Does Size Matter? The Effects of Supervisor Reduction on Minimal Communication Between Distributed Discrete-Event Agents

by

Sarah-Jane Whittaker

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
August 2005

Abstract

When applying control to a large and complex discrete-event system, traditional distributed supervisors may only be able to enforce a subset of the desired behaviour. If that subset is unacceptable, the agents must communicate with each other to achieve the required control action. However, the potential for delayed or lost messages and negative effects on the system demands that transmissions must be minimized. An algorithm that determines a minimal communication scheme which guarantees each supervisor is always certain of its current state does exist, but any options that may result in a further reduction in transmissions are worthy of exploration. One of these is to perform a state-size reduction on the agents, reducing their complexity and (hypothetically) the amount of communication they require to perform their control tasks.

This research examines several approaches to supervisor reduction and defines a new relation between agents known as *comparability* that encapsulates most of the concepts found in those methods. It then explores what occurs during the execution of the minimal communication algorithm for two pairs of agents, one pair of which is comparable to the other. The conjecture is that the original supervisors will communicate at least what the reduced supervisors communicate and most likely more. Unfortunately, this statement is not always correct. Both a counterexample and a "proof of concept" example are examined to illustrate both possible outcomes. Although there is no definitive result, performing a state-size reduction can lessen or even eliminate the need for communication. This fact would prompt the designer of any such system to perform such a reduction, but compare results before implementing the smaller agents.

Acknowledgments

I would like to thank my supervisor, Dr. Karen Rudie, for accepting me as her student and taking her role as my mentor so seriously. She demonstrated insight, faith, understanding and patience for two years despite numerous other demands that required her time and attention, and I am grateful for that.

My introduction to graduate work was assisted by the members of the DES Research group: Lenko Grigorov, Ying Huang, Michael Wood and Dr. Iakov Romanovski. Everyone in the group has not only proven to be considerate and helpful, but a lot of fun at meetings. Lenko is deserving of special thanks for his suggestions during a frustrating period in my work.

I would also like to recognize the contribution of Dr. Kai Salomaa, who made time in an already busy schedule to assist me at a crucial point in my research. He took great care in examining my work and addressing my concerns, which was very much appreciated.

My gratitude extends to Debby Robertson and Wendy Powley for enlightening me with their experience and knowledge of graduate work and research. I could always rely on them for an attentive ear and a sound piece of advice when I faced any sort of difficulty. Richard Linley and David Dove were also extremely helpful, providing candy, home repairs and encouragement when required. All of their efforts meant a great deal to me

Above all else, I would like to thank my husband, Benjamin Hall, for his limitless patience, love and support. He never phased out when I monopolized the conversation with discrete-event system theory and always knew how to neutralize my irritability after a poor research day. He is the best companion I could ever wish for and without him I would never have had the courage to attempt graduate work nor complete this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Motivation

It is a rainy morning in the world of research. The umbrella of systems control can be seen through the fog. Under the umbrella sits the field of discrete-event systems, waiting for the bus. In one hand, it holds the muffin of minimal communication between distributed agents. Unfortunately, it's never considered bringing the coffee of supervisor reduction as well. And although discrete-event systems still has breakfast, it would be much improved if it had brought coffee to go with the muffin.

Since its inception over 20 years ago, the original model for centralized control of a discrete-event system has been subject to numerous enhancements. None of these changes were conceived in frivolity; they were devised to expand the range of discrete-event processes to which control could be applied. Amongst the most powerful of these new concepts is that of distributed control: several supervisors working independently towards a common goal. This continues to be an important advancement in that numerous systems which are structurally distributed (whether by design or necessity) can be supervised.

It is thus rather unfortunate that the preceding sentence must use the term "supervised" instead of the phrase "fully controlled such that the desired behaviour is always enforced". The fact is that standard distributed supervision assumes that none of its agents can communicate the events they see to those that cannot. These agents must attempt to perform their function without any information beyond what they can observe, which may be a limited subsection of the system as a whole. As a result, distributed supervisors are often unable to fully enforce the desired behaviour and those attempting to implement the agents are forced to accept a limited amount of control, or perhaps none at all.

The need for communication is obvious, but communication itself is a wolf in sheep's clothing. Transmitting events between agents is intended to ensure that they always know enough to enforce the desired behaviour. The problem is that communication is never guaranteed; delayed or lost transmissions could confuse an agent and cause it to enable or disable events contrary to the desired action. However, if the need to communicate is minimized, then so is the risk associated with communication.

An algorithm that determines the minimal communication scheme between two distributed agents was proposed and proven to be correct by Rudie, Lafortune and Lin in [10]. Given two fully-defined automata that represent the behaviour of two distributed agents, the algorithm determines which events must be transmitted by the agents at what points during the process's operation. If all communications are properly sent and received, each automaton will always know its current state with absolute certainty. However, considering the risky nature of event transmissions, any possible enhancements to the scheme should be investigated.

Although the procedure returns a minimal communication scheme, there are several factors that should be considered when applying the algorithm, such as the efficiency of the given automata and the enablement/disablement policies of the given agents. Further analysis of these issues will yield strategies that may reduce the number of transmissions and allow the application of the algorithm to a wider problem set.

Problem Statement

The fact is that applying control to the majority of discrete-event systems is not difficult. The wealth of research done on the subject has resulted in a rich resource for those who must work with such event-driven processes. There are strategies for handling nearly every difficulty imaginable. Uncontrollable events? Unobservable events? Distributed system? Dynamic changes to the system? Not a problem; these can all be handled.

It is thus rather unfortunate that a tradeoff exists when it comes to the distributed control of discrete-event systems. As mentioned previously, too often only a subset of the desired behaviour can be coaxed out of the process while maintaining proper control. However, there are times when the desired behaviour is not up for discussion. Consider a shoe factory; all shoes must be glued and stitched, regardless of the distributed controllability of these actions.

[10] made great strides in addressing this issue with a method to determine a minimal communication scheme between two agents. It takes two fully-defined automata as input and returns two sets of transitions that must be communicated; one for each automaton. However, no examination is made of the automata themselves, both with respect to their efficiency in performing their control task and their control actions. Thus, there are two areas where the method of determining the event transmissions could be improved:

1. The current procedure takes two *fully-defined automata*, not discrete-event supervisors, as input. Thus, an agent that enforces control through implicit definition could not be used because it is not fully defined. In addition, a supervisor's feedback map (where defined) is not currently considered when determining the sets of transitions that are communicated, resulting in a scheme that may contain event occurences that will never occur.

2. The control action performed by a given agent gains new relevance when the plant that requires control is also considered. In reality, the given agent is but one of many possible supervisors that perform the same task, some of which are larger in state size than others. By "reducing" the given agent within the context of the plant, transitions will be eliminated, along with the need to have their associated events communicated by other supervisors.

The purpose of this research is thus to incorporate different types of supervisors into the algorithm and to prove that performing a state-space reduction on a set of distributed supervisors will never increase communication and will often result in a decrease of event transmissions. The desired goals are to achieve greater efficiency and make the method more applicable to discrete-event supervisors in particular and discrete-event systems in general.

Chapter 2

Discrete-Event Systems

2.1 Background

2.1.1 Centralized Discrete-Event Systems

A centralized discrete-event system (DES) is a process whose operation is defined solely in terms of its actions. It is characterized by sequences of events where, at each stage of the process's execution, what may occur depends entirely on what has occured previously. This type of system is completely time-independent; actions can neither result nor be predicted by any chronological function. Instead, the process could be termed a reactive system, as each event is a reaction to the occurrence of a preceeding event. A discourse on the fundamentals of DES theory can be found in [9], while a more comprehensive examination is offered in [1].

Modelling Discrete-Event Systems

There any many tools in the world of discrete mathematics and formal logic which can be employed to represent a discrete-event system; Petri nets, temporal logic and vectors are good examples. However, the standard for modelling requires the use of a finite-state automaton. There are several excellent reasons for this choice, including the following:

(i) Established language theory, algorithms and finite-state machine manipulation can be employed to assist in understanding and managing the process

(ii) The intuitive nature of automata provides a certain advantage; both in the model and in reality, different events lead the process into different states where other events may or may not occur

(iii) Events may be abstracted to a much higher level if the low-level actions are not relevant to the desired management of the system, *e.g.*, the event “grasp widget” may actually represent the sequence “open hand”, “position hand” and “close hand” if none of these actions are of individual importance to the process’s control policies

Thus, a discrete-event system is represented by a five-tuple

$$\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$$

where Σ is the set of events which may occur, Q is the set of states which the process may enter, δ is the set of event-transitions which take the operation from one state to another when an event occurs, q_0 is the process’s starting state and Q_m is the set of end (or marked) states where the process may terminate. The function δ may be totally or partially defined where

$$\delta : \Sigma \times Q \rightarrow Q$$

The function is applied in the following manner

$$\delta(\sigma, x) = y \text{ where } x, y \in Q,\ \sigma \in \Sigma$$

A transition is typically represented by a triple (x, σ, y) where the event σ leads from state x to state y. Note that as δ is a partial function, not every state has a transition defined for each event. In addition, some self-transitions (x, σ, x) may be defined where no change of state occurs. Finally, $\delta(\sigma, x)!$ is a common abbreviation for “the transition $\delta(\sigma, x)$ is defined.”

The automaton $\mathbf{G}$ mimics the behaviour of the process through its associated languages. The language $L(\mathbf{G})$ is the set of all event sequences (or strings of event labels) that lead to some state in $\mathbf{G}$ and $L_m(\mathbf{G})$ is the set of all event sequences that lead to marked or final states in $\mathbf{G}$. Note that either or both of these sets may be infinite but both are regular languages that can be respresented using finite-state automata. In terms of the process itself, $L(\mathbf{G})$ represents the possible behaviour of the system while $L_m(\mathbf{G})$ represents some recognized "completed" behaviour, *e.g.*, a widget is produced or a machine is turned off.

It is assumed that the modelled process does not contain any unreachable or "dead" states in which it may be unable to proceed to a marked state. Specifically, $\mathbf{G}$ must be *trim*—every state in $\mathbf{G}$ must have an event sequence that leads to some final state. With respect to languages, this means that the prefix-closure $\overline{L_m(\mathbf{G})}$ (which is the set of all prefixes of all words in $L_m(\mathbf{G})$) has the following property

$$\overline{L_m(\mathbf{G})} = L(\mathbf{G})$$

Example To properly illustrate the concepts discussed in this document, a running example will be built by incorporating, or examined using, the items described in the sections of each chapter. To begin, Figure 2.1 is the model of a plant $\mathbf{G}$. In this instance, $\Sigma = \{a_1, b_1, a_2, b_2\}$, $Q = \{A, B, C, D, E\}$,

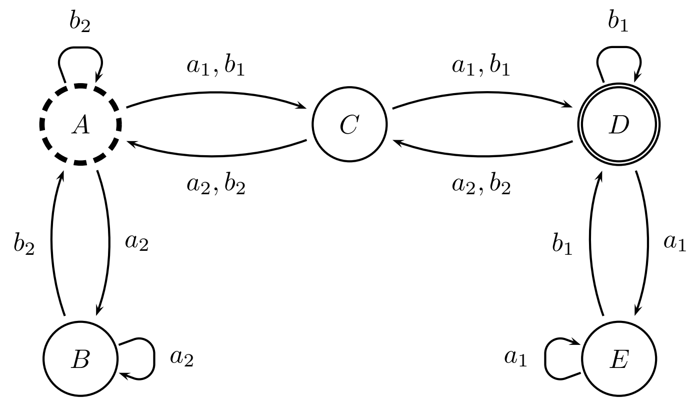


Figure 2.1: Running Example - Plant $\mathbf{G}$

$q_0 = A$ and $Q_m = \{D\}$. The initial and marked states are represented by dashed and double-lined borders, respectively. Although there is currently no control policy applied to the plant, some states do not have an outgoing transition for every event, *i.e.*, $\delta(a_1, B)$ and $\delta(b_1, B)$ are not defined. This does not imply that the plant is controlling its events; rather, it means that the plant's construction makes it impossible for those events to occur in those states.

Modelling Supervision

If DESs could only be employed as a modelling tool, research into this area would not have advanced very far. Fortunately, DESs offer a far more powerful feature: the ability to enforce some manner of control over the system. This is accomplished by enabling and disabling specific events in specific states. However, implementing control is not as simple as it sounds; applying a malformed control scheme will most likely affect the process negatively, resulting in unanticipated and undesirable behaviour such as blocking.

The first consideration when modelling a supervisor is the nature of the system's events. While some actions may be easily enabled or disabled, others may not. This leads to a necessary classification: events must be either controllable or uncontrollable.

$$\Sigma = \Sigma_c \cup \Sigma_{uc} \qquad \Sigma_c \cap \Sigma_{uc} = \emptyset$$

The difference between the two is that controllable events may be disabled at any given state while uncontrollable events can never be cancelled. This is sensible from a real-world perspective; a machine may be able to control the processing of a widget, but it cannot prevent a faulty widget from crumpling and jamming the machine.

At this point, an analogy in DES terminology must be introduced. So far, the operation to be controlled has been referred to as a process. Now think of it more as a factory or a *plant*, where a single machine or multiple machines perform tasks under the supervision of human controllers. It is the responsibility of these *supervisors* to monitor the plant and ensure that undesirable behaviour is not allowed to occur. Thus, from this point onwards the process will be referred to as a plant and any control scheme for that process will be referred to as a supervisor/agent.

A supervisor is defined as a pair

$$\mathbf{S} = (S, \phi)$$

where S is an automaton (over the same alphabet as $\mathbf{G}$) describing the behaviour of the supervisor and ϕ is a "feedback map" that determines when controllable events are disabled within S. Formally,

$$S = (\Sigma, X, \xi, x_0, X_m)$$

where Σ is the same set of event labels present in $\mathbf{G}$, X is the set of states, ξ is the set of event-transitions, x_0 is the process's start state and X_m is the set of end states. The mapping ϕ is defined as a total function where

$$\phi : \Sigma \times X \rightarrow \{0, 1\}$$

In this case, 0 denotes that an event is disabled and 1 denotes that is enabled. The feedback map's responsibility is to provide "control rules" constrained as follows: for all $\sigma \in \Sigma$ and $x \in X$, $\phi(\sigma, x) = 1$ if $\sigma \in \Sigma_{uc}$ and $\phi(\sigma, x) \in \{0, 1\}$ if $\sigma \in \Sigma_c$.

A supervisor may be defined in a straightforward manner if ϕ is determined implicitly by the transition structure of S. Specifically,

$$\phi(\sigma, x) = \begin{cases} 1 & \text{if } \sigma \in \Sigma_{uc} \\ 1 & \text{if } \sigma \in \Sigma_c \wedge \xi(\sigma, x)! \\ 0 & \text{otherwise} \end{cases}$$

Example At this point in the example, it is vital to note that every event in the plant $\mathbf{G}$'s alphabet Σ is considered controllable. Formally, $\Sigma_c = \Sigma$ and $\Sigma_{uc} = \emptyset$; this has been assumed for simplicity as factors other than controllability are the focus of this work.

Figure 2.2 is a diagram of an explicitly-defined supervisor $\mathbf{S}$. This supervisor is designed to act on $\mathbf{G}$, and thus possesses the same alphabet $\Sigma = \{a_1, b_1, a_2, b_2\}$. Its states are defined as follows: $X = \{1, 2, 3, 4\}$, $x_0 = 1$ and $X_m = \{4\}$. Nearly every event is enabled at each state in $\mathbf{S}$, the exceptions being $\phi(a_2, 2) = 0$, $\phi(b_1, 2) = 0$, $\phi(a_1, 3) = 0$ and $\phi(b_2, 3) = 0$. These are distinguished in the diagram by dotted edges; all transitions with enabled events have solid edges. An implicitly-defined, equivalent version of the same supervisor would appear as shown in Figure 2.3.

Applying Control

Thus far, two distinct processes have been defined: the uncontrolled actions of a plant and the event enablement/disablement policies of a supervisor. These behaviours must be merged so the agent may begin its supervision of the plant. With respect to automata, the supervisor must perform the following duties:

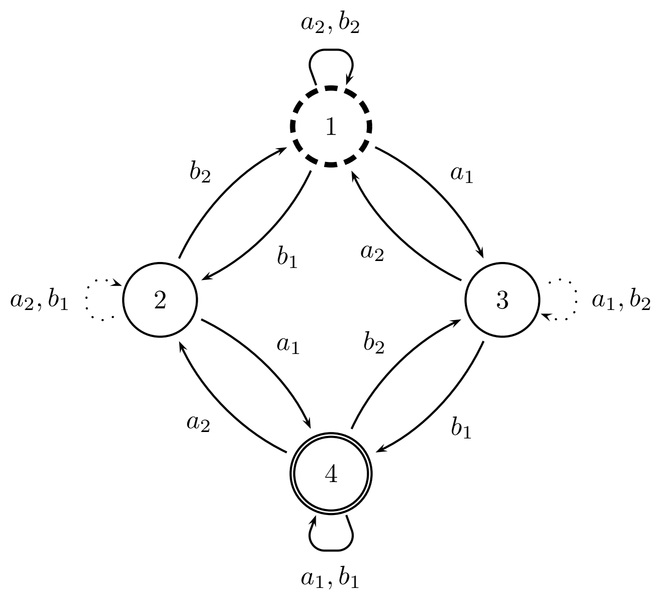


Figure 2.2: Running Example - Explicitly-Defined Supervisor **S**

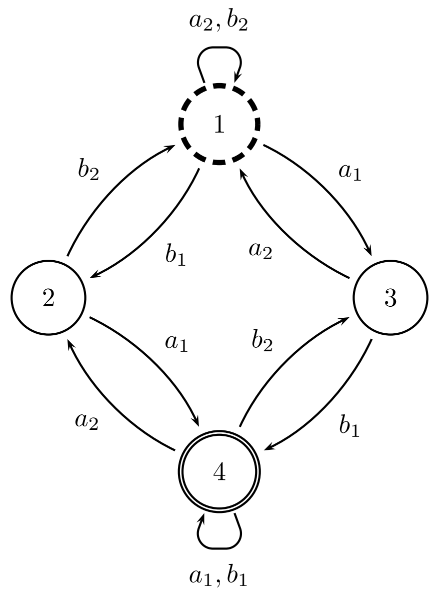


Figure 2.3: Running Example - Implictly-Defined Supervisor Equivalent to **S**

(i) Observe the events occurring in the plant and change state in its own automaton as the plant changes state

(ii) Prevent the plant from taking a transition if that event is to be disabled by the supervisor in its current state

To accomplish this, supervisor **S** is applied to plant **G** Cartesian-style, resulting in

$$\mathbf{S}/\mathbf{G} = (\Sigma, (Q \times X), (\delta \times \xi)^{\phi}, (q_0, x_0), (Q_m \times X_m))$$

where Σ is the defined set of event labels, $(Q \times X)$ is the combined set of states, (q_0, x_0) is the joint start state, and $(Q_m \times X_m)$ is the set of joint end states. The transition function $(\delta \times \xi)^{\phi}$ is defined as follows

$$\forall\, \sigma \in \Sigma,\ (q,x) \in (Q \times X),\ (\delta \times \xi)^{\phi}(\sigma,(q,x)) = \begin{cases} (\delta(\sigma,q), \xi(\sigma,x)) & \text{if } \delta(\sigma,q)! \wedge \\ & \xi(\sigma,x)! \wedge \phi(\sigma,x) = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

There is a key concept within this method of combining the behaviours of the plant and the supervisor that should be noted. The automaton **S**/**G** only allows a sequence of events to occur under the following circumstances:

1. The uncontrolled behaviour of the plant permits the sequence
2. The sequence does not violate the control scheme exerted by the supervisor

As a result, the language of the automaton $L(\mathbf{S}/\mathbf{G})$ must be a subset of (or equal to) $L(\mathbf{G})$ and $L(\mathbf{S})$ because it is restricted to what is possible for the plant to achieve and what the supervisor allows the plant to achieve.

Example Figure 2.4 illustrates the result of supervisor **S** acting on plant **G**. The first item of note is that even though $(Q \times X)$ contains 20 states, only those that are *reachable* are considered states in $(\mathbf{S}/\mathbf{G})$. This example also illustrates the case where **S** not only restricts the actions of **G**, but **G** also prohibits some of the behaviour specified by **S**. For example, $a_1a_1 \in L(\mathbf{G})$ but $a_1a_1 \notin L(\mathbf{S})$ since the second a_1 is disabled by the supervisor at state 3. On the other hand, $a_2a_1 \in L(\mathbf{S})$ but $a_2a_1 \notin L(\mathbf{G})$ since a_1 can not be generated by the plant at state B. Thus, $L(\mathbf{S}/\mathbf{G}) \subset L(\mathbf{G})$ and $L(\mathbf{S}/\mathbf{G}) \subset L(\mathbf{S})$.

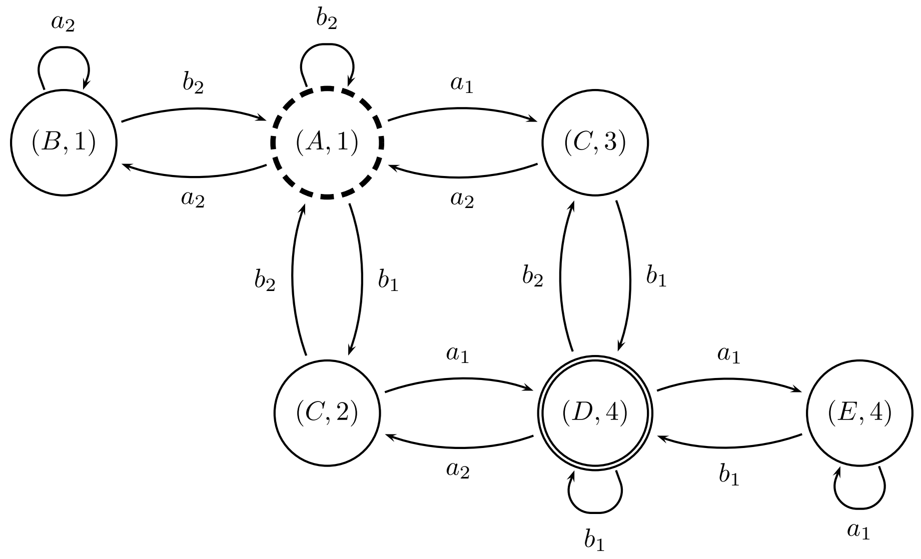


Figure 2.4: Running Example - Supervisor **S** Acting on Plant **G**

Proper Supervision

At this juncture, it is important to place some focus on constructing appropriate supervisors. As stated previously, a malformed supervisor may allow or even force the plant to generate undesirable events. To avoid such a malfunction, the first requirement for an appropriate supervior is that it be *complete*. Generally, this means that if a supervisor allows an event sequence to occur within the plant, a subsequent event that is accepted by the plant and allowed by the feedback map should be accepted by the supervisor. Formally,

$$\forall\, s \in \Sigma^*,\, \forall\, \sigma \in \Sigma,\, s \in L(\mathbf{S}/\mathbf{G}) \,\wedge\, s\sigma \in L(\mathbf{G}) \,\wedge\, \phi(\sigma, \xi(s, x_0)) = 1 \Rightarrow s\sigma \in L(\mathbf{S}/\mathbf{G})$$

The second requirement is that the supervisor be *nonblocking*. Generally speaking, this implies that once the supervisor is applied to the plant, the supervisor will not cause the plant to enter a dead state. Formally,

$$\overline{L_m(\mathbf{S}/\mathbf{G})} = L(\mathbf{S}/\mathbf{G})$$

In other words, $\mathbf{S}/\mathbf{G}$ is trim.

Example It is apparent that supervisor $\mathbf{S}$ is complete and nonblocking with respect to $\mathbf{G}$. In this case, $\mathbf{S}$'s simplicity allows it to avoid these two issues. However, in real-world systems with extremely large state spaces, these are significant issues which may prevent a suitable supervisor from being found. Having said that, only a few minor alterations are required to invalidate $\mathbf{S}$.

To make $\mathbf{S}$ incomplete, simply remove the definition $\xi(b_2, 1) = 1$. For $s = \epsilon$ and $\sigma = b_2$, $b_2 \in L(\mathbf{G}) \wedge \phi(b_2, 1) = 1$ *but* $b_2 \notin L(\mathbf{S}/\mathbf{G})$ due to the fact that $\xi(b_2, 1)$ is not defined. To make $\mathbf{S}$ blocking, simply disable b_1 at state 4. This will result in state $(E, 4)$ losing its only transition back to the marked state $(D, 4)$, thus reducing $(E, 4)$ to a dead state.

2.1.2 Distributed Discrete-Event Systems

The class of DESs discussed thus far is centralized in that a single supervisor observes, enables and disables all of the events in the system. However, there are instances where the nature of the plant renders it unrealistic to follow such a model. For example, a plant could have such a large state space as to render it computationally infeasible to determine a single supervisor that satifies all of the plant's control requirements. Another possibility is that the plant contains components that are physically separated by significant distances, thus disallowing a lone supervisory unit. This scenario was first formulated in [17], while [12] offers a recent summary of related research.

Modelling Modular Supervision

Modular supervision is an obvious solution to the problems described above. It requires the assignment of individual agents (such as $\mathbf{S_1}$ and $\mathbf{S_2}$) to the subprocesses of a plant $\mathbf{G}$. This method has several advantages over the centralized approach in that there is not as much computation to be done initially and alterations can be made to individual supervisors without affecting the other agents in the system. However, there is a significant disadvantage: it is possible that a set of modular supervisors that are nonblocking with respect to their individual tasks may not be nonblocking with respect to the system as a whole. In other words, local agents are not aware of their effect on the larger process and cannot know that together they are producing undesirable results. For some plants, a centralized supervisor may be able to apply more control than a series of local agents. However, where centralized supervision is simply not an option, there is no other choice.

