all $w \in L(G)$, $x \in \delta(x_0, w)$ if and only if $x \in \delta'(x_0, w)$. Thus $Q \cap \delta(x_0, s) = \emptyset$ and $x \in \delta(x_0, s)$.

If:

Suppose there exists an $s \in L(G)$ such that $x \in \delta(x_0, s)$ and $Q \cap \delta(x_0, s) = \emptyset$. Then, by definition of $G'$, $x \in \delta'(x_0, s)$ and $y \notin \delta'(x_0, s)$. So a decider for Problem 14 will return accept on input $\langle G', x, y \rangle$. 
CHAPTER 5. COMPUTING INDISTINGUISHABLE STATES

Problem 14 to Problem 12

Let \( \langle G, x, y \rangle \) denote an instance of Problem 14 where \( G = (X, \Sigma, \delta, x_0) \). We construct automaton \( G' \) from \( G \) by adding two new states with transitions to these states from \( x_0, x \) and \( y \). We add states \( x' \) and \( y' \). We add transitions \( (x, \varepsilon, x') \) and \( (y, \varepsilon, y') \). Thus any string leading to \( x \) (resp., \( y \)) also leads to \( x' \) (resp., \( y' \)), though \( x' \) (resp., \( y' \)) has no outgoing transitions. We also add new transitions to \( G \) that deliberately force there to exist a common string that leads to both \( x' \) and \( y' \). We add transition \( (x_0, \alpha, x') \) and transition \( (x_0, \alpha, y') \) such that \( \alpha \) is not in \( \Sigma \). Let \( \delta' \) denote the transition function of \( G' \). Tuple \( \langle G', x', \{y'\} \rangle \) denotes an instance of Problem 12.

Only If:

Suppose a decider for Problem 12 returns accept on input \( \langle G', x', \{y'\} \rangle \). Then there exists a string \( s \in L(G') \) such that \( x' \in \delta'(x_0, s) \) and \( \{y'\} \cap \delta'(x_0, s) = \emptyset \). By definition of \( G' \), it cannot be that \( s = \alpha \), since \( \{x', y'\} = \delta'(x_0, \alpha) \). Thus \( s \in L(G' \setminus \{\alpha\}) \). By definition of \( G' \), for all \( t \in L(G') \setminus \{\alpha\} \), \( x' \in \delta'(x_0, t) \) if and only if \( x \in \delta'(x_0, t) \). Symmetrically, for all \( w \in L(G') \setminus \{\alpha\} \), \( y' \in \delta'(x_0, w) \) if and only if \( y \in \delta'(x_0, w) \). Thus \( x \in \delta'(x_0, s) \) and \( \{y\} \cap \delta'(x_0, s) = \emptyset \). By definition of \( G' \), for all \( t \in L(G) \), \( \delta(x_0, t) \subseteq \delta'(x_0, t) \). Further \( L(G) \) is equal to \( L(G') \setminus \{\alpha\} \). Thus \( x \in \delta(x_0, s) \) and \( \{y\} \cap \delta(x_0, s) = \emptyset \). Stated equivalently, \( x \in \delta(x_0, s) \) and \( y \notin \delta(x_0, s) \).

If:

Suppose there exists an \( s \in L(G) \) such that \( x \in \delta(x_0, s) \) and \( y \notin \delta(x_0, s) \). Then, by definition of \( G' \), \( x' \in \delta'(x_0, s) \) and \( y' \notin \delta'(x_0, s) \). Stated equivalently, \( x' \in \delta'(x_0, s) \) and \( \{y'\} \cap \delta'(x_0, s) = \emptyset \). So a decider for Problem 12 will return accept on input \( \langle G', x', \{y'\} \rangle \).
5.9 Conclusions

In this chapter we introduced the application of the product for computing the indistinguishable state pairs of a given NFA. We provided a simple algorithm for computing the product of two given NFA and demonstrated the algorithm’s complexity. In the case of NFA with cycles of unobservable transitions, we defined a quotient automaton construction from which indistinguishable state pairs may be computed.

We demonstrated by example that the CLUSTER-TABLE algorithm of Wang et al. [50] for computing indistinguishable state pairs may be quartic in the state set and linear in the event set of a given NFA. In cases when the observable transitions of a given NFA are deterministic, computation of indistinguishable states via the product is preferable.

We demonstrated applications of computing indistinguishable state pairs using the product. The applications demonstrated include verifying observability, coobservability, eventual feasibility and computing the extended specification, which is used in the solution to problems of sensor activation. Though the problem of computing indistinguishable state pairs is trivially solved using the product, we proved that the problem of determining if there exists a string that leads to one state but not another is PSPACE-complete.
Chapter 6

Monotonic Observation Maps

6.1 Introduction

To solve problems of sensor activation and communication in DES it is usually required that the more an agent observes, the more it should know. Specifically, if an agent observes more event occurrences generated by a system, then the agent should know more about the current state of the system. When this requirement is satisfied, we can then consider the problem of reducing the number of event observations that an agent makes while permitting the agent to do its job (e.g., controlling a system, diagnosing failures in a system, distinguishing between certain states of a system). See Chapter 3 for previous work in this area.

In previous work, to guarantee that an agent knows more as it observes more, certain requirements are imposed on how an agent observes a system. That is, certain requirements are imposed on the agent’s observation map (a map from event sequences generated by the system to events that may be observed by the agent on
their occurrence). Examples of observation maps that have been considered in previous work are sensor activation maps and communication maps. A typical requirement is feasibility, i.e., if an agent cannot distinguish between two strings then the event observations made following both strings must not conflict.

In this chapter we consider different types of observation maps. For these observation maps we demonstrate that, if the observation maps satisfy feasibility, then the agent will be able to more precisely estimate the current state of the system as the agent observes more event occurrences. We demonstrate how most of these results may be applied for solving a problem of minimizing communication of event occurrences between two agents for satisfying some goal. Many of the results in this chapter concern only two agents. It remains open to generalize these results to the case of any number of agents.

6.2 Preliminaries

Here we introduce some definitions that are used throughout this chapter.

**Definition 1.** Let $L$ denote a formal language. Set $E(L)$ is used to denote the alphabet of language $L$. Specifically, set $E(L)$ is defined such that

- $L \subseteq E(L)^*$;

- There does not exist a set $A \subset E(L)$ where $L \subseteq A^*$.

We introduce the notion of an observation map, which generalizes sensor activation maps and communication maps that have been considered in the literature.
Definition 2. Let $L$ denote a formal language where $L$ is prefix-closed (i.e., $L = \overline{L}$) and $A \subseteq E(L)$. Map $O : L \rightarrow 2^A$ is an observation map (of $L$) if and only if
\[
\forall s \in L, O(s) \subseteq \{ e \in A : se \in L \}.
\]

We introduce the notion of containment of observation maps. This is required for modeling what it means for a given observation map to observe less (or more) than some other observation map.

Definition 3. Let $L$ denote a formal language where $L$ is prefix-closed and $A_1 \subseteq A_2 \subseteq E(L)$. Let $O_1 : L \rightarrow 2^{A_1}, O_2 : L \rightarrow 2^{A_2}$ denote observation maps of $L$.

Map $O_1$ is a subset of map $O_2$, i.e. $O_1 \subseteq O_2$, if and only if
\[
\forall s \in L, O_1(s) \subseteq O_2(s).
\]

Map $O_1$ is a strict subset of map $O_2$, i.e. $O_1 \subset O_2$, if and only if
\[
O_1 \subseteq O_2 \land \exists s \in L, O_1(s) \subset O_2(s).
\]

We define the union, intersection and difference of observation maps next.

Definition 4. Let $L$ denote a formal language where $L$ is prefix-closed and $A_1, A_2 \subseteq E(L)$. Let $O_1 : L \rightarrow 2^{A_1}, O_2 : L \rightarrow 2^{A_2}$ denote observation maps of $L$.

Map $[O_1 \cup O_2]$ is defined as follows
\[
\forall s \in L, [O_1 \cup O_2](s) = O_1(s) \cup O_2(s).
\]

Map $[O_1 \cap O_2]$ is defined as follows
\[
\forall s \in L, [O_1 \cap O_2](s) = O_1(s) \cap O_2(s).
\]
Map \([O_1 \setminus O_2]\) is defined as follows

\[ \forall s \in L, [O_1 \setminus O_2](s) = O_1(s) \setminus O_2(s). \]

Every observation map induces an information map. This is similar to notions of information map considered previously in this dissertation, and in previous works.

**Definition 5.** Let \(L\) denote a formal language where \(L\) is prefix-closed and \(A \subseteq E(L)\).

Map \(I : L \to A^*\) is an information map (of \(L\)) if

- \(I(\varepsilon) = \varepsilon;\)
- there exists observation map \(O : L \to 2^A\) such that

\[ \forall e \in E(L), \forall se \in L, I(se) = \begin{cases} I(s)e, & \text{if } e \in O(s) \\ I(s), & \text{if } e \notin O(s). \end{cases} \]

Map \(I\) is denoted by \(I_O\) when \(O\) is known.

Next we define precisely what it means for an agent to know more about a system (i.e., formal language). This is expressed in terms of information maps. Informally, an agent knows more if it has a more precise estimate of the sequence of events generated by the system. In the following definition, map \(I_2\) denotes the information map that yields the more precise estimate.

**Definition 6.** Let \(L\) denote a formal language where \(L\) is prefix-closed and \(I_1, I_2\) denote information maps of \(L\).

Information map \(I_1\) is a superset of information map \(I_2\), i.e. \(I_1 \supseteq I_2\), if and only if

\[ \forall s, t \in L, I_2(s) = I_2(t) \Rightarrow I_1(s) = I_1(t). \]
Relation $\subseteq$ between information maps is defined symmetrically.

In this chapter we show that, under certain circumstances, observing more event occurrences permits a more precise estimate. We express what it means for observation map $O$ to more precisely estimate a prefix-closed language $L$ than another observation map $O'$.

**Definition 7.** Given prefix-closed $L$ and two observation maps $O$ and $O'$ defined over $L$, we say that $O$ permits more precise estimates of $L$ than $O'$ if and only if

$$O' \subseteq O \Rightarrow I'_O \supseteq I_O.$$  

When the above holds, we also say that event observations are monotonic in the sense that observing more event occurrences permits more precise estimates.

In this chapter, we do not consider, for the most part, that observation maps are defined arbitrarily. Specifically, we consider that observation maps are feasible with respect to certain information maps. Next we provide a definition for feasibility of observation maps. This is similar to notions of feasibility discussed previously in this dissertation (e.g., feasibility of sensor activation maps, consistency of communication schemes). However, this notion is more general than those notions in that there is no assumed prior relation between the observation map and information map considered.

**Definition 8.** Let $L$ denote a formal language where $L$ is prefix-closed, $O$ denote an observation map of $L$ and $I$ denote an information map of $L$.

Map $O$ is feasible w.r.t. $I$ if

$$\forall e \in E(L), \forall s, t \in L, I(s) = I(t) \Rightarrow (e \in O(s) \Leftrightarrow e \in O(t)).$$

That $O$ is feasible w.r.t. $I$ is denoted by $O \mathcal{F} I$. That $O$ is not feasible w.r.t. $I$ is denoted by $O \not\mathcal{F} I$. 
In the sequel, we make frequent mention of an agent’s sensor activation map or communication map. Such maps are simply observation maps. A sensor activation map is an observation map that is intended to model how the agent observes occurrences of events in the agent’s observable event set using sensors that the agent can control. A communication map is an observation map that is intended to model how the agent communicates event occurrences that it observes to another agent.

6.3 When Observing More Event Occurrences Permits More Precise Estimates

In this section we provide some positive results regarding observation maps that satisfy feasibility and permit more precise estimates, i.e., as more is observed, more becomes known (i.e., more pairs of strings in the plant language become distinguishable). We also mention some negative results where observing more does not permit more precise estimates. Such negative results are similar to the lack of monotonicity for general communication maps introduced in the example of Wang [46] Section 2.2.6. In Section 6.4 we demonstrate how some of the positive results may be applied for solving a problem of communication between two agents. The proofs of some of the results in this section are accompanied by a proof idea to improve readability.

Theorems 15 and 16 demonstrate a basic property of observation maps where more precise estimates are obtained as agents observe more. Theorem 15 considers the case of two agents and is generalized in Theorem 16 to the case of more than two agents. The background is as follows. We are given a prefix-closed, formal language $L$ that models the set of possible behaviours of some discrete-event system. We are
also given multiple agents, $1 \ldots n$. Each agent $i$ has its own sensor activation map of $L$, denoted by $O_i$. The agents may choose to communicate every event that they observe to one another. This results in each agent observing the system language using information map $I_{[O_1 \cup \ldots \cup O_n]}$. In this case, any two agents observe the same sequence of event occurrences following any string generated in $L$. We then consider that each agent may choose to observe fewer event occurrences locally, denoted by sensor map $O_i'$ where $O_i' \subseteq O_i$. Further, each agent $i$ may choose to have fewer event occurrences communicated to another agent $j$, denoted by communication map $C_{ij}'$ where $C_{ij}' \subseteq O_i$. In this case, each agent now observes the system differently. Agent $i$ now observes the system using information map $I_i' = I_{[O_i' \cup \bigcup_{k \in \{1,\ldots,n\}\setminus\{i\}} C_{ki}']}$. Theorems 15 and [16] demonstrate that, if the smaller sensor activation maps $O_i'$ and communication maps $C_{ij}'$ are feasible w.r.t. information map $I_i'$, then the agents will know less than if they had chosen to observe more using $O_i$ and communicate everything they observe to one another. That is, information map $I_{[O_1 \cup \ldots \cup O_n]}$ permits the agents to distinguish between more pairs of strings in $L$ than when using $I_i'$. What is perhaps surprising about these results is that we do not require that agent $i$’s communication map $C_{ij}'$ be contained in $O_i'$. That is, there may exist event occurrences that agent $i$ does not observe locally using $O_i'$, yet these event occurrences may be communicated to another agent $j$. The only requirements are that the event occurrences that are communicated be visible to agent $i$ if $i$ decided to use the larger sensor activation map $O_i$, and that the decision to communicate such occurrences be feasible w.r.t. what $i$ observes. This is useful in situations in which agent $i$ may use some mechanism for triggering certain events to be communicated on their occurrence, yet $i$ does not observe these event occurrences directly.
Theorem 15. Let $E_1$, $E_2$ denote alphabets and $L$ denote a formal language where $L$ is prefix-closed and $E(L) \supseteq E_1 \cup E_2$. Let $i, j \in \{1, 2\}$ where $i \neq j$. Let $\overline{O_i} : L \rightarrow 2^E_i$.

$I = I_{(O_i \cup O_j)}$, $O_i' \subseteq O_i$, $C'_i : L \rightarrow 2^{E_i \backslash E_j}$ where $C'_i \subseteq O_i$ and $I'_i = I_{O'_i \cup C'_i}$.

Suppose $\overline{O_i} \nsubseteq I'_i$. (6.1)

Suppose $C'_i \nsubseteq I'_i$. (6.2)

$I \subseteq I'_i$. 

Proof Idea We show by contradiction that there does not exist two strings that are indistinguishable under $I$, but not $I'_i$. It follows symmetrically that there does not exist two strings that are indistinguishable under $I$, but not $I'_2$.

First, we assume there exists $s, t \in L$ such that $I(s) = I(t)$ and $I'_1(s) \neq I'_1(t)$.

There must exist a shortest prefix $s'$ of $s$ and shortest prefix $t'$ of $t$ such that $I(s') = I(t')$ and $I'_1(s') \neq I'_1(t')$. We show that $s'$ is specifically of the form $s''a$ and $t'$ is of the form $t''a$ where $s'', t'' \in L$ and $a \in E(L)$. Further, we show that the occurrences of $a$ following $s''$ and $t''$ are observed when $I$ is used, i.e., $I(s''a) = I(s''a)$ and $I(t''a) = I(t''a)$. However, since $I(s'') = I(t'')$, it cannot be that $I'_1(s') \neq I'_1(t')$, since $s'$ and $t'$ are the shortest such prefixes of $s$ and $t$ that satisfy this. So we demonstrate by contradiction that $I'_1(s'') = I'_1(t'')$. Using some of these facts, we show that $a \notin E_1$ by proving that if $a \in E_1$ then, since $O'_1$ is required to be feasible w.r.t. $I'_1$, $a$ must be in $O'_1(s'')$ and $O'_1(t'')$, which results in $I'_1(s') = I'_1(t')$, a contradiction. So $a$ must be in $E_2 \setminus E_1$. We then assume that $I'_2(s'') = I'_2(t'')$. By feasibility of $C'_2$ w.r.t. $I'_2$, we show that, if $I'_2(s'') = I'_2(t'')$, then $a$ must be in both $C'_2(s'')$ and $C'_2(t'')$, and so $I'_1(s') = I'_1(t')$, a contradiction. So $I'_2(s'') \neq I'_2(t'')$. 


So we have that \( I(s') = I(t') \) and \( I'_2(s') \neq I'_2(t') \). We then use a line of reasoning from this fact which is similar to the line of reasoning that we used from the initial assumption that \( I(s) = I(t) \) and \( I'_1(s) \neq I'_1(t) \) to the fact that \( I(s'') = I(t'') \) and \( I'_2(s'') \neq I'_2(t'') \). Using this line of reasoning, we ultimately show that there exists a strict prefix \( s''' \) of \( s' \) and a prefix \( t''' \) of \( t' \) such that \( I(s''') = I(t''') \) and \( I'_1(s'''') \neq I'_1(t''') \). However, by definition of \( s' \) and \( t' \), there cannot exist such prefixes where at least one of the prefixes is a strict prefix, so a contradiction is reached. This completes the proof that there does not exist \( s, t \in L \) such that \( I(s) = I(t) \) and \( I'_1(s) \neq I'_1(t) \).

**Proof.** (By Contradiction)

1. \( \exists s, t \in L, I(s) = I(t) \land I'_1(s) \neq I'_1(t) \) (Assumption)

1.1. Let \( s', t' \) be the shortest prefixes of \( s \) and \( t \) such that \( I(s') = I(t') \) and \( I'_1(s') \neq I'_1(t') \). That is, (by 1., definition of \( \overline{s} \) and definition of \( \overline{t} \))

\[
\exists s' \in \overline{s}, \exists t' \in \overline{t}, I(s') = I(t') \land I'_1(s') \neq I'_1(t') \land \\
\neg(\exists s'_1 \in \overline{s} \setminus \{s\}, \exists t'_1 \in \overline{t}, I(s'_1) = I(t'_1) \land I'_1(s'_1) \neq I'_1(t'_1)) \land \\
\neg(\exists s'_2 \in \overline{s}, \exists t'_2 \in \overline{t} \setminus \{t'\}, I(s'_2) = I(t'_2) \land I'_1(s'_2) \neq I'_1(t'_2)).
\]

1.2. \( I(s') = \varepsilon \) (Assumption)

1.2.1. \( I'_1(s') = \varepsilon \) (by 1.2. and def’n of \( I, I'_1 \))

1.2.2. \( I(t') = \varepsilon \) (by 1.1. and 1.2.)

1.2.3. \( I'_1(t') = \varepsilon \) (by 1.2.2. and def’n of \( I, I'_1 \))

1.2.4. \( I'_1(s') = I'_1(t') \) (by 1.2.1. and 1.2.3.)

1.2.5. \( I'_1(s') \neq I'_1(t') \) (by 1.1.)

1.2.6. \( \bot \) (by 1.2.4., 1.2.5.)
1.3. \(I(s') \neq \varepsilon\) \hspace{2cm} \text{(proof by contradiction 1.2. and 1.2.6.)}

1.4. \(s' = s''a\) where \(a \in E(L)\) \hspace{2cm} \text{(by 1.3. and def'n of I)}

1.5. \(I(s''a) = I(s'')\) \hspace{2cm} \text{(Assumption)}

1.5.1. \(I'_1(s''a) = I'_1(s'')\) \hspace{2cm} \text{(by 1.5. and def'n of I, I'$_1$)}

1.5.2. \(I'_1(s') = I'_1(s'')\) \hspace{2cm} \text{(by 1.4. and 1.5.1.)}

1.5.3. \(I'_1(s'') \neq I'_1(t')\) \hspace{2cm} \text{(by 1.1. and 1.5.2.)}

1.5.4. \(I(s') = I(s'')\) \hspace{2cm} \text{(by 1.4. and 1.5.)}

1.5.5. \(I(s'') = I(t')\) \hspace{2cm} \text{(by 1.1. and 1.5.4.)}

1.5.6. \(\neg(\exists s'_1 \in \overline{s} \setminus \{s'\}, \exists t'_1 \in \overline{t}, I(s'_1) = I(t'_1) \wedge I'_1(s'_1) \neq I'_1(t'_1))\) \hspace{2cm} \text{(by 1.1.)}

1.5.7. \(s'' \in \overline{s} \setminus \{s'\}\) \hspace{2cm} \text{(by 1.4.)}

1.5.8. \(s'' \in \overline{s} \setminus \{s'\} \wedge I(s'') = I(t') \wedge I'_1(s'') \neq I'_1(t')\) \hspace{2cm} \text{(by 1.5.3., 1.5.5. and 1.5.7.)}

1.5.9. \(\bot\) \hspace{2cm} \text{(by 1.5.6. and 1.5.8.)}

1.6. \(I(s''a) \neq I(s'')\) \hspace{2cm} \text{(proof by contradiction 1.5., 1.5.9.)}

1.7. \(I(s''a) = I(s'')a\) \hspace{2cm} \text{(by 1.6. and def'n of I)}

1.8. \(I(t') \neq \varepsilon\) \hspace{2cm} \text{(by 1.1., 1.3.)}

1.9. \(t' = t''b\) where \(b \in E(L)\) \hspace{2cm} \text{(by 1.8. and def'n of I)}

1.10. \(I(t''b) = I(t'')b\) \hspace{2cm} \text{(by 1.9. symmetry to 1.5., 1.7.)}

1.11. \(I(s') = I(s'')a\) \hspace{2cm} \text{(by 1.7. and 1.4.)}

1.12. \(I(t') = I(t'')b\) \hspace{2cm} \text{(by 1.9. and 1.10.)}

1.13. \(I(s'')a = I(t'')b\) \hspace{2cm} \text{(by 1.1., 1.11., 1.12. and transitivity)}

1.14. \(I(s'') = I(t'')\) \hspace{2cm} \text{(by 1.13.)}
1.15. \( a = b \) \hspace{1cm} (by \ 1.13.)

1.16. \( I'_1(s'') \neq I'_1(t'') \) \hspace{1cm} (Assumption)

1.16.1. \( s'' \in \overline{s} \setminus \{s'\} \land t'' \in \overline{t} \land I(s'') = I(t'') \land I'_1(s'') \neq I'_1(t'') \) \hspace{1cm} (by \ 1.4., 1.9., 1.14., 1.16.)

1.16.2. \( \neg(\exists s'_1 \in \overline{s} \setminus \{s'\}, \exists t'_1 \in \overline{t}, I(s'_1) = I(t'_1) \land I'_1(s'_1) \neq I'_1(t'_1)) \) \hspace{1cm} (by \ 1.1.)

1.16.3. \( \bot \) \hspace{1cm} (by 1.16.1. and 1.16.2.)

1.17. \( I'_1(s'') = I'_1(t'') \) \hspace{1cm} (proof by contradiction 1.16., 1.16.3.)

1.18. \( I'_1(s''a) \neq I'_1(t''b) \) \hspace{1cm} (1.1, 1.4, 1.9.)

1.19. \( I'_1(s''a) = I'_1(s'') \) \hspace{1cm} (Assumption)

1.19.1. \( I'_1(s''a) = I'_1(t'') \) \hspace{1cm} (1.17., 1.19.)

1.19.2. \( I'_1(t''b) = I'_1(t'') \) \hspace{1cm} (Assumption)

1.19.2.1. \( I'_1(s''a) = I'_1(t''b) \) \hspace{1cm} (1.19.1., 1.19.2.)

1.19.2.2. \( \bot \) \hspace{1cm} (1.18., 1.19.2.1.)

1.19.3. \( I'_1(t''b) \neq I'_1(t'') \) \hspace{1cm} (proof by contradiction 1.19.2., 1.19.2.2.)

1.19.4. \( I'_1(t''b) = I'_1(t'')b \) \hspace{1cm} (1.19.3., def’n of \( I'_1 \))

1.20. \( I'_1(s''a) = I'_1(s'') \Rightarrow I'_1(t''b) = I'_1(t'')b \) \hspace{1cm} (\Rightarrow_i, 1.19., 1.19.4.)

1.21. \( a \in E_1 \) \hspace{1cm} (Assumption)

1.21.1. \( I'_1(s''a) = I'_1(s'') \) \hspace{1cm} (Assumption)

1.21.1.1. \( a \notin O'_1(s'') \) \hspace{1cm} (1.21.1., def’n \( I'_1 \))

1.21.1.2. \( I'_1(t''b) = I'_1(t'')b \) \hspace{1cm} (\Rightarrow_e, 1.20., 1.21.1.)

1.21.1.3. \( b \in O'_1(t'') \cup C'_2(t'') \) \hspace{1cm} (1.21.1.2., def’n \( I'_1 \), Def. 5)

1.21.1.4. \( b \in E_1 \) \hspace{1cm} (1.15., 1.21.)
1.21.1.5. \( b \notin C_2'(t'') \) \( \overset{1.21.1.4.}{=} \text{defn } C_2' \)

1.21.1.6. \( b \in O_1'(t'') \) \( \overset{1.21.1.3., 1.21.1.5.}{=} \)

1.21.1.7. \( s''a \in L \land t''a \in L \land I_1'(s'') = I_1'(t'') \Rightarrow (a \in O_1'(s'') \iff a \in O_1'(t'')) \) \( (6.1) \)

1.21.1.8. \( a \in O_1'(s'') \iff a \in O_1'(t'') \) \( \overset{1., 1.1., 1.4., 1.9., 1.15., 1.17., 1.21.1.7.}{=} \)

1.21.1.9. \( a \in O_1'(s'') \iff a \in O_1'(t'') \) \( \overset{1.15., 1.21.1.6., 1.21.1.8.}{=} \)

1.21.1.10. \( \bot \) \( \overset{1.21.1.1., 1.21.1.9.}{=} \)

1.21.2. \( I_1'(s''a) \neq I_1'(s'') \) \( \overset{\text{proof by contradiction}}{=} \)

1.21.3. \( I_1'(s''a) = I_1'(s'')a \) \( \overset{1.21.2. \text{ defn } I_1'}{=} \)

1.21.4. \( I_1'(t''b) = I_1'(t'') \) \( \overset{\text{Assumption}}{=} \)

1.21.4.1. \( b \notin O_1'(t'') \) \( \overset{1.21.4. \text{ defn } I_1'}{=} \)

1.21.4.2. \( a \in O_1'(s'') \cup C_2'(s'') \) \( \overset{1.21.3. \text{ defn } I_1', \text{ Def. } 5}{=} \)

1.21.4.3. \( \bot \) \( \overset{\text{symmetric to } 1.21.1.5., 1.21.1.10.}{=} \)

1.21.5. \( I_1'(t''b) \neq I_1'(t'') \) \( \overset{\text{proof by contradiction}}{=} \)

1.21.6. \( I_1'(t''b) = I_1'(t'')b \) \( \overset{1.21.5. \text{ defn } I_1'}{=} \)

1.21.7. \( I_1'(s''a) = I_1'(t''b) \) \( \overset{1.15., 1.17., 1.21.3., 1.21.6.}{=} \)

1.21.8. \( \bot \) \( \overset{1.18., 1.21.7.}{=} \)

1.22. \( a \notin E_1 \) \( \overset{\text{proof by contradiction}}{=} \)

1.23. \( a \in E_2 \setminus E_1 \) \( \overset{1.7., 1.22. \text{ defn } I}{=} \)

1.24. \( I_2'(s'') = I_2'(t'') \) \( \overset{\text{Assumption}}{=} \)

1.24.1. \( I_1'(s''a) = I_1'(s'') \) \( \overset{\text{Assumption}}{=} \)
1.24.1.1. \( a \not \in C_2'(s'') \quad (\text{1.24.1, def’n } I_1')

1.24.1.2. \( I_1'(t''a) = I_1'(t'')a \quad (\Rightarrow \epsilon, \text{1.20, } \text{1.24.1, just implicitly assuming } b = a \text{ from now on}) \)

1.24.1.3. \( a \in O_1'(t'') \cup C_2'(t'') \quad (\text{1.24.1.2})

1.24.1.4. \( a \in C_2'(t'') \quad (\text{1.23, } \text{1.24.1.3, def’n } O_1', C_2')

1.24.1.5. \( s''a \in L \land t''a \in L \land I_2'(s'') = I_2'(t'') \Rightarrow (a \in C_2''(s'') \iff a \in C_2''(t'')) \quad (6.2)

1.24.1.6. \( a \in C_2'(s'') \iff a \in C_2'(t'') \quad (1, \text{1.1, } \text{1.4, } \text{1.9, } \text{1.15, } \text{1.24, } \text{1.24.1.5, } L = \overline{L})

1.24.1.7. \( a \in C_2'(s'') \quad (\text{1.24.1.4, } \text{1.24.1.6})

1.24.1.8. \bot \quad (\text{1.24.1.1, } \text{1.24.1.7})

1.24.2. \( I_1'(s''a) = I_1'(s'')a \quad (\text{proof by contradiction } \text{1.24.1.1. } \text{1.24.1.8, def’n } I_1')

1.24.3. \( I_1'(t''a) = I_1'(t'') \quad (\text{Assumption})

1.24.3.1. \( a \not \in C_2'(t'') \quad (\text{1.24.3, def’n } I_1')

1.24.3.2. \( a \in O_1'(s'') \cup C_2'(s'') \quad (\text{1.24.2, def’n } I_1', \text{Def. 5})

1.24.3.3. \( a \in C_2'(s'') \quad (\text{1.23, } \text{1.24.3.2, def’n } O_1', C_2')

1.24.3.4. \bot \quad (\text{symmetric to } \text{1.24.1.5, } \text{1.24.1.8})

1.24.4. \( I_1'(t''a) = I_1'(t'')a \quad (\text{proof by contradiction } \text{1.24.3, } \text{1.24.3.4, def’n } I_1')

1.24.5. \( I_1'(s''a) = I_1'(t''a) \quad (\text{1.17, } \text{1.24.2, } \text{1.24.4})

1.24.6. \bot \quad (\text{1.18, } \text{1.24.5})

1.25. \( I_2'(s'') \neq I_2'(t'') \quad (\text{proof by contradiction, } \text{1.24, } \text{1.24.6})

1.26. Let \( s''' \), \( t''' \) be the shortest prefixes of \( s'' \) and \( t'' \) such that \( I(s''') = I(t''') \) and \( I_2'(s''') \neq I_2'(t''') \). That is, (by \text{1.25, definition of } \overline{s'''} \text{ and definition of}
$I(t''') \neq \varepsilon$ (1.14., 1.25., 1.26.) then analogous to 1.2., 1.3.

1.28. $s''' = s'''c$ where $c \in E(L)$ (1.27. and def’n of $I$)

1.29. $I(s'''c) = I(s''')c$ (1.26., 1.28.) then analogous to 1.5., 1.7.

1.30. $I(t''') \neq \varepsilon$ (1.26., 1.27.)

1.31. $t'' = t'''d$ where $d \in E(L)$ (1.30. and def’n $I$)

1.32. $I(t'''d) = I(t'''')d$ (1.31. symmetry to 1.28., 1.29.)

1.33. $I(s''') = I(s''')c$ (1.28., 1.29.)

1.34. $I(t''') = I(t'''')d$ (1.31., 1.32.)

1.35. $I(s'''c) = I(t'''')d$ (1.26., 1.33., 1.34. and transitivity)

1.36. $I(s'''') = I(t'''')$ (1.35.)

1.37. $I_1'(s'''') \neq I_1'(t'''')$ (1.26., 1.36.) then analogous to 1.15., 1.25.

1.38. $s''' \in \overline{s'''} \setminus \{s''\} \land t''' \in \overline{t''} \land I(s'''') = I(t'''') \land I_1'(s'''') \neq I_1'(t'''')$ (by 1.28., 1.31., 1.36., 1.37.)

1.39. $\neg(\exists s'_1 \in \overline{s'}, \exists t'_1 \in \overline{t'}, I(s'_1) = I(t'_1) \land I_1'(s'_1) \neq I_1'(t'_1))$ (by 1.1.)

1.40. $\bot$ (1.38., 1.39.)

2. $\neg(\exists s, t \in L, I(s) = I(t) \land I_1'(s) \neq I_1'(t))$ (proof by contradiction 1.1., 1.40.)

3. $\forall s \in L, \neg(\exists t \in L, I(s) = I(t) \land I_1'(s) \neq I_1'(t))$ (2.)
CHAPTER 6. MONOTONIC OBSERVATION MAPS

4. \( \forall s, t \in L, \neg (I(s) = I(t) \land I'_1(s) \neq I'_1(t)) \) (3.)

5. \( \forall s, t \in L, I(s) \neq I(t) \lor I'_1(s) = I'_1(t) \) (4.)

6. \( \forall s, t \in L, I(s) = I(t) \Rightarrow I'_1(s) = I'_1(t) \) (5. and material implication)

7. \( \forall s, t \in L, I(s) = I(t) \Rightarrow I'_2(s) = I'_2(t) \) (symmetry to 5.)

Next, Theorem 15 is generalized to the case of \( n \) agents.

**Theorem 16.** Let \( E_1, \ldots, E_n \) denote alphabets and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq \bigcup_{k \in \{1, \ldots, n\}} E_k \). Let \( i, j \in \{1, \ldots, n\} \) where \( i \neq j \). Let \( O_i : L \rightarrow 2^{E_i}, I = I_{[O_1 \cup \cdots \cup O_n]}, O'_i \subseteq O_i, C'_{ij} : L \rightarrow 2^{E_i \setminus E_j} \) where \( C'_{ij} \subseteq O_i \) and \( I'_i = I_{[O'_i \cup \bigcup_{k \in \{1, \ldots, n\} \setminus \{i\}} C'_{ik}]} \).

Suppose \( O'_i \not\subseteq I'_i \). \( \text{(6.3)} \)

Suppose \( C'_{ij} \not\subseteq I'_i \). \( \text{(6.4)} \)

**Proof Idea** This proof uses a simple generalization of the proof technique used in Theorem 15. Similar to Theorem 15, first we assume that there exist strings \( s, t \in L \) and some agent \( i_1 \in \{1, \ldots, n\} \) such that \( I(s) = I(t) \) and \( I'_1(s) \neq I'_1(t) \). There must exist a shortest prefix \( s_1 \) of \( s \) and a shortest prefix \( t_1 \) of \( t \) such that \( I(s_1) = I(t_1) \) and \( I'_1(s_1) \neq I'_1(t_1) \). Using the same line of reasoning mentioned in the proof idea of Theorem 15 we show that there exists a prefix \( s'_1 \) of \( s \) and a prefix \( t'_1 \) of \( t \) such that \( I(s'_1) = I(t'_1) \) and \( I'_1(s'_1) \neq I'_1(t'_1) \) for some agent \( i_2 \in \{1, \ldots, n\} \). There must exist a shortest prefix \( s_2 \) of \( s'_1 \) and a shortest prefix \( t_2 \) of \( t'_1 \) such that \( I(s_2) = I(t_2) \) and \( I'_2(s_2) \neq I'_2(t_2) \). From this fact, we can show there exists a prefix \( s'_2 \) of \( s'_1 \) and a prefix
CHAPTER 6. MONOTONIC OBSERVATION MAPS

$t'_2$ of $t'_1$ such that $I(s'_2) = I(t'_2)$ and $I'_{i_3}(s'_2) \neq I'_{i_3}(t'_2)$ for some agent $i_3 \in \{1, \ldots, n\}$. This line of reasoning can be applied any number of times. We find that $n + 1$ times suffices in this proof. By the pigeonhole principle, there must exist an agent $i \in \{1, \ldots, n\}$ and indices $p, q \in \{1, \ldots, n + 1\}$ such that $i = i_p = i_q$ and $p > q$. Then we reach a contradiction since, during application of the line of reasoning $n + 1$ times, we assumed that $s_q$ and $t_q$ were the shortest prefixes such that $I(s_q) = I(t_q)$ and $I'_q(s_q) \neq I'_q(t_q)$.

Proof. (By Contradiction)

1. $\exists i_1 \in \{1, \ldots, n\}, \exists s, t \in L, I(s) = I(t) \land I'_{i_1}(s) \neq I'_{i_1}(t)$ (Assumption)

1.1. Let $s_1, t_1$ be the shortest prefixes of $s$ and $t$ such that $I(s_1) = I(t_1)$ and $I'_{i_1}(s_1) \neq I'_{i_1}(t_1)$. That is, (by 1, definition of $\overline{s}$ and definition of $\overline{t}$)

$$\exists s_1 \in \overline{s}, \exists t_1 \in \overline{t}, I(s_1) = I(t_1) \land I'_{i_1}(s_1) \neq I'_{i_1}(t_1)$$

(L1)

$$\neg (\exists s'_1 \in \overline{s_1} \setminus \{s_1\}, \exists t'_1 \in \overline{t_1}, I(s'_1) = I(t'_1) \land I'_{i_1}(s'_1) \neq I'_{i_1}(t'_1))$$

$$\neg (\exists s'_2 \in \overline{s_1}, \exists t'_2 \in \overline{t_1} \setminus \{t_1\}, I(s'_2) = I(t'_2) \land I'_{i_1}(s'_2) \neq I'_{i_1}(t'_2))$$

1.2. $s_1 = s'_1 a_1 \land t_1 = t'_1 a_1 \land I(s'_1) = I(t'_1) \land (\exists i_2 \in \{1, \ldots, n\}, I'_{i_2}(s'_1) \neq I'_{i_2}(t'_1))$ (by 1.1. and analogy to Theorem 151.1.25.)

1.3. Let $s_2, t_2$ be the shortest prefixes of $s'_1$ and $t'_1$ such that $I(s_2) = I(t_2)$ and $I'_{i_2}(s_2) \neq I'_{i_2}(t_2)$. That is, (by 1.2, definition of $\overline{s'_1}$ and definition of $\overline{t'_1}$)

$$\exists s_2 \in \overline{s'_1}, \exists t_2 \in \overline{t'_1}, I(s_2) = I(t_2) \land I'_{i_2}(s_2) \neq I'_{i_2}(t_2)$$

(L2)

1.4. $s_2 = s'_2 a_2 \land t_2 = t'_2 a_2 \land I(s'_2) = I(t'_2) \land (\exists i_3 \in \{1, \ldots, n\}, I'_{i_3}(s'_2) \neq I'_{i_3}(t'_2))$ (by 1.3. and analogy to Theorem 151.1.25.)
1.5. Let \( s_n, t_n \) be the shortest prefixes of \( s'_{n-1} \) and \( t'_{n-1} \) such that \( I(s_n) = I(t_n) \) and \( I'_{i_n}(s_n) \neq I'_{i_n}(t_n) \). That is,

\[
\exists s_n \in s'_{n-1}, \exists t_n \in t'_{n-1}, I(s_n) = I(t_n) \land I'_{i_n}(s_n) \neq I'_{i_n}(t_n) \land \ldots \quad \text{(Ln)}
\]

1.6. \( s_n = s'_na_n \land t_n = t'_na_n \land I(s_n) = I(t_n) \land (\exists i_{n+1} \in \{1, \ldots, n\}, I'_{i_{n+1}}(s_n) \neq I'_{i_{n+1}}(t_n)) \) (by 1.5. and analogy to Theorem 15 1.–1.25.)

1.7. Let \( s_{n+1}, t_{n+1} \) be the shortest prefixes of \( s'_n \) and \( t'_n \) such that \( I(s_{n+1}) = I(t_{n+1}) \) and \( I'_{i_{n+1}}(s_{n+1}) \neq I'_{i_{n+1}}(t_{n+1}) \). That is,

\[
\exists s_{n+1} \in s'_{n}, \exists t_{n+1} \in t'_{n}, I(s_{n+1}) = I(t_{n+1}) \land I'_{i_{n+1}}(s_{n+1}) \neq I'_{i_{n+1}}(t_{n+1}) \land \ldots \quad \text{(Ln+1)}
\]

1.8. \( \exists p, q \in \{1, \ldots, n+1\}, p > q \land i_p = i_q \) (by the pigeonhole principle and \( \forall r \in \{1, \ldots, n+1\}, i_r \in \{1, \ldots, n\} \))

1.9. \( s_p \in \overline{s_q} \setminus \{s_q\} \) (by 1.8. and definition of \( s_r \) for all \( r \in \{1, \ldots, n+1\} \))

1.10. \( t_p \in \overline{t_q} \) (by 1.8. and definition of \( t_r \) for all \( r \in \{1, \ldots, n+1\} \))

1.11. \( I(s_p) = I(t_p) \) (from (Lp))

1.12. \( I'_{i_p}(s_p) \neq I'_{i_p}(t_p) \) (from (Lp))

1.13. \( I'_{i_q}(s_p) \neq I'_{i_q}(t_p) \) (by 1.12. and \( i_p = i_q \) (from 1.8.))

1.14. \( s_p \in \overline{s_q} \setminus \{s_q\} \land t_p \in i_{q} \land I(s_p) = I(t_p) \land I'_{i_q}(s_p) \neq I'_{i_q}(t_p) \) (\( \land \) introduction, 1.9. – 1.11. and 1.13.)

1.15. \( \neg(\exists s^1_q \in \overline{s_q} \setminus \{s_q\}, \exists t^1_q \in t_q, I(s^1_q) = I(t^1_q) \land I'_{i_q}(s^1_q) \neq I'_{i_q}(t^1_q)) \) (by (Lq))

1.16. \( \bot \) (by 1.14. 1.15.)
2. \( \neg (\exists i_1 \in \{1, \ldots, n\}, \exists s, t \in L, I(s) = I(t) \land I'_i(s) \neq I'_i(t)) \) (proof by contradiction \[1.16\])

3. \( \neg (\exists i \in \{1, \ldots, n\}, \exists s, t \in L, I(s) = I(t) \land I'_i(s) \neq I'_i(t)) \) (2. and renaming \( i_1 \) to \( i \))

4. \( \forall i \in \{1, \ldots, n\}, \neg (\exists s, t \in L, I(s) = I(t) \land I'_i(s) \neq I'_i(t)) \) (by 3.)

5. \( \forall i \in \{1, \ldots, n\}, \forall s, t \in L, I(s) = I(t) \Rightarrow I'_i(s) = I'_i(t) \) (by 4. and analogy to Theorem \[15.2.6\])

Next, Theorem \[18\] is introduced. Theorem \[18\] is a generalization of Wang et al. \[54\] Theorem 1. First, we recall Wang et al. \[54\] Theorem 1 in order to show how Theorem \[18\] is a generalization of Wang et al. \[54\] Theorem 1.

Recall that Wang et al. \[54\] Theorem 1 demonstrates monotonicity of feasible sensor activation maps. That is, w.r.t. feasible sensor activation maps, as more event sensors are activated, more pairs of strings that were previously indistinguishable become distinguishable. Formally,

**Theorem 17** (\[54\] Theorem 1). Let \( E \) denote an alphabet and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E \). Let \( O : L \rightarrow 2^E \) and \( O' \subseteq O \).