Since the agents $\mathbf{S_1}$ and $\mathbf{S_2}$ are restricted to controlling subprocesses of $\mathbf{G}$, they are also restricted to observing the events that occur in their assigned subprocess. Hence, $\mathbf{S_1}$ and $\mathbf{S_2}$ are no longer capable of observing Σ but subsets $\Sigma_{1,o}$ and $\Sigma_{2,o}$ where $\Sigma_{1,o} \cup \Sigma_{2,o} = \Sigma_o$. Note that some events may not be observable to either agent, nor may the union of their observable events be disjoint; $\mathbf{S_1}$ and $\mathbf{S_2}$ may overlap in their supervision. Likewise, the set Σ_c must be divided into subsets $\Sigma_{1,c}$ and $\Sigma_{2,c}$; the set of uncontrollable events is then defined as $\Sigma_{uc} = \Sigma \setminus (\Sigma_{1,c} \cup \Sigma_{2,c})$. The set of globally unobservable events is defined similarly as $\Sigma_{uo} = \Sigma \setminus (\Sigma_{1,o} \cup \Sigma_{2,o})$.

Once the observable events for each supervisor have been determined, so can the language that each agent will see. Every sequence of events in $L(\mathbf{G})$ and $L_m(\mathbf{G})$ will appear differently to $\mathbf{S_1}$ and $\mathbf{S_2}$ because they will not be able to oberve some of the events in that sequence. A *projection* will be employed to determine exactly what $\mathbf{S_i}$ will see when $\mathbf{G}$ generates $s \in \Sigma^*$ and $\sigma \in \Sigma$.

$$\begin{aligned}
&P_i : \Sigma^* \rightarrow \Sigma_{i,o}^* \\
&P_i(\sigma) = \begin{cases} \sigma & \text{if } \sigma \in \Sigma_{i,o} \\ \epsilon & \text{otherwise} \end{cases} \\
&P_i(s\sigma) = P_i(s)P_i(\sigma)
\end{aligned}$$

Let $\mathbf{S_1} = (S_1, \phi_1)$ and $\mathbf{S_2} = (S_2, \phi_2)$ where $S_i = (\Sigma, X_i, \xi_i, x_{i,0}, X_{i,m})$ and $\phi_i : \Sigma \times X_i \rightarrow \{0, 1\}$, $i \in \{1, 2\}$. When both agents act on the plant $\mathbf{G}$, they work in *conjunction* with each other in the same manner as the following centralized supervisor

$$\mathbf{S} = \mathbf{S_1} \wedge \mathbf{S_2}$$
$$(S, \phi) = ((S_1 \times S_2), (\phi_1 \times \phi_2))$$
$$S = (\Sigma, X, \xi, x_0, X_m)$$

where $X \subseteq (X_1 \times X_2)$ is the set of reachable states, $x_0 = (x_{1,0}, x_{2,0})$ is the initial state and $X_m \subseteq (X_{1,m} \times X_{2,m})$ is the set of reachable marked states. The transition function ξ and feedback map ϕ are defined as follows for all $\sigma \in \Sigma_o$ and $x = (x_1, x_2) \in X$:

$$\xi(\sigma, x) = (\xi_1 \times \xi_2)(\sigma, (x_1, x_2)) = \begin{cases} (\xi_1(\sigma, x_1), \xi_2(\sigma, x_2)) & \text{if } \xi_1(\sigma, x_1)! \wedge \xi_2(\sigma, x_2)! \\ (\xi_1(\sigma, x_1), x_2) & \text{if only } \xi_1(\sigma, x_1)! \\ (x_1, \xi_2(\sigma, x_2)) & \text{if only } \xi_2(\sigma, x_2)! \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\phi(\sigma, x) = (\phi_1 \times \phi_2)(\sigma, (x_1, x_2)) = \begin{cases} 0 & \text{if } \phi_1(\sigma, x_1) = 0 \vee \phi_2(\sigma, x_2) = 0 \\ 1 & \text{otherwise} \end{cases}$$

The languages generated by $\mathbf{S_1}$ and $\mathbf{S_2}$ operating jointly are $L((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G}) = L(\mathbf{S_1}/\mathbf{G}) \cap L(\mathbf{S_2}/\mathbf{G})$ and $L_m((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G}) = L_m(\mathbf{S_1}/\mathbf{G}) \cap L_m(\mathbf{S_2}/\mathbf{G})$. In more general terms, the conjunction of two supervisors will only permit a sequence of events to occur if both supervisors permit it.

Example The example thus far has examined a single centralized supervisor acting on a plant. Now consider that, for some unspecified reason, additional behaviour must be enforced on $\mathbf{G}$ and a centralized solution is no longer feasible. Instead, two supervisors $\mathbf{S_1} = (S_1, \phi_1)$ and $\mathbf{S_2} = (S_2, \phi_2)$ are created where $S_i = (\Sigma, X_i, \xi_i, x_{i,0}, X_{i,m})$ and $\phi_i : \Sigma \times X_i \to \{0, 1\}$, $i \in \{1, 2\}$. The supervisor $\mathbf{S_1}$ will be identical to $\mathbf{S}$ as given in Figure 2.2 with one exception: while the agent's controllable events remain the same ($\Sigma_{1,c} = \Sigma$), what it can observe is now limited to $\Sigma_{1,o} = \{a_1, b_1\}$. As for $\mathbf{S_2}$, it is also an explicitly-defined supervisor as shown in Figure 2.5.

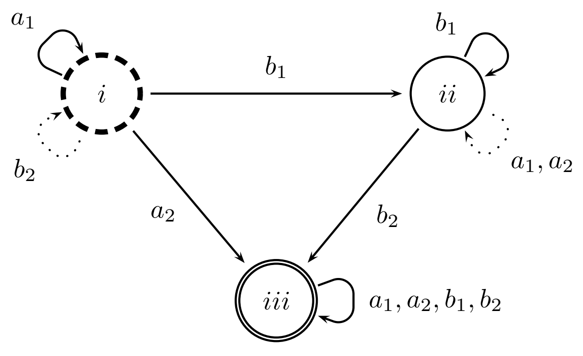


Figure 2.5: Running Example - Distributed Supervisor $\mathbf{S_2}$

Supervisor $\mathbf{S_2}$ observes the event set $\Sigma_{2,o} = \{a_2, b_2\}$, which is completely disjoint from that observed by $\mathbf{S_1}$. Agent $\mathbf{S_2}$ can also control nearly as much as its counterpart: $\Sigma_{2,c} = \{a_1, a_2, b_2\}$ and $\Sigma_{2,uc} = \{b_1\}$. In addition, the states of $\mathbf{S_2}$ are defined as $X = \{i, ii, iii\}$, $x_0 = i$ and $X_m = \{iii\}$. Like $\mathbf{S_1}$, $\mathbf{S_2}$ does not disable a great many events, the exceptions being $\phi_2(b_2, i) = 0$, $\phi_2(a_1, ii) = 0$ and $\phi_2(a_2, ii) = 0$.

When $\mathbf{S_1}$ and $\mathbf{S_2}$ work in conjunction with each other, they simulate the single centralized supervisor $\mathbf{S}$ illustrated in Figure 2.6. The resulting behaviour of the plant $\mathbf{G}$ when both $\mathbf{S_1}$ and $\mathbf{S_2}$

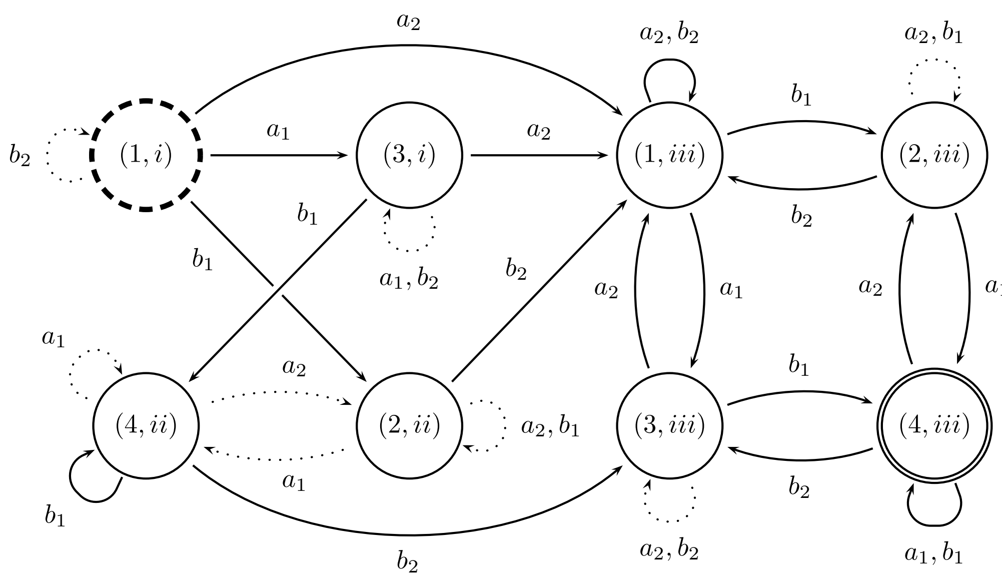


Figure 2.6: Running Example - Conjunction of $\mathbf{S_1}$ and $\mathbf{S_2}$

act in conjunction upon it is shown in Figure 2.7.

This example bears many similarities to the previous example where centralized supervisor $\mathbf{S}$ acted on plant $\mathbf{G}$, shown in Figure 2.4. Again, although there are 60 possible states between $\mathbf{S_1}$, $\mathbf{S_2}$ and $\mathbf{G}$, only 10 of those are actually reachable. In addition, the language $L(\mathbf{S_1} \wedge \mathbf{S_2}/\mathbf{G})$ is the result of both enablement/disablement policies placed on $\mathbf{G}$ by $\mathbf{S_1}$ and $\mathbf{S_2}$ *as well as* the inability of $\mathbf{G}$ to generate some of the behaviour sanctioned by $\mathbf{S_1}$ and $\mathbf{S_2}$.

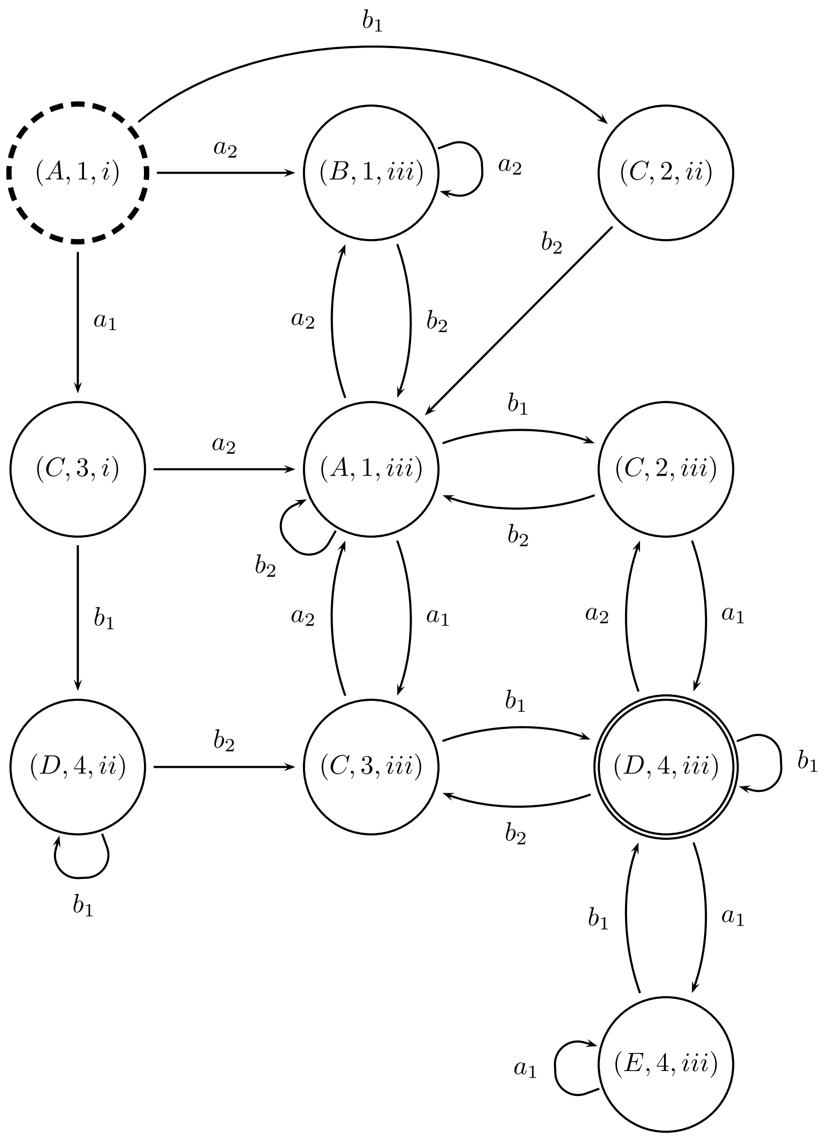


Figure 2.7: Running Example - Distributed Supervisors $\mathbf{S_1}$ and $\mathbf{S_2}$ Acting on Plant $\mathbf{G}$

Proper Modular Supervision

It has been proven that $(\mathbf{S_1} \wedge \mathbf{S_2})$ is complete with respect to $\mathbf{G}$ if $\mathbf{S_1}$ and $\mathbf{S_2}$ are both complete with respect to $\mathbf{G}$. However, $(\mathbf{S_1} \wedge \mathbf{S_2})$ is not necessarily nonblocking if $\mathbf{S_1}$ and $\mathbf{S_2}$ are. The supervisors

$\mathbf{S_1}$ and $\mathbf{S_2}$ must also be *nonconflicting* where

$$\overline{L_m(\mathbf{S_1}/\mathbf{G}) \cap L_m(\mathbf{S_2}/\mathbf{G})} = \overline{L_m(\mathbf{S_1}/\mathbf{G})} \cap \overline{L_m(\mathbf{S_2}/\mathbf{G})}$$

The two conditions above ensure that the centralized representation of $\mathbf{S_1}$ and $\mathbf{S_2}$ satisfies the nonblocking requirement outlined earlier.

Example As shown in Figures 2.6 and 2.7, it is apparent that both ($\mathbf{S_1} \wedge \mathbf{S_2}$) is nonblocking and $\mathbf{S_1}$ and $\mathbf{S_2}$ are nonconflicting since there are no dead states in either ($\mathbf{S_1} \wedge \mathbf{S_2}$) or (($\mathbf{S_1} \wedge \mathbf{S_2}$)/$\mathbf{G}$). However, only a minor change is required to make $\mathbf{S_1}$ and $\mathbf{S_2}$ conflict. Simply set $\phi_1(a_1, 2) = 0$ and $\phi_2(b_2, iii) = 0$. Although these changes do not cause $\mathbf{S_1}$ or $\mathbf{S_2}$ to block with respect to $\mathbf{G}$, it does render both $(B, 1, iii)$ and $(C, 2, iii)$ as dead states.

2.2 Research

It was stated previously that the research documented in this dissertation is dedicated to supervisor reduction and minimizing communication between distributed agents. Although this is certainly the case, not all of the new concepts are specific to those topics. The following definitions and associated results are based on the foundations of discrete-event systems with respect to supervised control and will be employed extensively in order to meet the stated research goals.

2.2.1 Synthesis-Accessibility

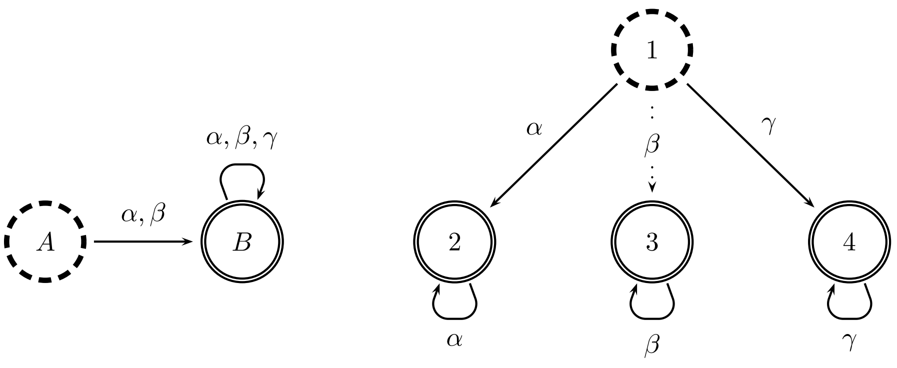
Definition 1. Given a plant $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$ and a supervisor $\mathbf{S} = (S, \phi)$ where $S = (\Sigma, X, \xi, x_0, X_m)$ and $\phi : \Sigma \times X \to \{0, 1\}$, a state $x \in X$ is considered to be *synthesis-accessible* with respect to $\mathbf{G}$ if $\exists\, s \in L(\mathbf{S}/\mathbf{G})$ such that $\xi(s, x_0) = x$.

Remarks. Synthesis-accessibility is as much a compound idea as it is a compound word: it applies the definitions of accessibility and language synthesis to the states in $\mathbf{S}$. Only states that are synthesis-accessible with respect to $\mathbf{G}$ are be considered in the pursuit of the primary research goals.

An obvious question at this point would be whether it is assumed that all given supervisors are trim with respect to synthesis-accessible states and their associated plant. The response is negative and the reason is relatively simple. Although many agents are custom-built for a certain plant and are thus trim, some supervisors could be designed for application to numerous plants. As a result, only a subset of the agents' potential control actions may be enforced on a given process. Hence, the states of interest in a supervisor become those that will have an opportunity to actually enforce control.

It should be noted that trimming a given supervisor with respect to synthesis-accessiblity is very simple: simply remove those states that are not synthesis-accessible for a given plant and eliminate all transitions that enter and exit these states. The result is a trim agent that enforces identical control on the plant in question.

Example From Figure 2.7, it is apparent that every state in both $\mathbf{S_1}$ and $\mathbf{S_2}$ is synthesis-accessible, as each state is visited while the supervisor is acting on the plant. In contrast, this is not the case in Figure 2.8. States 3 and 4 of the supervisor are synthesis-inaccessible; 3 because its only incoming



(a) Plant **G** (b) Supervisor **S**

Figure 2.8: Synthesis Inaccesibility Example - Plant **G** and Supervisor **S**

transition is disabled and 4 because γ is not permitted by the plant in its initial state.

2.2.2 Synthesis-Accessibility Set

Definition 2. Given a plant **G** and an associated supervisor **S**, a *synthesis-accessibility set* $[x]$ is defined for each state $x \in X$ where $[x] = \{s \mid s \in L(\mathbf{S}/\mathbf{G}) \wedge \xi(s, x_0) = x\}$.

The following facts can be ascertained from the definition of $[x]$:

1. $\forall\, x \in X,\ [x] = \emptyset \Rightarrow x$ is not synthesis-accessible
2. $\forall\, x \in X, \forall\, y \in X,\ [x] = [y] \Leftrightarrow x = y$
3. $\forall\, x \in X, \forall\, y \in X,\ [x] \cap [y] = \emptyset \Leftrightarrow x \neq y$

Remarks. In addition to considering which states of $\mathbf{S}$ are accessible during synthesis, it is also vital to categorize the strings that actually enter those states. These sets will be used extensively in the investigation of supervisor reduction to relate the states of the original agent to those in the modified agent.

It should be noted that $[x]$ is not an equivalence class; such a class would require an equivalence relation $\mathcal{R}$ over the set $(X \times X)$. Since the contents of $[x]$ are strings and not states, there is no such relation.

Example Many of the states in $((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G})$ have infinite synthesis-accessibility sets that are difficult to describe. However, some states have sets that can be expressed quite simply using regular expressions. Examples include the following:

- $[(A, 1, i)] = \epsilon$
- $[(C, 2, ii)] = b_1$
- $[(D, 4, ii)] = a_1 b_1 (b_1)^*$

Chapter 3

Supervisor Reduction

3.1 Background

The terms "supervisor reduction" and "supervisor minimization" are somewhat ambiguous in that they do not specify what aspect of supervision undergoes alteration. In [14], they refer to the elimination of superfluous distributed agents whose control actions are redundant. However, in more general terms, performing a *reduction* on a supervisor $\mathbf{S}$ will result in another agent $\hat{\mathbf{S}}$ that has fewer states but enforces the same control action as $\mathbf{S}$, while *minimization* results in a reduced agent with the smallest state size possible. There are several different approaches to supervisor reduction, all of which (that are known to the author) will be described in appropriate detail in this section. For a more in-depth examination of these methods, please refer to Appendix A, "Supervisor Reduction: A Quick Reference Guide".

3.1.1 Projection

Among the first to examine the effects of state-set reduction are Ramadge and Wonham in [8]. Their work produced the concept of a *projection.* Given two supervisors $\mathcal{S} = (S, \phi)$ and $\hat{\mathcal{S}} = (\hat{S}, \hat{\phi})$ where $S = (X, \Sigma, \xi, x_0, X_m)$ and $\phi : X \to \{0,1\}^\Sigma$ (and similarly for $\hat{S}$ and $\hat{\phi}$)[1], a total function $\pi : X \to \hat{X}$ is a projection from $\mathcal{S}$ to $\hat{\mathcal{S}}$ if the following conditions are satisfied:

[1]The notation $\phi : X \to \{0,1\}^\Sigma$ comes directly from [8]. It is equivalent to the form $\phi : \Sigma \times X \to \{0,1\}$ used previously in this dissertation.

(i) $\pi : X \to \hat{X}$ is surjective,

(ii) $\pi(x_0) = \hat{x}_0$ and $X_m = \pi^{-1}(\hat{X}_m)$,

(iii) $\hat{\xi} \circ (\mathrm{id}_\Sigma \times \pi)(\sigma, x) = \pi \circ \xi(\sigma, x)$ for all (σ, x) where $\xi(\sigma, x)$ is defined,

(iv) $\hat{\phi} \circ \pi = \phi$.

If such a state mapping exists, then $\hat{\mathcal{S}}$ is the *quotient* of $\mathcal{S}$ with respect to π. The concept of a quotient is slightly more rigid than that of a cover; the surjection requirement disallows a state in $\mathcal{S}$ from being mapped to multiple states in $\hat{\mathcal{S}}$. However, the end result is the same: given any plant $\mathcal{G}$, $L(\mathcal{S}/\mathcal{G}) = L(\hat{\mathcal{S}}/\mathcal{G})$ and $L_m(\mathcal{S}/\mathcal{G}) = L_m(\hat{\mathcal{S}}/\mathcal{G})$. Unfortunately, no algorithm is provided that determines a quotient supervisor from some given agent.

3.1.2 Cover

Further inroads into this topic were made by Wonham and Vaz in [16] with the introduction of *covers*. Given a supervisor $\mathbf{S} = (S, \psi)$ where $S = (\Sigma, X, \xi, x_0, X_m)$ and $\psi : (\Sigma \times X) \to \{0, 1, dc\}$ (where dc represents "don't care", *i.e.*, neither enabled nor disabled), a cover $\mathbf{C} = \{X_i \mid i \in I\}$ of $\mathbf{S}$ on index set I is defined as follows:

$$\begin{aligned}
\forall\, i, \quad & X_i \neq \emptyset; \\
\text{For a subset } I_m \subset I, \quad & X_m = \cup\{X_i \mid i \in I_m\}, \\
& X - X_m = \cup\{X_i \mid i \in I - I_m\}; \\
(\forall\, i, \sigma) : (\exists\, y \in X_i),\ \xi(\sigma, y)! \quad & \Rightarrow (\exists\, j)(\forall\, x \in X_i) \cdot \xi(\sigma, x)! \Rightarrow \xi(\sigma, x) \in X_j \\
& \text{(for brevity, we write this property as} \\
& (\forall\, i, \sigma)(\exists\, j)\, \xi(\sigma, X_i) \subset X_j) \\
(\forall\, i, \sigma)(\forall\, x, y \in X_i), \quad & \psi(\sigma, x) \neq dc \neq \psi(\sigma, y) \\
& \Rightarrow \psi(\sigma, x) = \psi(\sigma, y)
\end{aligned}$$

Informally speaking, a cover lumps together states from $\mathbf{S}$ that have similar marking and control actions such that the transition function and feedback map are both preserved. A *cover triple* (i, σ, j) stands in for transitions of the form (x, σ, y) where $x \in X_i$ and $y \in X_j$. Note that a state from $\mathbf{S}$ can be in more than one subset in $\mathbf{C}$.

From the cover and its triples, the reduced supervisor $\overline{\mathbf{S}} = (\overline{S}, \overline{\psi})$ is determined:

```
$\overline{S} = (\Sigma, I, \overline{\xi}, i_0, I_m)$
    Select $i_0 \in I$ such that $x_0 \in X_{i_0}$;
    Define $\overline{\xi} : \Sigma \times I \to I$ (pfn) as follows:
        For $\sigma \in \Sigma, i \in I$ select $j \in I$ such that $(i, \sigma, j)$
        is a cover triple and let $\overline{\xi}(\sigma, i) = j$;
    Define $\overline{\psi} : (\Sigma \times I) \to \{0, 1, dc\}$ as follows:
        For $\sigma \in \Sigma, i \in I$ if there exists $x \in X_i$ such that
        $\psi(\sigma, x) \neq dc$ then let $\overline{\psi}_i(\sigma) = \psi(\sigma, x)$;
        otherwise let $\overline{\psi}_i(\sigma) = dc$.
```

Covers are extremely powerful; not only does $L(\mathbf{S}/\mathbf{G}) = L(\overline{\mathbf{S}}/\mathbf{G})$ and $L_m(\mathbf{S}/\mathbf{G}) = L_m(\overline{\mathbf{S}}/\mathbf{G})$, but a cover can produce a reduced supervisor with the smallest state size possible for that control task. However, the algorithm to determine that supervisor is exponential in complexity and thus not feasible for the vast majority of control tasks.

3.1.3 Control Cover/Congruence

At this juncture, research into this topic appears to have languished until Su and Wonham addressed it again in [13]. Their approach is the first to consider reducing a supervisor using knowledge of both its and its associated plant's structures. As a result, their method is more complex than those described previously. This is offset by the implementation of their procedure in CTCT, the current standard for discrete-event software created and managed by Wonham and his Systems Control Group [3].

Given a plant $\mathbf{G} = (Y, \Sigma, \eta, y_0, Y_m)$ and its desired behaviour, represented by the automaton **SPEC**, the resulting supremal controllable sublanguage of the two is captured (using **supcon** from TCTC) in $\mathbf{SUPER} = (X, \Sigma, \xi, x_0, X_m)$. In contrast, an implicitly-defined agent $\mathbf{SIMSUP} = (Z, \Sigma, \zeta, z_0, Z_m)$ may exist that is much smaller in terms of state size than **SUPER** but is *control-equivalent*, where

$$L(\mathbf{G}) \cap L(\mathbf{SIMSUP}) = L(\mathbf{SUPER}) \tag{1a}$$

$$L_m(\mathbf{G}) \cap L_m(\mathbf{SIMSUP}) = L_m(\mathbf{SUPER}) \tag{1b}$$

The goal then becomes to determine such a **SIMSUP** from **SUPER**. It would be ideal to determine **MINSUP**, which has the smallest state size possible of all agents that enforce the best possible control (given in **SUPER**) on **G**. This task is possible, but unfortunately proven to be NP-hard [13].

The first task is to build relationships between the states of **SUPER** using its stucture and that of **G**. Four sets of states are defined as follows:

Let $E : X \to 2^{\Sigma}$ with

$$x \mapsto E(x) := \{\sigma \in \Sigma \mid \xi(x, \sigma)!\}$$

be the **SUPER**-enabled event set at state $x \in X$. Let $D : X \to 2^{\Sigma}$ with

$$x \mapsto D(x) := \{\sigma \in \Sigma \mid \neg\xi(x, \sigma)! \wedge (\exists\, s \in \Sigma^*)[\xi(x_0, s) = x \wedge \eta(y_0, s\sigma)!]\}$$

be the **SUPER**-disabled event set at state $x \in X$. Let $M : X \to \{true, false\}$ with

$$x \mapsto M(x) := true \text{ if } x \in X_m$$

Finally, let $T : X \to \{true, false\}$ with

$$x \mapsto T(x) := true \text{ if } (\exists\, s \in \Sigma^*)\xi(x_0, s) = x \wedge \eta(y_0, s) \in Y_m$$

From the above groups of states, the relation $\mathcal{R} \subseteq X \times X$ is defined. A state pair (x, x') is in $\mathcal{R}$ if the states $x, x' \in X$ satisfy the following conditions:

1. $E(x) \cap D(x') = E(x') \cap D(x) = \emptyset$
2. $T(x) = T(x') \Rightarrow M(x) = M(x')$

In essence, a pair of states in $\mathcal{R}$ is guaranteed not to conflict on enablement or disablement if the events are allowed to occur in the plant, nor on marking if the states correspond with marked state(s) in the plant.

A *control cover* $\mathbf{C} = \{X_i \mid i \in I\}$ on **SUPER** is defined in the same manner as the standard cover discussed previously, but with two additional requirements:

1. $(\forall\, i \in I)\, X_i \neq \emptyset \wedge (\forall\, x, x' \in X_i)\, (x, x') \in \mathcal{R}$
2. $(\forall\, i \in I)(\forall\, \sigma \in \Sigma)(\exists\, j \in I)[(\forall\, x \in X_i)\, \xi(x, \sigma)! \Rightarrow \xi(x, \sigma) \in X_j]$

As with the traditional definition of a cover, a state in **SUPER** can be a member of more than one subset in **C**. However, a *control congruence* is a control cover where each state in **SUPER** is contained in only one subset of **C**. In other words, a control congruence is a partition of X in **SUPER**.

An induced supervisor $\mathbf{J} = (I, \Sigma, \kappa, i_0, I_m)$ is then derived from **C** in a near-identical manner to that of an induced supervisor from a standard cover. The sole difference is with respect to marked states; while $X_m = \cup\{X_i \mid i \in I_m\}$ for a traditional cover, $I_m = \{i \in I \mid X_i \cap X_m \neq \emptyset\}$. This looser requirement reflects the properties enforced by $\mathcal{R}$: marking need not be uniform unless an associated state in the plant is marked as well. The end result is that **J** is control-equivalent to **SUPER**.

Arguably, the most important result in [13] is somewhat independent of control covers: proof that determining a minimal-state supervisor for any control task is NP-hard. This is a somewhat discouraging result, but there is a viable alternative. Although using control covers to determine a minimal agent is exponential-time, the same cannot be said for control congruences. Employing a partition of states reduces the time required for computation from exponential to polynomial and often results in a near-minimal agent, occasionally giving the actual minimum. The algorithm to determine a control congruence has been implemented in TCTC as **simsup**.

3.2 Research

As stated previously, one of the primary aims of this research is to prove that reducing the state-space of a set of distributed supervisors while preserving their control action will not result in an increase of communication and will often decrease the number of event transmissions. On one hand, some relationship between the original and the reduced supervisor will have to be assumed. Without some relationship, it would be extremely difficult (if not impossible) to properly compare the communication results between the two agents. On the other hand, the solution should be as inclusive as possible. There is no real purpose in proposing a solution so exclusive that it becomes impractical and essentially useless. With all of these factors in mind, the following definitions are made.

3.2.1 Language-Equivalence

Definition 3. Two supervisors $\mathbf{S}$ and $\hat{\mathbf{S}}$ are *language-equivalent* respect to a plant $\mathbf{G}$ if $L(\mathbf{S}/\mathbf{G}) = L(\hat{\mathbf{S}}/\mathbf{G})$ and $L_m(\mathbf{S}/\mathbf{G}) = L_m(\hat{\mathbf{S}}/\mathbf{G})$.

Remarks. This definition formally states a basic requirement that must be enforced when discussing the reduction of supervisors in this context. A pair of agents, one original and one reduced, should only be considered if the control action of the first is preserved in the second. Otherwise, the purpose of the reduction is negated and the result is not relevant.

Although traditional covers and projections are certainly language-equivalent, the fact that control covers are as well is not immediately obvious. However, control-equivalence implies language-equivalence between **SUPER** and **SIMSUP**. Since **SUPER** is the supremal controllable sublanguage of **SPEC** and **G**, $L(\mathbf{SUPER}/\mathbf{G}) = L(\mathbf{SUPER})$. In addition, the language generated by **SIMSUP** acting on **G** is identical to the intersection of those languages. Thus,

$$\begin{aligned} L(\mathbf{G}) \cap L(\mathbf{SIMSUP}) &= L(\mathbf{SUPER}) \\ L(\mathbf{G}) \cap L(\mathbf{SIMSUP}) &= L(\mathbf{SUPER}/\mathbf{G}) \\ L(\mathbf{SIMSUP}/\mathbf{G}) &= L(\mathbf{SUPER}/\mathbf{G}) \end{aligned}$$

The proof for the synthesized marked languages from **SUPER** and **SIMSUP** is similar. Hence, **SUPER** and **SIMSUP** are language-equivalent. Of note is the fact that **SPEC** and **SIMSUP** are not. A fuller language is generated when **SPEC** acts on the plant, of which the largest controllable subset is selected for **SUPER**.

Example At this juncture, some adjustments will be made to the running example to illustrate the concepts that will be defined in both this and subsequent chapters. To review, Figure 3.1 is a diagram of the plant $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$. Define the agent $\hat{\mathbf{S}}_\mathbf{1} = (\hat{S}_1, \hat{\phi}_1)$ as shown in Figure 3.2. Although it is nearly identical to the supervisor $\mathbf{S_1}$ given in the previous chapter, it has one vital difference: synthesis-inaccessible state 5, which will demonstrate an important point in the near future.

Now define the agent $\mathbf{S_1} = (S_1, \phi_1)$ as shown in Figure 3.3 and compare it to $\hat{\mathbf{S}}_\mathbf{1}$ with respect to $\mathbf{G}$. The first item of note is that both $\mathbf{S_1}$ and $\hat{\mathbf{S}}_\mathbf{1}$ have synthesis-inaccessible states with respect to

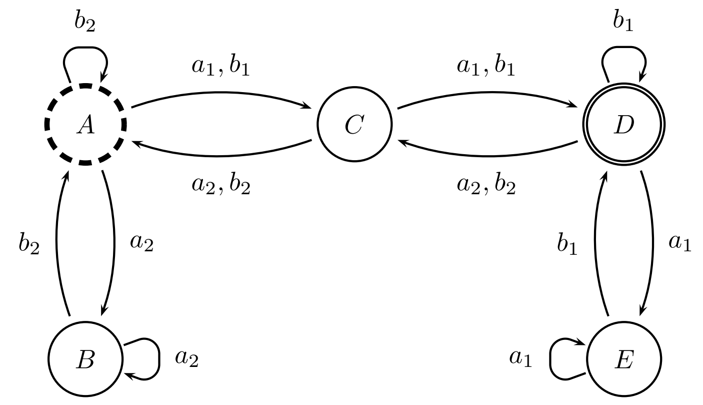


Figure 3.1: Running Example - Plant $\mathbf{G}$ (Review of Figure 2.1)

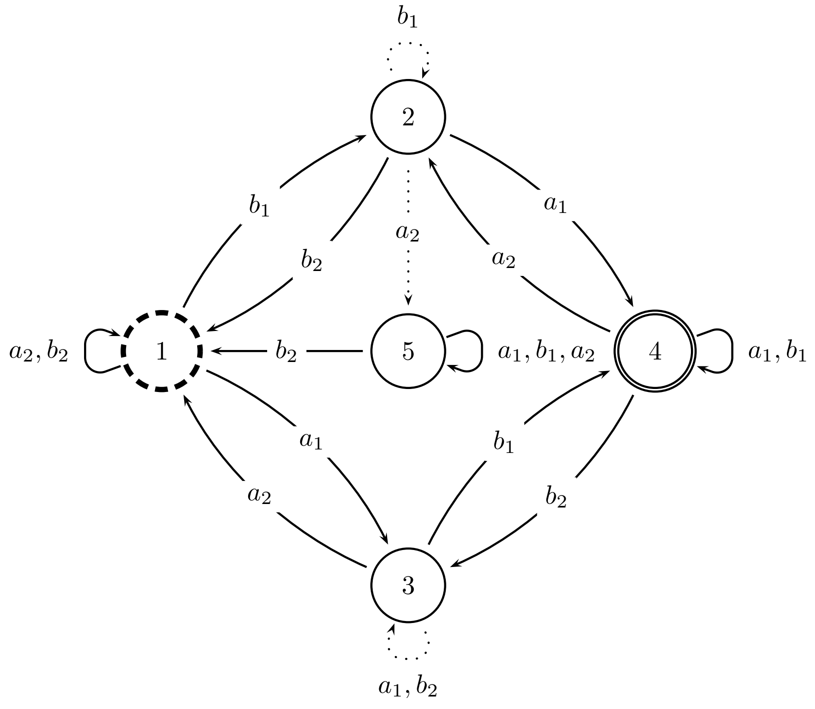


Figure 3.2: Running Example - Supervisor $\mathbf{\hat{S}_1}$

$\mathbf{G}$. The second item of note is that the first item means virtually nothing. Not only are $\mathbf{S_1}$ and $\mathbf{\hat{S}_1}$ language-equivalent with respect to $\mathbf{G}$, but the graphs $(\mathbf{S_1}/\mathbf{G})$ and $(\mathbf{\hat{S}_1}/\mathbf{G})$ that result from these agents acting on the plant are actually isomorphic, as shown in Figures 3.4 and 3.5

Although $\mathbf{S_1}$ and $\mathbf{\hat{S}_1}$ may not appear to be closely related when viewed individually, it is apparent

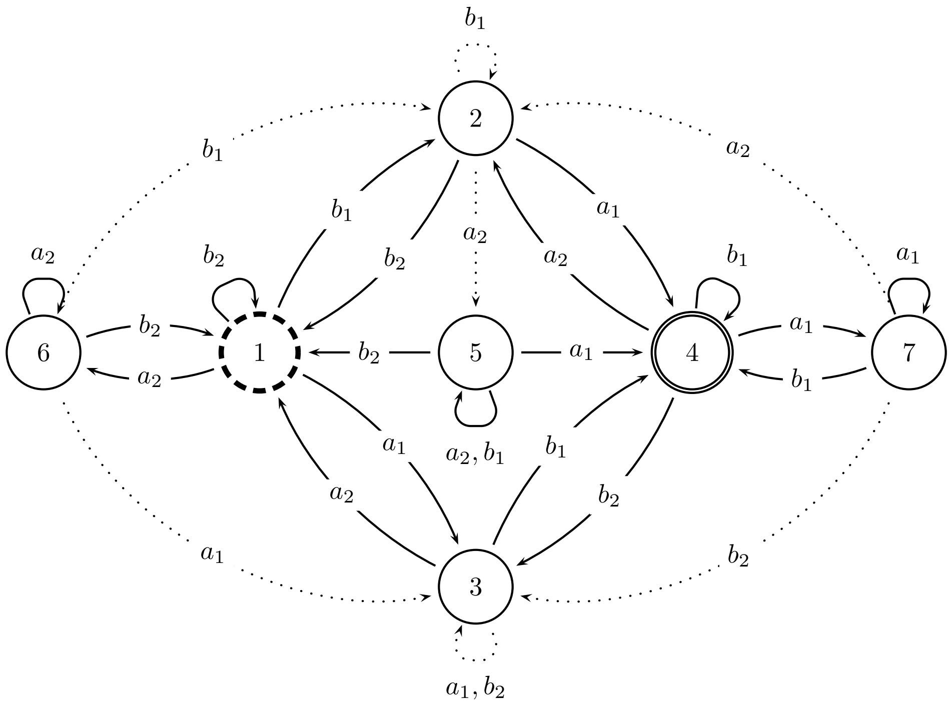


Figure 3.3: Running Example - Supervisor $\mathbf{S_1}$

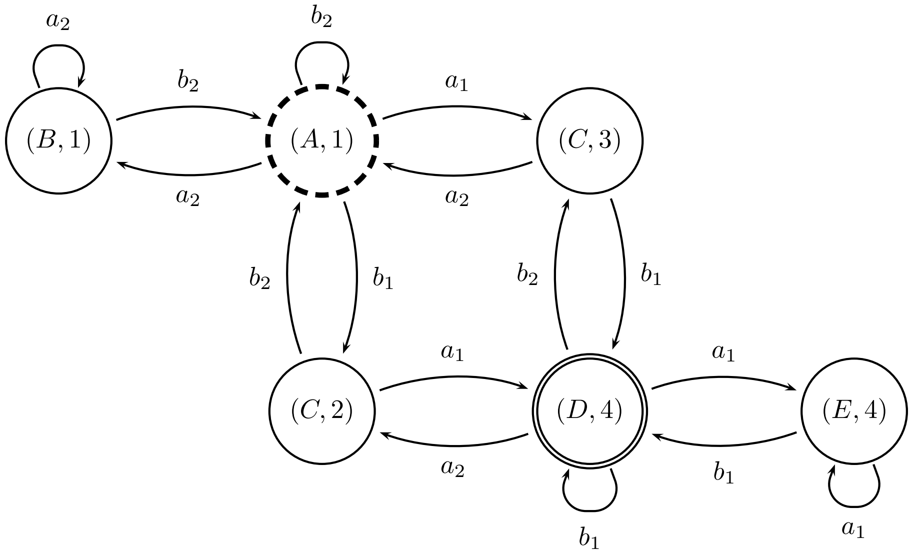


Figure 3.4: Running Example - Supervisor $\hat{\mathbf{S}}_\mathbf{1}$ Acting on Plant $\mathbf{G}$

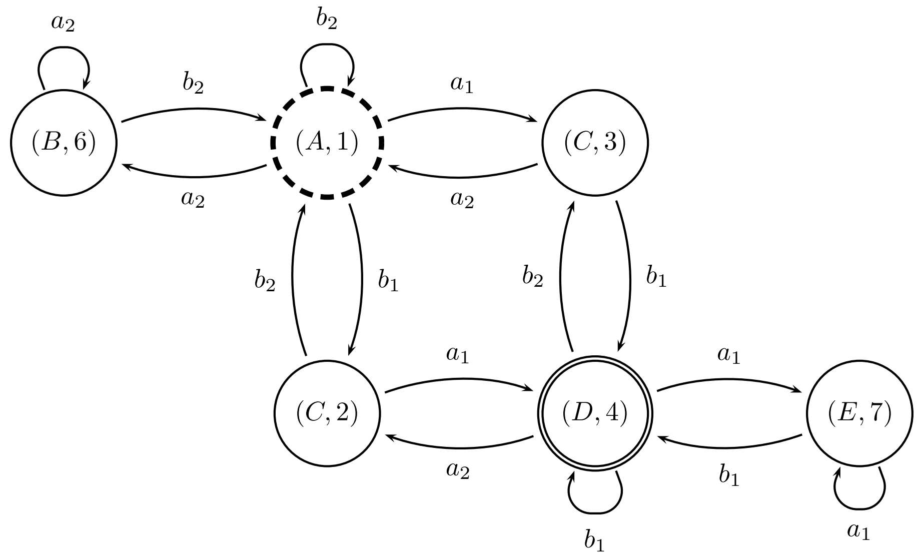


Figure 3.5: Running Example - Supervisor $\mathbf{S_1}$ Acting on Plant $\mathbf{G}$

that they have similar properties where $\mathbf{G}$ is concerned. The next step is to formally define the relationship.

3.2.2 Comparability

Definition 4. Consider two supervisors $\mathbf{S}$ and $\hat{\mathbf{S}}$ whose respective behaviours are given by $S = (\Sigma, X, \xi, x_0, X_m)$ and $\hat{S} = (\Sigma, \hat{X}, \hat{\xi}, \hat{x}_0, \hat{X}_m)$. The agents $\mathbf{S}$ and $\hat{\mathbf{S}}$ may be implicitly or explicitly-defined; in the latter case, their respective feedback maps are given by $\phi : \Sigma \times X \rightarrow \{0,1\}$ and $\hat{\phi} : \Sigma \times \hat{X} \rightarrow \{0,1\}$.

Supervisor $\mathbf{S}$ is considered to be *comparable* to $\hat{\mathbf{S}}$ with respect to $\mathbf{G}$ if there exists a state mapping $\mu : X \rightarrow \hat{X}$ that satisifies the following properties:

1. $\mathbf{S}$ and $\hat{\mathbf{S}}$ are language-equivalent with respect to $\mathbf{G}$
2. μ is surjective with respect to synthesis-accessible states
3. $\mu(x_0) = \hat{x}_0$
4. $\forall\, \sigma \in \Sigma,\, x \in X,\, \hat{\xi}(\sigma, \mu(x)) = \mu(\xi(\sigma, x))$ where $\xi(\sigma, x)!$ and x is synthesis-accessible

Remarks. This definition is essentially that of a relaxed projection. The third requirement is taken verbatim from the original definition of a projection, while the second and fourth are less stringent than in the original. Here, the transition function need only translate properly where allowed by ξ. Comparability also differs from projections in that no relationship is established between ϕ and $\hat{\phi}$. The result is a property that implies the following about $\mathbf{S}$ and $\hat{\mathbf{S}}$:

1. $\mathbf{S}$ and $\hat{\mathbf{S}}$ act in an identical manner when either is applied to $\mathbf{G}$
2. The number of synthesis-accessible states in $\hat{\mathbf{S}}$ with respect to $\mathbf{G}$ is no greater than the number of synthesis-accessible states in $\mathbf{S}$ with respect to $\mathbf{G}$
3. The initial state in $\mathbf{S}$ maps to the initial state in $\hat{\mathbf{S}}$
4. Every transition defined in $\mathbf{S}$ that originates from a synthesis-accessible state is related to a transition in $\hat{\mathbf{S}}$ where the start and end states are correctly mapped by μ [2]

Note that comparability is not a symmetric property by definition. In fact, unless μ is bijective with respect to synthesis-accessible states, it is guaranteed that $\hat{\mathbf{S}}$ is *not* comparable to $\mathbf{S}$.

The purpose of defining this property is to be as inclusive as possible with respect to the different methods of supervisor reduction. Unfortunately, neither traditional covers nor control covers will satisfy the requirements for comparability unless the cover represents a partition of the states in the original agent. The good news is that both projections and control congruences imply that their associated supervisors are comparable.

Lastly, it is convenient to define an inverse mapping to μ as many of the proofs in this document will begin with states in $\hat{X}$ as well as X. Hence, $\mu^{-1} : \hat{X} \to 2^X$ where

$$\forall\, \hat{x} \in \hat{X},\ \mu^{-1}(\hat{x}) = \{x \mid x \in X \wedge \mu(x) = \hat{x}\}$$

Example The mapping μ_1 from the states of $\mathbf{S_1}$ (Figure 3.3) to the states of $\hat{\mathbf{S}}_\mathbf{1}$ (Figure 3.2) is defined in Table 3.1. Under this state mapping, $\mathbf{S_1}$ is comparable to $\hat{\mathbf{S}}_\mathbf{1}$ with respect to $\mathbf{G}$. Note that state 5 from $\hat{\mathbf{S}}_\mathbf{1}$ is mapped to state 5 in $\mathbf{S_1}$; this preserves $\hat{\xi}_1(a_2, \mu_1(2)) = \mu_1(\xi_1(a_2, 2))$. However, the same rule is not preserved in synthesis-inaccessible states $5 \in X_1$ and $5 \in \hat{X}_1$ where

[2] The definition of this particular property was adapted from a suggestion made by Lenko Grigorov.

$x \in X_1$	$\mu_1(x)$
1	1
2	2
3	3
4	4
5	5
6	1
7	4

Table 3.1: Running Example - State Mapping μ_1

$\hat{\xi}_1(a_1, \mu_1(5)) = \hat{\xi}_1(a_1, 5) = 5$ but $\mu_1(\xi_1(a_1, 5)) = \mu_1(4) = 4$. This is permitted under comparability since neither of these states are synthesis-accessible.

To complete the example, two additional supervisors $\hat{\mathbf{S}}_\mathbf{2} = (\hat{S}_2, \hat{\phi}_2)$ and $\mathbf{S}_\mathbf{2} = (S_2, \phi_2)$ are defined as illustrated in Figures 3.6 and 3.7, respectively. Agent $\hat{\mathbf{S}}_\mathbf{2}$ is identical to supervisor $\mathbf{S}_\mathbf{2}$ from the previous chapter; its associated diagram is included here for review.

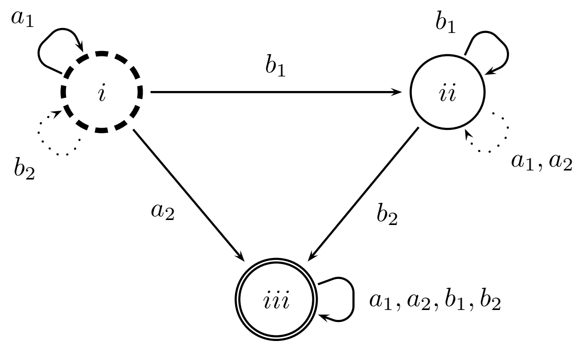


Figure 3.6: Running Example - Supervisor $\hat{\mathbf{S}}_\mathbf{2}$ (Review of Figure 2.5)

It will not come as a surprise that $\mathbf{S}_\mathbf{2}$ and $\hat{\mathbf{S}}_\mathbf{2}$ are language-equivalent with respect to $\mathbf{G}$. All that is left is to define the mapping μ_2 in Table 3.2. Under this state mapping, $\mathbf{S}_\mathbf{2}$ is comparable to $\hat{\mathbf{S}}_\mathbf{2}$ with respect to $\mathbf{G}$.

At this juncture, the primary elements of the running example are fully defined. The remainder of the example will be concerned with performing operations and analysis on these agents.

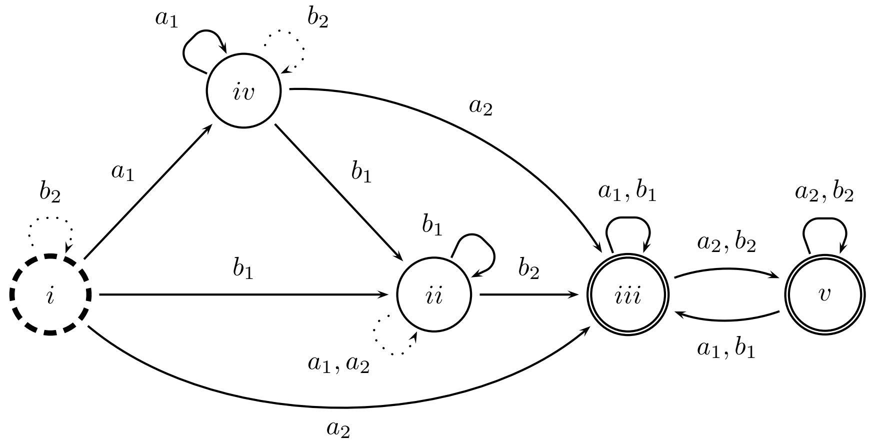


Figure 3.7: Running Example - Supervisor $\mathbf{S_2}$

$x \in X_2$	$\mu_2(x)$
i	i
ii	ii
iii	iii
iv	i
v	iii

Table 3.2: Running Example - State Mapping μ_2

Projection

Proposition 1. Consider two supervisors $\mathcal{S} = (S, \phi)$ and $\hat{\mathcal{S}} = (\hat{S}, \hat{\phi})$ where $S = (X, \Sigma, \xi, x_0, X_m)$, $\hat{S} = (\hat{X}, \Sigma, \hat{\xi}, \hat{x}_0, \hat{X}_m)$, $\phi : X \to \{0,1\}^\Sigma$ and $\hat{\phi} : \hat{X} \to \{0,1\}^\Sigma$. Let $\pi : \mathcal{S} \to \hat{S}$ denote a projection from $\mathcal{S}$ to $\hat{\mathcal{S}}$.