Suppose \( O \supseteq I_O \) and \( O' \supseteq I_{O'} \).

\[
I_O \subseteq I_{O'}.
\]

Theorem \[18\] is a generalization in that \( O \) (the larger sensor activation map) is not required to be feasible. Only \( O' \) (the smaller sensor activation map) is required to be feasible.
To generalize further, we permit the agent to observe some occurrences of events for which the agent does not have a sensor (e.g., event occurrences communicated to the agent in question by other agents). We express these observations as an observation map $C$ in Theorem 18. These observations are made independent of which sensor activation map ($O$ or $O'$) is employed.

**Theorem 18.** Let $E$ denote an alphabet and $L$ denote a formal language where $L$ is prefix-closed and $E(L) \supseteq E$. Let $O : L \rightarrow 2^E$, $O' \subseteq O$, $C : L \rightarrow 2^{E(L) \setminus E}$, $I = I_{[O \cup C]}$ and $I' = I_{[O' \cup C]}$.

Suppose $O' \not\subseteq I'$.

\[ I \subseteq I'. \]

**Proof Idea** A proof by contradiction is used. This proof borrows much from the proof of Theorem 15. First, we assume there exists $s, t \in L$ such that $I(s) = I(t)$ and $I'(s) \neq I'(t)$. There must exist a shortest prefix $s'$ of $s$ and shortest prefix $t'$ of $t$ such that $I(s') = I(t')$ and $I'(s') \neq I'(t')$. We show that $s'$ is specifically of the form $s''a$ and $t'$ is of the form $t''a$ where $s'', t'' \in L$ and $a \in E(L)$. Further, we show that the occurrences of $a$ following $s''$ and $t''$ are observed when $I$ is used, i.e., $I(s''a) = I(s'')a$ and $I(t''a) = I(t'')a$. However, since $I(s'') = I(t'')$, it cannot be that $I'(s'') \neq I'(t'')$, since $s'$ and $t'$ are the shortest such prefixes of $s$ and $t$ that satisfy this. So we demonstrate by contradiction that $I'(s'') = I'(t'')$. We show that $a \notin E$ by proving that if $a \in E$ then, since $O'$ is required to be feasible w.r.t. $I'$, $a$ must be in $O'(s'')$ and $O'(t'')$, which results in $I'(s') = I'(t')$, a contradiction. So $a$ must be in $E(L) \setminus E$. Then $a$ must be in $C(s'')$ and $C(t'')$ since $I(s''a) = I(s'')a$ and $I(t''a) = I(t'')a$. Then, by definition of $I'$, we have $I'(s') = I'(t')$, a contradiction. **Proof.** (By Contradiction)
1. ∃s, t ∈ L, I(s) = I(t) ∧ I'(s) ≠ I'(t)  
   (Assumption)

1.1. Let s', t' be the shortest prefixes of s, t such that I(s') = I(t') and I'(s') ≠ I'(t').

   Steps 1.2. - 1.8 follow by 1.1. and analogy to Theorem 15 steps 1.1. - 1.23.

1.2. s' = s''a where a ∈ E(L)

1.3. I(s''a) = I(s'')a

1.4. t' = t''a

1.5. I(t''a) = I(t'')a

1.6. I'(s'') = I'(t'')

1.7. I'(s''a) ≠ I'(t''a)

1.8. a ∈ E(L) \ E

   The remainder is proven from the above.

1.9. a ∈ C(s'')  
   (by 1.3. - 1.8. and def’n of I)

1.10. I'(s''a) = I'(s'')a  
   (by 1.9. and def’n of I’)

1.11. a ∈ C(t'')  
   (by 1.5. - 1.8. and def’n of I)

1.12. I'(t''a) = I'(t'')a  
   (by 1.11. and def’n of I’)

1.13. I'(t''a) = I'(s'')a  
   (by 1.6. - 1.12.)

1.14. I'(t''a) = I'(s''a)  
   (by 1.10. - 1.13.)

1.15. ⊥  
   (by 1.7. - 1.14.)

2. ¬(∃s, t ∈ L, I(s) = I(t) ∧ I'(s) ≠ I'(t))  
   (proof by contradiction 1.1. - 1.15.)
3. \( \forall s, t \in L, I(s) = I(t) \Rightarrow I'(s) = I'(t). \) (by 2 and analogy to Theorem 15 steps 2, 6)

Next we present a result similar to Theorem 18. It is not a generalization of Theorem 18 in that the larger local observation map must be feasible w.r.t. what is observed. However, the restrictions on the communication map are weaker. Instead of requiring that the codomain of the communication map be distinct from the local observation map, we only require that the communication map does not produce observation of the same event occurrences as the local observation map. We conjecture that this result may be used in a solution to Problem 17 of Section 6.4, which is left as an open problem.

**Theorem 19.** Let \( L \) denote a formal language where \( L \) is prefix-closed. Let \( O : L \rightarrow 2^{E(L)} \), \( O' \subseteq O \), \( C : L \rightarrow 2^{E(L)} \) where \( \forall s \in L, C(s) \cap O(s) = \emptyset \), \( I = I_{[O \cup C]} \) and \( I' = I_{[O' \cup C]} \).

Suppose \( O' \nsubseteq I' \). Suppose \( O \nsubseteq I \).

\[
I \subseteq I'.
\]

**Proof Idea** A proof by contradiction is used. This proof is similar to the proof of Theorem 18 and borrows from the proof of Theorem 15. That \( O \) is feasible w.r.t. \( I \) and \( C \) observes different event occurrences than \( O \) is important in this proof.

We assume there exists \( s, t \in L \) such that \( I(s) = I(t) \) and \( I'(s) \neq I'(t) \). There must exist a shortest prefix \( s' \) of \( s \) and shortest prefix \( t' \) of \( t \) such that \( I(s') = I(t') \) and \( I'(s') \neq I'(t') \). We show that \( s' \) is specifically of the form \( s''a \) and \( t' \) is of the form \( t''a \) where \( s'', t'' \in L \) and \( a \in E(L) \). We show that \( I(s''a) = I(s'')a, I(t''a) = I(t'')a, I'(s'') = I'(t'') \) and \( I'(s''a) \neq I'(t''a) \).
We show that $a \not\in O'(s'')$ for otherwise feasibility of $O'$ w.r.t. $I'$ would also require that $a \in O'(t')$, which implies $I'(s'a) = I'(t'a)$, a contradiction. Symmetrically, $a \not\in O'(t'')$. So it must be that $a \in C(s'')$ or (exclusively) $a \in C(t'')$ since either $I'(s''a) = I'(s'')a$ or (exclusively) $I'(t''a) = I'(t'')a$.

Assuming $I'(s''a) = I'(s'')a$, we have $a \in C(s'')$. It cannot also be that $a \in C(t'')$, for otherwise $I'(s''a) = I'(t''a)$. So $a \not\in C(t'')$, and yet $I(t''a) = I(t'')a$. So $a \in O(t'')$. By feasibility of $O$ w.r.t. $I$, we also have that $a \in O(s'')$. However, by definition of $C$, if $a \in C(s'')$ then $a \not\in O(s'')$, so a contradiction is reached. So $I'(s''a) = I'(s'')a$.

Symmetrically, we can argue that $I'(t''a) = I'(t'')a$ using a proof by contradiction from the assumption that $I'(t''a) = I'(t'')a$. Finally, from the above two facts, we have $I'(s''a) = I'(t''a)$, a contradiction.

Proof. (By Contradiction)

1. $\exists s, t \in L, I(s) = I(t) \land I'(s) \neq I'(t)$ (Assumption)

1.1. Let $s', t'$ be the shortest prefixes of $s, t$ such that $I(s') = I(t')$ and $I'(s') \neq I'(t')$.

Steps 1.2–1.9 follow by 1., 1.1 and analogy to Theorem 15 steps 1–1.20.

1.2. $s' = s''a$ where $a \in E(L)$

1.3. $I(s''a) = I(s'')a$

1.4. $t' = t''a$

1.5. $I(t''a) = I(t'')a$

1.6. $I(s'') = I(t'')$

1.7. $I'(s'') = I'(t'')$

1.8. $I'(s''a) \neq I'(t''a)$
1.9. \( I'(s''a) = I'(s'') \Rightarrow I'(t''a) = I'(t'')a \)

The remainder is proven from the above.

1.10. \( I'(t''a) = I'(t'')a \) \hspace{1cm} (Assumption)

1.10.1. \( I'(s''a) = I'(s'')a \) \hspace{1cm} (Assumption)

1.10.1.1. \( I'(s''a) = I'(t'')a \) \hspace{1cm} (1.7, 1.10.1.)

1.10.1.2. \( I'(s''a) = I'(t''a) \) \hspace{1cm} (1.10, 1.10.1.1.)

1.10.1.3. \( \bot \) \hspace{1cm} (1.8, 1.10.1.2.)

1.10.2. \( I'(s''a) \neq I'(s'')a \) \hspace{1cm} (proof by contradiction 1.10.1, 1.10.1.3.)

1.10.3. \( I'(s''a) = I'(s'')\) \hspace{1cm} (1.10.2, def'n \( I' \))

1.11. \( I'(s''a) = I'(s'') \Leftarrow I'(t''a) = I'(t'')a \) \hspace{1cm} (\Leftrightarrow i, 1.10, 1.10.3.)

1.12. \( I'(s''a) = I'(s'') \iff I'(t''a) = I'(t'')a \) \hspace{1cm} (1.9, 1.11.)

1.13. \( I'(s''a) \neq I'(s'') \iff I'(t''a) \neq I'(t'')a \) \hspace{1cm} (1.12.)

1.14. \( I'(s''a) = I'(s'')a \iff I'(t''a) = I'(t'') \) \hspace{1cm} (1.13, def'n \( I' \))

1.15. \( a \in O'(s'') \) \hspace{1cm} (Assumption)

1.15.1. \( a \in O'(s'') \cup C(s'') \) \hspace{1cm} (1.15.)

1.15.2. \( a \in [O' \cup C](s'') \) \hspace{1cm} (1.15.1, Def. 4)

1.15.3. \( I'(s''a) = I'(s'')a \) \hspace{1cm} (1.15.2, def'n \( I' \))

1.15.4. \( I'(t''a) = I'(t'') \) \hspace{1cm} (\Rightarrow e, 1.14, 1.15.3.)

1.15.5. \( a \notin [O' \cup C](t'') \) \hspace{1cm} (1.15.4, def'n \( I' \))

1.15.6. \( a \notin O'(t'') \) \hspace{1cm} (1.15.5, Def. 4)

1.15.7. \( s''a \in L \land t''a \in L \land I'(s'') = I'(t'') \Rightarrow (a \in O'(s'') \iff a \in O'(t'')) \) \hspace{1cm} (\( O' \vDash I' \))
1.15.8. \( s''a \in L \) 
(\( s''a \) prefix of \( s, s \in L \))

1.15.9. \( t''a \in L \) 
(\( t''a \) prefix of \( t, t \in L \))

1.15.10. \( a \in O'(s'') \Leftrightarrow a \in O'(t'') \) 
(\( \Rightarrow \epsilon, 1.15.7, 1.7, 1.15.8, 1.15.9 \).)

1.15.11. \( a \in O''(t'') \) 
(\( \Rightarrow \epsilon, 1.15.10, 1.15 \).)

1.15.12. \( \bot \) 
(\( 1.15.6, 1.15.11 \).)

1.16. \( a \notin O'(s'') \) 
(proof by contradiction \( 1.15, 1.15.12 \).)

1.17. \( a \notin O'(t'') \) 
(symmetric to \( 1.15, 1.16 \).)

1.18. \( I'(s''a) = I'(s'')a \) 
(Assumption)

1.18.1. \( a \in [O' \cup C](s'') \) 
(\( 1.18 \) def’n \( I' \))

1.18.2. \( a \in O'(s'') \cup C(s'') \) 
(\( 1.18.1 \) Def. \( 4 \))

1.18.3. \( a \in C(s'') \) 
(\( 1.16, 1.18.2 \).)

1.18.4. \( a \notin O(s'') \) 
(\( 1.18.3 \) \( \forall u \in L, O(u) \cap C(u) = \emptyset \))

1.18.5. \( I'(t''a) = I'(t'') \) 
(\( \Rightarrow \epsilon, 1.14, 1.18 \).)

1.18.6. \( a \notin [O' \cup C](t'') \) 
(\( 1.18.5 \) def’n \( I \))

1.18.7. \( a \notin C(t'') \) 
(\( 1.18.6 \) Def. \( 4 \))

1.18.8. \( a \in [O \cup C](t'') \) 
(\( 1.5 \) def’n \( I \))

1.18.9. \( a \in O(t'') \cup C(t'') \) 
(\( 1.18.8 \) Def. \( 4 \))

1.18.10. \( a \in O(t'') \) 
(\( 1.18.7, 1.18.9 \).)

1.18.11. \( s''a \in L \land t''a \in L \land I(s'') = I(t'') \Rightarrow (a \in O(s'') \Leftrightarrow a \in O(t'')) \) 
(\( O \forall I \))

1.18.12. \( s''a \in L \) 
(\( s''a \) prefix of \( s, s \in L \))

1.18.13. \( t''a \in L \) 
(\( t''a \) prefix of \( t, t \in L \))

1.18.14. \( a \in O(s'') \Leftrightarrow a \in O(t'') \) 
(\( \Rightarrow \epsilon, 1.18.11, 1.6, 1.18.12, 1.18.13 \).)

1.18.15. \( a \in L \) 
(\( \epsilon \) \( 1.18.10 \).)

1.18.16. \( a \notin O(s'') \) 
(\( 1.18.15 \) \( \forall u \in L, O(u) \cap C(u) = \emptyset \))

1.18.17. \( a \notin O(t'') \) 
(\( 1.18.16 \) \( \Rightarrow \epsilon, 1.14, 1.18 \).)
CHAPTER 6. MONOTONIC OBSERVATION MAPS

1.18.15. \( a \in O(s'') \)  \((\Leftarrow_e, 1.18.10., 1.18.14.)\)

1.18.16. \( \perp \)  \((1.18.4., 1.18.15.)\)

1.19. \( I'(s''a) \neq I'(s'')a \)  \(>(\text{proof by contradiction } 1.18., 1.18.16.)\)

1.20. \( I'(s''a) = I'(s'') \)  \((1.19., \text{def’} n’ I')\)

1.21. \( I'(t''a) = I'(t'') \)  \(>(\text{symmetric to } 1.18., \text{I} 1.20.)\)

1.22. \( I'(s''a) = I'(t'') \)  \((1.7., 1.20.)\)

1.23. \( I'(s''a) = I'(t''a) \)  \((1.21., 1.22.)\)

1.24. \( \perp \)  \((1.8., 1.23.)\)

2. \( \neg (\exists s, t \in L, I(s) = I(t) \land I'(s) \neq I'(t)) \)  \(>(\text{proof by contradiction } 1.1.124.)\)

3. \( \forall s, t \in L, I(s) = I(t) \Rightarrow I'(s) = I'(t) \)  \(>(\text{by } 2. \text{ and analogy to Theorem } 15 \text{ steps } 2.6.)\)

Theorem 20 is similar to Theorem 19. However, it differs in that we consider two agents with their own observation maps, \( O_i \), instead of one agent. We demonstrate that the information map defined by the union of \( O_1 \), \( O_2 \) and \( C \) distinguishes more than information maps defined by the union of just one of \( O_i \) and \( C \). Also, we only require that \( O_i \) be feasible w.r.t. \( I_{[O_i \cup C]} \), and we don’t explicitly require that \( O_1 \cup O_2 \) be feasible w.r.t. \( I_{[O_1 \cup O_2 \cup C]} \). This contrasts with Theorem 19 where the larger observation map must be feasible. However, it turns out that \( O_1 \cup O_2 \) is indeed feasible w.r.t. \( I_{[O_1 \cup O_2 \cup C]} \), which we demonstrate in Theorem 27.

**Theorem 20.** Let \( L \) denote a formal language where \( L \) is prefix-closed. Let \( O_1 : L \to 2^{E(L)} \), \( O_2 : L \to 2^{E(L)} \), \( C : L \to 2^{E(L)} \) where \( \forall s \in L, C(s) \cap O_1(s) = \emptyset \land C(s) \cap O_2(s) = \emptyset \). Let \( I_1 = I_{[O_1 \cup C]} \), \( I_2 = I_{[O_2 \cup C]} \) and \( I = I_{[O_1 \cup O_2 \cup C]} \).
Suppose \( O_1 \subseteq I_1 \) and \( O_2 \subseteq I_2 \).

\[ I \subseteq I_1 \land I \subseteq I_2. \]

**Proof Idea** A proof by contradiction is used. This proof borrows much from the proof of Theorem 15. First, we assume there exists \( s, t \in L \) such that \( I(s) = I(t) \) and \( I_k(s) \neq I_k(t) \) where \( k \) is one of the two agents. There must exist a shortest prefix \( s' \) of \( s \) and shortest prefix \( t' \) of \( t \) such that \( I(s') = I(t') \) and \( I_i(s') \neq I_i(t') \) for some agent \( i \). Further, we can assume without loss of generality that there does not exist any strict prefix \( \hat{s} \) of \( s' \) and prefix \( \hat{t} \) of \( t' \) such that \( I(\hat{s}) = I(\hat{t}) \) and \( I_1(\hat{s}) \neq I_1(\hat{t}) \) or \( I_2(\hat{s}) \neq I_2(\hat{t}) \), for otherwise we can simply use \( \hat{s} \) and \( \hat{t} \) in place of \( s' \) and \( t' \), and adjust \( i \) if necessary.

Like previous proofs in this section, \( s' \) is specifically of the form \( s''a \) and \( t' \) is of the form \( t''a \) where \( s'', t'' \in L \) and \( a \in E(L) \). Further, the occurrences of \( a \) following \( s'' \) and \( t'' \) are observed when \( I \) is used, i.e., \( I(s''a) = I(s'')a \) and \( I(t''a) = I(t'')a \). However, since \( I(s'') = I(t'') \), it cannot be that \( I_i(s'') \neq I_i(t'') \), since \( s' \) and \( t' \) are the shortest such prefixes of \( s \) and \( t \) that satisfy this. So \( I_i(s'') = I_i(t'') \). By our choice of \( s' \) and \( t' \), it must also be that \( I_j(s'') = I_j(t'') \) where \( j \) denotes the other agent (i.e., \( j \neq i \) and \( j \in \{1, 2\} \)).

One can demonstrate by contradiction that \( a \) is not in \( O_i(s'') \) or \( O_i(t'') \) using the fact that \( O_i \) must be feasible w.r.t. \( I_i \). We then prove by contradiction that \( a \) is not observed following \( s'' \) under information map \( I_i \) (i.e., \( I_i(s''a) = I_i(s'') \)). Assume \( I_i(s''a) = I_i(s'')a \). It must be that \( a \in C(s'') \) by definition of \( I_i \) and \( a \notin O_i(s'') \). Since \( a \in C(s'') \), it cannot be that \( a \in O_j(s'') \), since the events to be observed following the same string must be different between \( O_j \) and \( C \) (i.e., \( C(s'') \cap O_j(s'') = \emptyset \)), which is
given in the theorem statement). We have that \( I_i(t''a) = I_i(t'') \) since \( I_i(s''a) = I_i(s'')a \) and \( I_i(s') \neq I_i(t') \). Then \( a \notin C(t'') \) by definition of \( I_i \). Since \( I(t''a) = I(t'')a \), we have that \( a \) must be in at least one of \( O_i(t'') \), \( O_j(t'') \) or \( C(t'') \) by definition of \( I \). Then it must be that \( a \) is in \( O_j(t'') \), since \( a \) is not in \( O_i(t'') \) or \( C(t'') \). By feasibility of \( O_j \) w.r.t. \( I_j \), it must be that \( a \in O_j(s'') \), since \( I_j(s'') = I_j(t'') \), \( s''a \in L \), \( t''a \in L \) and \( a \in O_j(t'') \) This results in a contradiction.

Thus \( a \) is not observed following \( s'' \) under information map \( I_i \). Symmetrically, one can prove that \( a \) is not observed following \( t'' \) under information map \( I_i \). These two facts imply \( I_i(s') = I_i(t') \), a contradiction.

Proof. (By Contradiction)

Note that this proof is very similar to the proof of Theorem 19.

1. \( \exists k \in \{1, 2\}, \exists s, t \in L, I(s) = I(t) \land I_k(s) \neq I_k(t) \) (Assumption)

1.1. Let \( s', t' \) be the shortest prefixes of \( s, t \) such that \( I(s') = I(t') \) and \( I_i(s') \neq I_i(t') \) for some \( i \in \{1, 2\} \).

Steps 1.2.–1.9. follow by 1.1. and analogy to Theorem 15 steps 1.–1.20.

1.2. \( s' = s''a \) where \( a \in E(L) \)

1.3. \( I(s''a) = I(s'')a \)

1.4. \( t' = t''a \)

1.5. \( I(t''a) = I(t'')a \)

1.6. \( I(s'') = I(t'') \)

1.7. \( I_i(s'') = I_i(t'') \)

1.8. \( I_i(s''a) \neq I_i(t''a) \)
1.9. \( I_i(s''a) = I_i(s'') \Rightarrow I_i(t''a) = I_i(t'')a \)

* Let \( j \in \{1, 2\}, j \neq i. \)

1.10. \( I_j(s'') = I_j(t'') \)  \hspace{1em} (by 1.1 and analogy to Theorem 15 steps 1.16, 1.17)

* Steps 1.11, 1.14 follow by 1.10 and analogy to Theorem 19 steps 1.10, 1.17.

1.11. \( I_i(s''a) = I_i(s'') \Leftrightarrow I_i(t''a) = I_i(t'')a \)

1.12. \( I_i(s''a) = I_i(s'')a \Leftrightarrow I_i(t''a) = I_i(t'') \)

1.13. \( a \notin O_i(s'') \)

1.14. \( a \notin O_i(t'') \)

* The remainder is proven from the above.