The existence of a projection between $\mathcal{S}$ and $\hat{\mathcal{S}}$ implies that there exists a state mapping μ such that $\mathcal{S}$ is comparable to $\hat{\mathcal{S}}$ with respect to $\mathbf{G}$.

Proof. Let μ be an identical mapping to π. It was previously established in the remarks for Definition 3 that the existence of a projection implies that $\mathcal{S}$ and $\hat{\mathcal{S}}$ are language-equivalent. Proving that μ is valid is trivial; requirements (2) and (3) for comparability are immediately satisfied due to the fact that they are identical to or less stringent than the similar requirements for a projection.

The third requirement for a projection,

$$\hat{\xi}(\sigma, \pi(x)) = \pi(\xi(\sigma, x)) \text{ for all } (\sigma, x) \text{ where } \xi(\sigma, x) \text{ is defined}$$

is nearly identical to its counterpart in comparability. However, it enforces a stricter rule in that all defined transitions must map correctly, versus only those that are defined on synthesis-accessible states. □

Control Congruence

Proposition 2. Consider two implicitly-defined supervisors **SUPER** $= (X, \Sigma, \xi, x_0, X_m)$ and **J** $= (I, \Sigma, \kappa, i_0, I_m)$ where **J** was induced from a control congruence **C** $= \{X_i \mid i \in I\}$ on **SUPER**.

The existence of a control congruence implies that there exists a state mapping μ such that **SUPER** is comparable to **J** with respect to **G**.

Proof. Note that since **C** is a control congruence, the following properties hold

$$\begin{aligned} \forall\, i \in I, \quad & X_i \subseteq X \\ \cup\{X_i \mid i \in I\} \quad & = X \\ \forall\, i, j \in I,\, i \neq j,\, X_i \cap X_j \quad & = \emptyset \end{aligned}$$

Define μ using **C**: $\forall\, x \in X$, $\mu(x) = i$ such that $x \in X_i$. To prove that μ is valid, each of the requirements for comparability will be examined in turn with respect to the properties of **C** and the corresponding induced supervisor.

1. **SUPER** and **J** are language-equivalent with respect to **G**

   It was previously established in the remarks for Definition 3 that the existence of a control congruence implies that **SUPER** and **J** are language-equivalent.

2. μ is surjective with respect to synthesis-accessible states

   By definition, a control cover is a collection of subsets of of X. Since there are $2^{|X|} - 1$ possible nonempty subsets of X, this does not necessarily imply that the mapping μ is surjective. However, a control congruence is actually a partition of states; no state from **SUPER** can appear in more than one subset of **C**. Hence, at worst μ is one-to-one and otherwise it is onto; these both satisfy the definition of a surjection for *all* states, including those that are synthesis-accessible.

3. $\mu(x_0) = \hat{x}_0$

The initial state i_0 of **J** is arbitrarily chosen; it can be any state subset X_i of which x_0 is a member. Since a control congruence is a partition of X, x_0 can only be in one possible set in **C** and the index of that set is the only choice for i_0. Hence, this requirement is also satisfied.

4. $\forall\, \sigma \in \Sigma,\ x \in X,\ \hat{\xi}(\sigma, \mu(x)) = \mu(\xi(\sigma, x))$ where $\xi(\sigma, x)!$ and x is synthesis-accessible

A control cover must satisfy the following requirement from its definition

$$(\forall\, i \in I)(\forall\, \sigma \in \Sigma)(\exists\, j \in I)[(\forall\, x \in X_i)\, \xi(x, \sigma)! \Rightarrow \xi(x, \sigma) \in X_j] \tag{3.2a}$$

Likewise, an induced supervisor defines its transition function κ from **C** as follows

$$\begin{aligned}\kappa(i, \sigma) = j \quad \text{provided} \quad & (\exists\, x \in X_i)\, \xi(x, \sigma) \in X_j \text{ and} \\ & (\forall\, x' \in X_i)[\xi(x', \sigma)! \Rightarrow \xi(x', \sigma) \in X_j]\end{aligned} \tag{3.2b}$$

Consider some $x \in X$, $\sigma \in \Sigma$ such that $\xi(x, \sigma)!$ in **SUPER** and x is synthesis-accessible. Let $y \in X$ such that $y = \xi(x, \sigma)$. Assume that $x \in X_i$ and $y \in X_j$ for some X_i, X_j in **C** and corresponding i, j in I. To prove the final requirement for comparability is satisfied, it must be shown that $\kappa(\sigma, \mu(x)) = \mu(\xi(\sigma, x))$.

$$\begin{aligned} LHS &= \kappa(\sigma, \mu(x)) \\ &= \kappa(\sigma, i) && \text{(Assumption)} \\ &= j && \text{((3.2a) and (3.2b))} \end{aligned}$$

$$\begin{aligned} RHS &= \mu(\xi(\sigma, x)) \\ &= \mu(y) && \text{(Assumption)} \\ &= j && \text{(Assumption)} \end{aligned}$$

□

Associated Properties

Lemma 1. *Consider two supervisors* $\mathbf{S} = (S, \phi)$ *and* $\hat{\mathbf{S}} = (\hat{S}, \hat{\phi})$ *where* $S = (\Sigma, X, \xi, x_0, X_m)$ *and* $\phi : \Sigma \times X \to \{0, 1\}$*;* $\hat{S}$ *and* $\hat{\phi}$ *are similarly defined. If* $\mathbf{S}$ *is comparable to* $\hat{\mathbf{S}}$ *with respect to* $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$ *via* μ*,*

(i) $\forall\, x \in X, \forall\, \hat{x} \in \hat{X}, \; \mu(x) = \hat{x} \Rightarrow [x] \subseteq [\hat{x}]$

(ii) $\forall\, \hat{x} \in \hat{X}, \; [\hat{x}] = \bigcup_{x \in \mu^{-1}(\hat{x})} [x]$

Proof: (i). Since $\mathbf{S}$ is comparable to $\hat{\mathbf{S}}$ with respect to $\mathbf{G}$, the following holds for all $x \in X$ and $s \in [x]$,

$$\begin{aligned} \hat{\xi}(s, \mu(x_0)) &= \mu(\xi(s, x_0)) \\ \hat{\xi}(s, \hat{x}_0) &= \mu(x) && (s \in [x]) \\ \hat{\xi}(s, \hat{x}_0) &= \hat{x} && (\text{Given}) \end{aligned}$$

The above result proves that every string in a given $[x]$ will also lead to $\hat{x}$ where $\mu(x) = \hat{x}$. Hence, $[x] \subseteq [\hat{x}]$. □

Proof: (ii). There are two parts to this proposition: $[\hat{x}] \subseteq \bigcup_{x \in \mu^{-1}(\hat{x})} [x]$ and $\bigcup_{x \in \mu^{-1}(\hat{x})} [x] \subseteq [\hat{x}]$. The second follows from (i), but the first requires a more in-depth examination.

Consider any $s \in [\hat{x}]$; by definition, $s \in L(\hat{\mathbf{S}}/\mathbf{G})$. Since $\mathbf{S}$ is comparable to $\hat{\mathbf{S}}$, $s \in L(\mathbf{S}/\mathbf{G})$ and thus $\xi(s, x_0)$ is defined. Suppose $\xi(s, x_0) = y$ where $\mu(y) \neq \hat{x}$ and thus $y \notin \mu^{-1}(\hat{x})$. This implies that $s \in [y]$. Since $\mathbf{S}$ is comparable to $\hat{\mathbf{S}}$,

$$\begin{aligned} \hat{\xi}(s, \mu(x_0)) &= \mu(\xi(s, x_0)) \\ \hat{\xi}(s, \hat{x}_0) &= \mu(\xi(s, x_0)) && (\mu(x_0) = \hat{x}_0 \text{ since } \mathbf{S} \text{ is comparable to } \hat{\mathbf{S}}) \\ \hat{\xi}(s, \hat{x}_0) &= \mu(y) && (\text{Assumption that } \xi(s, x_0) = y) \\ \hat{x} &= \mu(y) && (s \in [\hat{x}]) \end{aligned}$$

The above is a clear contradiction to the assumption that $\mu(y) \neq \hat{x}$. Thus, the reverse is true and $y \in \mu^{-1}(\hat{x})$. Since $s \in [y]$, this means that $s \in \bigcup_{x \in \mu^{-1}(\hat{x})} [x]$. Therefore, $[\hat{x}] \subseteq \bigcup_{x \in \mu^{-1}(\hat{x})} [x]$. □

Example To add a somewhat less formal dimension to the items proven in Lemma 1, the following list will provide examples of each point from $\mathbf{S_1}$ and $\hat{\mathbf{S}}_\mathbf{1}$ or $\mathbf{S_2}$ and $\hat{\mathbf{S}}_\mathbf{2}$. Note that the strings used in these examples are taken from $L(\mathbf{S_1}/\mathbf{G})$ or $L(\mathbf{S_2}/\mathbf{G})$.

(i) For $6, 1 \in X_1$ and $1 \in \hat{X}_1$ where $\mu_1(6) = \mu_1(1) = 1$, $a_2 \in [6]$ and $a_2b_2 \in [1]$ while both $a_2, a_2b_2 \in [1]$

(ii) For $i \in \hat{X}_2$ and $i, iv \in X_2$ where $\mu_2^{-1}(i) = \{i, iv\}$, $[i] = (a_1)^*$ which is the union of $[iv] = a_1(a_1)^*$ and $[i] = \{\epsilon\}$

3.2.3 Reachable Product

As discussed in the previous chapter, the operation used to combine decentralized supervisors into a centralized counterpart is known as *product*. Incidentally, product is essentially the first (and possibly the most important) operation performed in the minimal communication algorithm devised by Rudie, Lafortune and Lin in [10]. Given two sets of supervisors ($\mathbf{S_1}$, $\mathbf{S_2}$) and ($\hat{\mathbf{S}}_\mathbf{1}$, $\hat{\mathbf{S}}_\mathbf{2}$) as input for the algorithm where $\mathbf{S_i}$ is comparable to $\hat{\mathbf{S}}_\mathbf{i}$ with respect to $\mathbf{G}$, it is vital to establish a similar relationship between ($\mathbf{S_1} \times \mathbf{S_2}$) and ($\hat{\mathbf{S}}_\mathbf{1} \times \hat{\mathbf{S}}_\mathbf{2}$). Happily, it is the case that comparability "survives" product, the proof of which is the subject of this section.

Definition 5. Consider two pairs of supervisors $(\mathbf{S_1}, \mathbf{S_2})$ and $(\hat{\mathbf{S}}_\mathbf{1}, \hat{\mathbf{S}}_\mathbf{2})$ where $\mathbf{S_i}$ is comparable to $\hat{\mathbf{S}}_\mathbf{i}$ via μ_i with respect to a plant $\mathbf{G}$ ($i \in \{1, 2\}$). The *reachable* product $(\mathbf{S_1} \times \mathbf{S_2})$ is represented by $\mathbf{S} = (S, \phi)$ and is defined as follows for $S = (\Sigma, X, \xi, x_0, X_m)$ and $\phi : \Sigma \times X \to \{0, 1\}$:

$$\begin{aligned}
\Sigma &= \Sigma_1 \cup \Sigma_2 \\
x_0 &= (x_{1,0}, x_{2,0}) \\
X &= \{(x_1, x_2) \mid x_1 \in X_1 \wedge x_2 \in X_2 \wedge \\
&\qquad (\exists\, s \in \Sigma^*)(\xi_1(s, x_{1,0}) = x_1 \wedge \xi_2(s, x_{2,0}) = x_2))\} \\
\forall\, \sigma \in \Sigma,\, x \in X,\, \xi(\sigma, x) &= (\xi_1(\sigma, x_1), \xi_2(\sigma, x_2)) \\
X_m &= \{x = (x_1, x_2) \mid x \in X \wedge x_1 \in X_{1,m} \wedge x_2 \in X_{2,m}\}
\end{aligned}$$

$$\forall\, \sigma \in \Sigma,\, x \in X,\, \phi(\sigma, x) = \begin{cases} 0 & \text{if } \phi_1(\sigma, x) = 0 \vee \phi_2(\sigma, x) = 0 \\ 1 & \text{otherwise} \end{cases}$$

The reachable product $(\hat{\mathbf{S}}_\mathbf{1} \times \hat{\mathbf{S}}_\mathbf{2})$ is denoted by $\hat{\mathbf{S}} = (\hat{S}, \hat{\phi})$. Note that the reachable product $\mathbf{S}$ is equivalent to the conjunction of $\mathbf{S_1}$ and $\mathbf{S_2}$ and thus $L(\mathbf{S}/\mathbf{G}) = L((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G})$. Although it may seem redundant, this definition is included to match this operation to the one used in [10]: the "accessible" part of the "product" of the two agents.

Define the state mapping $\mu : X \to \hat{X}$ where $\mu = (\mu_1 \times \mu_2)$ as follows

$$\forall\, \sigma \in \Sigma,\, x \in X,\, \mu(x) = (\mu_1(x_1), \mu_2(x_2)) \text{ where } x = (x_1, x_2),\ x_1 \in X_1,\, x_2 \in X_2$$

Example The diagrams in Figures 3.8 and 3.9 represent the reachable products $\hat{\mathbf{S}} = (\hat{\mathbf{S}}_\mathbf{1} \times \hat{\mathbf{S}}_\mathbf{2}) = (\hat{S}, \hat{\phi})$ and $\mathbf{S} = (\mathbf{S_1} \times \mathbf{S_2}) = (S, \phi)$, respectively.

It is somewhat impressive how quickly the benefits of supervisor reduction emerge; while Figure 3.8 is reasonable in appearance and complexity for the purposes of an example for human visualization, Figure 3.9 very nearly is not. This is a typical case of state-space explosion, which is difficult to avoid when performing operations such as the reachable product. It is fortunate to have state-size reduction tools (such as those described in this document) that help alleviate the problem, but unfortunate that a more general solution does not exist.

Associated Properties

Lemma 2. *Using the parameters and properties defined in Definition 5,*

(i) $\forall\, x \in X,\, [x] = [x_1] \cap [x_2]$ *where* $x = (x_1, x_2),\ x_1 \in X_1,\, x_2 \in X_2$

(ii) $L(\mathbf{S}/\mathbf{G}) = L(\hat{\mathbf{S}}/\mathbf{G})$

(iii) $\forall\, \hat{x} \in \hat{X},\, [\hat{x}] = \bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\}$

Proof: (i). Consider that for the above to be true, $[x] \subseteq [x_1] \cap [x_2]$ must also be true. Assume

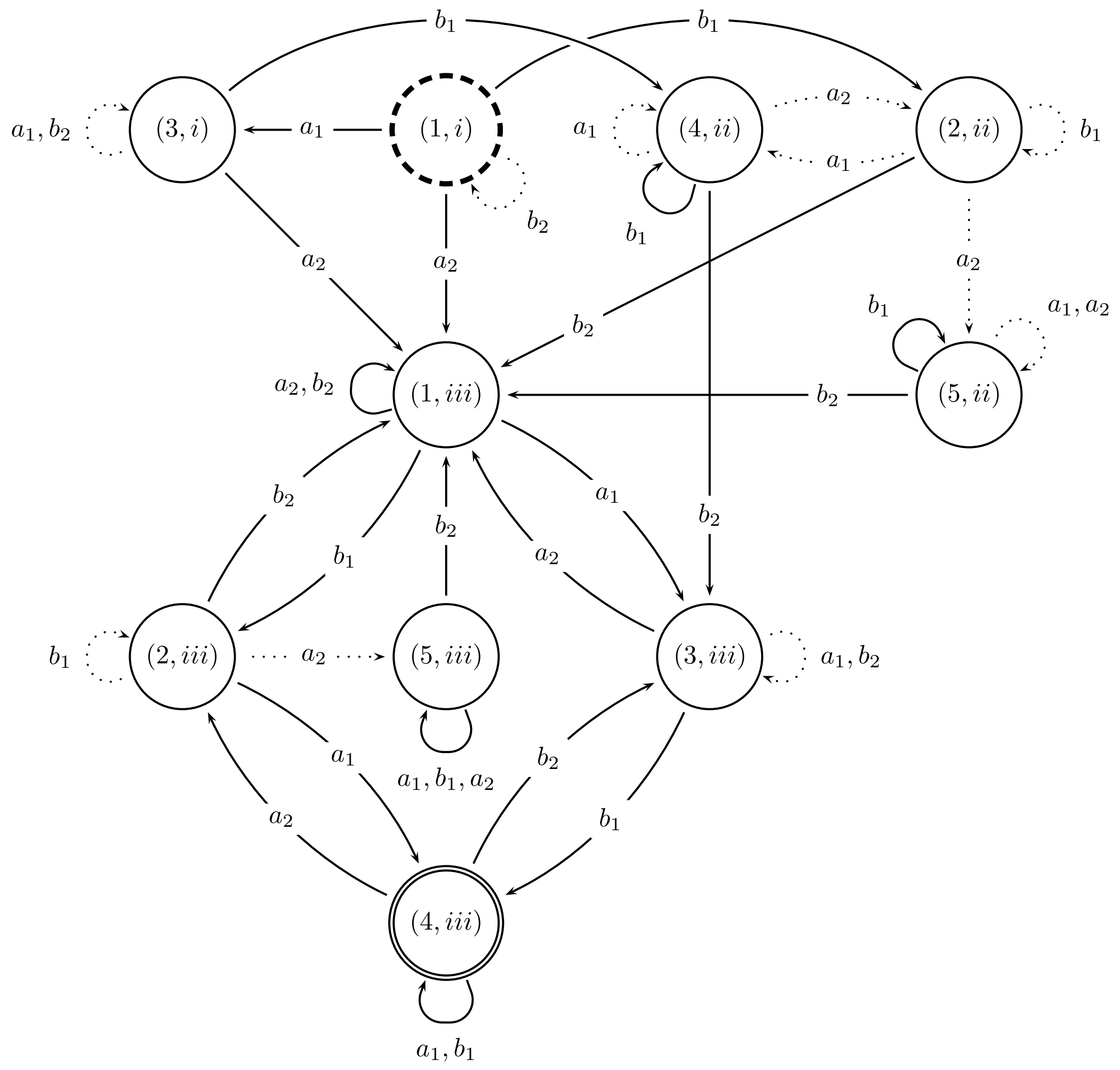


Figure 3.8: Running Example - Reachable Product of $\hat{\mathbf{S}}_\mathbf{1}$ and $\hat{\mathbf{S}}_\mathbf{2}$

$\exists\, s \in [x]$ where $s \notin [x_1] \cap [x_2]$ such that $x = (x_1, x_2)$. This implies that $s \notin [x_1]$ or $s \notin [x_2]$ or both.

$$\begin{aligned}
s \in [x] \quad &\Rightarrow \quad s \in L(\mathbf{S_1} \wedge \mathbf{S_2}/\mathbf{G}) \wedge \xi(s, x_0) = x && \text{(Definition 2)}\\
&\Rightarrow \quad s \in L(\mathbf{S_1}/\mathbf{G}) \wedge s \in L(\mathbf{S_2}/\mathbf{G}) && \text{(Definition of } L(\mathbf{S_1} \wedge \mathbf{S_2}/\mathbf{G}))\\
&\qquad \wedge\, \xi(s, x_0) = x\\
&\Rightarrow \quad s \in L(\mathbf{S_1}/\mathbf{G}) \wedge L(\mathbf{S_2}/\mathbf{G}) && \text{(Definition 5)}\\
&\qquad \wedge\, (\xi_1(s, x_{1,0}), \xi_2(s, x_{2,0})) = (x_1, x_2)\\
&\Rightarrow \quad s \in L(\mathbf{S_1}/\mathbf{G}) \wedge \xi_1(s, x_{1,0}) = x_1\\
&\qquad \wedge\, s \in L(\mathbf{S_2}/\mathbf{G}) \wedge \xi_2(s, x_{2,0}) = x_2
\end{aligned}$$

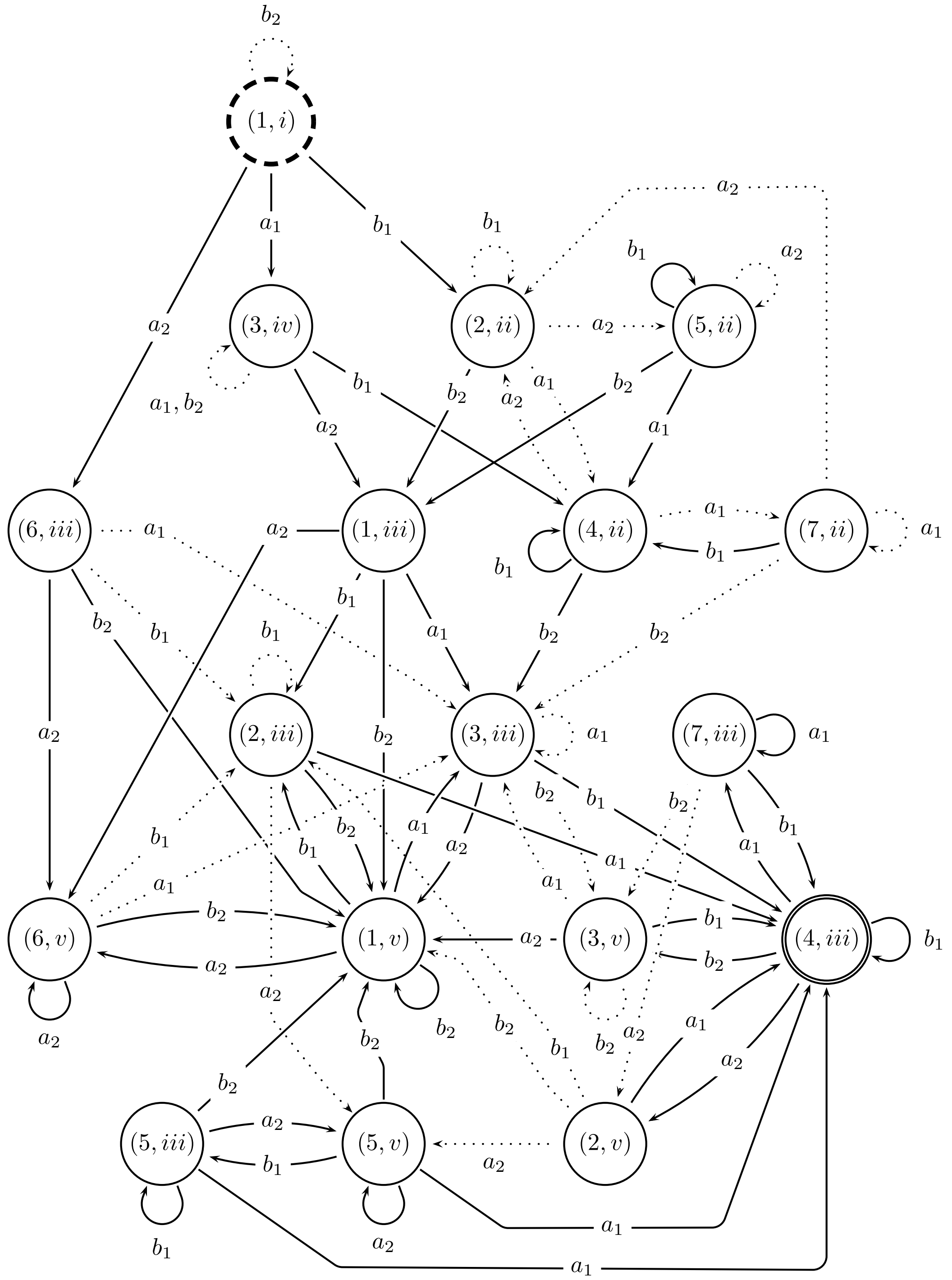


Figure 3.9: Running Example - Reachable Product of $\mathbf{S_1}$ and $\mathbf{S_2}$

If $s \in L(\mathbf{S_1}/\mathbf{G})$ and $\xi_1(s, x_{1,0}) = x_1$, then $s \in [x_1]$ (and similarly for $[x_2]$). This contradicts the assumption and thus proves $[x] \subseteq [x_1] \cap [x_2]$ to be true.

Now consider the reverse of the above: $[x_1] \cap [x_2] \subseteq [x]$. Assume $\exists\, s \in [x_1] \cap [x_2]$ where $s \notin [x]$ such that $x = (x_1, x_2)$. This implies that $s \in [x_1]$ and $s \in [x_2]$. Since $s \in [x_1]$, by the definition of $[x_1]$ the following are all true: $\xi_1(s, x_{1,0})!$, $\phi_1(s, x_{1,0}) \neq 0$ and $\xi_1(s, x_{1,0}) = x_1$. A similar statement can be made with respect to s and $[x_2]$.

$$\begin{aligned}
\xi(s, x_0) &= (\xi_1(s, x_{1,0}), \xi_2(s, x_{2,0})) && \text{(Definition 5)}\\
&= (x_1, x_2) && \text{(Definitions of } [x_1] \text{ and } [x_2])\\
&= x && \text{(Assumption)}
\end{aligned}$$

If $\xi(s, x_0) = x$ and $s \in L(\mathbf{S_1} \wedge \mathbf{S_2}/\mathbf{G})$, then $s \in [x]$. This contradicts the assumption and thus proves $[x_1] \cap [x_2] \subseteq [x]$ to be true.