1.15. \( I_i(s''a) = I_i(s'')a \)  \hspace{1em} (Assumption)

1.15.1. \( a \in [O_i \cup C](s'') \)  \hspace{1em} (1.15, def’n \( I_i \))

1.15.2. \( a \in O_i(s'') \cup C(s'') \)  \hspace{1em} (1.15.1, Def. 4)

1.15.3. \( a \in C(s'') \)  \hspace{1em} (1.13, 1.15.2)

1.15.4. \( a \notin O_j(s'') \)  \hspace{1em} (1.15.3, \( \forall u \in L, O_j(u) \cap C(u) = \emptyset \))

1.15.5. \( I_i(t''a) = I_i(t'') \)  \hspace{1em} (\( \Rightarrow \epsilon \), 1.12, 1.15)

1.15.6. \( a \notin [O_i \cup C](t'') \)  \hspace{1em} (1.15.5, def’n \( I_i \))

1.15.7. \( a \notin C(t'') \)  \hspace{1em} (1.15.6, Def. 4)

1.15.8. \( a \in [[O_1 \cup O_2] \cup C](t'') \)  \hspace{1em} (1.5, def’n \( I \))

1.15.9. \( a \in [O_1 \cup O_2](t'') \cup C(t'') \)  \hspace{1em} (1.15.8, Def. 4)

1.15.10. \( a \in O_1(t'') \cup O_2(t'') \cup C(t'') \)  \hspace{1em} (1.15.9, Def. 4)

1.15.11. \( a \in O_i(t'') \cup O_j(t'') \cup C(t'') \)  \hspace{1em} (1.15.10, i, j \in \{1, 2\}, i \neq j)
1.15.12. \( a \in O_j(t'') \cup C(t'') \) 

1.15.13. \( a \in O_j(t'') \) 

1.15.14. \( s''a \in L \land t''a \in L \land I_j(s'') = I_j(t'') \Rightarrow (a \in O_j(s'') \Leftrightarrow a \in O_j(t'')) \) 

\((O_j \not\supseteq I_j)\)

1.15.15. \( s''a \in L \) 

1.15.16. \( t''a \in L \) 

1.15.17. \( a \in O_j(s'') \Leftrightarrow a \in O_j(t'') \) 

\((\Rightarrow e, 1.15.14., 1.10., 1.15.15., 1.15.16.)\)

1.15.18. \( a \in O_j(s'') \) 

1.15.19. \( \bot \) 

1.16. \( I_i(s''a) \neq I_i(s'')a \) 

(proof by contradiction 1.15.15, 1.15.19)

1.17. \( I_i(s''a) = I_i(s'') \) 

(1.16, def’n \( I_i \))

1.18. \( I_i(t''a) = I_i(t'') \) 

(symmetrical to 1.15.17)

1.19. \( I_i(s''a) = I_i(t'') \) 

(1.7, 1.17)

1.20. \( I_i(s''a) = I_i(t''a) \) 

(1.18, 1.19)

1.21. \( \bot \) 

(1.8, 1.20)

2. \( \neg (\exists k \in \{1, 2\}, \exists s, t \in L, I(s) = I(t) \land I_k(s) \neq I_k(t)) \) 

(proof by contradiction 1.15, 1.21)

3. \( \forall k \in \{1, 2\}, \neg (\exists s, t \in L, I(s) = I(t) \land I_k(s) \neq I_k(t)) \) 

(2)

4. \( \forall k \in \{1, 2\}, \forall s, t \in L, I(s) = I(t) \Rightarrow I_k(s) = I_k(t) \) 

(3 and analogy to Theorem 15 steps 2, 6)

\( \square \)

Theorem 21 is a negative result. It demonstrates that, if agents decide to observe more locally and communicate more event occurrences between one another, then the
agents may not necessarily know more (i.e., the agents may not necessarily be able to
distinguish between more pairs of strings in a system language \( L \)). This is true even
when an agent’s local sensor activations and communications to another agent are
feasible w.r.t. what the agent observes. This contrasts with Wang et al. [54] Theorem
1 where each agent does not communicate anything to another agent, but each agent
just chooses to observe more locally. In this latter, simpler situation, it turns out that
the agent does know more when it chooses to observe more when the agent’s sensor
activations are feasible w.r.t. what the agent observes (Wang et al. [54] Theorem 1).

A result similar to Theorem 21 is demonstrated in the example of Wang [46] Sec-
tion 2.2.6. There it is demonstrated that, as more is communicated between two
agents, it is not necessarily the case that more pairs of strings are distinguished
(Wang [46] Section 2.2.6). However, Theorem 21 is stronger than this result in two
ways. First, we consider observation maps in general for modeling an agent’s local
event observations. In the example of Wang [46] Section 2.2.6, each of the two agents
considered observe all of their local event occurrences. Thus, each agent’s local obser-
vation map degenerates to a natural projection. Second, each agent observation map
considered in Theorem 21 is feasibile w.r.t. what each agent observes. This is not
the case for the example in Wang [46] Section 2.2.6, where one of the agent’s smaller
communication maps is not feasible w.r.t. what the agent observes. However, the
point of Wang [46] Section 2.2.6 was to show that, by removing the communication of
individual event occurrences, it is not necessarily the case that the agents will know
less than before; for their example, feasibility or lack thereof was not relevant.

**Theorem 21.** Let \( E_1, E_2 \) denote alphabets and \( L \) denote a formal language where \( L \)
is prefix-closed and \( E(L) = E_1 \cup E_2 \). Let \( i, j \in \{1, 2\} \) where \( i \neq j \). Let \( O_i : L \to 2^{E_i} \).
CHAPTER 6. MONOTONIC OBSERVATION MAPS

$C_i : L \rightarrow 2^{E_i}$ where $C_i \subseteq O_i$, $I_i = I_{[O_i \cup C_i]}$, $O'_i \subseteq O_i$, $C'_i \subseteq C_i$ and $I'_i = I_{[O'_i \cup C'_i]}$.

Suppose $O_i \not\subset I_i$, $O'_i \not\subset I'_i$, $C_i \not\subset I_i$ and $C'_i \not\subset I'_i$.

It is not necessarily the case that $I_i \subseteq I'_i$.

Proof. (By Example)

Let $E_1 = \{a_1, a_2, a'_1, a'_2, a'_3, c_1, c_2\}$, $E_2 = \{b_1, b_2, b_3, b'_1, b'_2, c_1, c_2\}$. Note that $E_1 \cap E_2 = \{c_1, c_2\}$.

Let $L = \{c_1b_3, c_1a'_3, c_2c_1, a_1b_1a_2b_3, b_1a_1a_2b_2b_3, b'_1a'_1b'_2a'_3, a'_1b'_1b'_2a'_3\}$. An automaton generator of $L$ is depicted in Figure 6.1. Let $O_2 : L \rightarrow 2^{E_2}$ be defined as follows

- $O_2(a_1) = \{b_1\}$;
- $O_2(b_1a_1a_2) = \{b_2\}$;
- $O_2(b_1a_1a_2b_2) = O_2(a_1b_1a_2) = \{b_3\}$;
- $O_2(\epsilon) = \{b_1, b'_1, c_1, c_2\}$;
- $O_2(a'_1) = \{b'_1\}$;
- $O_2(a'_1b'_1) = O_2(b'_1a'_1) = \{b'_2\}$;
- for all other strings $s \in L$, $O_2(s) = \emptyset$.

Observation map $O_2$ is visually represented in Figure 6.1 as follows. For all $\sigma \in E_2$, for all $s\sigma \in L$, $\sigma \in O_2(s)$ if and only if the state reached by $s$ in Figure 6.1 has an outgoing transition labeled by $\sigma$ or $\sigma'$.
Observation map $C_2 : L \rightarrow 2^{E_2 \setminus E_1}$ is represented similarly. For all $\sigma \in E_2 \setminus E_1$, for all $s\sigma \in L$, $\sigma \in C_2(s)$ if and only if the state reached by $s$ in Figure 6.1 has an outgoing transition labeled by $\sigma$.

If $\sigma \in C_1(s)$ then the occurrence of $\sigma$ following $s$ is communicated to agent 2 from agent 1 (i.e., $I_{\{O_2 \cup C_1\}}(s\sigma) = I_{\{O_2 \cup C_1\}}(s)\sigma$). This is visually represented in Figure 6.1 by the state reached by $s$ in Figure 6.1 having an outgoing transition labeled by $\sigma$. 
We suppose $O_1, C_1$ are defined according to the visual representation in Figure 6.2.

The visual representation permits Figure 6.1 to model $I_{[O_2 \cup C_1]}$ and Figure 6.2 to model $I_{[O_1 \cup C_2]}$, with any $\square$ denoting an observed occurrence of $\sigma$ and the colour of the corresponding box indicating which observation map is responsible for the observation as outlined above.

We suppose $O'_2, C'_2$ are defined according to the visual representation in Figure 6.3. We suppose $O'_1, C'_1$ are defined according to the visual representation in Figure 6.4.
One can verify given the visual representation in Figures 6.1–6.4 that \( C_i \subset O_i \), \( O'_i \subset O_i \), \( C'_i \subset O'_i \) and \( C'_i \subset C_i \) for \( i \in \{1, 2\} \). One can verify given these representations that \( O_i, C_i \) are feasible w.r.t. \( I_{[O_i \cup C_j]} \) and \( O'_i, C'_i \) are feasible w.r.t. \( I'_{[O'_i \cup C'_j]} \) for \( i, j \in \{1, 2\}, i \neq j \). \textbf{One can verify that none of} \( O_i, C_i, O'_i \) or \( C'_i \) \textbf{are projections} for \( i \in \{1, 2\} \).

Observe \( \text{I}_{[O_1 \cup C_2]}(a_1b_1a_2b_3) = \text{I}_{[O_1 \cup C_2]}(b_1a_1a_2b_2b_3) = a_1b_3 \) and \( \text{I}_{[O'_1 \cup C'_2]}(b_1a_1a_2b_2b_3) = a_1 \). Symmetrically, observe \( \text{I}_{[O_2 \cup C_1]}(a'_1b'_1b'_2a'_3) = \text{I}_{[O_2 \cup C_1]}(a'_1b'_1b'_2b'_3) = b'_1b'_2a'_3, \text{I}_{[O'_1 \cup C'_2]}(b'_1a'_1b'_2b'_3) = b'_1a'_3 \) and \( \text{I}_{[O'_2 \cup C'_1]}(a'_1b'_1b'_2a'_3) = b'_1 \).

The theorem statement follows.

Theorem 22 demonstrates an interesting property where agent communications are fixed but their local observations can increase. Theorem 22 demonstrates that, if two agents initially decide to communicate all of their local event observations to one another and afterwards they decide to observe extra event occurrences locally (and not communicate these extra occurrences to one another), then they will individually be able to distinguish more pairs of strings in a system language \( L \) than before. We only require that the agents’ communications are feasible w.r.t. what they observed before deciding to observe additional event occurrences locally.

**Theorem 22.** Let \( E_1, E_2 \) denote alphabets and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E_1 \cup E_2 \). Let \( i, j \in \{1, 2\} \) where \( i \neq j \). Let \( O_i : L \to 2^{E_i} \), \( C_i : L \to 2^{E_i \setminus E_j} \) where \( C_i \subseteq O_i \), \( I_i = I_{[O_i \cup C_j]} \) and \( I' = I_{[C_i \cup C_j]} \).

\[
\text{Suppose } C_i \nsubseteq I' \tag{6.5}
\]

\[
I_i \subseteq I'.
\]

**Proof Idea** A proof by contradiction is used. First, we assume there exists \( s, t \in L \) such
that \( I(s) = I(t) \) and \( I_i(s) \neq I_i(t) \) where \( i \) is one of the two agents. As with previous proofs in this chapter, we can obtain the following. There must exist a shortest prefix \( s' \) of \( s \) and shortest prefix \( t' \) of \( t \) such that \( I_i(s') = I_i(t') \) and \( I'(s') \neq I'(t') \). String \( s' = s''a \) and string \( t' = t''a \) where \( s'', t'' \in L \) and \( a \in E(L) \). The occurrence of \( a \) is observed when \( I_i \) is used, i.e., \( I_i(s''a) = I_i(s'')a \) and \( I_i(t''a) = I_i(t'')a \). String \( I_i(s'') \) is equal to string \( I_i(t'') \).

We prove by contradiction that neither \( a \in E_i \) nor \( a \in E_j \). From these two facts it follows that \( I_i(s') = I_i(t') \), which is a contradiction of the initial assumption.

Next we describe how we prove \( a \notin E_i \) (the proof that \( a \notin E_j \) follows symmetrically). Assuming \( a \in E_i \), we make a further assumption that \( a \) is not observed following \( s'' \) under \( I' \). Then \( a \) is not in \( C_i(s'') \) by definition of \( I' \). Since \( a \) is not observed following \( s'' \) under \( I' \), it must be that \( a \) is observed following \( t'' \) under \( I' \), for otherwise \( I'(s') = I'(t') \) and a contradiction is reached. Since \( a \) is observed following \( t'' \) under \( I' \), it must be that \( a \) is in \( C_i(t'') \) or \( C_j(t'') \) where \( j \neq i \). However, since \( a \in E_i \), it cannot be that \( a \in C_j(t'') \) since only events in \( E_j \setminus E_i \) are in \( C_j(t'') \). Thus \( a \in C_i(t'') \). Then, since \( C_i \) is feasible w.r.t. \( I' \), we have that \( a \in C_i(s'') \) and a contradiction is reached. Thus \( a \) must be observed following \( s'' \) under \( I' \). Symmetrically, one can demonstrate that \( a \) must be observed following \( t'' \) under \( I' \) using a symmetric proof by contradiction. Thus \( I'(s') = I'(t') \), a contradiction. Thus, \( a \notin E_i \).

**Proof.** (By Contradiction)

1. \( \exists s, t \in L, I_i(s) = I_i(t) \land I'(s) \neq I'(t) \) (Assumption)

1.1. Let \( s', t' \) be the shortest prefixes of \( s, t \) such that \( I_i(s') = I_i(t') \) and \( I'(s') \neq I'(t') \).

Steps [1.2] [1.8] follow by [1.1] and analogy to Theorem [15] steps [1.1.20].
1.2. \( s' = s''a \) where \( a \in E(L) \)

1.3. \( I_i(s''a) = I_i(s'')a \)

1.4. \( t' = t''a \)

1.5. \( I_i(t''a) = I_i(t'')a \)

1.6. \( I'(s'') = I'(t'') \)

1.7. \( I'(s''a) \neq I'(t''a) \)

1.8. \( I'(s''a) = I'(s'') \Rightarrow I'(t''a) = I'(t'')a \)

The remainder is proven from the above.

1.9. \( a \in E_i \) (Assumption)

1.9.1. \( I'(s''a) = I'(s'') \) (Assumption)

1.9.1.1. \( a \notin C_i(s'') \) (1.9.1. def’n \( I' \))

1.9.1.2. \( I'(t''a) = I'(t'')a \) \( \Rightarrow \) (1.8. 1.9.1.)

1.9.1.3. \( a \in C_i(t'') \cup C_j(t'') \) (1.9.1.2. def’n \( I' \), Def. 5)

1.9.1.4. \( a \notin C_j(t'') \) (1.9. def’n \( C_j \))

1.9.1.5. \( a \in C_i(t'') \) (1.9.1.3. 1.9.1.4.)

1.9.1.6. \( s''a \in L \land t''a \in L \land I'(s'') = I'(t'') \Rightarrow (a \in C_i(s'') \iff a \in C_i(t'')) \)

\((6.5)\)

1.9.1.7. \( a \in C_i(s'') \iff a \in C_i(t'') \) (1.1. 1.2. 1.4. 1.6. 1.9.1.6.)

1.9.1.8. \( a \in C_i(s'') \) \( \iff \) (1.9.1.5. 1.9.1.7.)

1.9.1.9. \( \bot \) (1.9.1.1. 1.9.1.8.)

1.9.2. \( I'(s''a) \neq I'(s'') \) (proof by contradiction 1.9.1. 1.9.1.9.)

1.9.3. \( I'(s''a) = I'(s'')a \) (1.9.2. def’n \( I' \))
1.9.4. \( I'(t''a) = I'(t'') \)  
   (Assumption)

1.9.4.1. \( a \not\in C_i(t'') \)  
   (1.9.4, def'n \( I' \))

1.9.4.2. \( a \in C_i(s'') \cup C_j(s'') \)  
   (1.9.3, def'n \( I' \), Def. 5)

1.9.4.3. \( \perp \)  
   (symmetric to 1.9.1.4.–1.9.1.9.)

1.9.5. \( I'(t''a) \neq I'(t'') \)  
   (proof by contradiction 1.9.4, 1.9.4.3.)

1.9.6. \( I'(t''a) = I'(t'')a \)  
   (1.9.5, def'n \( I' \))

1.9.7. \( I'(s''a) = I'(t''a) \)  
   (1.6, 1.9.3, 1.9.6)

1.9.8. \( \perp \)  
   (1.7, 1.9.7)

1.10. \( a \not\in E_i \)  
   (proof by contradiction 1.9, 1.9.8)

1.11. \( a \in E_j \)  
   (Assumption)

\[ \ldots \]

1.11.1. \( \perp \)  
   (by 1.8, 1.11, and symmetry to 1.9, 1.9.8)

1.12. \( a \not\in E_j \)  
   (proof by contradiction 1.11, 1.11.1)

1.13. \( a \not\in E_i \cup E_j \)  
   (1.10, 1.12)

1.14. \( a \in E_i \cup E_j \)  
   (1.3, def'n \( I \))

1.15. \( \perp \)  
   (by 1.13, 1.14)

2. \( \neg(\exists s, t \in L, I_i(s) = I_i(t) \land I'(s) \neq I'(t)) \)  
   (proof by contradiction 1.14, 1.15)

3. \( \forall s, t \in L, I_i(s) = I_i(t) \Rightarrow I'(s) = I'(t) \).  
   (by 2, and analogy to Theorem 15 steps 2, 6)

Theorem 23 demonstrates a positive special case where two agents can know more by both choosing to observe more locally and by communicating more with one another. Similar to previous Theorems in this section, we require that if an agent
decides to make fewer observations locally, then the agent’s local observations must be feasible w.r.t. what the agent observes. A special requirement that is considered in Theorem 23 is that, if the agents communicate less with one another, then their communications must be feasible w.r.t. only those event observations that are associated with the communications. This requirement is stronger than the requirement of Theorem 21 where the smaller communication maps used by an agent must be feasible w.r.t. what the agent observes. Theorem 23 also differs from Theorem 21 in that we do not impose any restriction on agents when they observe more locally or communicate more with one another.

**Theorem 23.** Let \( E_1, E_2 \) denote alphabets and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E_1 \cup E_2 \). Let \( i, j \in \{1, 2\} \) where \( i \neq j \). Let \( O_i : L \rightarrow 2^{E_i} \), \( C_i : L \rightarrow 2^{E_i \setminus E_j} \) where \( C_i \subseteq O_i \), \( O_i' \subseteq O_i \), \( C_i' \subseteq [O_i' \cap C_i] \), \( I_i = I_{[O_i \cup C_j]} \) and \( I_i' = I_{[O_i' \cup C_j']} \). Suppose \( O_i' \not\subseteq I_i' \).

Suppose \( C_i' \not\subseteq I_i'[C_i' \cup C_j'] \) \hfill (6.6)

\[ I_i \subseteq I_i' . \]

**Proof Idea** A proof by contradiction is used. We assume there exists \( s, t \in L \) such that \( I_i(s) = I_i(t) \) and \( I_i'(s) \neq I_i'(t) \) where \( i \) is one of the two agents. As with previous proofs in this chapter, we can obtain the following. There must exist a shortest prefix \( s' \) of \( s \) and shortest prefix \( t' \) of \( t \) such that \( I_i(s') = I_i(t') \) and \( I_i'(s') \neq I_i'(t') \). String \( s' = s''a \) and string \( t' = t''a \) where \( s'', t'' \in L \) and \( a \in E(L) \). The occurrence of \( a \) is observed when \( I_i \) is used, i.e., \( I_i(s''a) = I_i(s'')a \) and \( I_i(t''a) = I_i(t'')a \). String \( I_i'(s'') \) is equal to string \( I_i'(t'') \). Also, event \( a \) is not in \( E_i \).
CHAPTER 6. MONOTONIC OBSERVATION MAPS

We apply Theorem 22 to show that \( s'' \) appears identical to \( t'' \) under \( I_{C_i \cup C_j} \) since \( I'_i(s'') = I'_i(t'') \). Basically, by Theorem 22, any two strings indistinguishable under \( I'_i \) are also indistinguishable under \( I_{C_i \cup C_j} \). The requirements of Theorem 22 are fulfilled since \( C'_i \) and \( C'_j \) are both feasible w.r.t. \( I_{C_i \cup C_j} \) and \( C_i \subseteq O_i \).

From the important fact that \( I_{[C'_i \cup C'_j]}(s'') = I_{[C'_i \cup C'_j]}(t'') \), we prove by contradiction that \( I'_i(s''a) = I'_i(s')a \) and, symmetrically, \( I'_i(t''a) = I'_i(t')a \). If we assume \( I'_i(s''a) = I'_i(s'') \), then \( a \notin C'_j(s'') \) by definition of \( I'_i \). Also, since \( I'_i(s') \neq I'_i(t') \), we have that \( I'_i(t''a) = I'_i(t')a \) and so \( a \in C'_j(t'') \). However, feasibility of \( C'_j \) w.r.t. \( I_{[C'_i \cup C'_j]} \) and the important fact that \( I_{[C'_i \cup C'_j]}(s'') = I_{[C'_i \cup C'_j]}(t'') \) require that \( a \in C'_j(s'') \) if \( a \in C'_j(t'') \), so a contradiction is reached.

Thus \( I'_i(s''a) = I'_i(s')a \) and, symmetrically, \( I'_i(t''a) = I'_i(t')a \). These two facts imply \( I'_i(s') = I'_i(t') \), a contradiction.

**Proof.** (By Contradiction)

1. \( \exists s, t \in L, I_i(s) = I_i(t) \land I'_i(s) \neq I'_i(t) \) \hspace{1cm} (Assumption)

1.1. Let \( s', t' \) be the shortest prefixes of \( s, t \) such that \( I_i(s') = I_i(t') \) and \( I'_i(s') \neq I'_i(t') \).

Steps 1.2. - 1.8. follow by 1.1. and analogy to Theorem 15 steps 1.1 - 1.22.

1.2. \( s' = s''a \) where \( a \in E(L) \)

1.3. \( I_i(s''a) = I_i(s'')a \)

1.4. \( t' = t''a \)

1.5. \( I_i(t''a) = I_i(t')a \)

1.6. \( I'_i(s'') = I'_i(t'') \)

1.7. \( I'_i(s''a) \neq I'_i(t''a) \)
1.8. \( I_i'(s''a) = I_i'(s'') \Rightarrow I_i'(t''a) = I_i'(t'')a \)

1.9. \( a \notin E_i \)

The remainder is proven from the above.

1.10. \( I_{[C'_i \cup C'_j]}(s'') = I_{[C'_i \cup C'_j]}(t'') \) \( \Box \) def’n \( I_i' \), \( C'_i \subseteq O'_i \), \( 6.6 \) and Theorem 22

1.11. \( I_i'(s''a) = I_i'(s'') \) (Assumption)

1.11.1. \( a \notin C'_j(s'') \) \( \Box \) def’n \( I_i' \)

1.11.2. \( I_i'(t''a) = I_i'(t'')a \) \( \Rightarrow e, 1.8, 1.11. \)

1.11.3. \( a \in O'_i(t'') \cup C'_j(t'') \) \( \Box \) def’n \( I_i' \), Def. 5

1.11.4. \( a \notin O'_i(t'') \) \( \Box \) def’n \( O'_i \)

1.11.5. \( a \in C'_j(t'') \) \( \Box \) def’n \( I_i' \), Def. 5

1.11.6. \( s''a \in L \land t''a \in L \land I_{[C'_i \cup C'_j]}(s'') = I_{[C'_i \cup C'_j]}(t'') \Rightarrow (a \in C'_j(s'') \Leftrightarrow a \in C'_j(t'')) \) \( (6.6) \)

1.11.7. \( a \in C'_j(s'') \Leftrightarrow a \in C'_j(t'') \) \( \Box \) def’n \( I_i' \), Def. 5

1.11.8. \( a \in C'_j(s'') \) \( \Leftrightarrow e, 1.11.5, 1.11.7. \)

1.11.9. \( \perp \) \( \Box \) symmetric to 1.11.4.–1.11.9.

1.12. \( I_i'(s''a) \neq I_i'(s'') \) \( \Box \) proof by contradiction 1.11.1, 1.11.9.

1.13. \( I_i'(s''a) = I_i'(s'')a \) \( \Box \) def’n \( I_i' \)

1.14. \( I_i'(t''a) = I_i'(t'') \) (Assumption)

1.14.1. \( a \notin C'_j(t'') \) \( \Box \) def’n \( I_i' \)

1.14.2. \( a \in O'_i(t'') \cup C'_j(s'') \) \( \Box \) def’n \( I_i' \), Def. 5

1.14.3. \( \perp \) \( \Box \) symmetric to 1.11.4, 1.11.9.

1.15. \( I_i'(t''a) \neq I_i'(t'') \) \( \Box \) proof by contradiction 1.14, 1.14.3.
1.16. \( I_i'(t''a) = I_i'(t''a) \) \hspace{1cm} (1.15. def’n \( I_i' \))

1.17. \( I_i'(s''a) = I_i'(t''a) \) \hspace{1cm} (1.6. 1.13. 1.16.)

1.18. \( \perp \) \hspace{1cm} (1.7. 1.17.)

2. \( \neg(\exists s,t \in L, I_i(s) = I_i(t) \land I_i'(s) \neq I_i'(t)) \) \hspace{1cm} (proof by contradiction 1. 1.18.)

3. \( \forall s,t \in L, I_i(s) = I_i(t) \Rightarrow I_i'(s) = I_i'(t) \). \hspace{1cm} (by 2. and analogy to Theorem 15 steps 2.-6.)

In the remainder of this section we demonstrate some properties of the union of observation maps. These properties are used in Section 6.4 for proving the correctness of a solution to a problem of minimal communication between two agents.

The following demonstrates the simple property that the union of observation maps is associative.

**Lemma 19.** Let \( L \) denote a formal language where \( L \) is prefix-closed and \( A, B, C \subseteq E(L) \). Let \( O_A : L \rightarrow 2^A \), \( O_B : L \rightarrow 2^B \), \( O_C : L \rightarrow 2^C \).

The union of observation maps \( O_A, O_B, O_C \) is associative. That is,

\[
[[O_A \cup O_B] \cup O_C] = [O_A \cup [O_B \cup O_C]].
\]

**Proof.**

1. \( s \in L \) \hspace{1cm} (Assumption)

1.1. \( [[O_A \cup O_B] \cup O_C](s) = [O_A \cup O_B](s) \cup O_C(s) \) \hspace{1cm} (1. Def. 4)

1.2. \( [[O_A \cup O_B] \cup O_C](s) = O_A(s) \cup O_B(s) \cup O_C(s) \) \hspace{1cm} (1.1, Def. 4)

1.3. \( [[O_A \cup O_B] \cup O_C](s) = O_A(s) \cup [O_B \cup O_C](s) \) \hspace{1cm} (1.2, Def. 4)
1.4. $[[O_A \cup O_B] \cup O_C](s) = [O_A \cup [O_B \cup O_C]](s)$ \hspace{1cm} (1.3. Def. [4])

2. $\forall t \in L, [[O_A \cup O_B] \cup O_C](t) = [O_A \cup [O_B \cup O_C]](t)$ \hspace{3cm} (\forall_1, 1., 1.4.) \hfill \Box$

Lemma 20 demonstrates a simple property of observation maps. It demonstrates that if an observation map is feasible w.r.t. a given information map $I'$, then it will be feasible w.r.t. any information map $I$ that distinguishes between strings better than $I'$. This should be obvious, but a simple proof is provided to clarify why this is true.

**Lemma 20.** Let $L$ denote a formal language where $L$ is prefix-closed. Let $O : L \to 2^{E(L)}$ and $I, I'$ denote information maps of $L$ where $I \subseteq I'$.

$$O \subseteq I' \Rightarrow O \subseteq I.$$  

**Proof.**

1. $O \subseteq I'$ (Assumption)

1.1. $\forall s, t \in L, I(s) = I(t) \Rightarrow I'(s) = I'(t)$ \hspace{1cm} ($I \subseteq I'$, Def. [6])

1.2. $O \subseteq I$ (Assumption)

1.2.1. $\exists e \in E(L), \exists u, v \in L, I(u) = I(v) \land e \in O(u) \land e \notin O(v)$ \hspace{1cm} (1.2. Def. [8])

1.2.2. $I(u) = I(v)$ \hspace{1cm} ($\land_e 1.2.1.$)

1.2.3. $I'(u) = I'(v)$ \hspace{1cm} ($\Rightarrow_e 1.2.2, 1.1.$)

1.2.4. $\forall a \in E(L), \forall p, q \in L, I'(p) = I'(q) \Rightarrow (a \in O(p) \Leftrightarrow a \in O(q))$ \hspace{1cm} (1. Def. [8])

1.2.5. $e \in E(L)$ \hspace{1cm} (1.2.1.)

1.2.6. $u, v \in L$ \hspace{1cm} (1.2.1.)
1.2.7. \( e \in \text{O}(u) \Leftrightarrow e \in \text{O}(v) \)  
\( (\Rightarrow e, 1.2.4., 1.2.5., 1.2.6., 1.2.3.) \)

1.2.8. \( e \in \text{O}(u) \)  
(\( \wedge e 1.2.1. \))

1.2.9. \( e \in \text{O}(v) \)  
\( (\Rightarrow e, 1.2.7., 1.2.8.) \)

1.2.10. \( e \notin \text{O}(v) \)  
\( (\wedge e 1.2.1.) \)

1.2.11. \( \bot \)  
\( (1.2.9., 1.2.10.) \)

1.3. \( \text{OFI} \)  
(proof by contradiction 1.2.-1.2.11.)

2. \( \text{OFI}' \Rightarrow \text{OFI} \)  
\( (\Rightarrow e, 1.3.) \)

Theorem 24 demonstrates that feasibility of observation maps is closed under union when we fix the information map used. This result is similar to Wang et al. [54] Theorem 2. However, it differs in that the information map used is fixed, and is not dependent on the particular observation map used.

**Theorem 24.** Let \( E_1, E_2 \) denote alphabets and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E_1 \cup E_2 \). Let \( I \) denote an information map of \( L \) and \( C_i : L \rightarrow 2^{E_i} \) where \( i \in \{1, 2\} \).

\[
(C_1 \text{FI} \land C_2 \text{FI}) \Rightarrow [C_1 \cup C_2] \text{FI}.
\]

**Proof Idea** We show that, for any two strings \( ua \) and \( va \) in \( L \) where \( a \in E(L) \), if \( I(u) = I(v) \) then \( a \in [C_1 \cup C_2](u) \) if and only if \( a \in [C_1 \cup C_2](v) \).

We are given that \( C_1 \text{FI} \) and \( C_2 \text{FI} \). Thus, if \( I(u) = I(v) \) then \( a \) is in \( C_1(u) \) (resp., \( C_2(u) \)) if and only if \( a \) is in \( C_1(v) \) (resp., \( C_2(v) \)). We assume that \( a \in C_1(u) \cup C_2(u) \) and, using the previous two facts, we show that \( a \in C_1(v) \cup C_2(v) \). The reverse direction can also be proven symmetrically. Thus, \( a \in [C_1 \cup C_2](u) \) if and only
if \( a \in [C_1 \cup C_2](v) \) when \( I(u) = I(v) \). By definition of feasibility, it follows that \( [C_1 \cup C_2]F \).

**Proof.**

1. \( C_1F \land C_2F \) (Assumption)

2. \( C_1F \) (\( \land_e 1. \))

3. \( C_2F \) (\( \land_e 1. \))

4. \( \forall e \in E(L), \forall se, te \in L, I(s) = I(t) \Rightarrow (e \in C_1(s) \iff e \in C_1(t)). \) (2. Def. 8)

5. \( \forall e \in E(L), \forall se, te \in L, I(s) = I(t) \Rightarrow (e \in C_2(s) \iff e \in C_2(t)). \) (3. Def. 8)

6. \( a \in E(L) \) (Assumption)

6.1. \( ua \in L \) (Assumption)

6.1.1. \( va \in L \) (Assumption)

6.1.1.1. \( I(u) = I(v) \) (Assumption)

6.1.1.2. \( a \in C_1(u) \iff a \in C_1(v) \) (\( \Rightarrow_e, 4. \) 6. 6.1.1.1.)

6.1.1.3. \( a \in C_2(u) \iff a \in C_2(v) \) (\( \Rightarrow_e, 5. \) 6. 6.1.1.1.)

6.1.1.4. \( a \in C_1(u) \cup C_2(u) \) (Assumption)

6.1.1.5. \( a \in C_1(u) \lor a \in C_2(u) \) (6.1.1.4. def’n \( \cup \))

6.1.1.6. \( a \in C_1(u) \) (Assumption)

6.1.1.7. \( a \in C_1(v) \) (\( \Rightarrow_e, 6.1.1.2., 6.1.1.6. \))

6.1.1.8. \( a \in C_1(v) \lor a \in C_2(v) \) (\( \lor_i, 6.1.1.7. \))

6.1.1.9. \( a \in C_1(v) \cup C_2(v) \) (6.1.1.8. def’n \( \cup \))

6.1.1.10. \( a \in C_1(u) \Rightarrow a \in C_1(v) \cup C_2(v) \) (\( \Rightarrow_i, 6.1.1.6., 6.1.1.9. \))
6.1.1.11. \( a \in C_2(u) \Rightarrow a \in C_1(v) \cup C_2(v) \) (symmetric to 6.1.1.6., 6.1.1.10.)

6.1.1.12. \( a \in C_1(v) \cup C_2(v) \) (\( \forall \), 6.1.1.5., 6.1.1.10., 6.1.1.11.)

6.1.1.13. \( a \in C_1(u) \cup C_2(u) \Rightarrow a \in C_1(v) \cup C_2(v) \) (\( \Rightarrow \), 6.1.1.4., 6.1.1.12.)

6.1.1.14. \( a \in C_1(u) \cup C_2(u) \Leftarrow a \in C_1(v) \cup C_2(v) \) (symmetric to 6.1.1.4.–6.1.1.13.)

6.1.1.15. \( a \in C_1(u) \cup C_2(u) \iff a \in C_1(v) \cup C_2(v) \) (6.1.1.13., 6.1.1.14.)

6.1.1.16. \( a \in [C_1 \cup C_2](u) \iff a \in [C_1 \cup C_2](v) \) (6.1.1.15., Def. 4)

6.1.1.17. \( I(u) = I(v) \Rightarrow (a \in [C_1 \cup C_2](u) \iff a \in [C_1 \cup C_2](v)) \) (\( \Rightarrow \),
6.1.1.11., 6.1.1.16.)

6.1.2. \( \forall a \in L, I(u) = I(v) \Rightarrow (a \in [C_1 \cup C_2](u) \iff a \in [C_1 \cup C_2](v)) \) (\( \forall \),
6.1.1., 6.1.1.17.)

6.2. \( \forall a \in L, \forall a \in L, I(u) = I(v) \Rightarrow (a \in [C_1 \cup C_2](u) \iff a \in [C_1 \cup C_2](v)) \)
(\( \forall \), 6.1., 6.1.2.)

7. \( \forall a \in E(L), \forall a \in L, I(u) = I(v) \Rightarrow (a \in [C_1 \cup C_2](u) \iff a \in [C_1 \cup C_2](v)) \)
(\( \forall \), 6., 6.2.)

8. \([C_1 \cup C_2]FI \) (7. Def. 8) \( \square \)

Theorem 25 demonstrates the opposite direction of Theorem 24. That is, it demonstrates that if the union of two observation maps is feasible w.r.t. a given, fixed information map, then the individual observation maps are each feasible w.r.t. the information map.

**Theorem 25.** Let \( E_1, E_2 \) denote alphabets where \( E_1 \cap E_2 = \emptyset \) and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E_1 \cup E_2 \). Let \( I \) denote an information map of \( L \) and \( C_i : L \rightarrow 2^{E_i} \) where \( i \in \{1, 2\} \).
\( (C_1 \mathbb{F} I \land C_2 \mathbb{F} I) \Leftrightarrow [C_1 \cup C_2] \mathbb{F} I \).

Proof Idea: We show that, for any two strings \( ua \) and \( va \) in \( L \) where \( a \in E(L) \), if \( I(u) = I(v) \) then \( a \in C_1(u) \) if and only if \( a \in C_1(v) \).

We are given that \( [C_1 \cup C_2] \mathbb{F} I \). Thus, if \( I(u) = I(v) \) then \( a \) is in \( [C_1 \cup C_2](u) \) if and only if \( a \) is in \( [C_1 \cup C_2](v) \). We assume that \( a \in C_1(u) \). Then \( a \in C_1(u) \cup C_2(u) \) and so \( a \in C_1(v) \cup C_2(v) \) by feasibility of \( [C_1 \cup C_2] \) w.r.t. \( I \) and \( I(u) = I(v) \). Since \( a \in C_1(u) \), we have \( a \in E_1 \). Also, if \( a \in E_1 \) then \( a \notin E_2 \) by since \( E_1 \cap E_2 = \emptyset \) (given in theorem statement). Thus, \( a \in C_1(v) \) by this fact, \( a \in C_1(v) \cup C_2(v) \) and \( C_2 : L \rightarrow 2^{E_2} \). So, if \( a \in C_1(u) \) then \( a \in C_1(v) \). The reverse direction can be proven symmetrically. Finally, we have that if \( I(u) = I(v) \) then \( a \in C_1(u) \) if and only if \( a \in C_1(v) \). It follows that \( C_1 \mathbb{F} I \).

That \( C_2 \mathbb{F} I \) can be proven symmetric to above.

Proof:

1. \([C_1 \cup C_2] \mathbb{F} I \) \hspace{1cm} (Assumption)

2. \( \forall e \in E(L), \forall s, t \in L, I(s) = I(t) \Rightarrow (e \in [C_1 \cup C_2](s) \Leftrightarrow e \in [C_1 \cup C_2](t)) \) \hspace{1cm} (1. Def. 8)

2.1. \( a \in E(L) \) \hspace{1cm} (Assumption)

2.1.1. \( ua \in L \) \hspace{1cm} (Assumption)

2.1.1.1. \( va \in L \) \hspace{1cm} (Assumption)

2.1.1.2. \( I(u) = I(v) \) \hspace{1cm} (Assumption)

2.1.1.3. \( a \in [C_1 \cup C_2](u) \Leftrightarrow a \in [C_1 \cup C_2](v) \) \hspace{1cm} (\Rightarrow, 2.1, 2.1.1.2.)
2.1.1.4. \( a \in C_1(u) \cup C_2(u) \iff a \in C_1(v) \cup C_2(v) \)  
\( \text{(2.1.1.3., Def. 4)} \)

2.1.1.5. \( a \in C_1(u) \) 
\( \text{(Assumption)} \)

2.1.1.6. \( a \in C_1(u) \cup C_2(u) \)  
\( \text{(2.1.1.5., def'n \cup)} \)

2.1.1.7. \( a \in C_1(v) \cup C_2(v) \)  
\( \Rightarrow \text{e, 2.1.1.4, 2.1.1.6.} \)

2.1.1.8. \( a \in E_1 \)  
\( \text{(2.1.1.5., def'n C_1)} \)

2.1.1.9. \( a \notin E_2 \)  
\( \text{(2.1.1.8, E_1 \cap E_2 = \emptyset)} \)

2.1.1.10. \( a \notin C_2(v) \)  
\( \text{(2.1.1.9, def'n C_2)} \)

2.1.1.11. \( a \in C_1(v) \)  
\( \text{(2.1.1.7, 2.1.1.10.)} \)

2.1.1.12. \( a \in C_1(u) \Rightarrow a \in C_1(v) \)  
\( \Rightarrow i, 2.1.1.5, 2.1.1.11. \)

2.1.1.13. \( a \in C_1(u) \iff a \in C_1(v) \)  
\( \text{symmetric to 2.1.1.5, 2.1.1.12.} \)

2.1.1.14. \( a \in C_1(u) \iff a \in C_1(v) \)  
\( \text{(2.1.1.12, 2.1.1.13.)} \)

2.1.1.15. \( I(u) = I(v) \Rightarrow (a \in C_1(u) \iff a \in C_1(v)) \)  
\( \Rightarrow i, 2.1.1.2, 2.1.1.14. \)

2.1.1.16. \( \forall \nu a \in L, I(u) = I(v) \Rightarrow (a \in C_1(u) \iff a \in C_1(v)) \)  
\( \forall_i, 2.1.1.1, 2.1.1.15. \)

2.1.2. \( \forall u a \in L, \forall v a \in L, I(u) = I(v) \Rightarrow (a \in C_1(u) \iff a \in C_1(v)) \)  
\( \forall_i, 2.1.1, 2.1.1.16. \)

2.2. \( \forall a \in E(L), \forall u a \in L, \forall v a \in L, I(u) = I(v) \Rightarrow (a \in C_1(u) \iff a \in C_1(v)) \)  
\( \forall_i, 2.1, 2.1.2. \)

3. \( C_1 \mathcal{FI} \)  
\( \text{(2.2, Def. 8)} \)

4. \( C_2 \mathcal{FI} \)  
\( \text{(symmetric to 1.3.)} \)
5. \( C_1 \mathbb{F} I \land C_2 \mathbb{F} I \) \( 3, 4 \) □

Theorem 26 demonstrates that feasibility of observation maps is closed under union when the information map considered is derived using the observation map considered. This result is a generalization of [54] Theorem 2. It is more general in that the agent also observes events communicated to it that it cannot observe locally using its observation maps. The communications may originate from some other agent(s), or some other source.

**Theorem 26.** Let \( E_1, E_2 \) denote alphabets and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E_1 \cup E_2 \). Let \( i \in \{1, 2\} \). Let \( O_i : L \rightarrow 2^{E_i} \), \( C : L \rightarrow 2^{E(L) \setminus (E_1 \cup E_2)} \), \( O = [O_1 \cup O_2] \), \( I = I_{O \cup C} \) and \( I_i = I_{O_i \cup C} \).

\[ O_1 \mathbb{F} I_1 \land O_2 \mathbb{F} I_2 \Rightarrow O \mathbb{F} I. \]

**Proof Idea** Follows by proof by contradiction. We assume that \( O_1 \mathbb{F} I_1 \) and \( O_2 \mathbb{F} I_2 \), but \( O \mathbb{F} I \). By \( O \mathbb{F} I \), there must exist \( ue \) and \( ve \) in \( L \) where \( e \in E_1 \cup E_2 \) such that \( I(u) = I(v) \) and \( e \in O(u) \) but \( e \notin O(v) \). Theorem 18 is crucial here. By Theorem 18 that \( I(u) = I(v) \) implies \( I_i(u) = I_i(v) \), where \( i \in \{1, 2\} \). One can verify that the requirements for Theorem 18 are met for this application. If we assume that \( e \in O_1(u) \) then, by feasibility of \( O_1 \) w.r.t. \( I_i \) and \( I_i(u) = I_i(v) \), we have \( e \in O_1(v) \), a contradiction. Symmetrically, it cannot be that \( e \in O_2(u) \). By these two facts and definition of \( O \), we have \( e \notin O(u) \), a contradiction. Thus, \( O \mathbb{F} I \).