Finally, since $[x] \subseteq [x_1] \cap [x_2]$ and $[x_1] \cap [x_2] \subseteq [x]$ have both been proven to be true, then $[x] = [x_1] \cap [x_2]$ must also be true. □

Proof: (ii). Consider that for $L(\mathbf{S}/\mathbf{G}) = L(\hat{\mathbf{S}}/\mathbf{G})$ to be true, $L(\mathbf{S}/\mathbf{G}) \subseteq L(\hat{\mathbf{S}}/\mathbf{G})$ must also be true. For any $s \in L(\mathbf{S}/\mathbf{G})$,

$$\begin{aligned}
s \in L(\mathbf{S}/\mathbf{G}) &\Rightarrow s \in [x] \text{ for some } x \in X && \text{(Definition 2)}\\
&\Rightarrow s \in [x_1] \;\wedge\; s \in [x_2] && (x = (x_1, x_2))\\
&\Rightarrow s \in [\mu_1(x_1)] \;\wedge\; s \in [\mu_2(x_2)] && \text{(Lemma 1(i))}\\
&\Rightarrow s \in [\mu(x)] && \text{(Definition of } \mu)\\
&\Rightarrow s \in L(\hat{\mathbf{S}}/\mathbf{G})
\end{aligned}$$

The proof for $L(\hat{\mathbf{S}}/\mathbf{G}) \subseteq L(\mathbf{S}/\mathbf{G})$ proceeds in a near-identical manner, except Lemma 1(ii) is used at the third step.

Thus, if $L(\mathbf{S}/\mathbf{G}) \subseteq L(\hat{\mathbf{S}}/\mathbf{G})$ and $L(\hat{\mathbf{S}}/\mathbf{G}) \subseteq L(\mathbf{S}/\mathbf{G})$ are both true, then $L(\mathbf{S}/\mathbf{G}) = L(\hat{\mathbf{S}}/\mathbf{G})$ must also be true. □

Proof: (iii). This proof will proceed in a near-identical manner to that in (i).

Consider that for $[\hat{x}] = \bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\}$ to be true, $[\hat{x}] \subseteq \bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\}$

must also be true. Assume $\exists\, s \in [\hat{x}]$ where $s \notin \bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\}$; note that $\xi(s, x_0)!$ due to (ii). This implies that $s \notin [x]$ for all x where $\mu(x) = \hat{x}$ and that $\xi(s, x_0) = y$ where $\mu(y) \neq \hat{x}$. Let $\hat{x} = (\hat{x}_1, \hat{x}_2)$ where $\hat{x}_i \in \hat{X}_i$, let $y = (y_1, y_2)$ where $y_i \in X_i$, and let $\mu(y) = \hat{y} = (\hat{y}_1, \hat{y}_2)$.

$$\begin{aligned}
s \in [y] &\Rightarrow s \in [y_1] \cap [y_2] && \text{(i)}\\
&\Rightarrow s \in [y_1] \wedge s \in [y_2] && \text{(Definition of } \cap\text{)}\\
&\Rightarrow s \in [\hat{y}_1] \wedge s \in [\hat{y}_2] && \text{(Lemma 1(i))}\\
&\Rightarrow s \in [\hat{y}_1] \cap [\hat{y}_2] &&\\
&\Rightarrow s \in [\hat{y}] &&
\end{aligned}$$

Since $s \in [\hat{x}]$, the above result implies that $\hat{x} = \hat{y}$ (Definition 2, Fact 2) and thus $\hat{x} = \mu(y)$. This contradicts the assumption and proves $[\hat{x}] \subseteq \bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\}$ to be true.

Now consider the reverse of the above: $\bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\} \subseteq [\hat{x}]$. Take any $s \in \bigcup[x]$; there must exist a $x = (x_1, x_2)$ such that $\mu(x) = \hat{x}$ and $s \in [x]$.

$$\begin{aligned}
s \in [x] &\Rightarrow s \in [x_1] \cap [x_2] && \text{(i)}\\
&\Rightarrow s \in [x_1] \wedge s \in [x_2] && \text{(Definition of } \cap\text{)}\\
&\Rightarrow s \in [\hat{x}_1] \wedge s \in [\hat{x}_2] && \text{(Lemma 1(i))}\\
&\Rightarrow s \in [\hat{x}_1] \cap [\hat{x}_2] &&\\
&\Rightarrow s \in [\hat{x}] &&
\end{aligned}$$

Finally, since $[\hat{x}] \subseteq \bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\}$ and $\bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\} \subseteq [\hat{x}]$ have both been proven to be true, then $[\hat{x}] = \bigcup\{[x] \mid x \in X \wedge \mu(x) = \hat{x}\}$ must also be true. □

Example For the items proven in Proposition 2, the following list will provide examples of all but one from reachable products $\hat{\mathbf{S}}$ and $\mathbf{S}$ as shown in Figures 3.8 and 3.9, respectively. Note that for the strings used in these examples, $s \in L(\mathbf{S}/\mathbf{G}) \Rightarrow s \in L((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G})$ and similarly for $\hat{\mathbf{S}}$, $\hat{\mathbf{S}}_\mathbf{1}$ and $\hat{\mathbf{S}}_\mathbf{2}$.

(i) For $3 \in X_1$ and $iv \in X_2$, sets $[3]$ and $[iv]$ are nearly disjoint. However, they do share one element: a_1, which is the sole element in $[(3, iv)]$.

(iii) For $(1, iii), (6, iii) \in X$ and $(1, iii) \in \hat{X}$ where $\mu((1, iii)) = \mu((6, iii)) = (1, iii)$, $[(1, iii)] =$

$\{a_1a_2, b_1b_2\}$ and $[(6, iii)] = \{a_2\}$, which are both subsets of $[(1, iii)]$ in $\mathbf{\hat{S}}$.

Language-Equivalence

Proposition 3. Consider two sets of distributed agents $(\mathbf{S_1}, \mathbf{S_2})$ and $(\mathbf{\hat{S}_1}, \mathbf{\hat{S}_2})$ where $\mathbf{S_i}$ is comparable to $\mathbf{\hat{S}_i}$ with respect to $\mathbf{G}$ through the state mapping μ_i. Let $\mathbf{S}$ and $\mathbf{\hat{S}}$ represent the reachable products $(\mathbf{S_1} \times \mathbf{S_2})$ and $(\mathbf{\hat{S}_1} \times \mathbf{\hat{S}_2})$.

Using the definitions of reachable product and the state mapping μ outlined in Definition 5 and Proposition 2, respectively, $\mathbf{S}$ is language-equivalent to $\mathbf{\hat{S}}$ with respect to $\mathbf{G}$.

Proof. The fact that $L(\mathbf{S}/\mathbf{G}) = L(\mathbf{\hat{S}}/\mathbf{G})$ was proven in Lemma 2(ii). Additionally, it must be proven that $L_m(\mathbf{S}/\mathbf{G}) = L_m(\mathbf{\hat{S}}/\mathbf{G})$.

$$\begin{aligned}
s \in L_m(\mathbf{S}/\mathbf{G}) \quad &\Leftrightarrow \quad s \in L_m((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G}) && \text{(Definition 5)}\\
&\Leftrightarrow \quad s \in L_m(\mathbf{S_1}/\mathbf{G}) \wedge s \in L_m(\mathbf{S_2}/\mathbf{G}) && \text{(Definition of } \wedge)\\
&\Leftrightarrow \quad s \in L_m(\mathbf{\hat{S}_1}/\mathbf{G}) \wedge s \in L_m(\mathbf{\hat{S}_2}/\mathbf{G}) && \text{(Language-Equivalence via } \mu_i)\\
&\Leftrightarrow \quad s \in L_m((\mathbf{\hat{S}_1} \wedge \mathbf{\hat{S}_2})/\mathbf{G}) && \text{(Definition of } \wedge)\\
&\Leftrightarrow \quad s \in L_m(\mathbf{\hat{S}}/\mathbf{G}) && \text{(Definition 5)}
\end{aligned}$$

□

Example From Proposition 3, it is apparent that $\mathbf{S}$ is language-equivalent to $\mathbf{\hat{S}}$ with respect to $\mathbf{G}$. Unlike $(\mathbf{\hat{S}_1}/\mathbf{G})$ and $(\mathbf{S_1}/\mathbf{G})$ in Figures 3.4 and 3.5, respectively, the graphs $(\mathbf{\hat{S}}/\mathbf{G})$ and $(\mathbf{S}/\mathbf{G})$ are not isomorphic, as shown in Figures 3.10 and 3.11.

Comparability

Theorem 1. *Using the parameters outlined in Proposition 3,* $\mathbf{S}$ *is comparable to* $\mathbf{\hat{S}}$ *with respect to* $\mathbf{G}$ *via* μ.

Proof. To prove that μ satisfies the conditions for comparability, each of the requirements for comparability will be examined in turn with respect to the properties of $\mathbf{S_i}$ and $\mathbf{\hat{S}_i}$.

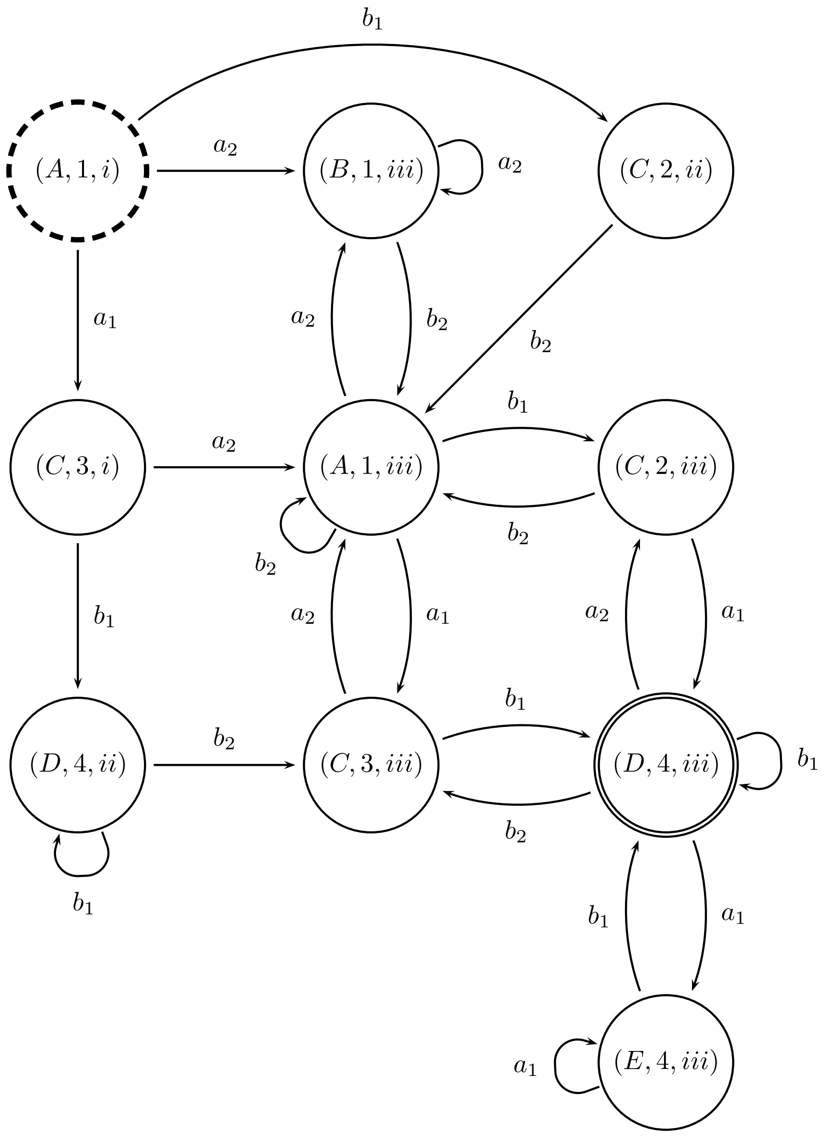


Figure 3.10: Running Example - Supervisor $\mathbf{\hat{S}} = (\mathbf{\hat{S}_1} \times \mathbf{\hat{S}_2})$ Acting on Plant $\mathbf{G}$

1. $\mathbf{S}$ and $\mathbf{\hat{S}}$ are language-equivalent with respect to $\mathbf{G}$

   This was proven in Proposition 3.

2. μ is surjective with respect to synthesis-accessible states

   For the above statement to be true, μ must satisfy the following property: $\forall \hat{x} \in \hat{X}$ where $\hat{x}$ is

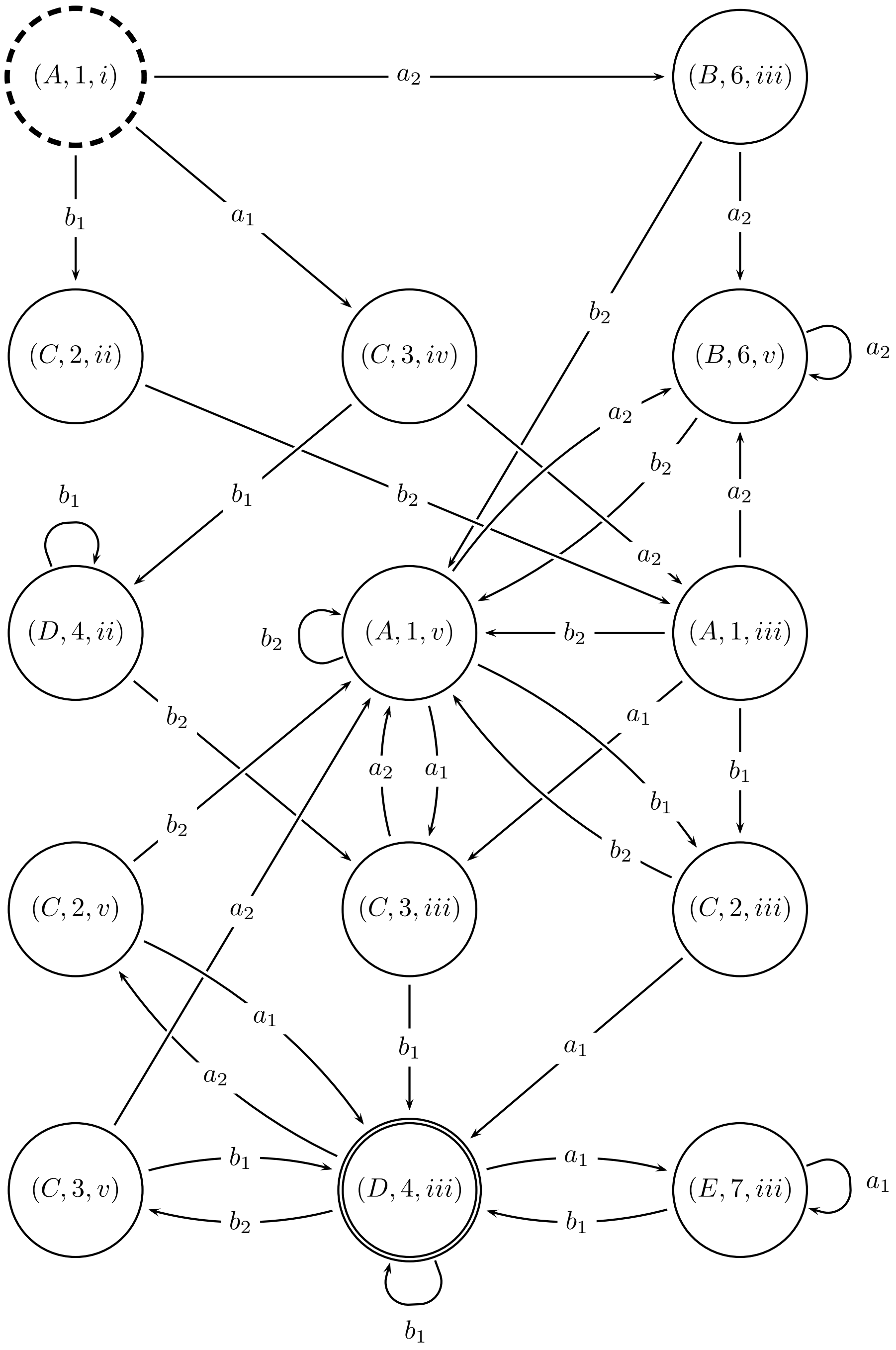


Figure 3.11: Running Example - Supervisor $\mathbf{S} = (\mathbf{S_1} \times \mathbf{S_2})$ Acting on Plant $\mathbf{G}$

synthesis-accessible, $\exists\, x \in X$ such that $\mu(x) = \hat{x}$ and x is synthesis-accessible.

Assume there exists some $\hat{x} \in \hat{X}$ that is synthesis-accessible, but every state in $\mu^{-1}(\hat{x}) = \{x_1, x_2, \ldots, x_n\}$ is not. This implies that $[\hat{x}] \neq \emptyset$ but $[x] = \emptyset$ for all $x \in \mu^{-1}(\hat{x})$. However, this contradicts Lemma 2(iii). Thus, the assumption is false and the original statement is true.

3. $\mu(x_0) = \hat{x}_0$

   This is true by definition; $\mu(x_0) = (\mu_1(x_{1,0}), \mu_2(x_{2,0})) = (\hat{x}_{1,0}, \hat{x}_{2,0}) = \hat{x}_0$.

4. $\forall\, \sigma \in \Sigma,\ x \in X,\ \hat{\xi}(\sigma, \mu(x)) = \mu(\xi(\sigma, x))$ where $\xi(\sigma, x)!$ and x is synthesis-accessible

   Consider all $\sigma \in \Sigma$ and $x \in X$ where x is synthesis-accessible and $\xi(\sigma, x)$ is defined. The fact that $\xi(\sigma, x)$ is defined implies that $\xi_1(\sigma, x)$ is defined and $\xi_2(\sigma, x)$ is defined.

   For all transitions that satisfy the above criteria, examine both sides of the equation $\hat{\xi}(\sigma, \mu(x)) = \mu(\xi(\sigma, x))$.

$$\begin{aligned}
LHS &= \hat{\xi}(\sigma, \mu(x)) \\
&= \hat{\xi}(\sigma, \mu(x_1, x_2)) && (x = (x_1, x_2)) \\
&= \hat{\xi}(\sigma, (\mu_1(x_1), \mu_2(x_2))) && (\text{Definition of } \mu) \\
&= (\hat{\xi}_1(\sigma, \mu_1(x_1)), \hat{\xi}_2(\sigma, \mu_2(x_2))) && (\text{Definition of } \hat{\xi})
\end{aligned}$$

$$\begin{aligned}
RHS &= \mu(\xi(\sigma, x)) \\
&= \mu(\xi(\sigma, (x_1, x_2))) && (x = (x_1, x_2)) \\
&= \mu(\xi_1(\sigma, x_1), \xi_2(\sigma, x_2)) && (\text{Definition of } \xi) \\
&= (\mu_1(\xi_1(\sigma, x_1)), \mu_1(\xi_2(\sigma, x_2))) && (\text{Definition of } \mu)
\end{aligned}$$

   Since $\mathbf{S_i}$ is comparable to $\mathbf{\hat{S}_i}$ with respect to $\mathbf{G}$, $\hat{\xi}_i(\sigma, \mu_i(x_i)) = \mu_i(\xi_i(\sigma, x_i))$. This implies that $LHS = RHS$ and that the requirement is satisfied.

□

Example Using Theorem 1, it can be concluded that $\mathbf{S}$ is comparable to $\mathbf{\hat{S}}$ with respect to $\mathbf{G}$ via the state mapping $\mu = (\mu_1 \times \mu_2)$.

Now that the “rules and regulations” associated with supervisor reduction have been established, focus will be shifted to the minimal communication algorithm for which the reduction is intended. An introduction and examination of the algorithm is the subject of the next chapter.

Chapter 4

Minimal Communication

4.1 Background

Although modular supervision has its advantages, it also has a significant disadvantage: distributed agents may not be able to safely implement the same level of control as a centralized supervisor. The crux of the problem is that different event sequences generated by the plant may appear identical to a supervisor who can only see certain events. If one of those sequences is illegal, the supervisor will disable it and all others that appear to be the same. Some synthesized languages satisfy certain conditions that allow them to avoid this issue, as shown in [7], [2], [11] and [6]. However, if the desired behaviour does *not* meet these conditions, the agents will need to know the occurrences of events that help them distinguish legal sequences from illegal ones. If a supervisor cannot observe these directly, but another supervisor can, it must be the responsibility of the latter agent to communicate those events to the former.

4.1.1 Issues With Communication

Introducing communication into DESs is not necessarily a panacea; it solves the issue at hand but introduces two classic network problems. The first is *latency*, where transmissions are delayed and can arrive later than expected or required. In DESs, this could result in dire consequences. Consider $s \in \Sigma^*$ and $\alpha, \beta \in \Sigma$ where $s\alpha\beta$ is legal and $s\beta\alpha$ is not. If α is communicated to a local agent but experiences a delay, and β is observed before α arrives, then the agent would believe $s\beta$ had occurred and incorrectly disable α. The second concern is that of *reliability*; no communication network is guaranteed to function perfectly in all circumstances. If a malfunction occurs and a transmission between agents is lost, coordination between the distributed supervisors is essentially lost as well; the $s\beta$ scenario given earlier applies to this situation as well. Unfortunately, the occurrence of either of these scenarios could result in the modular agents applying a control scheme that is not appropriate for the actions of the system.

At the moment, no framework has been introduced in DESs which can detect and resolve the delay or loss of event transmissions. Focus has been placed on first *minimizing* the number of communications that must be sent between agents. This is a logical approach to take since, from a security perspective, a reduction in the number of messages that must be sent also reduces the probability that those messages will be intercepted. With respect to the network, the fewer transmissions that must be sent, the less likely it is that a loss or delay will adversely affect the system. Until such a framework is introduced, it will be assumed that the communication network used by the supervisors does *not* experience latency or transmission loss. In a real-world scenario however, priorities would be established and a balance would be achieved where the capacity for control is somewhat diminished but some communication is performed to ensure the most vital control policies are preserved.

4.1.2 The Algorithm

At this juncture, focus will be placed on the method used to determine a minimal communication scheme between two agents, introduced in [10] and hereafter known as "the algorithm". In fact, "the" is an appropriate article as no other function that accomplishes the same task has yet been proposed.

Agents

The minimal communication algorithm only requires one thing to perform its job: a pair of distributed discrete-event supervisors ($\mathbf{R_1}$, $\mathbf{R_2}$) where $\mathbf{R_i} = (R_i, \psi_i)$, $R_i = (\Sigma, X_i, \xi_i, x_{i,0})$ and $\psi_i : \Sigma \times X_i \to \Psi_i$, $i \in \{1, 2\}$, where these parameters are all defined in a similar manner to those for other agents discussed previously in this document. However, there remain three items of note about this particular pair of supervisors:

1. Only two agents are considered, versus some other fixed number of supervisors. As with the fault-free communication structure, this is assumed in the interests of simplicity. Even so, the procedure to determine a minimal transmission scheme is extremely complex, as will be shown shortly.

2. It is also assumed that ($\mathbf{R_1}$, $\mathbf{R_2}$) are fully defined over the entire alphabet. In other words, every state has an outgoing transition defined for every event in Σ. Again, the assumption is made to reduce the complexity of the problem by allowing an agent to handle any event that may be communicated unexpectedly.

3. The feedback map ψ_i is defined over the set Ψ_i, whose contents are unspecified. This is designed to not limit the type of supervisor that may be employed. A “standard” agent is concerned with event enablement and disablement (where $\Psi_i = \{0, 1\}$), while other types of supervisors charged with tasks such as error detection will employ a different set of values.

To inspire the need to communicate, it is a given that each agent $\mathbf{R_i}$ is only able to observe a subset of all possible events. The set $\Sigma_{i,o} \subseteq \Sigma$, $i \in \{1, 2\}$ is defined to specify which events supervisor i is capable of detecting. Without communication, for any string s generated by the plant $\mathbf{G}$, $\mathbf{R_i}$ would observe the projected sequence $P_i(s)$. However, the transmission of events to $\mathbf{R_i}$ means that it will see more of s; defining just how much is the subject of the next section.

Mappings

The first task is to distinguish those events selected for transmission from those that are not. A *communication mapping* com_{ij} is established between $\mathbf{R_i}$ and $\mathbf{R_j}$, $i, j \in \{1, 2\}, i \neq j$. For all $s \in (\Sigma_{i,o} \cup \Sigma_{j,o})^*$, $com_{ij}(s)$ is the set of events $\alpha \in \Sigma_{i,o}$ that, upon their occurrence after sequence s, will be transmitted by $\mathbf{R_i}$ to $\mathbf{R_j}$. Readers may wonder why s is defined over $(\Sigma_{i,o} \cup \Sigma_{j,o})$ and not just $\Sigma_{i,o}$. The reasoning is that $\mathbf{R_i}$ will see all events in $\Sigma_{i,o}$ as well as some in $\Sigma_{j,o}$ that are sent to $\mathbf{R_i}$ by $\mathbf{R_j}$. Hence, s may be composed of more events than are observed by $\mathbf{R_i}$ alone.