**Proof.** (By Contradiction)

1. \( O_1 \mathbb{F} I_1 \land O_2 \mathbb{F} I_2 \). (Assumption)

1.1. \( O \mathbb{F} I \) (Assumption)
1.1.1. \( \exists e \in E_1 \cup E_2, \exists u, v \in L, I(u) = I(v) \land e \in O(u) \land e \notin O(v) \) \hspace{2em} (1.1. Def. 8)

1.1.2. \( I(u) = I(v) \) \hspace{2em} (\wedge e, 1.1.1.)

1.1.3. \( I_i(u) = I_i(v) \) (Theorem[18] \( O_i \subseteq O, O : L \rightarrow 2^{E_1 \cup E_2}, C : L \rightarrow 2^{E(L) \setminus (E_1 \cup E_2)}, 1 \). 1.1.2.)

1.1.4. \( e \in O(u) \) \hspace{2em} (\wedge e, 1.1.1.)

1.1.5. \( e \in O_1(u) \cup O_2(u) \) \hspace{2em} (1.1.4. def’n \( O \))

1.1.6. \( e \in O_1(u) \lor e \in O_2(u) \) \hspace{2em} (1.1.5. def’n \( \cup \))

1.1.7. \( e \in O_1(u) \) \hspace{2em} (Assumption)

1.1.7.1. \( \forall a \in E(L), \forall p, q \in L, I_1(p) = I_1(q) \Rightarrow (a \in O_1(p) \iff a \in O_1(q)) \) \hspace{2em} (1. Def. 8)

1.1.7.2. \( e \in E(L) \) \hspace{2em} (1.1.1. \( E_1 \cup E_2 \subseteq E(L) \))

1.1.7.3. \( u, v \in L \) \hspace{2em} (1.1.1.)

1.1.7.4. \( e \in O_1(u) \iff e \in O_1(v) \) \hspace{2em} (1.1.7.2. 1.1.7.3. 1.1.3. 1.1.7.1.)

1.1.7.5. \( e \in O_1(v) \) \hspace{2em} (\Rightarrow e, 1.1.7.4. 1.1.7.)

1.1.7.6. \( e \in O(v) \) \hspace{2em} (1.1.7.5. def’n \( O \))

1.1.7.7. \( e \notin O(v) \) \hspace{2em} (\wedge e, 1.1.1.)

1.1.7.8. \( \perp \) \hspace{2em} (1.1.7.6. 1.1.7.7.)

1.1.8. \( e \in O_2(u) \) \hspace{2em} (Assumption)

1.1.8.1. \( \perp \) \hspace{2em} (symmetric to 1.1.7. 1.1.7.8.)

1.1.9. \( \perp \) \hspace{2em} (V e, 1.1.6. 1.1.7. 1.1.7.8. 1.1.8. 1.1.8.1.)

1.2. \( \Box \)

2. \( O_1 \Box I_1 \land O_2 \Box I_2 \Rightarrow \Box \) \hspace{2em} (\Rightarrow i, 1.2.)
As mentioned before, the observation map \( O_1 \cup O_2 \) of Theorem 20 is feasible w.r.t. \( I_{[O_1 \cup O_2 \cup C]} \), though we did not explicitly require it in Theorem 20. This is demonstrated in the following, which is a generalization of Theorem 26.

\[ \text{Theorem 27. Let } L \text{ denote a formal language where } L \text{ is prefix-closed. Let } O_1 : L \rightarrow 2^{E(L)} \text{ and } O_2 : L \rightarrow 2^{E(L)} \text{ and } C : L \rightarrow 2^{E(L)} \text{ where } \forall s \in L, C(s) \cap O_1(s) = \emptyset \land C(s) \cap O_2(s) = \emptyset \text{. Let } O = [O_1 \cup O_2], I = I_{[O \cup C]} \text{ and } I_i = I_{[O_i \cup C]} \text{.} \]

\[ O_1 \text{FI}_1 \land O_2 \text{FI}_2 \Rightarrow OFI. \]

\[ \text{Proof. Identical to the proof of Theorem 26 but where Theorem 20 is applied in place of Theorem 18.} \]

6.4 Minimal Observation for Satisfying Feasibility and Specifications

Exploiting the results of Section 6.3, we propose three general problems of minimal observation as well as solutions to two of these problems. The problems are presented in a language-based setting. Algorithms are presented for computing solutions to two of these problems. The results of Section 6.3 may find their application to solving other problems of observation as well.

We note that the proposed problems and algorithms for solving these problems are for demonstration purposes only. There are practical limitations that prohibit application of the algorithms since the problems are defined in a language-based setting (e.g., the algorithms may not terminate). These practical limitations are discussed in
detail later. We note that, if the problems considered are defined over finite, state-based models of systems (e.g., finite-state automata, pushdown automata) instead of languages, then it may be possible to define algorithms for computing solutions to such problems that do not encounter the practical limitations. We leave this as future work, but conjecture that such algorithms could follow the same approach for minimizing observations as the algorithms presented in this section.

Until now in this chapter, we have only considered that observation maps be feasible w.r.t. one information map or another. This requirement alone is not sufficient to permit application of the results. We must impose additional requirements for purposes of applying the results to practical applications. We propose a general requirement, which we refer to as a specification. A specification may be refined for a number of practical scenarios. For purposes of controlling a system, observability [17], a requirement for solving problems of centralized control with partial observation, may be characterized using a specification. More generally, coobservability [24], a requirement for solving problems of decentralized control with partial observation, may be characterized using a specification. For purposes of diagnosing failures in a system, diagnosability [25] or codiagnosability [5] (referred to as “diagnosability under Protocol 3” in Debouk et al. [5]) may be characterized using a specification. These are only some example application areas for which our proposed specification may be used.

Informally, a specification is a predicate defined over a tuple of information maps. However, the predicate is not defined arbitrarily. It is required that, if a specification is not true for a given tuple of information maps \((I_1, \ldots, I_n)\), then it should not be true for any tuple of information maps \((I'_1, \ldots, I'_n)\) where information map \(I'_i\) is not
CHAPTER 6. MONOTONIC OBSERVATION MAPS

better at distinguishing between strings than \( I_i \). Essentially, if the predicate is true for some tuple of information maps, then it should be true when a better tuple of information maps is used. A tuple of information maps is used in this section to model the collective observations of a number of agents.

**Definition 9.** Let \( L \) denote a formal language where \( L \) is prefix-closed and \( n \in \mathbb{Z}^+ \).

Map \( \Phi : (L \rightarrow E(L))^n \rightarrow \{ \text{true}, \text{false} \} \) is a specification of \( L \) if and only if,

\[
\forall I_1, \ldots, I_n : L \rightarrow E(L), \neg \Phi(I_1, \ldots, I_n) \Rightarrow \\
\forall I'_1 \supseteq I_1, \ldots, I'_n \supseteq I_n, \neg \Phi(I'_1, \ldots, I'_n).
\]

The arity of \( \Phi \) is denoted by \( |\Phi| \).

To motivate our use of specifications we show how a generalization of observability [17], a requirement for centralized control of partially-observed DES, may be modeled using a specification when the languages considered are prefix-closed. This example is simple, since it only considers specifications defined over one information map. We are given two prefix-closed languages, \( K \) and \( G \), defined over an alphabet \( \Sigma \). Language \( K \) is a subset of \( G \). We are given sets of events \( \Sigma_c \) and \( \Sigma_o \), which are subsets of \( \Sigma \). Given an information map \( I : G \rightarrow \Sigma_o^* \) (presumably defined from some observation map \( O : G \rightarrow 2^{\Sigma_o} \)), we say that \( K \) is observable w.r.t. \( G, \Sigma_c \) and \( I \) if, for all \( \sigma \in \Sigma_c \), for all \( s, s' \in K \),

\[ s\sigma \in K \land I(s) = I(s') \Rightarrow s'\sigma \notin G \setminus K. \tag{6.7} \]

One can verify that, for information maps \( I, I' \) where \( I \subseteq I' \), if (6.7) does not hold under \( I \), then (6.7) does not hold under \( I' \). Then we can model (6.7) using a specification \( \Phi : (G \rightarrow \Sigma_o^*) \). Specifically, \( \Phi \) would map all \( I : G \rightarrow \Sigma_o^* \) which satisfies (6.7) to \text{true}, and would map any other information maps of type \( G \rightarrow \Sigma^* \) to \text{false}.
Specifications are useful in the context of this chapter as they provide a uniform way for modeling different application-dependant goals (e.g., satisfying observability, coobservability, diagnosability). However, they are only useful in the language-based setting of this chapter, and cannot be computed in general.

Next we introduce observation schemes and containment of observation schemes. Observation schemes are used to formalize the problems to be studied in this section and the solution to one of these problems. An observation scheme is simply a tuple of observation maps. That is, if $O_1, \ldots, O_n$ are observation maps, then $(O_1, \ldots, O_n)$ denotes an observation scheme. Next we define what it means for one observation scheme to be contained in another.

**Definition 10.** Let $L$ denote a formal language where $L$ is prefix-closed, $A_1, \ldots, A_n \subseteq E(L)$ where $n \in \mathbb{N}$, $O_1 : L \to 2^{A_1}$, $O'_1 : L \to 2^{A_1}$, $\ldots$, $O_n : L \to 2^{A_n}$, $O'_n : L \to 2^{A_n}$ be observation maps.

Observation scheme $(O'_1, \ldots, O'_n) \subseteq (O_1, \ldots, O_n)$ if and only if

$$\forall i \in \{1, \ldots, n\}, O'_i \subseteq O_i.$$

Observation scheme $(O'_1, \ldots, O'_n) \subset (O_1, \ldots, O_n)$ if and only if

$$(O'_1, \ldots, O'_n) \subseteq (O_1, \ldots, O_n) \land \exists i \in \{1, \ldots, n\}, O'_i \subset O_i.$$

The first problem we consider is one where we wish to minimize communication between two agents for purposes of satisfying a certain feasibility condition and for satisfying a certain specification. This problem considers the case of two agents. We are given observation maps, $O_i$, denoting each agent’s local observations (i.e., sensor activations). We suppose that these observation maps are projections. That
is, the agents view every occurrence of an event that is locally observable to them. We are also given a specification, $\Phi$. We suppose that if the agents communicate all of their event observations to each other then the specification is satisfied. That is, $\Phi(I_{[O_1 \cup O_2]}, I_{[O_1 \cup O_2]})$ is true. The problem is to determine if we can minimize the communication of event occurrences from one agent to the other while still satisfying the specification and while the communication maps satisfy a certain feasibility condition. The feasibility condition that we consider is that communications should be feasible w.r.t. only those event observations that result from sending or receiving a communication. Recall that this feasibility condition appears in Theorems 22 and 23 where it is used to demonstrate that, for certain observation maps, more precise estimates are obtained as more event occurrences are observed. These results will be used in the solution to this problem.

**Problem 15** (Minimal Communication). Let $E_1$, $E_2$ denote alphabets and $L$ denote a formal language where $L$ is prefix-closed and $E(L) \supseteq E_1 \cup E_2$. Let $i, j \in \{1, 2\}$ where $i \neq j$. Let $O_i : L \rightarrow 2^{E_i}$ be an observation map such that, for all $e \in E_i$, for all $s \in L$, $se \in L$ if and only if $e \in O_i(s)$. Let $\Phi$ be a specification of $L$ where $|\Phi| = 2$.

Suppose $\Phi(I_{[O_1 \cup O_2]}, I_{[O_1 \cup O_2]})$. 

CHAPTER 6. MONOTONIC OBSERVATION MAPS

Compute $C_i : L \rightarrow 2^{E_i \setminus E_j}$, $C_i \subseteq [O_i \setminus O_j]$, $C_j : L \rightarrow 2^{E_j \setminus E_i}$, $C_j \subseteq [O_j \setminus O_i]$ such that

1. $C_1^F I_{[C_1 \cup C_2]} \land C_2^F I_{[C_1 \cup C_2]}$;

2. $\Phi(I_{[O_1 \cup C_2]}, I_{[O_2 \cup C_1]})$;

3. there does not exist $C' \subseteq C_i$, $C' \subseteq C_j$ such that
   3.1. $C'_1^F I_{[C'_1 \cup C'_2]} \land C'_2^F I_{[C'_1 \cup C'_2]}$;
   3.2. $\Phi(I_{[O_1 \cup C'_2]}, I_{[O_2 \cup C'_1]})$.

Suppose that $(C_1, C_2)$ is a solution to Problem 15. We have that the specification is satisfied, i.e., $\Phi(I_{[O_1 \cup C_2]}, I_{[O_2 \cup C_1]})$. We also have that the communication maps are feasible w.r.t. what is communicated, i.e., $C_1^F I_{[C_1 \cup C_2]} \land C_2^F I_{[C_1 \cup C_2]}$. However, one may question whether the communication maps can in fact be used by the agents. That is, can $C_i$ be used by agent $i$ without any confusion if what agent $i$ observes is $I_{[O_i \cup C_j]}$ where $i, j \in \{1, 2\}$, $i \neq j$? In other words, is $C_i^F I_{[O_i \cup C_j]}$? The answer is affirmative, and is demonstrated in Proposition 1. Thus agent $i$ will not be confused as to when it should communicate an observed event occurrence to agent $j$.

Proposition 1. Let $C_1, C_2$ denote a solution to Problem 15.

$$C_1^F I_{[O_1 \cup C_2]} \land C_2^F I_{[O_2 \cup C_1]}.$$  

Proof.

1. $C_1, C_2$ is a solution to Problem 15 (Given)
2. \( C_1 : L \rightarrow 2^{E_1 \setminus E_2} \land C_2 : L \rightarrow 2^{E_2 \setminus E_1} \) \hspace{1cm} (1.)

3. \( C_1 \subseteq [O_1 \setminus O_2] \) \hspace{1cm} (1.)

4. \( C_1 \subseteq O_1 \) \hspace{1cm} (3., 26)

5. \( C_2 \subseteq O_2 \) \hspace{1cm} (symmetric to 3., 4.)

6. \( C_1 F I_{[C_1 \cup C_2]} \land C_2 F I_{[C_1 \cup C_2]} \) \hspace{1cm} (1., Problem 15 Criteria 1.)

7. \( I_{[O_1 \cup O_2]} \subseteq I_{[C_1 \cup C_2]} \land I_{[O_2 \cup O_1]} \subseteq I_{[C_1 \cup C_2]} \) \hspace{1cm} (2., 4., 5., 6., Theorem 22)

8. \( C_1 F I_{[O_1 \cup C_2]} \) \hspace{1cm} (6., 7., Lemma 20)

9. \( C_2 F I_{[O_2 \cup C_1]} \) \hspace{1cm} (6., 7., Lemma 20)

Our solution to Problem 15 relies on reducing Problem 15 to Problem 16, a simpler problem of minimal observation. In this simpler problem, rather than computing a distinct communication map for each agent (i.e., \( C_1 \) and \( C_2 \)), we instead compute a single observation map that is shared by both agents, denoted by \( S \). The objective is to minimize \( S \) while satisfying the same specification as well as a similar feasibility requirement as considered in Problem 15. For practical purposes, map \( S \) is intended to model the union of the agent communication maps from Problem 15 (i.e., \( C_1 \cup C_2 \)). The reduction that we propose later (in Algorithm 4) from Problem 15 to Problem 16 works by decomposing \( S \), a solution to Problem 16, into \( C_1 \) and \( C_2 \), which turns out to be a solution to Problem 15.

**Problem 16.** Let \( E_1, E_2 \) denote alphabets; and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E_1 \cup E_2 \). Let \( i, j \in \{1, 2\} \) where \( i \neq j \). Let \( O_i : L \rightarrow 2^{E_i} \) be an observation map such that, for all \( e \in E_i \), for all \( s \in L \), \( se \in L \) if and only if \( e \in O_i(s) \). Let \( \Phi \) be a specification of \( L \) where \( |\Phi| = 2 \).

Suppose \( \Phi(I_{[O_1 \cup O_2]}, I_{[O_1 \cup O_2]}) \).
CHAPTER 6. MONOTONIC OBSERVATION MAPS

Compute \( S : L \to 2^{(E_1 \setminus E_2) \cup (E_2 \setminus E_1)} \), \( S \subseteq [O_1 \setminus O_2] \cup [O_2 \setminus O_1] \), such that

1. \( S \not\subseteq I_S \);
2. \( \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \);
3. there does not exist \( S' \subset S \) such that
   3.1. \( S' \not\subseteq I_{S'} \);
   3.2. \( \Phi(I_{[O_1 \cup S']}, I_{[O_2 \cup S']}) \).

Definition 11 is necessary for purposes of computing a solution to Problem 16.

Given fixed observation maps \( O \) and \( C \), we define the set of observation maps contained in \( O \) that are feasible w.r.t. themselves and \( C \). This set is denoted by \( \mathcal{O} \) where \( \mathcal{O} = \{ O' \subseteq O : O' \not\subseteq I_{O' \cup C} \} \). Applying Theorem 26, we show that \( \mathcal{O} \) has a largest element, which is precisely the union of all members of \( \mathcal{O} \). We denote this element by \( O^* \) and refer it as the maximum feasible sub-map of \( O \) (w.r.t. \( C \)). Note that, if \( C = L \to \emptyset \) (i.e., if the agent receives no communications from the fixed communication map), then the maximum feasible sub-map of \( O \) is analogous to the maximum feasible sub-policy of Wang et al. [54] Theorem 3. Note that we consider two separate cases for \( C \) of increasing generality. For the second, more general case, the existence of the maximum feasible sub-map of \( O \) is justified using Theorem 27.

**Definition 11.** Let \( E \) denote an alphabet and \( L \) denote a formal language where \( L \) is prefix-closed and \( E(L) \supseteq E \). Let \( O : L \to 2^E \) and \( C : L \to 2^{E(L) \setminus E} \) (more generally, \( C : L \to 2^{E(L)} \) where \( \forall s \in L, C(s) \cap O(s) = \emptyset \)). Let \( \mathcal{O} = \{ O' \subseteq O : O' \not\subseteq I_{O' \cup C} \} \). Let
\( \tau : L \rightarrow \emptyset \). Trivially, we have \( \tau \in I_{[F \cup C]} \). Then \( \tau \in O \), so \( O \) is non-empty.

By Theorem 26 (more generally, Theorem 27), there exists an \( O^* \in O \) such that, for any \( O' \in O \), \( O' \subseteq O^* \). When \( O, C \) are known, \( O^* \) is denoted by \( (O, C)^\uparrow f \). If \( C \) is not stated explicitly then \( C = L \rightarrow \emptyset \). Map \( (O, \emptyset)^\uparrow f \) is denoted by \( O^\uparrow f \).

We propose Algorithm 3 for computing a solution to Problem 16. We prove that Algorithm 3 computes a solution to Problem 16 in Theorem 28. The algorithm works by setting \( S \) initially to \([O_1 \setminus O_2] \cup [O_2 \setminus O_1]\). Then we repeatedly attempt to remove the observation of certain event occurrences in \( S \), compute the maximum feasible sub-map of the resulting map, and test if the specification is satisfied. If so, we proceed. If not, we record the fact that removing observation of the particular event occurrence does not work.

Note that Algorithm 3, though correct in principle, does not yield an effective implementation. The drawbacks of Algorithm 3 are as follows. First, we are removing the observation of individual event occurrences from \( S \), one at a time. If the size of the system language is unbounded, then this algorithm will not terminate. For restricted types of observation maps, effective approaches may be used for removing sets of individual event observations (potentially of unbounded size). For example, for sensor activation policies defined over the transitions of a particular automaton, algorithm Wang et al. 54 MIN-SEN-ACT may be used to remove sets of individual event observations. Second, we do not detail how to compute the maximum feasible sub-map of a given map. For sensor activation policies, algorithm Wang et al. 54 Algorithm MAX-FEA-SUB may be used for computing the maximum feasible sub-map. Computing the maximum feasible sub-map of other types of observation maps is an open problem. Third, we do not detail how the specification \( \Phi \) should be tested.
This is application-dependent.

**Algorithm 3** Compute a Solution to Problem 16

**Input**

- \( \Phi : (L \rightarrow E(L)^*)^2 \rightarrow \{\text{true}, \text{false}\} \)

1: For \( i \in \{1, 2\} \), let \( O_i : L \rightarrow 2^{E_i} \) be an observation map such that, for all \( e \in E_i \), for all \( s \in L \), \( se \in L \) if and only if \( e \in O_i(s) \).

2: Let \( S = [[O_1 \setminus O_2] \cup [O_2 \setminus O_1]] \).

3: Let \( L_S = \emptyset \)

4: while \( \exists e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \exists se \in L \setminus L_S, e \in S(s) \) do

5: Let \( S_{se} : L \rightarrow 2^{(E_1 \setminus E_2) \cup (E_2 \setminus E_1)}, S_{se}(s) = \{e\} \) and for all \( t \in L \setminus \{s\}, S_{se}(t) = \emptyset \).

6: Let \( S' = [S \setminus S_{se}]^f \).

7: if \( \neg \Phi(I_{[O_1 \cup S']}, I_{[O_2 \cup S']}) \) then

8: Let \( L_S = L_S \cup \{se\} \).

9: else

10: Let \( S = S' \).

11: end if

12: end while

13: return \( S \)

**Theorem 28.** Algorithm 3 computes a solution to Problem 16.

**Proof.** This proof utilizes Hoare logic \[8\] to demonstrate the correctness of Algorithm 3. We refer the reader to Tennent \[43\] for an introductory reference on Hoare logic.

1. Let \( C_1 = [O_1 \setminus O_2], C_2 = [O_2 \setminus O_1] \).

2. \( \exists e \in E_1, \exists se \in L, e \in C_2(s) \) (Assumption)

2.1. \( e \in [O_2 \setminus O_1](s) \) (1, 2)

2.2. \( e \in O_2(s) \setminus O_1(s) \) (2.1, Def. 4)

2.3. \( e \notin O_1(s) \) (2.2, def’n \( \setminus \))

2.4. \( se \in L \Leftrightarrow e \in O_1(s) \) (def’n \( O_1, e \in E_1, s \in L \))

2.5. \( se \notin L \) (2.3, 2.4)
CHAPTER 6. MONOTONIC OBSERVATION MAPS

2.6. \( \perp \)

3. \( \forall e \in E_1, \forall s \in L, e \notin C_2(s) \) (proof by contradiction [2.5., 2.6.])

4. \( C_2 : L \rightarrow 2^{E_2} \) (\( C_2 \subseteq O_2, O_2 : L \rightarrow 2^{E_2} \))

5. \( C_1 : L \rightarrow 2^{E_1 \setminus E_2} \) (symmetric to 2.5.)

6. \( I_{[O_1 \cup O_2]} = I_{[[O_1 \cup O_1 \setminus O_2] \cup O_2]} \) ([\( O_1 \setminus O_2 \subseteq O_1 \)])

7. \( I_{[O_1 \cup O_2]} = I_{[[O_1 \cup O_1 \setminus O_2] \cup O_2]} \) (defn \( \cap, \setminus \))

8. \( I_{[O_1 \cup O_2]} = I_{[[O_1 \cup O_1 \setminus O_2] \cup O_2]} \) (Lemma 19)

9. \( I_{[O_1 \cup O_2]} = I_{[[O_1 \cup O_1 \setminus O_2] \cup O_2]} \) (Lemma 19)

10. \( I_{[O_1 \cup O_2]} = I_{[[O_1 \cup O_1 \setminus O_2] \cup O_2]} \) (Lemma 19)

11. \( I_{[O_1 \cup O_2]} = I_{[[O_1 \cup O_1 \setminus O_2] \cup O_2]} \) ([\( O_2 \cap O_1 \subseteq O_1 \)])

12. \( I_{[O_1 \cup O_2]} = I_{[[O_1 \cup O_1 \setminus O_2] \cup O_2]} \) (1.12.)

13. \( I_{[O_1 \cup O_2]} = I_{[O_1 \cup [C_1 \cup C_2]]} \) (symmetry to 7.13.)

14. \( I_{[O_1 \cup O_2]} = I_{[O_1 \cup [C_1 \cup C_2]]} \)

15. \( \Phi(I_{[O_1 \cup [C_1 \cup C_2]]}, I_{[O_1 \cup [C_1 \cup C_2]]}) \) (13, 14, \( \Phi(I_{[O_1 \cup O_2]}, I_{[O_1 \cup O_2]}) \))

16. \( I : L \rightarrow E(L)^* \) (Assumption)

16.1. \( C_1 \Phi I \) (Assumption)

16.1.1. \( \exists e \in E_1 \setminus E_2, \exists s, t \in L, I(s) = I(t) \land e \in C_1(s) \land e \notin C_1(t) \) (6, 16.1, Def. 8)

16.1.2. \( e \in C_1(s) \) (\( \land e \) 16.1.1.)

16.1.3. \( e \in [O_1 \setminus O_2](s) \) (1.16.1.2.)

16.1.4. \( e \in O_1(s) \setminus O_2(s) \) (16.1.3, Def. 4)
16.1.5. $e \in O_1(s)$

16.1.6. $te \in L \iff e \in O_1(t)$ (def’n $O_1$, $e \in E_1$, $t \in L$)

16.1.7. $e \in O_1(t)$ ($\Rightarrow_e$, 16.1.6., $te \in L$)

16.1.8. $e \not\in E_2$ (16.1.1.)

16.1.9. $e \not\in O_2(t)$ (16.1.8., def’n $O_2$)

16.1.10. $e \not\in O_1(t)$ (16.1.7., 16.1.9.)

16.1.11. $e \not\in O_1(t) \setminus O_2(t)$ (16.1.10., Def. 4)

16.1.12. $e \not\in C_1(t)$ (1., 16.1.11.)

16.1.13. $e \not\in C_1(t)$ ($\land_e$, 16.1.1.)


16.2. $C_1FI$ (proof by contradiction 16.1., 16.1.14.)

17. $\forall I : L \to E(L)^*, C_1FI$ ($\forall_i$, 16., 16.2.)

18. $\forall I : L \to E(L)^*, C_2FI$ (symmetric to 16.17.)

19. $C_1FI_{C_1}$ (17. $I_{C_1} : L \to E(L)^*$)

20. $C_2FI_{C_2}$ (18. $I_{C_2} : L \to E(L)^*$)

21. $[C_1 \cup C_2]FI_{[C_1 \cup C_2]}$ (19., 20., Theorem 26)

* Notation: Let $A_{3n}$ denote the state of Algorithm 3 immediately following execution of step $n$ and before further execution ($n \in \{1, \ldots, 12\}$). That proposition $P$ is true at state $A_{3n}$ is denoted by $A_{3n} \models P$.

22. $A_{31} \models O_1, O_2 : L \to 2^{E(L)}$ (Algorithm 3 Step 1)

23. $A_{31} \models [O_1 \setminus O_2], [O_2 \setminus O_1] : L \to 2^{E(L)}$ (22. Def. 4)
24. \( A_{31} \models C_1, C_2 : L \to 2^{E(L)} \)  

25. \( A_{31} \models [C_1 \cup C_2] : L \to 2^{E(L)} \)  

26. \( A_{31} \models \Phi(I_{[O_1 \cup [C_1 \cup C_2]]}, I_{[O_2 \cup [C_1 \cup C_2]]}) \)  

27. \( A_{31} \models [C_1 \cup C_2] \mathcal{F} I_{[C_1 \cup C_2]} \)  

28. \( A_{31} \models [C_1 \cup C_2] : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup [C_1 \cup C_2]]}, I_{[O_2 \cup [C_1 \cup C_2]]}) \land [C_1 \cup C_2] \mathcal{F} I_{[C_1 \cup C_2]} \)  

29. \( A_{31} \models S : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \land \mathcal{S} \mathcal{F} \mathcal{I} S \)  

30. \( A_{31} \models S : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \land \mathcal{S} \mathcal{F} \mathcal{I} S \)  

31. \( A_{31} \models S : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \land \mathcal{S} \mathcal{F} \mathcal{I} S \)  

31.1. \( A_{31} \models S : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \land \mathcal{S} \mathcal{F} \mathcal{I} S \)  

31.2. \( A_{31} \models S_{se} : L \to 2^{E(L)} \)  

31.3. \( A_{31} \models [S \setminus S_{se}] : L \to 2^{E(L)} \)  

31.4. \( A_{31} \models [S \setminus S_{se}]^{\uparrow f} : L \to 2^{E(L)} \)  

31.5. \( \forall T : L \to 2^{E(L)}, T^{\uparrow f} \mathcal{F} \mathcal{I} T^{\uparrow f} \)  

31.6. \( A_{31} \models [S \setminus S_{se}]^{\uparrow f} \mathcal{F} \mathcal{I} [S \setminus S_{se}]^{\uparrow f} \)  

31.7. \( A_{31} \models [S \setminus S_{se}]^{\uparrow f} : L \to 2^{E(L)} \land [S \setminus S_{se}]^{\uparrow f} \mathcal{F} \mathcal{I} [S \setminus S_{se}]^{\uparrow f} \)  

31.8. \( A_{31} \models S : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \land \mathcal{S} \mathcal{F} \mathcal{I} S \)  

31.9. \( A_{31} \models S' : L \to 2^{E(L)} \land S^* \mathcal{F} \mathcal{I} S' \)  

31.10. \( A_{31} \models S : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \land \mathcal{S} \mathcal{F} \mathcal{I} S \)  

31.11. \( A_{31} \models S : L \to 2^{E(L)} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \land \mathcal{S} \mathcal{F} \mathcal{I} S \)  

31.12. \( A_{31} \models \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \) (Algorithm 3 Step 9 holds when Step 7 does not hold)
CHAPTER 6. MONOTONIC OBSERVATION MAPS

31.13. $A^3_9 \models S' : L \to 2^{E(L)} \land S' \mathcal{F} I_{S'}$ \hspace{1cm} (31.9. Algorithm 3 Step 9)

31.14. $A^3_9 \models S' : L \to 2^{E(L)} \land \Phi(I_{O_1 \cup S'}, I_{O_2 \cup S'}) \land S' \mathcal{F} I_{S'}$ \hspace{1cm} ($\land_i$, 31.12., 31.13.)

31.15. $A^3_{10} \models S : L \to 2^{E(L)} \land \Phi(I_{O_1 \cup S'}, I_{O_2 \cup S'}) \land S \mathcal{F} I_{S}$ \hspace{1cm} (31.14. Algorithm 3 Step 10)

31.16. $A^3_{11} \models S : L \to 2^{E(L)} \land \Phi(I_{O_1 \cup S'}, I_{O_2 \cup S'}) \land S \mathcal{F} I_{S}$ \hspace{1cm} if...else...end if of Steps 7–11, def’n of if...else...end if)

Steps 4–12, def’n of while...end while)

32. $A^3_4 \models S : L \to 2^{E(L)} \land \Phi(I_{O_1 \cup S'}, I_{O_2 \cup S'}) \land S \mathcal{F} I_{S} \Rightarrow A^3_{11} \models S : L \to 2^{E(L)} \land \Phi(I_{O_1 \cup S'}, I_{O_2 \cup S'}) \land S \mathcal{F} I_{S}$ \hspace{1cm} ($\Rightarrow_i$, 31., 31.16.)

33. $A^3_{12} \models S : L \to 2^{E(L)} \land \Phi(I_{O_1 \cup S'}, I_{O_2 \cup S'}) \land S \mathcal{F} I_{S}$ \hspace{1cm} while...end while of Steps 4–12, def’n of while...end while)

* Thus, $S$ computed by Algorithm 3 satisfies criterion 1., 2. of Problem 16

34. $I : L \to E(L)^*$ \hspace{1cm} (Assumption)

34.1. $O_1 \mathcal{F} I$ \hspace{1cm} (Assumption)

34.1.1. $\exists e \in E_1, \exists se, te \in L, I(s) = I(t) \land e \in O_1(s) \land e \notin O_1(t)$ \hspace{1cm} (34.1. $O_1 : L \to 2^{E_1}$, Def. 8)

34.1.2. $te \in L \iff e \in O_1(t)$ \hspace{1cm} (def’n $O_1$, $e \in E_1$, $t \in L$)

34.1.3. $e \in O_1(t)$ \hspace{1cm} ($\Rightarrow_e$, 34.1.2., $te \in L$)

34.1.4. $e \notin O_1(t)$ \hspace{1cm} ($\land_e$, 34.1.1.)

34.1.5. $\bot$ \hspace{1cm} (34.1.3., 34.1.4.)

34.2. $O_1 \mathcal{F} I$ \hspace{1cm} (proof by contradiction 34.1., 34.1.5.)

35. $\forall I : L \to E(L)^*, O_1 \mathcal{F} I$ \hspace{1cm} ($\forall_i$, 34., 34.2.)

36. $\forall I : L \to E(L)^*, O_2 \mathcal{F} I$ \hspace{1cm} (symmetric to 34., 35.)

37. $O_1 \mathcal{F} I_{O_1}$ \hspace{1cm} (35. $I_{O_1} : L \to E(L)^*$)
38. $O_2 \mathcal{F} I_{O_2}$

39. Let $S_{12} : L \to 2^{E(L)}$, $A_{3_{12}} \models S_{12} = S$

40. Let $L_{S_{12}} \subseteq L$, $A_{3_{12}} \models L_{S_{12}} = L_S$

41. $\exists t_a \in L_{S_{12}}, \Phi(I_{[O_1 \cup [S_{12} \setminus S_{ta}]^f]} \cup I_{[O_2 \cup [S_{12} \setminus S_{ta}]^f]})$ (Assumption; $S_{ta}$ defined using $t_a$ as $S_{se}$ is defined using $se$ in Algorithm 3 Step 5)

41.1. $S_{12} : L \to 2^{E(L)}$ (39.)

41.2. $S_{ta} : L \to 2^{E(L)}$ (def’n $S_{ta}$; $S_{ta}$ defined using $t_a$ as in Algorithm 3 Step 5)

41.3. $S_{12} \setminus S_{ta} : L \to 2^{E(L)}$ (41.1., 41.1., Def. 4)

41.4. $[S_{12} \setminus S_{ta}]^f : L \to 2^{E(L)}$ (41.3., Def. 11)

41.5. $\forall T : L \to 2^{E(L)}, T^f \mathcal{F} I_T^f$ (Def. 11)

41.6. $[S_{12} \setminus S_{ta}]^f \mathcal{F} I_{[S_{12} \setminus S_{ta}]^f}$ (41.4., 41.5.)

41.7. $[O_1 \cup [S_{12} \setminus S_{ta}]^f] \mathcal{F} I_{[O_1 \cup [S_{12} \setminus S_{ta}]^f]}$ (37., 41.6., Theorem 26)

41.8. $[O_2 \cup [S_{12} \setminus S_{ta}]^f] \mathcal{F} I_{[O_2 \cup [S_{12} \setminus S_{ta}]^f]}$ (38., 41.6., Theorem 26)

41.9. $A_{3_{12}} \models \neg \Phi(I_{[O_1 \cup [S \setminus S_{ta}]^f]} \cup I_{[O_2 \cup [S \setminus S_{ta}]^f]})$ (ta $\in L_{S_{12}}$, step 8 of Algorithm 3 is the only step where elements are added to $L_S$)

41.10. $A_{3_{12}} \models S_{12} \subseteq S$ (From step 8 to termination of Algorithm 3 $S$ changes only via application of step 10. $A_{3_{12}} \models S' \subseteq S$ by step 6, Def. 4 and Def. 11. Then one can show $A_{3_{12}} \models S' \subseteq S$. Thus, after applying step 10, $S$ is smaller than before w.r.t. Def. 3.)

41.11. $A_{3_{12}} \models [S_{12} \setminus S_{ta}] \subseteq [S \setminus S_{ta}]$ (41.10., Def. 4)

41.12. $A_{3_{12}} \models [S_{12} \setminus S_{ta}]^f \subseteq [S \setminus S_{ta}]^f$ (41.11., Def. 11)
41.13. \( A_{37} \models [O_1 \cup [S_{12} \setminus S_{ta}]^{\uparrow}] \subseteq [O_1 \cup [S \setminus S_{ta}]^{\uparrow}] \) (41.12. Def. 4)

41.14. \( A_{37} \models [O_2 \cup [S_{12} \setminus S_{ta}]^{\uparrow}] \subseteq [O_2 \cup [S \setminus S_{ta}]^{\uparrow}] \) (41.12. Def. 4)

41.15. \( A_{37} \models I_{[O_1 \cup [S_{12} \setminus S_{ta}]^{\uparrow}]} \supseteq I_{[O_1 \cup [S \setminus S_{ta}]^{\uparrow}]} \) (41.7, 41.13. Theorem 18)

41.16. \( A_{37} \models I_{[O_2 \cup [S_{12} \setminus S_{ta}]^{\uparrow}]} \supseteq I_{[O_2 \cup [S \setminus S_{ta}]^{\uparrow}]} \) (41.8, 41.14. Theorem 18)

41.17. \( A_{37} \models \neg \Phi(I_{[O_1 \cup [S_{12} \setminus S_{ta}]^{\uparrow}]}), I_{[O_2 \cup [S_{12} \setminus S_{ta}]^{\uparrow}]} \) (41.9, 41.15, 41.16. Def. 9)

41.18. \( \bot \) (41. 41.17.

42. \( \forall ta \in L_{S_{12}}, \neg \Phi(I_{[O_1 \cup [S_{12} \setminus S_{ta}]^{\uparrow}]}, I_{[O_2 \cup [S_{12} \setminus S_{ta}]^{\uparrow}]}) \) (proof by contradiction 41. 41.18.)

43. \( S_{12} \subseteq [[O_1 \setminus O_2] \cup [O_2 \setminus O_1]] \) (A3 \( \models S = [[O_1 \setminus O_2] \cup [O_2 \setminus O_1]] \) and the only operation conducted on \( S \) in the remainder of the algorithm following step 2 is removing elements)

44. \( S_{12} : L \rightarrow 2^{(E_1 \setminus E_2) \cup (E_2 \setminus E_1)} \) (1., 5., 6., 43.,)

45. \( \exists b \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \exists ub \in L, b \in S_{12}(u) \land ub \notin L_{S_{12}} \) (Assumption)

45.1. \( A_{312} \models \forall e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \forall se \in L \setminus L_{S}, e \notin S(s) \) (while...end while

Algorithm 3 steps 4–12, def’n while...end while)

45.2. \( \forall e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \forall se \in L \setminus L_{S_{12}}, e \notin S_{12}(s) \) (39. 45.1.)

45.3. \( b \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1) \) (45.)

45.4. \( ub \in L \setminus L_{S_{12}} \) (45.)

45.5. \( b \notin S_{12}(u) \) (45.2. 45.4.)

45.6. \( \bot \) (45. 45.5.)

46. \( \forall b \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \forall ub \in L, \neg(b \in S_{12}(u) \land ub \notin L_{S_{12}}) \) (proof by contradiction 45. 45.6.)
CHAPTER 6. MONOTONIC OBSERVATION MAPS

47. \( \forall b \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \forall ub \in L, b \in S_{12}(u) \Rightarrow ub \in L_{S_{12}} \) \hspace{1cm} (46. DeMorgan’s and material implication)

48. \( \exists S'_{12} \subset S_{12}, S'_{12} \not\subset I_{S_{12}} \land \Phi(I_{[O_1 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (Assumption)

48.1. \( S'_{12} \subset S_{12} \) \hspace{1cm} (48.)

48.2. \( \exists v \in L, S'_{12}(v) \subset S_{12}(v) \) \hspace{1cm} (44., 48.1., Def. 3)

48.3. \( \exists c \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \exists uc \in L, c \notin S'_{12}(v) \land c \in S_{12}(v) \) \hspace{1cm} (44., 48.2., Def. 2, Def. 3)

48.4. \( S'_{12} \subset S_{12} \subset S_{vc} \) \hspace{1cm} \( S_{vc} \) defined using \( vc \) as in Algorithm 3 Step 5)

48.5. \( S'_{12} \not\subset I_{S_{12}} \) \hspace{1cm} (48.1., 48.3., Def. 11)

48.6. \( S'_{12} \not\subset I_{S_{12}} \) \hspace{1cm} (48.4., 48.5., Def. 11)

48.7. \( \Phi(I_{[O_1 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.7., 48.9., Theorem 18)

48.8. \( \Phi(I_{[O_2 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)

48.9. \( \Phi(I_{[O_2 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)

48.10. \( \Phi(I_{[O_1 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)

48.11. \( \Phi(I_{[O_1 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)

48.12. \( \Phi(I_{[O_2 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)

48.13. \( \Phi(I_{[O_1 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)

48.14. \( \Phi(I_{[O_2 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)

48.15. \( \Phi(I_{[O_1 \cup S'_{12}], I_{[O_2 \cup S'_{12}]}}) \) \hspace{1cm} (48.11., 48.12., Theorem 26)
48.19. \( \Phi(I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}) \) (\( \wedge e, 48. \))

48.20. \( \bot \) (48.18., 48.19.)

49. \( S'_{12} \subset S_{12}, S'_{12} \not\subseteq I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}} \) (proof by contradiction 48., 48.20.)

* Thus, \( S \) computed by Algorithm 3 satisfies criteria 3. of Problem 16. \( \square \)

We propose Algorithm 4 for computing a solution to Problem 15. This algorithm directly applies Algorithm 3 for computing a solution to Problem 16, then decomposes the solution, \( S \), into \( (C_1, C_2) \), a solution to Problem 15.

**Algorithm 4** Compute a Solution to Problem 15

**Input**
- \( \Phi : (L \rightarrow E(L))^2 \rightarrow \{\text{true, false}\} \)

1. For \( i \in \{1, 2\} \), let \( O_i : L \rightarrow 2^{E_i} \) be an observation map such that, for all \( e \in E_i \), for all \( s \in L \), \( se \in L \) if and only if \( e \in O_i(s) \).
2. Compute \( S \) by applying Algorithm 3 on input \( \Phi \).
3. Let \( C_1 = [S \setminus O_2], C_2 = [S \setminus O_1] \).
4. return \( C_1, C_2 \)

We prove in Theorem 29 that Algorithm 4 computes a solution to Problem 15.

**Theorem 29.** Algorithm 4 computes a solution to Problem 15.

**Proof.**

1. \( S : L \rightarrow 2^{(E_1 \setminus E_2) \cup (E_2 \setminus E_1)} \land S \subseteq ([O_1 \setminus O_2] \cup [O_2 \setminus O_1]) \land S \not\subseteq I_{S_{12}} \land \Phi(I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}) \land (\exists S' \subset S, S' \not\subseteq I_{S_{12}} \land \Phi(I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}, I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}) \land (\exists S' \subset S, S' \not\subseteq I_{S_{12}} \land \Phi(I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}, I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}) \land (\exists S' \subset S, S' \not\subseteq I_{S_{12}} \land \Phi(I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}, I_{[O_1 \cup S'_1], I_{[O_2 \cup S'_1]}}) ) \) (Algorithm 4 step 2, Theorem 28)
2. \( C_1 = [S \setminus O_2] \) (Algorithm 4 step 3)
3. \( C_2 = [S \setminus O_1] \) (Algorithm 4 step 3)
4. \( \exists e \in E_1, \exists se \in L, e \in C_2(s) \) (Assumption)
4.1. \( e \in [S \setminus O_1](s) \) \hfill (3, 4.)