The observable event sets of $\mathbf{R_1}$ and $\mathbf{R_2}$, in combination with their communication mappings com_{12} and com_{21}, can be used to define their *information mappings* θ_1 and θ_2. The purpose of these mappings is to determine, for any given $s = \alpha_1\alpha_2\ldots\alpha_n$, which events α_i $\mathbf{R_i}$ knows about (through observation or communication) and which α_j it does not know about. Thus, for all $s \in \Sigma^*$ and $\alpha \in \Sigma$,

$$\begin{aligned}
\theta_i : &\quad \Sigma^* \rightarrow \Sigma_o^* \\
\theta_i(\varepsilon) &= \varepsilon \\
\theta_i(s\alpha) &= \begin{cases} \theta_i(s\alpha) & \text{if } \alpha \in (\Sigma_{i,o} \cup com_{ji}(\theta_j(s)) \\ \theta_i(s) & \text{otherwise} \end{cases}
\end{aligned}$$

The information mapping pair (θ_1, θ_2) is derived from a given communication mapping pair (com_{12}, com_{21}) using the function $\mathcal{C}(com_{12}, com_{21})$.

As always, it is far simpler to define a solution than to propose a method for finding it. A communication mapping where all events that are not observed are transmitted may function, but it is almost certain that needless event transmissions will occur. Conversely, a mapping where nothing is communicated will result in bewilderment on the part of the supervisors. As a result, formal requirements must be defined to ensure that a transmission scheme properly coordinates the agents and does not put the operation of the system at risk.

Conditions

There are three requirements that a communication mapping (com_{12}, com_{21}) must satisfy for the purposes of this algorithm. It should be noted that for any given $(\mathbf{R_1}, \mathbf{R_2})$, many alternate transmission schemes will exist that do not meet all of these conditions. One of them may even be satisfactory to the implementor of the system. However, none of these requirements are totally unreasonable or overly restrictive; they could be described as common sense. It is ultimately the implementor's decision whether to pursue this avenue or not, but it is difficult to imagine many scenarios where the following requirements would be an issue.

The first condition is known as *feasibility* and requires that any two event sequences which appear to be identical to an agent will be followed with identical event transmissions. This guarantees that no agent will be expected to distinguish between event sequences that it finds indistinguishable.

Formally, for all $s, s' \in \Sigma^*$,

$$\theta_i(s) = \theta_i(s') \quad \Rightarrow \quad com_{ij}(P(s)) = com_{ij}(P(s')) \quad i, j \in \{1, 2\}, i \neq j$$

The second requirement is known as *validity.* It ensures that any event sequences which are indistinguishable to an agent lead to the same state in that agent's automaton. Formally, for all $s, s' \in \Sigma^*$

$$\theta_i(s) = \theta_i(s') \Rightarrow \xi_i(x_{i,0}, s) = \xi_i(x_{i,0}, s'),\ i \in \{1, 2\}$$

This may seem redundant, and it is with respect to an agent's projection, but it is not with respect to a supervisor's information mapping. For example, consider the case where $\alpha \notin \Sigma_{i,o}$ and $\xi_i(\alpha, x) = y$ where $y \neq x$. If α is not communicated, then what state is agent i really in? If α has occurred, it is in y, but if it has not, it is in x. Validity serves the important purpose of preventing this very quandary.

The third requirement is known as *implementability.* It dictates that for all event sequences leading to the same state in both agents, all event transmissions following those sequences will be identical. This guarantees that both agents exchange information according to their current state, which can easily distinguished, unlike event sequences. Formally, for all $s, s' \in \Sigma^*$

$$\begin{aligned}(\xi_1(s, x_{1,0}), \xi_2(s, x_{2,0})) = (\xi_1(s', x_{1,0}), \xi_2(s', x_{2,0})) \quad \Rightarrow \quad & com_{12}(P(s)) = com_{12}(P(s')) \wedge \\ & com_{21}(P(s)) = com_{21}(P(s'))\end{aligned}$$

The criteria outlined in this section will ensure that any solution which satisfies these conditions is operationally appropriate, *i.e.*, can be applied to a real-world situation without fear of system failure. However, this only represents half the battle; the rest is concerned with finding such a solution.

Formal Problem Declaration

As stated previously, the goal of this exercise is to determine a minimal communication system between two distributed discrete-event supervisors. This gives rise to an important question: minimal in what sense? Fewest events flagged for communication? Least number of transmissions sent in the worst case? Smallest number of transmissions made during standard operation of the system?

The following definition removes all possible ambiguity from the term "minimal". A pair of communication maps $(\overline{com}_{12}, \overline{com}_{21})$ transmits *strictly less than* (com_{12}, com_{12}) if the following conditions are met:

$$\begin{aligned} \forall\, s \in \Sigma_o^*, \quad & \overline{com}_{12}(s) \subseteq com_{12}(s) \,\wedge\, \overline{com}_{21}(s) \subseteq com_{21}(s) \wedge \\ \exists\, t \in \Sigma_o^*, \quad & \overline{com}_{12}(t) \subset com_{12}(t) \,\vee\, \overline{com}_{21}(t) \subset com_{21}(t) \end{aligned}$$

The notational representation of this situation is the obvious choice: $(\overline{com}_{12}, \overline{com}_{21})$ $< (com_{12}, com_{12})$. The transmission scheme (com^*_{12}, com^*_{12}) is considered *minimal* if no other pair of communication mappings exists that transmits strictly less than this scheme. Note that this definition of minimality involves strings and event subsets, as opposed to the overall number of events transmitted. It is far different than the possible options suggested at the beginning of this section.

Thus, given a pair of supervisors $(\mathbf{R_1}, \mathbf{R_2})$ with associated observable event projection (P_1, P_2), the goal is to produce a minimal communication scheme (com^*_{12}, com^*_{21}) that is *feasible*, *valid* and *implementable* with respect to $(\mathbf{R_1}, \mathbf{R_2})$ and (P_1, P_2). The reasons behind these conditions have been fully explored; what remains is to demonstrate how such a solution is determined.

Technique

When one is tackling a large complex problem, the hardest question is usually the first: where to start? In the case of this particular algorithm, the answer is to first examine the combined behaviour of agents $\mathbf{R_1}$ and $\mathbf{R_2}$. This is accomplished by taking the accessible portion of the product of the two supervisors, as described in Definition 5. Formally,

$$\begin{gathered} \mathbf{R} = (\mathbf{R_1} \times \mathbf{R_2}) = (R, \psi) \\ R = (\Sigma, X, \xi, x_0, X_m) \\ \psi : \Sigma \times X \to \Psi \end{gathered}$$

where every state $x = (x_1, x_2)$, $x_i \in X_i$, $i \in \{1, 2\}$. This centralized representation of $(\mathbf{R_1}, \mathbf{R_2})$ is the crux of the algorithm; it is the first operation to be performed and all subsequent operations depend upon this one.

The sets V_{12} and V_{21} represent events that must be communicated from one agent to another. Each set V_{ij} is composed of transitions taken from $\mathbf{R}$ that agent i can observe but agent j cannot:

$$V_{ij} = \{\ldots, (x, \sigma, \xi(s, x)), \ldots\}\ \sigma \in \Sigma_{i,o} - \Sigma_{j,o},\ i, j \in \{1, 2\}, i \neq j$$

The advantage to defining transmited events over $\mathbf{R}$ is that each transition not only describes where agent i is, but where agent i believes agent j is in its execution. This is extremely advantageous from a "finding a solution" perspective, but readers may be concerned from an "implementing the solution" viewpoint. After all, how can agent $\mathbf{R_i}$ perform a control or communication action designed for agent $(\mathbf{R_1} \times \mathbf{R_2})$? Rest assured that this issue will be addressed and resolved, but until then, some patience may be required.

Assume for a moment that some (V_{ij}, V_{ji}) has been derived by an unknown method. There are two obvious questions that beg to be asked:

1. What does $\mathbf{R_i}$ know about the status of $\mathbf{R_j}$?

2. What does $\mathbf{R_i}$ know about its own status?

The answers to the above can be determined through a short series of operations on $\mathbf{R}$ that essentially narrow its scope to $\mathbf{R_i}$'s viewpoint.

The first step is to "eliminate" any transition (x, σ, y) in $\mathbf{R}$ that $\mathbf{R_i}$ can neither directly observe $(\sigma \notin \Sigma_{i,o})$ nor receive via communication $((x, \sigma, y) \notin V_{ji})$. This is accomplished by replacing the events on these transitions with ϵ. The resulting automaton is defined as

$$R_i^{\epsilon}(V_{ji}) = (\Sigma, X, \xi_i^{\epsilon}, x_0, X_m)$$

After some ϵ-replacement has taken place, this automaton is uncertain of its current state. For any given (x, σ, y) where σ is not observed by or communicated to $\mathbf{R_i}$ (represented by (x, ϵ, y)), it is impossible to say whether $R_i^{\epsilon}(V_{ji})$ should be in state x or state y. This mimics the real-life confusion an agent may experience in a similar situation: if an event is unobservable and not communicated, it is impossible to know whether or not it has occured.

The next step is to determine exactly which states in $\mathbf{R}$ are indistinguishable to $\mathbf{R_i}$. Due to the ϵ-transitions inserted in the last step, $R_i^\epsilon(V_{ji})$ is a nondeterministic finite-state automaton (NFA). It is converted into a deterministic one (DFA) in the standard manner [4], resulting in

$$\tilde{R}_i = DA(R_i^\epsilon(V_{ji})) = (\Sigma, \tilde{X}_i, \tilde{\xi}_i, \tilde{x}_{i,0}, \tilde{X}_{i,m})$$

This action creates states $\tilde{x} \in \tilde{X}_i$ which are composed of a series of states $x, y, z, \ldots \in X$ from $\mathbf{R}$. These "composite" states essentially embody $\mathbf{R_i}$'s uncertainity; for any given $\tilde{x}$, $\mathbf{R_i}$ does not know which state $x \in \tilde{x}$ $\mathbf{R}$ is actually in.

The final step is not so much for agent $\mathbf{R_i}$'s benefit as for that of the minimal communication algorithm. Although $\mathbf{R_i}$ and $\mathbf{R}$ are totally defined over their alphabets, the NFA-DFA operation will most likely eliminate some of these transitions from $\tilde{R}_i$. Thus, it must be completed by inserting self-loops for every event at every state where a transition on that event is not currently defined. The resulting automaton is represented in the following manner

$$\tilde{R}_i^{SL} = SL(\tilde{R}_i) = (\Sigma, \tilde{X}_i, \xi_i^{SL}, \tilde{x}_{i,0}, \tilde{X}_{i,m})$$

The final product $\tilde{R}_i^{SL}$ of the operations given above embodies $\mathbf{R_i}$'s view of the system. However, that does not imply that $\mathbf{R_i}$ has sufficient knowledge for it to properly perform both supervisory actions and event communication. That requires $\tilde{R}_i^{SL}$ to have certain physical properties, which are the subject of the next section.

Correctness and Consistency

Previously, the concepts of feasibility, validity and implementability were introduced to ensure that a communication mapping met a certain level of quality. These were expressed in terms of (com_{12}^*, com_{21}^*), (θ_1, θ_2), $(\mathbf{R_1}, \mathbf{R_2})$ and $\mathbf{R}$. The goal of this section is to express them in terms of (V_{12}, V_{21}), $\mathbf{R}$ and $(\tilde{R}_1^{SL}, \tilde{R}_2^{SL})$, then finish by bridging the disconnect between these two definitions.

For agent $\mathbf{R_i}$ to perform its supervisory and communication responsibilities, it must always be absolutely certain of its current state. This is essential; both the feedback map ψ_i and implementability are dependent on the state of the agent. Fortunately, this is also easily expressed in terms of

$\tilde{R}_i^{SL}$.

Every composite state $\tilde{x} = \{x_1, x_2, \ldots x_n\}$ from $\tilde{R}_i^{SL}$ consists of product states $x_k = (x_{k,1}, x_{k,2})$, $1 \leq k \leq n$ where $x_{k,i} \in X_i$ from agent $\mathbf{R_i}$. For $\mathbf{R_i}$ to precisely know its current state at every point of its execution, it must be the case that $x_{1,i} = x_{2,i} = \ldots = x_{n,i}$ for every $\tilde{x}$. Informally speaking, this means that $\mathbf{R_i}$ will always know its own state, and will have an idea what state $\mathbf{R_j}$ is in, but will not be sure on the latter point. If this requirement is satisfied, $\tilde{R}_i^{SL}$ is said to be *correct*.

Ensuring correctness is actually quite a simple task. Define the "correctness set" for $\mathbf{R_i}$ as follows:

$$\begin{aligned} C_{ij} \quad = \quad & \{((x_1, x_2), \sigma, (y_1, y_2)) \mid \\ & ((x_1, x_2), \sigma, (y_1, y_2)) \in \text{ Transition}(\mathbf{R}) \\ & \wedge \sigma \in \Sigma_{i,o} - \Sigma_{j,o} \wedge x_j \neq y_j\} \end{aligned}$$

This set consists of every transition in $\mathbf{R}$ where $\mathbf{R_i}$ can observe the event, $\mathbf{R_j}$ can not and $\mathbf{R_j}$ needs to change state. It is proven in [10] that if $C_{ij} \subseteq V_{ij}$, then $\tilde{R}_j^{SL}$ is correct and $\mathbf{R_j}$ will definitively know its own state during every step of its execution. Thus, the feedback map $\tilde{\psi}_i : \Sigma \times \tilde{X}_i \rightarrow \Psi_i$ can be defined where for all $\tilde{x} \in \tilde{X}_i$, $(x_1, x_2) \in \tilde{x}$,

$$\tilde{\psi}_i(\alpha, \tilde{x}) = \psi_i(\alpha, x_i) \tag{4.1}$$

The second condition, that of *consistency*, is conceptually as straightforward as correctness. Informally put, it states that if agent $\mathbf{R_i}$ knows that it must transmit an event to $\mathbf{R_j}$ when $\mathbf{R_j}$ is state x, but $\mathbf{R_i}$ does not know whether $\mathbf{R_j}$ is in state x or y, then $\mathbf{R_i}$ must err on the side of caution and transmit the event regardless. This can be more formally stated using $\tilde{R}_i^{SL}$: for all $\sigma \in (\Sigma_{i,o} - \Sigma_{j,o})$ and $\tilde{x} \in \tilde{X}_i$,

$$\forall\, x \in \tilde{x},\ (x, \sigma, \xi(\sigma, x)) \in V_{ij}\ \vee\ (x, \sigma, \xi(\sigma, x)) \notin V_{ij}$$

Unlike correctness, the set of transitions N_{ij} required to achieve consistency for $\mathbf{R_i}$ cannot be easily expressed mathematically. Instead, Algorithm 1 is employed to determine this set. It performs the operations described above to create $\tilde{R}_i^{SL}$, then searches the composite states $\tilde{x}$ for two states x, x' where x' has an event σ flagged for transmission and x does not. This search is repeated until no more instances are found. The resulting set N_{ij} contains all of those instances that are necessary to communicate for consistency.

```
Algorithm 1: Consistency Function $\mathcal{N}_{ij}(V_{ij}, V_{ji})$
  Data: $\mathbf{R}$, $V_{ij}$ and $V_{ji}$
  Result: $N_{ij}$
1 $N_{ij} := \emptyset$;
2 Transition($R_i^\epsilon$) := $\emptyset$;
3 forall $(x, \sigma, \xi(\sigma, x)) \in$ Transition($\mathbf{R}$) do
      if $\sigma \in (\Sigma - \Sigma_{i,o}) \wedge (x, \sigma, \xi(\sigma, x)) \notin V_{ji}$ then
          Transition($R_i^\epsilon$) := Transition($R_i^\epsilon$) $\cup \{(x, \epsilon, \xi(\sigma, x))\}$;
      else
          Transition($R_i^\epsilon$) := Transition($R_i^\epsilon$) $\cup \{(x, \sigma, \xi(\sigma, x))\}$;
4 $R_i^\epsilon := (\Sigma, X, \text{Transition}(R_i^\epsilon), x_{i,0})$;
5 $\tilde{R}_i = DA(R_i^\epsilon(V_{ji})) = (\Sigma, \tilde{X}_i, \tilde{\xi}_i, \tilde{x}_{i,0}, \tilde{X}_{i,m})$;
6 $W_{ij} := \emptyset$;
7 forall $\tilde{x} \in \tilde{X}_1$ do
      if $(\exists\, x, x' \in \tilde{x})\, (x, \sigma, \xi(\sigma, x)) \notin (W_{ij} \cup V_{ij}) \wedge (x', \sigma, \xi(\sigma, x')) \in (W_{ij} \cup V_{ij})$ then
          $N_{ij} := N_{ij} \cup \{(x, \sigma, \xi(\sigma, x))\}$;
8 if $N_{ij} \neq W_{ij}$ then
      $W_{ij} := N_{ij}$;
      Go to 7;
  else
      return;
```

The combination of consistency and correctness results in an agent that is always certain of its own state (and through that its supervisory action) and is never uncertain about event transmissions. All that remains is to connect these concepts to the three hard requirements introduced earlier.

Define the mapping $\phi_{ij} : \tilde{X}_i \to 2^{\Sigma_{i,o} - \Sigma_{j,o}}$ as a state-based communication set where

$$\sigma \in \phi_{ij}(\tilde{x}) \Leftrightarrow (\exists\, x \in \tilde{x})\, (x, \sigma, \xi(\sigma, x))) \in V_{ij} \tag{4.2}$$

Of note is that if $\mathcal{N}_{ij}(V_{ij}, V_{ji}) = \emptyset$ (*i.e.*, further calls to the consistency function do not yield any additional transitions), then

$$(\exists\, x \in \tilde{x})\, (x, \sigma, \xi(\sigma, x))) \in V_{ij} \Rightarrow (\forall\, x \in \tilde{x})\, (x, \sigma, \xi(\sigma, x))) \in V_{ij}$$

Using both ϕ_{ij} and $\tilde{R}_i^{SL}$, the communication mapping com_{ij}^* is defined as follows for all $s \in \Sigma^*$

$$com_{ij}^*(s) = \phi_{ij}(\xi_i^{SL}(\tilde{x}_{i,0}, s)) \tag{4.3}$$

It is proven in [10] that if $\tilde{R}_1^{SL}$ and $\tilde{R}_2^{SL}$ are both consistent, then the communication mappings derived from ϕ_{12} and ϕ_{21} in the manner desribed above are feasible with respect to (P_1, P_2) and implementable with respect to $(\mathbf{R_1}, \mathbf{R_2})$. Finally, if $\tilde{R}_1^{SL}$ and $\tilde{R}_2^{SL}$ are both correct *and* consistent, then the (com_{12}^*, com_{21}^*) derived in the above manner are also valid with respect to $(\mathbf{R_1}, \mathbf{R_2})$.

This concludes the definition of the mappings, conditions, requirements, statements, sets and operations necessary to discuss the actual algorithm that determines a minimal communication scheme between two agents $(\mathbf{R_1}, \mathbf{R_2})$. However, before broaching that topic, it might be prudent to demonstrate the concepts introduced thus far in this section.

Example To begin, a slight renaming of the agents given in the example thus far is required to "fit in" with the naming convention of the minimal communication algorithm. Where a supervisor was known as $\mathbf{S_i}$ previously, it will now be known as $\mathbf{R_i}$ and similarly for $\hat{\mathbf{S}}_\mathbf{i}$ and $\hat{\mathbf{R}}_\mathbf{i}$.

To review, the reachable product of $\hat{\mathbf{R}}_\mathbf{1}$ and $\hat{\mathbf{R}}_\mathbf{2}$ yields the supervisor $\hat{\mathbf{R}}$ given in Figure 4.1. From this, the transitions that must be communicated to maintain correctness are listed in Table 4.1.

$\hat{C}_{12}$	$\hat{C}_{21}$		
$((1,i), b_1, (2,ii))$	$((3,i), a_2, (1,iii))$	$((2,ii), a_2, (5,ii))$	$((2,ii), b_2, (1,iii))$
$((3,i), b_1, (4,ii))$	$((4,ii), a_2, (2,ii))$	$((4,ii), b_2, (3,iii))$	$((5,ii), b_2, (1,iii))$
	$((2,iii), a_2, (5,iii))$	$((2,iii), b_2, (1,iii))$	$((3,iii), a_2, (1,iii))$
	$((4,iii), a_2, (2,iii))$	$((4,iii), b_2, (3,iii))$	$((5,iii), b_2, (1,iii))$

Table 4.1: Running Example - Correctness Sets $\hat{C}_{12}$ and $\hat{C}_{21}$

For the purposes of determining the transitions necessary for consistency, initially set $\hat{V}_{ij} = \hat{C}_{ij}$. Assume a function call to $\mathcal{N}_{12}$ has been placed with parameters $\hat{V}_{12}$, $\hat{V}_{21}$ and $\hat{\mathbf{R}}$. The first two lines of the method are concerned with setting initial values for $\hat{N}_{12}$ and $\hat{\mathbf{R}}$. Line **3**, however, is where things start to get interesting. That loop results in the ϵ-transitions given in Table 4.2. From this list, $\hat{R}_1^\epsilon(\hat{V}_{21})$ is generated on line **4** as shown in Figure 4.2. The automaton $\hat{R}_1^\epsilon(\hat{V}_{21})$ is then converted

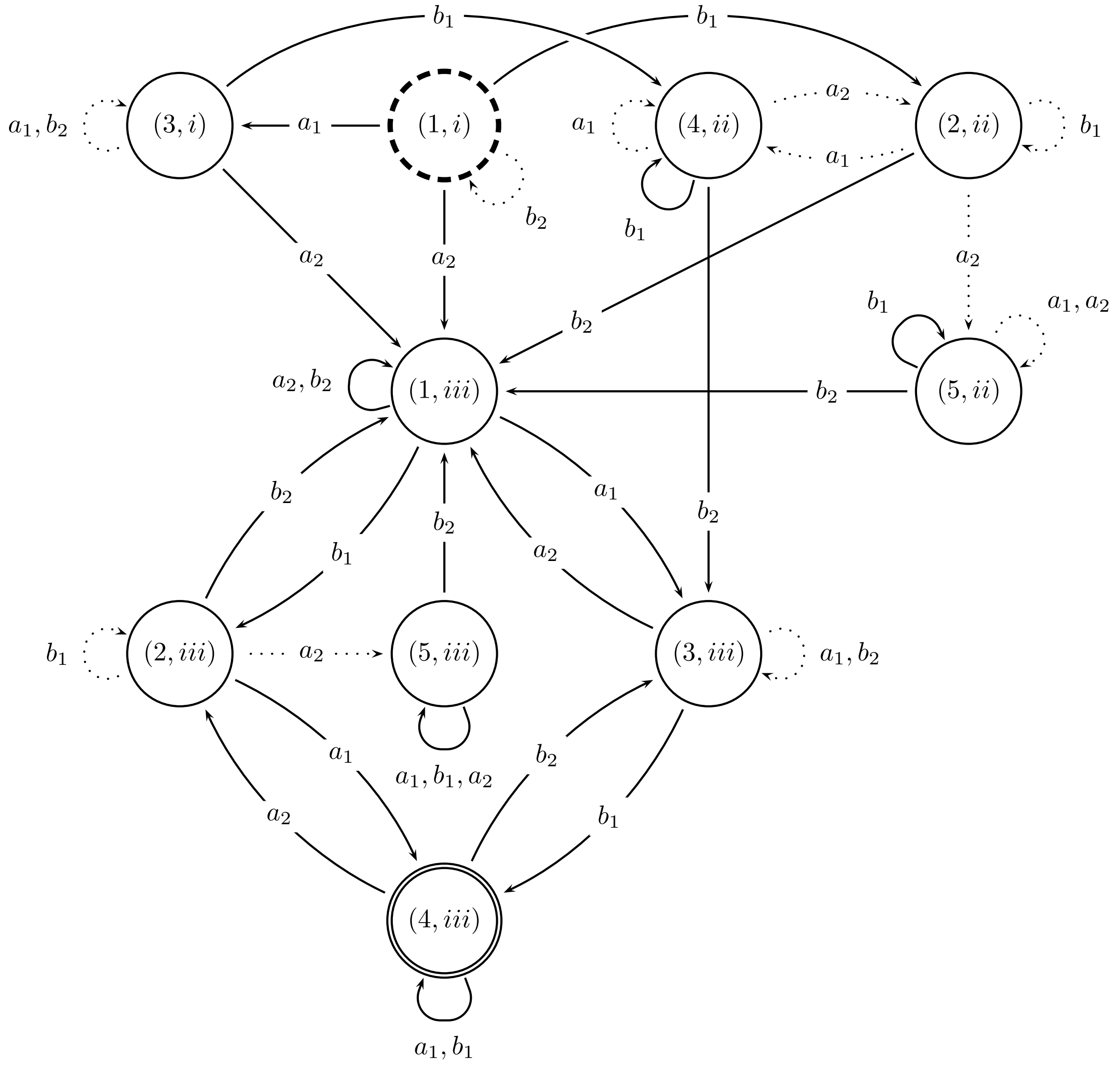


Figure 4.1: Running Example - Reachable product of $\hat{\mathbf{R}}_\mathbf{1}$ and $\hat{\mathbf{R}}_\mathbf{2}$ (Review of Figure 3.8)

$((1,i),\cancel{a_2}\epsilon,(1,iii))$	$((1,iii),\cancel{a_2}\epsilon,(1,iii))$	$((1,iii),\cancel{b_2}\epsilon,(1,iii))$	$((3,i),\cancel{b_2}\epsilon,(3,i))$
$((3,iii),\cancel{b_2}\epsilon,(3,iii))$	$((5,ii),\cancel{a_2}\epsilon,(5,ii))$	$((5,iii),\cancel{a_2}\epsilon,(5,iii)$	

Table 4.2: Running Example - ϵ-Transitions in $\hat{R}_1^\epsilon(\hat{V}_{21})$ for $\hat{N}_{12}$

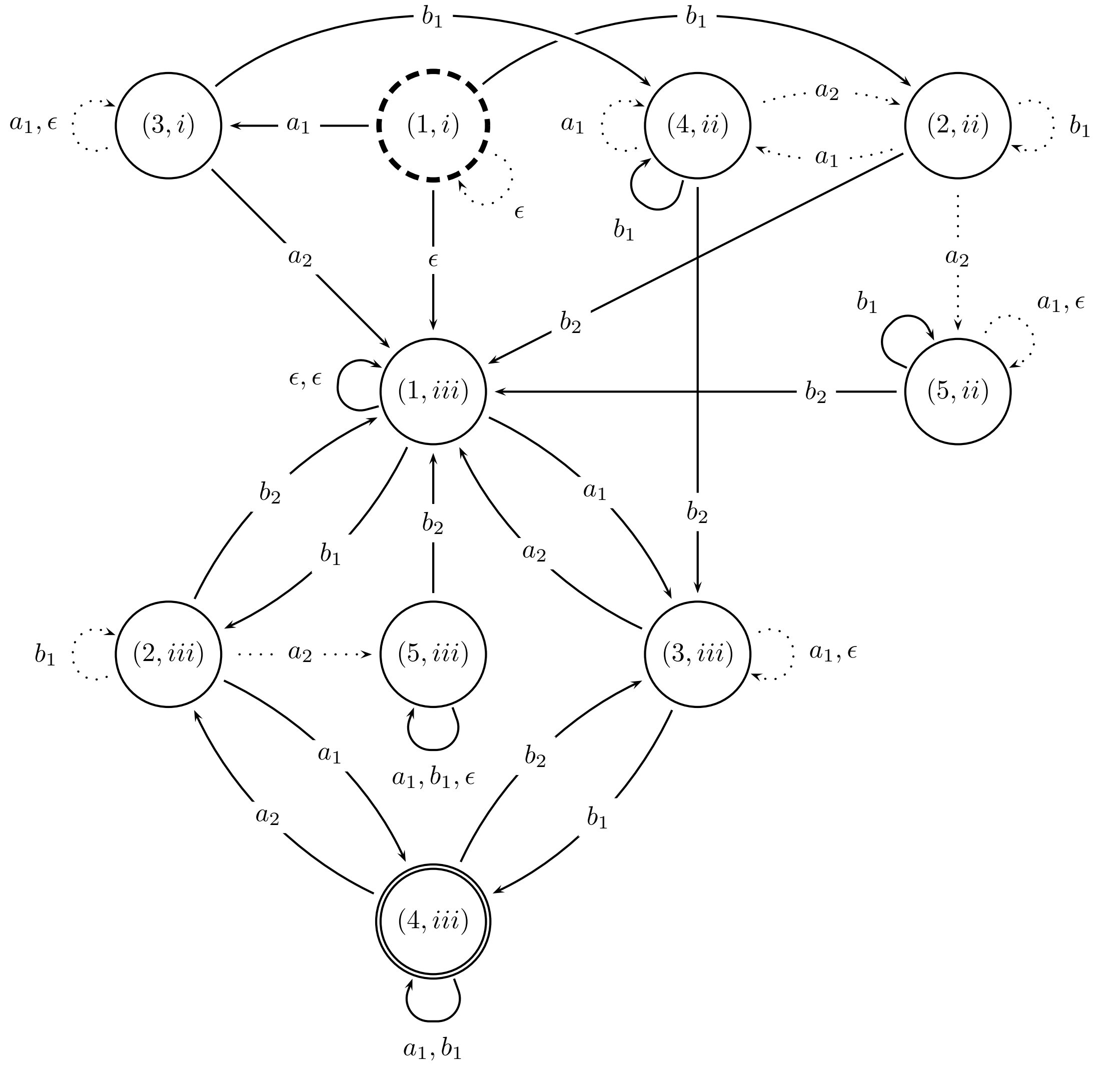


Figure 4.2: Running Example - $\hat{R}_1^{\epsilon}(\hat{V}_{21})$ Generated by $\mathcal{N}_{12}(\hat{C}_{12}, \hat{C}_{21})$

into $\tilde{\tilde{R}}_1$ on line **5**, as shown in Figure 4.3.

It is important to note that $\tilde{\tilde{R}}_1$'s enablement/disablement policies (represented by solid/dotted transitions) and marking policies (represented by double-bordered states) are strictly based upon $\mathbf{\hat{R}_1}$'s decisions, instead of both $\mathbf{\hat{R}_1}$'s and $\mathbf{\hat{R}_2}$'s. The reason is simple: while $\mathbf{\hat{R}}$ represents the combined efforts of both agents, $\tilde{\tilde{R}}_1$ is meant to illustrate only $\mathbf{\hat{R}_1}$'s point of view, and as such, should reflect

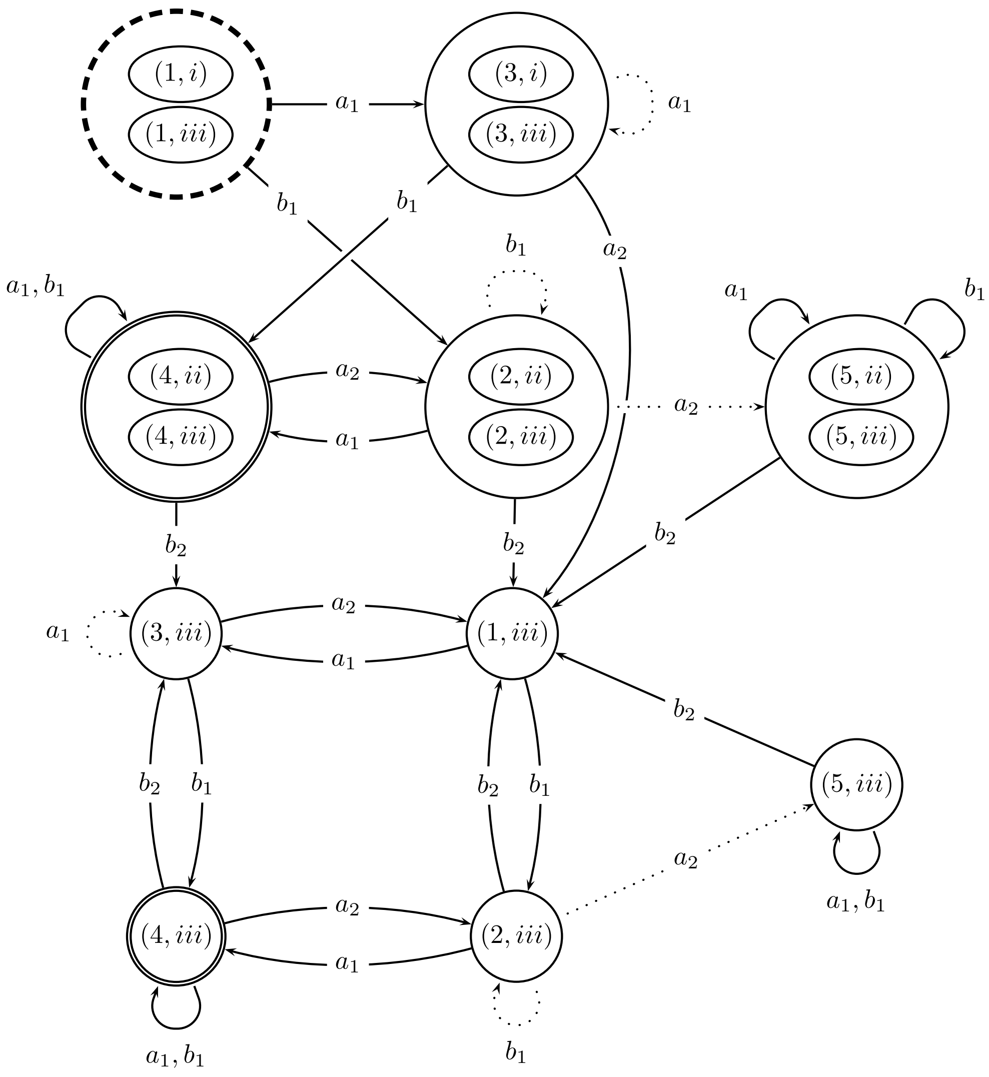


Figure 4.3: Running Example - $\tilde{\tilde{R}}_1$ Generated by $\mathcal{N}_{12}(\hat{C}_{12}, \hat{C}_{21})$

its feedback map and marking scheme alone. This will be the case with all such diagrams in this dissertation.

As previously stated in (4.1), the feedback map for $\tilde{\tilde{R}}_1$ is created by setting $\tilde{\tilde{\psi}}_1(\sigma, \tilde{x}) = \hat{\psi}_1(\sigma, x_1)$ where $(x_1, x_2) \in \tilde{x}$, $x_i \in X_i$, $i \in \{1, 2\}$. For example, event a_1 is enabled at $\{(5, ii), (5, iii)\}$ because a_1 is enabled at state 5 in $\hat{\mathbf{R}}_\mathbf{1}$; the fact that a_1 is disabled at state ii in $\hat{\mathbf{R}}_\mathbf{2}$ does not affect the control policy for this composite state. A similar statement can be made with respect to marking.

Since there are only five states in $\tilde{\hat{R}}_1$ that contain more than one state from $\hat{\mathbf{R}}$, it is very simple to "eyeball" the results of line **7** from the algorithm, as given in Table 4.3. Note that each row in the table contains the following entries: a transition that must be communicated for consistency (in this case, a member of $\hat{N}_{12}$); the transition with which consistency is necessary, *i.e.*, the "cause"; and the composite state that contains the originating states for both of these transitions (in this case, a state in $\tilde{\hat{R}}_1$). This will be the format for all future tables related to N_{ij} and $\hat{N}_{ij}$.

$\hat{N}_{12}$		
Transition	**Cause**	**State in $\tilde{\hat{R}}_1$**
$((1,iii),b_1,(2,iii))$	$((1,i),b_1,(2,ii)) \in \hat{C}_{12}$	$\{(1,i),(1,iii)\}$
$((3,iii),b_1,(4,iii))$	$((3,i),b_1,(4,ii)) \in \hat{C}_{12}$	$\{(3,i),(3,iii)\}$

Table 4.3: Running Example - Consistency Set $\hat{N}_{12}$

At line **8**, $\hat{N}_{12} \neq W_{12}$, so W_{12} is assigned the value of $\hat{N}_{12}$ and the algorithm returns to line **7**. However, in this round no new transitions are added to $\hat{N}_{12}$; they were all caught the first time. Thus, at line **8**, $\hat{N}_{12} = W_{12}$ and $\mathcal{N}_{12}(\hat{V}_{12}, \hat{V}_{21})$ terminates with the $\hat{N}_{12}$ given above.

Main Function

All the items discussed previously in this section come together in a single function known simply as *Main*. It requires only two supervisors as input, but returns far more: the communications required for correctness and consistency for both agents as well as the automata that represent each agent's view of the system given the communicated events.

To begin, the following is a high-level description of each step in the algorithm:

1. Combine the behaviour of $\mathbf{R_1}$ and $\mathbf{R_2}$ into $\mathbf{R}$
2. Determine the transitions required for correctness (C_{12}, C_{21})
3. Find the maximum set of additional transitions that $\mathbf{R_1}$ must communicate for consistency given C_{12} and C_{21} (N_{12}^{max})
4. Decide the minimal set of transitions that $\mathbf{R_2}$ must communicate for consistency given C_{12}, C_{21} and N_{12}^{max} (N_{21}^{*})

5. Determine the (potentially) insufficient minimal set of transitions that $\mathbf{R_1}$ must communicate for consistency given C_{12}, C_{21} and N^*_{21} (N^{min}_{12})

6. Examine every set of transitions between the range of N^{min}_{12} and N^{max}_{12} and flag those sets that do *not* violate the consistency requirement

7. Select the minimal set of transitions from those that were flagged (N^*_{12})

8. Define the "point of view" automata for agents $\mathbf{R_1}$ and $\mathbf{R_2}$ using $DA(R^{\epsilon}_i(C_{ji} \cup N^*_{ji})$ ($\tilde{R}_1$ and $\tilde{R}_2$)

9. Add self-loops to $\tilde{R}_1$ and $\tilde{R}_2$ (R^*_1 and R^*_2)

10. Determine the minimal communication maps using the state-based mapping ϕ_{ij} and automaton R^*_i (com^*_{12} and com^*_{21})

11. Return C_{12}, C_{21}, N^*_{12}, N^*_{21}, R^*_1 and R^*_2

With this informal description in mind, the formal definition of the *Main* function is given in Algorithm 2. Arguably, the most interesting item of note in this algorithm is the manner in which N^*_{12} and N^*_{21} are determined. Not only is no iteration is required for N^*_{21}, but it is minimal despite the fact that it is calculated using N^{max}_{12}, which likely does not contain the fewest number of transitions. The set N^*_{12} does require some iteration, but it is bounded by the range over the largest and smallest possible sets satisfying consistency. It is somewhat surprising that more calculations are not required over both sets. That is, it would be reasonable to expect that once a potential N^*_{ij} was determined, N^*_{ji} would require adjustments, then so would N^*_{ij} to compensate for those changes, then so would N^*_{ji} again, and so on. Perhaps this solution could be applied to the well-worn "I know, but you know that I know, but I know that you know that I know ..." act used so often in films and television.

Example It is only fitting to complete this section with the application of the *Main* function to the running example. Steps **1** through **3** were described in the last appearance of the example, so focus will be immediately shifted to line **4**, which performs the function call $\mathcal{N}_{21}(\hat{C}_{21}, \hat{C}_{12} \cup \hat{N}^{max}_{12})$.

Inside the consistency function, a new set of ϵ-transitions is determined from the point of view of agent $\hat{\mathbf{R}}_\mathbf{i}$. Given communicated transitions ($\hat{C}_{12} \cup \hat{N}^{max}_{12}$), the transitions in Table 4.4 are not

Algorithm 2: *Main* Function

```
   Data: $\mathbf{R_1}$, $\mathbf{R_2}$
   Result: $C_{12}$, $C_{21}$, $N_{12}^*$, $N_{21}^*$, $R_1^*$ and $R_2^*$
1  $\mathbf{R} := (\mathbf{R_1} \times \mathbf{R_2})$;
2  $C_{21} := \{((x_1, x_2), \sigma, (y_1, y_2)) \in \text{Transition}(\mathbf{R}) \mid \sigma \in (\Sigma_{2,o} - \Sigma_{1,o}) \wedge x_1 \neq y_1\}$;
   $C_{12} := \{((x_1, x_2), \sigma, (y_1, y_2)) \in \text{Transition}(\mathbf{R}) \mid \sigma \in (\Sigma_{1,o} - \Sigma_{2,o}) \wedge x_2 \neq y_2\}$;
3  $N_{12}^{max} := \mathcal{N}_{12}(C_{12}, C_{21})$;
4  $N_{21}^* := \mathcal{N}_{21}(C_{21}, C_{12} \cup N_{12}^{max})$;
5  $N_{12}^{min} := \mathcal{N}_{12}(C_{12}, C_{21} \cup N_{21}^*)$;
6  forall $N$ such that $N_{12}^{min} \subseteq N \subseteq N_{12}^{max}$ do
       if $\mathcal{N}_{21}(C_{21}, C_{12} \cup N) = N_{21}^* \wedge \mathcal{N}_{12}(C_{12} \cup N, C_{21} \cup N_{21}^*) = \emptyset$ then flag($N$);
7  Pick $N_{12}^*$ such that flag($N_{12}^*$) $= 1$ and for all $N \subset N_{12}^*$, flag($N$) $\neq 1$;
   (i.e., $N_{12}^*$ is a minimal element in the range that satisfies the two conditions of step 6)
8  $\tilde{R}_1 := DA(R_1^\epsilon(C_{21} \cup N_{21}^*))$;
   $\tilde{R}_2 := DA(R_2^\epsilon(C_{12} \cup N_{12}^*))$;
9  $R_1^* := SL(\tilde{R}_1)$;
   $R_2^* := SL(\tilde{R}_2)$;
10 Apply (4.2) and (4.3) and define $(\phi_{12}, \phi_{21})$ and $(com_{12}^*, com_{21}^*)$ according to
   $\sigma \in \phi_{12}(\tilde{x}) \Leftrightarrow (\exists\, x \in \tilde{x})\, (x, \sigma, \xi(\sigma, x)) \in (C_{12} \cup N_{12}^*)$;
   $\sigma \in \phi_{21}(\tilde{x}) \Leftrightarrow (\exists\, x \in \tilde{x})\, (x, \sigma, \xi(\sigma, x)) \in (C_{21} \cup N_{21}^*)$;
   $com_{12}^*(s) := \phi_{12}(\xi_1^*(\tilde{x}_{1,0}, s))$;
   $com_{21}^*(s) := \phi_{21}(\xi_2^*(\tilde{x}_{1,0}, s))$;
11 end
```

$((1, i), \cancel{a_1}\, \epsilon, (3, i))$	$((3, i), \cancel{a_1}\, \epsilon, (3, i))$	$((2, ii), \cancel{a_1}\, \epsilon, (4, ii))$	$((2, ii), \cancel{b_1}\, \epsilon, (2, ii))$
$((4, ii), \cancel{a_1}\, \epsilon, (4, ii))$	$((4, ii), \cancel{b_1}\, \epsilon, (4, ii))$	$((5, ii), \cancel{a_1}\, \epsilon, (5, ii))$	$((5, ii), \cancel{b_1}\, \epsilon, (5, ii)$
$((1, iii), \cancel{a_1}\, \epsilon, (3, iii))$	$((3, iii), \cancel{a_1}\, \epsilon, (3, iii))$	$((2, iii), \cancel{a_1}\, \epsilon, (4, iii))$	$((2, iii), \cancel{b_1}\, \epsilon, (2, iii)$
$((4, iii), \cancel{a_1}\, \epsilon, (4, iii))$	$((4, iii), \cancel{b_1}\, \epsilon, (4, iii))$	$((5, iii), \cancel{b_1}\, \epsilon, (5, iii)$	

Table 4.4: Running Example - ϵ-Transitions in $\hat{R}_2^\epsilon(\hat{V}_{12})$ for $\hat{N}_{21}^*$

observed. From this list, $\hat{R}_2^\epsilon(\hat{V}_{12})$ is generated as shown in Figure 4.4. The automaton $\hat{R}_2^\epsilon(\hat{V}_{12})$ is then converted into $\tilde{\hat{R}}_2$ as shown in Figure 4.5. In contrast to $\tilde{\hat{R}}_1$ from $\hat{N}_{12}^{max}$, all of the states in $\tilde{\hat{R}}_2$ contain more than one state from $\hat{\mathbf{R}}$. However, like $\hat{N}_{12}^{max}$, only one pass is needed to determine the transitions required for consistency. The results are shown in Table 4.5.

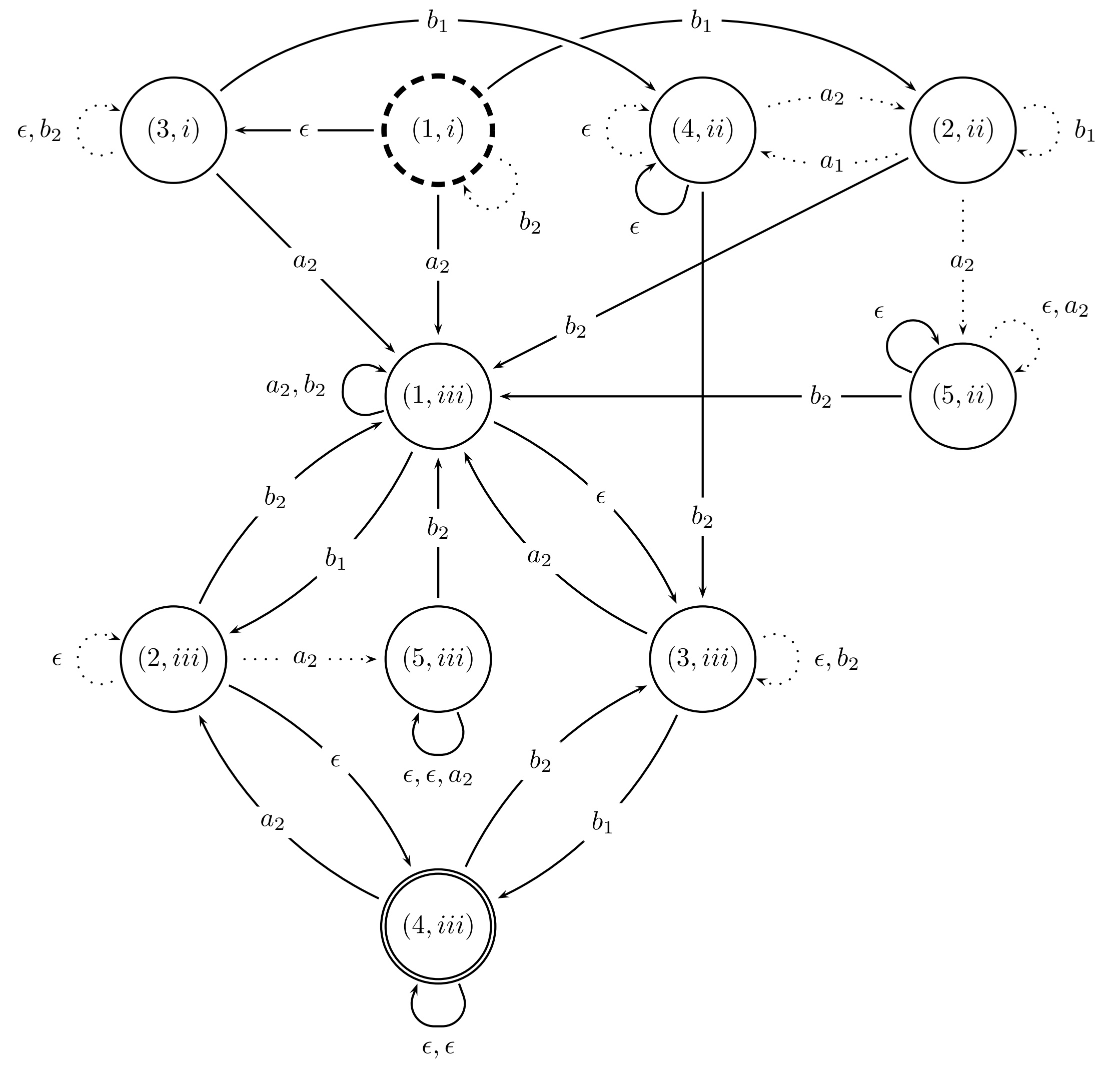


Figure 4.4: Running Example - $\hat{R}_2^{\epsilon}(\hat{V}_{12})$ Generated by $\mathcal{N}_{21}(\hat{C}_{21}, \hat{C}_{12} \cup \hat{N}_{12}^{max})$

At line **5** of the algorithm, $\hat{N}_{12}^{min}$ is determined using the function call $\mathcal{N}_{12}(\hat{C}_{12}, \hat{C}_{21} \cup \hat{N}_{21}^{*})$. Inside the method, the routine of first establishing ϵ-transitions is obeyed, as shown in Table 4.6. There is a trend in this list that is worthy of note: only self-transitions are reduced to ϵ. This fact greatly simplifies the consistency results in that the states of $\tilde{\hat{R}}_1$ are identical to those in $\hat{\mathbf{R}}$. As a result, there are no composite states in $\tilde{\hat{R}}_1$ and therefore no transitions that must be added for consistency. Thus, $\hat{N}_{12}^{min} = \emptyset$.