4.2. \( e \in S(s) \setminus O_1(s) \) \hfill (4.1, Def. 4)

4.3. \( e \not\in O_1(s) \) \hfill (4.2, def’n \( \) )

4.4. \( se \in L \iff e \in O_1(s) \) \hfill (def’n \( O_1, e \in E_1, s \in L \) )

4.5. \( se \not\in L \) \hfill (4.3, 4.4.)

4.6. \( \perp \) \hfill (4, 4.5.)

5. \( \forall e \in E_1, \forall se \in L, e \not\in C_2(s) \) \hfill (proof by contradiction 4, 4.6.)

6. \( C_2 \subseteq S \) \hfill (3, Def. 4)

7. \( C_2 : L \rightarrow 2^{(E_1 \setminus E_2) \cup (E_2 \setminus E_1)} \) \hfill (1, 6.)

8. \( C_2 : L \rightarrow 2^{E_2 \setminus E_1} \) \hfill (5, 7.)

9. \( C_1 : L \rightarrow 2^{E_1 \setminus E_2} \) \hfill (symmetric to 4.8.)

10. \( \exists e \in (E_1 \setminus E_2), \exists se \in L, e \in S(s) \land e \not\in C_1(s) \) \hfill (Assumption)

10.1. \( e \in (E_1 \setminus E_2) \) \hfill (10.)

10.2. \( e \not\in E_2 \) \hfill (10.1, def’n \( \) )

10.3. \( e \not\in O_2(s) \) \hfill (10.2, \( O_2 : L \rightarrow 2^{E_2} \) )

10.4. \( e \in S(s) \) \hfill (10.)

10.5. \( e \in S(s) \setminus O_2(s) \) \hfill (10.3, 10.4.)

10.6. \( e \in [S \setminus O_2](s) \) \hfill (10.5, Def. 4)

10.7. \( e \in C_1(s) \) \hfill (2, 10.6.)

10.8. \( e \not\in C_1(s) \) \hfill (10.)
10.9. \( \bot \quad \text{[10.7], [10.8.]} \)

11. \( \forall e \in (E_1 \setminus E_2), \forall s \in L, \neg (e \in S(s) \land e \notin C_1(s)) \) \hspace{1cm} \text{(proof by contradiction [10], [10.9.])}

12. \( \forall e \in (E_1 \setminus E_2), \forall s \in L, e \in S(s) \Rightarrow e \in C_1(s) \quad \text{[11], DeMorgan’s and material implication} \)

13. \( \forall e \in (E_2 \setminus E_1), \forall s \in L, e \in S(s) \Rightarrow e \in C_2(s) \) \hspace{1cm} \text{(symmetric to 10., 12.)}

14. \( e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1) \hspace{1cm} \text{(Assumption)} \)

14.1. \( s \in L \hspace{1cm} \text{(Assumption)} \)

14.1.1. \( e \in S(s) \hspace{1cm} \text{(Assumption)} \)

14.1.1.1. \( e \in (E_1 \setminus E_2) \lor e \in (E_2 \setminus E_1) \hspace{1cm} \text{[14], def’n } \cup \)

14.1.1.2. \( e \in (E_1 \setminus E_2) \hspace{1cm} \text{(Assumption)} \)

14.1.1.3. \( e \in S(s) \Rightarrow e \in C_1(s) \hspace{1cm} \text{[12], 14.1., 14.1.1.2.} \)

14.1.1.4. \( e \in C_1(s) \hspace{1cm} \Rightarrow e, 14.1.1., 14.1.1.3. \)

14.1.1.5. \( e \in C_1(s) \lor e \in C_2(s) \hspace{1cm} \text{[\forall_i, 14.1.1.4.]} \)

14.1.1.6. \( e \in C_1(s) \cup e \in C_2(s) \hspace{1cm} \text{[14.1.1.5., def’n } \cup \)

14.1.1.7. \( e \in (E_1 \setminus E_2) \Rightarrow e \in C_1(s) \cup e \in C_2(s) \hspace{1cm} \Rightarrow e, 14.1.1.2., 14.1.1.6. \)

14.1.1.8. \( e \in (E_2 \setminus E_1) \Rightarrow e \in C_1(s) \cup e \in C_2(s) \hspace{1cm} \text{(symmetric to 14.1.1.2., 14.1.1.7.)} \)

14.1.1.9. \( e \in C_1(s) \cup e \in C_2(s) \hspace{1cm} \langle \forall_e, 14.1.1.1., 14.1.1.7., 14.1.1.8. \rangle \)

14.1.2. \( e \in S(s) \Rightarrow e \in C_1(s) \cup e \in C_2(s) \hspace{1cm} \Rightarrow e, 14.1.1., 14.1.1.9. \)

14.2. \( \forall s \in L, e \in S(s) \Rightarrow e \in C_1(s) \cup e \in C_2(s) \hspace{1cm} \langle \forall_i, 14.1., 14.1.2. \rangle \)

15. \( \forall e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \forall s \in L, e \in S(s) \Rightarrow e \in C_1(s) \cup e \in C_2(s) \hspace{1cm} \langle \forall_i, 14.1., 14.2. \rangle \)
16. \( \forall e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1), \forall se \in L, e \in S(s) \Rightarrow e \in [C_1 \cup C_2](s) \) \hspace{1cm} (15. Def. 4)

17. \( \forall e \in E(L), \forall se \in L, e \in S(s) \Rightarrow e \in [C_1 \cup C_2](s) \) \hspace{1cm} (16. \( S : L \rightarrow 2^{(E_1 \setminus E_2) \cup (E_2 \setminus E_1)} \))

18. \( \forall s \in L, S(s) \subseteq [C_1 \cup C_2](s) \) \hspace{1cm} (17.)

19. \( S \subseteq [C_1 \cup C_2] \) \hspace{1cm} (18. Def. 3)

20. \( C_1 \subseteq S \) \hspace{1cm} (2. Def. 4)

21. \( [C_1 \cup C_2] \subseteq S \) \hspace{1cm} (6. 20.)

22. \( [C_1 \cup C_2] = S \) \hspace{1cm} (19. 21.)

23. \( [C_1 \cup C_2] \mathcal{F} I_{[C_1 \cup C_2]} \) 

24. \( C_1 \mathcal{F} I_{[C_1 \cup C_2]} \land C_2 \mathcal{F} I_{[C_1 \cup C_2]} \) \hspace{1cm} \( ((E_1 \setminus E_2) \cap (E_2 \setminus E_1) = \emptyset, \) (8., 9., 23. Theorem 25)

25. \( C_2 \subseteq [(O_1 \setminus O_2) \cup (O_2 \setminus O_1)] \) \hspace{1cm} (1. 6.)

26. \( C_2 \subseteq [O_1 \cup O_2] \) \hspace{1cm} (25. Def. 4)

27. \( C_2 \subseteq O_2 \) \hspace{1cm} (8. 26. \( O_1 : L \rightarrow 2^{E_1} \))

28. \( C_1 \subseteq O_1 \) \hspace{1cm} (symmetric to 25., 27.)

29. \( I_{[O_1 \cup S]} = I_{[O_1 \cup (C_1 \cup C_2)]} \) \hspace{1cm} (22.)

30. \( I_{[O_1 \cup S]} = I_{[O_1 \cup (C_1 \cup C_2)]} \) \hspace{1cm} (29. Lemma 19)

31. \( I_{[O_1 \cup S]} = I_{[O_1 \cup C_2]} \) \hspace{1cm} (28. 30.)

32. \( I_{[O_2 \cup S]} = I_{[O_2 \cup C_1]} \) \hspace{1cm} (symmetric to 29., 31.)

33. \( \Phi(I_{[O_1 \cup C_2]}, I_{[O_2 \cup C_1]}) \) \hspace{1cm} (1. 32.)

* By 24. and 33., \( C_1, C_2 \) computed by Algorithm 4 satisfies criteria 1., 2. of Problem 15.

34. \( \exists C'_1 \subseteq C_1, \exists C'_2 \subseteq C_2, C'_1 \mathcal{F} I_{[C'_1 \cup C_2]} \land C'_2 \mathcal{F} I_{[C'_1 \cup C_2]} \land \Phi(I_{[O_1 \cup C'_2]}, I_{[O_2 \cup C'_1]}) \) \hspace{1cm} (Assumption)
34.1. \([C'_1 \cup C'_2][F_{I[C'_1 \cup C'_2]}] \) (34. Theorem 24)

34.2. \(I_{[O_1 \cup C'_2]} = I_{[O_1 \cup C'_1 \cup C'_2]} \) (28, 34.)

34.3. \(I_{[O_1 \cup C'_2]} = I_{[O_1 \cup C'_1 \cup C'_2]} \) (34.2. Lemma 19)

34.4. \(I_{[O_2 \cup C'_2]} = I_{[O_2 \cup C'_1 \cup C'_2]} \) (symmetric to 34.2.–34.3.)

34.5. \(\Phi(I_{[O_1 \cup C'_1 \cup C'_2]}), I_{[O_2 \cup C'_1 \cup C'_2]} \) (34., 34.3., 34.4.)

34.6. \([C'_1 \cup C'_2] \subset [C_1 \cup C_2] \) (34.1, def'n 4)

34.7. \([C'_1 \cup C'_2] \subset S \) (22, 34.6.)

34.8. \(\not\exists S' \subset S, S' \cap F_{I[S]} \land \Phi(I_{[O_1 \cup S]}, I_{[O_2 \cup S]}) \) (\&e, 1.)

34.9. \([C'_1 \cup C'_2] \subset S \land [C'_1 \cup C'_2][F_{I[C'_1 \cup C'_2]}] \land \Phi(I_{[O_1 \cup C'_1 \cup C'_2]}), I_{[O_2 \cup C'_1 \cup C'_2]} \) (\&i, 34.1, 34.5., 34.6.)

34.10. \(\bot \) (34.8, 34.9.)

35. \(\neg(\exists C'_1 \subset C_1, \exists C'_2 \subset C_2, C'_1[I[C'_1 \cup C'_2]] \land C'_2[I[C'_1 \cup C'_2]], \Phi(I_{[O_1 \cup C'_2]}, I_{[O_2 \cup C'_2]}) \) (proof by contradiction 34., 34.10.)

36. \(\neg(\exists C'_1 \subset C_1, \exists C'_2 \subset C_2, C'_1[I[C'_1 \cup C'_2]] \land C'_2[I[C'_1 \cup C'_2]], \Phi(I_{[O_1 \cup C'_2]}, I_{[O_2 \cup C'_2]}) \) (symmetric to 34., 35.)

* By 35. and 36, \(C_1, C_2 \) computed by Algorithm 4 satisfies criteria 3 of Problem 15.

Algorithm 4 and Theorem 29 demonstrate a reduction from Problem 15 to Problem 16. Also, Problem 16 is reducible to Problem 15. That is, the two problems are computationally equivalent, as demonstrated in Theorem 30.

**Theorem 30.** Problem 15 and Problem 16 are computationally equivalent. That is,
Proof.

(i) Follows directly by Theorem 29 and step 2 of Algorithm 4.

(ii)

1. Let $C_1, C_2$ be computed by applying Algorithm 4 on input $\phi$.

2. $C_1, C_2$ is a solution to Problem 15. (Theorem 29)

3. Let $S$ be computed by step 2 of Algorithm 4 on input $\phi$.

4. $S$ is a solution to Problem 16. (Theorem 28)

5. $[C_1 \cup C_2] = S$ (Theorem 29 steps 1–22.)

6. $[C_1 \cup C_2]$ is a solution to Problem 16 (4., 5.)

* Thus, Problem 16 is reducible to Problem 15.

Next we propose the following extension of Problem 15. Problem 15 considered minimizing communications while satisfying feasibility (w.r.t. what is communicated) and a specification. Problem 17 takes this a step further by also requiring that an agent’s local observations be minimized. We still require that communications be feasible and the specification be satisfied, but now we also require that an agent’s local observations be feasible w.r.t. what the agent observes.

**Problem 17** (Minimal Communication & Sensor Activation). Let $E_1, E_2$ denote alphabets and $L$ denote a formal language where $L$ is prefix-closed and $E(L) \supseteq E_1 \cup E_2$. Let $i, j \in \{1, 2\}$ where $i \neq j$. Let $O_i^{alt} : L \rightarrow 2^{E_i}$ be an observation map such that, for
all \( e \in E_i \), for all \( s \in L \), se \( \in L \) if and only if \( e \in O_i^{all}(s) \). Let \( \Phi \) be a specification of \( L \) where \( |\Phi| = 2 \).

Suppose \( \Phi(I_{[O_1^{all} \cup O_2^{all}]}), I_{[O_1^{all} \cup O_2^{all}]})) \).

Compute \( O_1 : L \rightarrow 2^{E_1}, O_1 \subseteq O_1^{all}, O_2 : L \rightarrow 2^{E_2}, O_2 \subseteq O_2^{all}, C_1 : L \rightarrow 2^{E_1 \setminus E_2}, C_1 \subseteq [O_1 \setminus O_2], C_2 : L \rightarrow 2^{E_2 \setminus E_1}, C_2 \subseteq [O_2 \setminus O_1] \) such that

1. \( C_1 \mathbb{F} I_{[C_1 \cup C_2]} \wedge C_2 \mathbb{F} I_{[C_1 \cup C_2]} \);
2. \( O_1 \mathbb{F} I_{[O_1 \cup C_2]} \wedge O_2 \mathbb{F} I_{[O_2 \cup C_1]} \);
3. \( \Phi(I_{[O_1 \cup C_2]}, I_{[O_2 \cup C_1]}) \);
4. there does not exist observation scheme \( (O'_1, C'_1, O'_2, C'_2) \subset (O_1, C_1, O_2, C_2) \) where
   4.1. \( C'_1 \subseteq [O'_1 \setminus O'_2] \wedge C'_2 \subseteq [O'_2 \setminus O'_1] \);
   4.2. \( C'_1 \mathbb{F} I_{[C'_1 \cup C'_2]} \wedge C'_2 \mathbb{F} I_{[C'_1 \cup C'_2]} \);
   4.3. \( O'_1 \mathbb{F} I_{[O'_1 \cup C'_2]} \wedge O'_2 \mathbb{F} I_{[O'_2 \cup C'_1]} \);
   4.4. \( \Phi(I_{[O'_1 \cup C'_2]}, I_{[O'_2 \cup C'_1]}) \).

We leave Problem 17 as an open problem. However, we conjecture that Algorithm 5 computes a solution to Problem 17. We conjecture that the results of Section 6.3 may be used to demonstrate that Algorithm 5 is correct. We conjecture that a similar proof technique as used in the proof of Theorems 28 and 29 may be used to demonstrate the correctness of Algorithm 5.
Algorithm 5 is described in the following. First we apply Algorithm 4 for computing $C_1, C_2$. Then we attempt to remove an agent’s local observation of some event occurrence, compute the maximum feasible sub-map of the resulting observation map (w.r.t., $C_1 \cup C_2$), and test to see if the predicate $\Phi$ is satisfied. If $\Phi$ is not satisfied, then we record the fact that removing observation of the particular event occurrence was unsuccessful, so that it is not attempted again. These two steps are repeated until no local observation of an event occurrence can be removed without violating the predicate $\Phi$, at which point the algorithm terminates, returning the computed local observation maps for each agent as well as $C_1, C_2$.

6.5 Conclusions

In this chapter we formalized the concept of an observation map, information map, and feasibility of observation maps w.r.t. a given information map. We also introduced containment relations between observation maps and between information maps. In Section 6.3 we presented several results, most of which demonstrate that, under certain settings where feasibility of observation maps is satisfied, seeing more permits an agent to know more. More precisely, if an agent uses a larger observation map, then an information map derived from the larger observation map will permit the agent to distinguish between more pairs of strings of a given prefix-closed formal language than if the agent used an information map derived from a smaller observation map. In Section 6.4 we introduced specifications, which are predicates defined over tuples of observation maps. We applied the results of Section 6.3 for solving a problem of computing minimal communication maps for satisfying a given specification and feasibility.
Algorithm 5 Compute Solution to Problem 17

Input

$\Phi: (L \rightarrow E(L))^2 \rightarrow \{\text{true, false}\}$

1: For $i \in \{1, 2\}$, let $O^i_{all}: L \rightarrow 2^{E_i}$ be an observation map such that, for all $e \in E_i$, for all $s \in L$, $se \in L$ if and only if $e \in O^i_{all}(s)$.
2: Compute $C_1, C_2$ by applying Algorithm 4 on input $\Phi$.
3: For $i \in \{1, 2\}$, let $S_i = [O^i_{all} \setminus C_i]$, $L_{S_i} = \emptyset$.
4: while $\exists k \in \{1, 2\}, \exists e \in E_k, \exists se \in L \setminus L_{S_k}, e \in S_k(s)$ do
5: Let $S_{k,se}: L \rightarrow 2^{E_k}$, $S_{k,se}(s) = \{e\}$ and for all $t \in L \setminus \{s\}$, $S_{k,se}(t) = \emptyset$.
6: Let $S_k' = ([S_k \setminus S_{k,se}], [C_1 \cup C_2])^f$.
7: if $k = 1$ then
8: if $\neg \Phi(I_{[S_1' \cup (C_1 \cup C_2)]}, I_{[S_2' \cup (C_1 \cup C_2)]})$ then
9: Let $L_{S_k} = L_{S_k} \cup \{se\}$
10: else
11: Let $S_k = S_k'$
12: end if
13: else
14: if $\neg \Phi(I_{[S_1' \cup (C_1 \cup C_2)]}, I_{[S_2' \cup (C_1 \cup C_2)]})$ then
15: Let $L_{S_k} = L_{S_k} \cup \{se\}$
16: else
17: Let $S_k = S_k'$
18: end if
19: end if
20: end while
21: Let $O_1 = [S_1 \cup C_1]$, $O_2 = [S_2 \cup C_2]$.
22: return $O_1, C_1, O_2, C_2$
As future work, one could consider studying other problems of minimal observation. For example, we proposed Problem 17 as an extension of Problem 16 where we additionally require that the use of an agent’s sensors for obtaining local event observations be minimized in addition to minimizing communication between agents. Also, the problems of Section 6.4 only consider the case of two agents, and could be generalized to the case of more than two agents. Also, one could consider studying other cases where observation maps provide more precise estimates as more event occurrences are observed. Such results may be applied for solving other problems of minimal observation.
Chapter 7

Conclusions and Future Work

In this chapter we summarize the contributions of the dissertation and propose some future work extending the results of this dissertation.

In Chapter 4 we investigated the problem of computing deterministic finite-state automata representations of sensor activation maps from associated sensor activation policies. Such sensor activation maps can be used by an agent to determine when an agent should turn an event sensor on or off based on the event observations made by the agent that are consistent with the associated sensor activation policy. We demonstrated that, when two strong notions of feasibility of sensor activation policies are satisfied, such finite representations may be computed in polynomial-time (and the associated feasibility conditions were introduced in Sections 4.3 and 4.4). This contrasts with the conventional approach for computing such maps where NFA to DFA conversion is employed, and is in exponential-time in the worst case. When one of the feasibility conditions studied is generalized slightly, the problem of verifying if a
given sensor activation policy satisfies the resulting feasibility condition is PSPACE-complete. For each feasibility condition considered, we demonstrated how a finite-state automaton representation of a sensor activation map associated with the input sensor activation policy may be computed.

In Chapter 5 we studied the problem of computing indistinguishable state pairs. We reviewed the CLUSTER-TABLE algorithm [50] for conducting this computation, and demonstrated its asymptotic complexity for a certain example. We proposed using the product of NFA for computing indistinguishable state pairs, and demonstrated that the worst-case asymptotic complexity of applying the product is not worse than CLUSTER-TABLE. In the case when the input automaton has cycles of unobservable transitions (i.e., $\varepsilon$-transitions), we demonstrated how computing a quotient automaton may first be conducted prior to application of the product. This potentially reduces the size of the product automaton, depending on the number and size of the cycles of unobservable transitions of the input automaton. We demonstrated how indistinguishable state pairs of automata may be used to verify properties such as observability [17], coobservability [24] and the eventual feasibility condition studied in Chapter 4. We demonstrated how the extended specification, which is studied in some problems of sensor activation in DES, may be computed by computing indistinguishable states using the product of automata. Also, we studied the problem of determining if a there exists a string generated by an NFA leads to one state in the NFA but not another state and demonstrated that this problem is PSPACE-complete.

In Chapter 6 we studied conditions where observation maps are monotonic in the sense that, as more is observed, the agent’s estimate of the current string generated
by a DES becomes more precise. Most results presented were positive in this regard, though we demonstrated one general case where observation maps are feasible with respect to what is observed, but this monotonicity property does not hold. We introduced a problem where communication between two agents must be minimized while satisfying certain feasibility conditions. We proposed algorithms for solving these problems, and proved that one of the algorithms is correct.

As future work, we propose the following. With respect to Chapter 4, we propose investigating other classes of sensor activation policies from which computation of a deterministic map from observed event sequences to sensor activation decisions can be done efficiently. The conditions studied in Chapter 4 are fairly strong, so it would be worthwhile to investigate other general, incomparable conditions which permit sensor activation maps to be computed from policies efficiently.

With respect to Chapter 6, we propose studying other cases where observation maps are monotonic in the sense that observing more permits distinguishing more. Such results may be applied for solving other problems of minimal observation outside of the communication problem studied in Section 6.4. We proposed Problem 17 as an extension of Problem 16 and proposed an algorithm for computing a solution to Problem 17 whose proof of correctness we leave open. The communication problem studied in Section 6.4 only considers the case of two agents, and could be generalized to the case of more than two agents. Also, these problems are fairly general and proposed in a language-based setting. As a result, the proposed solutions are fairly general, and there are limitations prohibiting their application. We conjecture that if specific, finite, state-based models of DES are considered, then other solutions to these problems that exploit such models may be constructed. Such solutions would
likely yield implementations with fewer practical limitations.
Bibliography