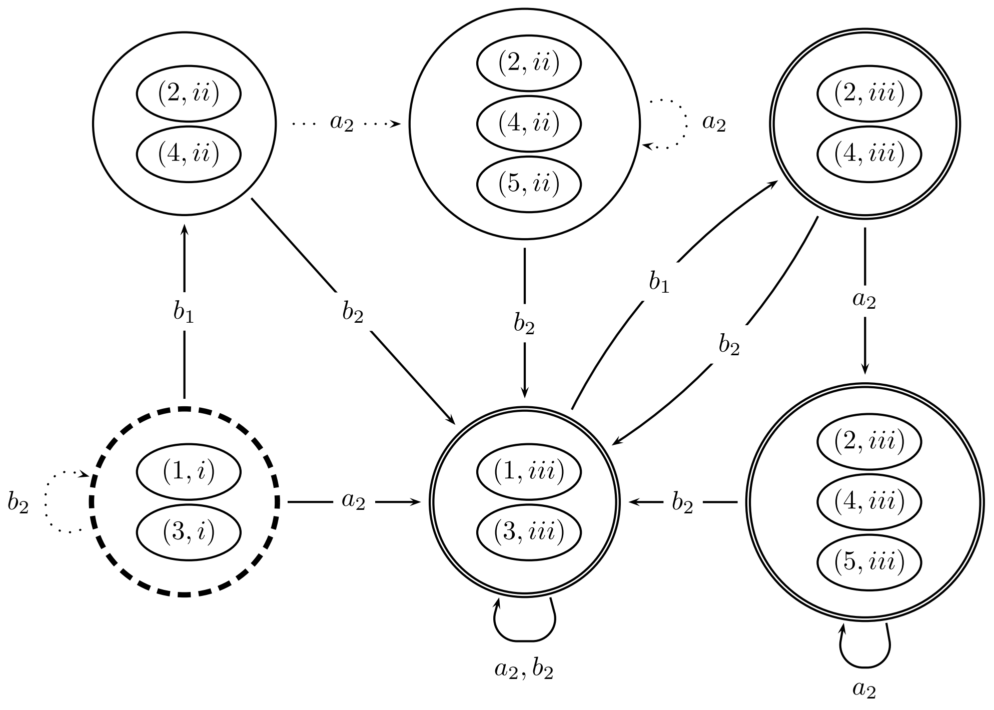


Figure 4.5: Running Example - $\tilde{\hat{R}}_2$ Generated by $\mathcal{N}_{21}(\hat{C}_{21}, \hat{C}_{12} \cup \hat{N}_{12}^{max})$

$\hat{N}_{21}^{*}$		
Transition	**Cause**	**State in $\tilde{\hat{R}}_1$**
$((1,i), a_2, (1,iii))$	$((3,i), a_2, (1,iii)) \in \hat{C}_{21}$	$\{(1,i), (3,i)\}$
$((5,ii), a_2, (5,ii))$	$((2,ii), a_2, (5,ii)) \in \hat{C}_{21}$	$\{(2,ii), (4,ii), (5,ii)\}$
$((1,iii), a_2, (1,iii))$	$((3,iii), a_2, (1,iii)) \in \hat{C}_{21}$	$\{(1,iii), (3,iii)\}$
$((5,iii), a_2, (5,iii))$	$((2,iii), a_2, (5,iii)) \in \hat{C}_{21}$	$\{(2,iii), (4,iii), (5,iii)\}$

Table 4.5: Running Example - Consistency Set $\hat{N}_{21}^{*}$

$((1,i), \cancel{b_2}\,\epsilon, (1,i))$	$((3,i), \cancel{b_2}\,\epsilon, (3,i))$	$((1,iii), \cancel{b_2}\,\epsilon, (1,iii))$
$((1,iii), \cancel{b_2}\,\epsilon, (1,iii))$	$((3,iii), \cancel{b_2}\,\epsilon, (3,iii))$	

Table 4.6: Running Example - ϵ-Transitions in $\hat{R}_1^{\epsilon}(\hat{V}_{21})$ for $\hat{N}_{12}^{min}$

The next step in the algorithm is **7**, where transition sets within the range bounded by $\hat{N}_{12}^{max}$ and $\hat{N}_{12}^{min}$ that do *not* violate consistency with $\hat{C}_{12}$, $\hat{C}_{21}$ and $\hat{N}_{21}^{*}$ are flagged. There are four possible sets:

$N_1 = \emptyset = \hat{N}_{12}^{min}$

$N_2 = \{((1, iii), b_1, (2, iii))\}$

$N_3 = \{((3, iii), b_1, (4, iii))\}$

$N_4 = \{((1, iii), b_1, (2, iii)), ((3, iii), b_1, (4, iii))\} = \hat{N}_{12}^{max}$

In the interests of brevity, the calculations performed by consistency function calls $\mathcal{N}_{21}(\hat{C}_{21}, \hat{C}_{12} \cup N_k)$ and $\mathcal{N}_{12}(\hat{C}_{12} \cup N_k, \hat{C}_{21} \cup \hat{N}_{21}^*)$ will not be detailed. Instead, it suffices to informally examine what would occur if either $((1, iii), b_1, (2, iii))$ or $((3, iii), b_1, (4, iii))$ were not communicated.

Consider $\mathcal{N}_{21}(\hat{C}_{21}, \hat{C}_{12} \cup N_k)$ first with reference to Figure 4.5. Eliminating either or both of these transitions would results in the amalgamation of states $\{(1, iii), (3, iii)\}$ and $\{(2, iii), (4, iii)\}$ as well as the addition of $\{(1, iii), (3, iii)\}$ to state $\{(2, iii), (4, iii), (5, iii)\}$. Neither of these operations changes the outcome since all of the transitions in $\hat{N}_{21}^*$ are added for consistency with transitions communicated for correctness and no states in $\tilde{\hat{R}}_i$ are split. Thus, $\mathcal{N}_{21}(\hat{C}_{21}, \hat{C}_{12} \cup N_k) = \hat{N}_{21}^*$ for all $1 \leq k \leq 4$.

Now consider $\mathcal{N}_{12}(\hat{C}_{12} \cup N_k, \hat{C}_{21} \cup \hat{N}_{21}^*)$ with reference to Figure 4.3. Both of these transitions were added to be consistent with $((1, i), b_1, (2, ii))$ and $((3, i), b_1, (4, ii))$ due to composite states $\{(1, i), (1, iii)\}$ and $\{(3, i), (3, iii)\}$. Eliminating either $((1, iii), b_1, (2, iii))$ or $((3, iii), b_1, (4, iii))$ or both does not split these composite states. As a result, any one of these transitions will be included for consistency by $\mathcal{N}_{12}$ if it is not already present in the set N_k. Thus, $\mathcal{N}_{12}(\hat{C}_{12} \cup N_k, \hat{C}_{21} \cup \hat{N}_{21}^*) \neq \emptyset$ for $1 \leq k \leq 3$ *but* $\mathcal{N}_{12}(\hat{C}_{12} \cup \hat{N}_{12}^{max}, \hat{C}_{21} \cup \hat{N}_{21}^*) = \emptyset$. The conclusion of this step is to flag $\hat{N}_{12}^{max}$ while all other possible sets remain unflagged.

Due to the simple result from the previous step, **7** through **10** are relatively simple. Again, to be brief, a high-level description will be employed. Step **7** sets $\hat{N}_{12}^* = \hat{N}_{12}^{max}$ and step **8** creates automata $\hat{R}_1^*$ and $\hat{R}_2^*$ that are identical to Figures 4.3 and 4.5, respectively. Step **9** completes the states of these "viewpoint" automata with self-loops (where necessary), and step **10** calculates the communication mappings $(\widehat{com}_{12}^*, \widehat{com}_{21}^*)$ based on the state-based mappings (ϕ_{12}, ϕ_{21}) as given in Table 4.7.

$\tilde{\hat{x}}_1 \in \tilde{\hat{X}}_1$	$\phi_{12}(\tilde{\hat{x}}_1)$	$\tilde{\hat{x}}_2 \in \tilde{\hat{X}}_2$	$\phi_{21}(\tilde{\hat{x}}_2)$
$\{(1,i),(1,iii)\}$	$\{b_1\}$	$\{(1,i),(3,i)\}$	$\{a_2\}$
$\{(3,i),(3,iii)\}$	$\{b_1\}$	$\{(2,ii),(4,ii)\}$	$\{a_2,b_2\}$
$\{(2,ii),(2,iii)\}$	$\emptyset$	$\{(1,iii),(3,iii)\}$	$\{a_2\}$
$\{(4,ii),(4,iii)\}$	$\emptyset$	$\{(2,ii),(4,ii),(5,ii)\}$	$\{a_2,b_2\}$
$\{(5,ii),(5,iii)\}$	$\emptyset$	$\{(2,iii),(4,iii)\}$	$\{a_2,b_2\}$
$(1,iii)$	$\{b_1\}$	$\{(2,iii),(4,iii),(5,iii)\}$	$\{a_2,b_2\}$
$(2,iii)$	$\emptyset$		
$(3,iii)$	$\{b_1\}$		
$(4,iii)$	$\emptyset$		
$(5,iii)$	$\emptyset$		

Table 4.7: Running Example - State-Based Communication Mappings ϕ_{12} and ϕ_{21} for $\hat{R}_1^*$ and $\hat{R}_2^*$

4.2 Research

4.2.1 Considering Control

A discrete-event agent is defined in [10] as follows:

> We use the term “agent” to mean a process of interest whose behaviour is described by sequences of events or actions. ... What we have in mind for supervisory control is that agents of interest are supervisors or controllers that make control decisions or that do diagnostics.
>
> Associated with an agent i ($i = 1, 2$) is a (finite-state) automaton R_i
>
> $$R_i = (\Sigma, X_i, \xi_i, x_{i,0})$$
>
> where Σ is an alphabet of event labels, X_i is a set of states, $x_{i,0} \in X_i$ is the initial state, and $\xi_i : X_i \times \Sigma \to X_i$ is the transition function. (Note that here ξ_i is assumed to be defined over its entire domain. This is in contrast to the definition of automata usually used in the DESs literature). ... The objective of Agent i is defined as a state feedback mapping $\psi_i : X_i \to \Psi_i$, where Ψ_i is some set, which we will not specify. In other words, we do not commit ourselves to a specific type of objective (such as supervisory control or diagnosis).

This definition shares much with respect to the standard definition of a discrete-event supervisor but differs in two regards: no marked states and full definition across ξ_i. It was stated previously that one of this project's goals is to make the minimal communication algorithm applicable to discrete-event supervisors, both implicitly and explicitly-defined, and not simply fully-defined automata. Thus, each of these differences must be addressed and resolved.

To ensure the clarity of the following section, certain points must be made before examining each of the aforementioned issues. First, it is assumed that the goal of the supervisors considered here is control, hence the examination to follow will deal exclusively with enabled and disabled events. However, the same principles would apply to other feedback map policies, such as fault detection. Second, it is important to note that the solutions to both problems are based upon the principle of correctness. To recap, the following definition appears in [10]

> Formally, consider any state $\tilde{x}$ in $\tilde{R}_1$. The state $\tilde{x}$ is a subset of states in R: $\tilde{x} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. In order for Agent 1 to achieve its objective defined by the state feedback map ψ_1, it is required that all of the pairs in $\tilde{x}$ have the same first component $x_1 = x_2 = \ldots = x_n$. When this requirement is satisfied, we say that $\tilde{R}_1$ is *correct*. In that case, a unique feedback mapping $\tilde{\psi_1} : \tilde{X}_1 \to \Psi_1$ can be defined as
>
> $$\tilde{\psi_1}(\tilde{x}_1) = \psi_(x_1)$$
>
> This mapping will achieve the objective defined by ψ_1.

The first problem, a lack of marked states, actually has one solution that covers two schools of thought. The discrete-event system research community is somewhat split on whether marked states are required at all. Some would argue that a plant and its supervisors are built to know when a process is complete and thus defining completed states is redundant. Others would argue that the definition is important, especially in distributed control, because each supervisor may not have a complete view of the process and may only be able to know when it itself is "done".

Thanks to the correctness requirement, the answer lies in apathy, *i.e.*, not changing a single thing. For those that are anti-marking, there is not even an issue to begin with. For those that are pro-marking, the correctness requirement will guarantee that each supervisor is always certain of its current state. Thus, it will be able to enforce its marking strategy as usual in conjunction with the plant and other supervisors.

The second problem, full definition across Σ, also has a simple solution whose foundation is laid in [10]. Informally put, for any given implicit supervisor, "complete" it by adding self-loops for those events that do not have a transition defined at each state. To preserve control, define (or make additions to) an accompanying feedback map where all previously-defined transitions are enabled and all newly-defined self-loops are disabled. Use the supervisor's automaton (which represents its behaviour) as input for the algorithm, then trim the resulting communication scheme of all disabled events and apply it to the original supervisor.

Upon first reading, the above method, in which both enabled and disabled transitions are given as input for the algorithm, may appear problematic for two reasons:

(i) Only enabled transitions should be considered when determining a minimal communication scheme; including disabled transitions is not efficient and significantly increases the average running time of the algorithm

(ii) The result of the algorithm may no longer be minimal; the addition of a disabled transition to the communication set may cause the addition of one or more enabled transitions in order to maintain consistency with an event that never takes place

On the subject of average running time, there is no argument; including the disabled transitions only to remove them at the end is, on average, a waste of computation. However, the simplicity of the solution offers a distinct advantage in that the algorithm does not need to be re-tooled and re-proven. As for the possibility that the proposed method may result in a less-than-minimal communication scheme, the following definition and proposition will formally prove that that will never be the case.

4.2.2 Explicit Function

Standard Explicit Function

Definition 6. Consider a plant $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$ and implicitly-defined supervisor $S = (\Sigma, X, \xi, x_0, X_m)$. The *standard explicit function* $\upsilon : S \mapsto \mathbf{S}'$ that converts S into an explicitly-defined supervisor $\mathbf{S}'$ by adding a feedback map ϕ is defined as follows:

$$\mathbf{S}' = (T', \phi)$$

$$T' = (\Sigma, X, \xi', x_0, X_m)$$

$$\phi : \Sigma \times X \to \{0,1\}$$

$$\forall\, \sigma \in \Sigma,\, \forall\, x \in X,\, \xi'(\sigma, x) = \begin{cases} \xi(\sigma, x) & \text{if } \xi(\sigma, x)! \\ x & \text{otherwise} \end{cases}$$

$$\forall\, \sigma \in \Sigma,\, \forall\, x \in X,\, \phi(\sigma, x) = \begin{cases} 1 & \text{if } \xi(\sigma, x)! \\ 0 & \text{otherwise} \end{cases}$$

The following implications are immediate from the definitions of ξ' and ϕ.

$$\forall\, \sigma \in \Sigma,\, \forall\, x \in X,\, \xi(\sigma, x)! \Rightarrow \xi'(\sigma, x)! \tag{4.4}$$

$$\forall\, \sigma \in \Sigma,\, \forall\, x \in X,\, \xi(\sigma, x)! \Rightarrow \phi(\sigma, x) = 1 \tag{4.5}$$

$$\forall\, \sigma \in \Sigma,\, \forall\, x \in X,\, \xi(\sigma, x)! \Rightarrow \xi'(\sigma, x) = \xi(\sigma, x) \tag{4.6}$$

$$\forall\, \sigma \in \Sigma,\, \forall\, x \in X,\, \phi(\sigma, x) = 1 \Rightarrow \xi(\sigma, x)! \tag{4.7}$$

Theorem 2. *Given a plant* $\mathbf{G}$*, an implicitly-defined supervisor* S *and an explicitly-defined supervisor* $\mathbf{S}' = (T', \phi)$ *where* $\mathbf{S}' = \upsilon(S)$*,*

(i) $\forall\, \sigma \in \Sigma,\, \forall\, q \in Q,\, \forall\, x \in X,\, (\delta \times \xi)(\sigma, q, x) = (\delta \times \xi')^{\phi}(\sigma, q, x)$

(ii) $L(S/\mathbf{G}) = L(\mathbf{S}'/\mathbf{G})$

(iii) $L_m(S/\mathbf{G}) = L_m(\mathbf{S}'/\mathbf{G})$

(iv) $\forall\, \sigma \in \Sigma,\, \forall\, q \in Q,\, \forall\, x_1 \in X_1,\, \forall\, x_2 \in X_2,\, (\delta \times (\xi_1 \wedge \xi_2))(\sigma, q, x_1, x_2) =$
$(\delta \times (\xi_1' \wedge \xi_2'))^{\phi_1 \wedge \phi_2}(\sigma, q, x_1, x_2)$

(v) $L(S_1 \wedge S_2/\mathbf{G}) = L(\mathbf{S_1'} \wedge \mathbf{S_2'}/\mathbf{G})$

(vi) $L_m(S_1 \wedge S_2/\mathbf{G}) = L_m(\mathbf{S_1'} \wedge \mathbf{S_2'}/\mathbf{G})$

Proof: (i). Consider the definition of the transition function $(\delta \times \xi)$.

$$\forall \sigma \in \Sigma, \forall q \in Q, \forall x \in X,$$

$$\begin{aligned}
(\delta \times \xi)(\sigma, q, x) &= \begin{cases} (\delta(\sigma,q), \xi(\sigma,x)) & \text{if } \delta(\sigma,q)! \text{ and } \xi(\sigma,x)! \\ \text{undefined} & \text{otherwise} \end{cases} \\
&= \begin{cases} (\delta(\sigma,q), \xi'(\sigma,x)) & \text{if } \delta(\sigma,q)! \text{ and } \xi(\sigma,x)! \qquad \text{(From (4.6))} \\ \text{undefined} & \text{otherwise} \end{cases} \\
&= \begin{cases} (\delta(\sigma,q), \xi'(\sigma,x)) & \text{if } \delta(\sigma,q)!,\ \xi(\sigma,x)!, \qquad \text{(From (4.5))} \\ & \text{and } \phi(\sigma,x) = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \\
&= \begin{cases} (\delta(\sigma,q), \xi'(\sigma,x)) & \text{if } \delta(\sigma,q)!,\ \xi'(\sigma,x)!, \qquad \text{(From (4.4))} \\ & \text{and } \phi(\sigma,x) = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \\
&= (\delta \times \xi')^{\phi}(\sigma, q, x)
\end{aligned}$$

Now consider the definition of the transition function $(\delta \times \xi')^{\phi}$.

$$\forall \sigma \in \Sigma, \forall q \in Q, \forall x \in X,$$

$$\begin{aligned}
(\delta \times \xi')^{\phi}(\sigma, q, x) &= \begin{cases} (\delta(\sigma,q), \xi'(\sigma,x)) & \text{if } \delta(\sigma,q)!,\ \xi'(\sigma,x)!, \\ & \text{and } \phi(\sigma,x) = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \\
&= \begin{cases} (\delta(\sigma,q), \xi'(\sigma,x)) & \text{if } \delta(\sigma,q)!,\ \xi'(\sigma,x)!, \qquad \text{(From (4.7))} \\ & \text{and } \xi(\sigma,x)! \\ \text{undefined} & \text{otherwise} \end{cases} \\
&= \begin{cases} (\delta(\sigma,q), \xi(\sigma,x)) & \text{if } \delta(\sigma,q)! \text{ and } \xi(\sigma,x)! \qquad \text{(From (4.6))} \\ \text{undefined} & \text{otherwise} \end{cases} \\
&= (\delta \times \xi)(\sigma, q, x)
\end{aligned}$$

□

Proof: (ii). This follows from (i); if $(S/\mathbf{G})$ and $(\mathbf{S}'/\mathbf{G})$ have identical state spaces and equivalent transition functions, then their respective languages must also be identical. □

Proof: (iii). Similar argument to the one presented in (ii). □

Proof: (iv). This proof is nearly identical to that for (i); the only difference is that (4.4), (4.5), (4.6) and (4.7) must all be applied seperately with respect to ξ_1, ξ_2, ϕ_1 and ϕ_2. □

Proof: (v) and (vi). These both follow from (iv) and are similar to the arguments presented in (ii) and (iii). □

Theorem 2 is appropriate and correct when applied to a standalone pair of supervisors $\mathbf{S_1}$ and $\mathbf{S_2}$ for input to the minimal communication algorithm. However, comparability adds another dimension: if supervisor $\mathbf{S_i}$ is comparable to supervisor $\hat{\mathbf{S}}_\mathbf{i}$ via μ_i, then their explicit counterparts should also be comparable via the same mapping. Unfortunately, υ may not preserve this relationship.

Consider the two agents given in Figure 4.6. Agent S is comparable to $\hat{S}$ with respect to the plant $\mathbf{G}$ shown in Figure 4.7 via the mapping $\mu(1) = 1$ and $\mu(2) = \mu(3) = 2$. Apply the explicit

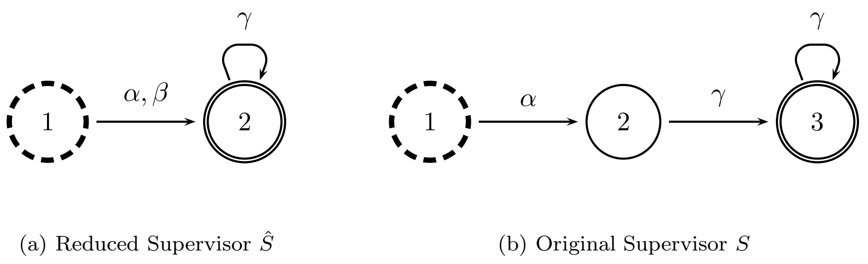


Figure 4.6: Explicit Function Example - Implicit Supervisors $\hat{S}$ and S

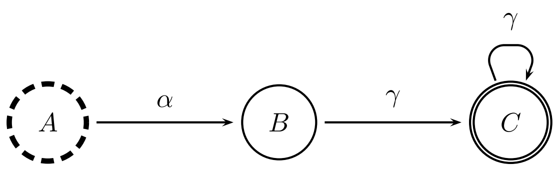
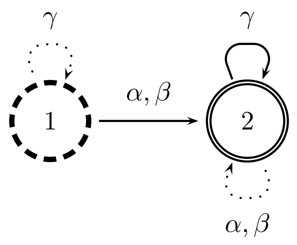
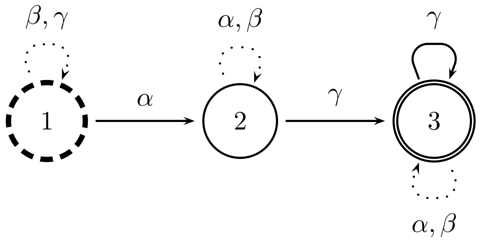


Figure 4.7: Explicit Function Example - Plant $\mathbf{G}$

function υ to S and $\hat{S}$, yielding $\mathbf{S'} = (T', \phi)$ and $\hat{\mathbf{S}}' = (\hat{T}', \hat{\phi}')$ given in Figure 4.8. The fourth



(a) Explicit Reduced Supervisor $\hat{\mathbf{S}}' = \upsilon(\hat{S})$



(b) Explicit Original Supervisor $\mathbf{S}' = \upsilon(S)$

Figure 4.8: Explicit Function Example - Explicit Supervisors $\hat{\mathbf{S}}'$ and $\mathbf{S}'$

requirement of comparability enforces the following rule:

$$\forall\, \sigma \in \Sigma,\, x \in X,\, \hat{\xi}(\sigma, \mu(x)) = \mu(\xi(\sigma, x)) \text{ where } \xi(\sigma, x)! \text{ and } x \text{ is synthesis-accessible}$$

However, Figure 4.8 illustrates that this is not the case for $\mathbf{S}'$ and $\hat{\mathbf{S}}'$. Specifically, for states $1 \in X$, $1 \in \hat{X}$ and event β,

$$\begin{aligned} \hat{\xi}'(\beta, \mu(1)) &= \hat{\xi}'(\beta, 1) \\ &= 2 \\ \mu(\xi'(\beta, 1)) &= \mu(1) \\ &= 1 \end{aligned}$$

This is a clear violation of the fourth requirement of comparability, demonstrating that $\hat{\mathbf{S}}'$ and $\mathbf{S}'$ are no longer comparable via μ. Thus, a slight adjustment must be made to the definition of $\upsilon(\ )$to handle just such a case by taking μ into account.

Comparable Explicit Function

Definition 7. Consider a plant $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$ and two supervisors $S = (\Sigma, X, \xi, x_0, X_m)$ and $\hat{S} = (\Sigma, \hat{X}, \hat{\xi}, \hat{x}_0, \hat{X}_m)$ where S is comparable to $\hat{S}$ with respect to $\mathbf{G}$ via the state mapping μ. The *comparable explicit function* $\upsilon^{\mu} : S \mapsto \mathbf{S}'$ converts S into an explicitly-defined supervisor

$\mathbf{S}' = (S', \phi')$ in the same manner as υ with the exception of ξ', which is defined as follows:

$$\forall\, \sigma \in \Sigma,\, x \in X,\, \xi'(\sigma, x) = \begin{cases} \xi(\sigma, x) & \text{if } \xi(\sigma, x)! \\ \text{any state in } \mu^{-1}(\hat{\xi}(\sigma, \mu(x))) & \text{if } \neg\xi(\sigma, x)! \wedge \hat{\xi}(\sigma, \mu(x))! \\ x & \text{otherwise} \end{cases}$$

Remarks. Note that if $\xi(\sigma, x)$ is not defined but $\hat{\xi}(\sigma, \mu(x)) = \hat{y}$ is, then any $y \in \mu^{-1}(\hat{y})$ is sufficient to preserve the fourth requirement of comparability; the choice does not really matter as (x, σ, y) in $\mathbf{S}'$ will be disabled regardless.

Needless to say, all of the associated properties of υ hold for υ^{μ} as well. The only change is in the definition of ξ' with respect to undefined transitions in $\mathbf{S}$ that are defined in $\hat{\mathbf{S}}$. Although some disabled transitions in $\mathbf{S}'$ may no longer be self-loops, the fact that they remain disabled implies that its resulting behaviour is no different whether υ, υ^{μ} or no function whatsoever had been applied to $\mathbf{S}$.

Explicit Functions and Minimal Communication

To complete the task of proving that both "explicit" functions defined here are appropriate for use in conjunction with the minimal communication algorithm, it is necessary to reiterate some of the algorithm's established properties. The following lemmas from [10] will be used in the propositions that will finish this section.

Lemma 1 ([10]). *If $C_{21} \subseteq V_{21}$, then $\tilde{R}_1 = DA(R_1^{\epsilon}(V_{21}))$ is correct, that is, all the pairs in a state $\tilde{x}$ of $\tilde{R}_1$ have the same first component.*

Lemma 2 ([10]). $(x, \sigma, \xi(x, \sigma)) \in \mathcal{N}_{12}(V_{12}, V_{21})$ *if and only if*

$$(\exists\, n \geq 1)\, (\exists\, \tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^n \in \tilde{X}_1)\, (\exists\, x^1, x^2, \ldots, x^n \in X)$$
$$(x^1, x^2 \in \tilde{x}^1) \;\wedge\; (x^2, x^3 \in \tilde{x}^2) \;\wedge\; \ldots \;\wedge\; (x^n, x \in \tilde{x}^n)$$
$$\wedge\; (x^1, \sigma, \xi(x^1, \sigma)) \in V_{21} \;\wedge\; (x, \sigma, \xi(x, \sigma)) \notin V_{21}$$

Proposition 4. Let $\tilde{R}_1 = DA(R_2^\epsilon(V_{12}))$ where $C_{21} \subseteq V_{21}$; by Lemma 1, $\tilde{R}_1$ is correct. Consider a transition $(y, \sigma, \xi(y, \sigma)) \in V_{12}$. If it causes the inclusion of another transition $(z, \sigma, \xi(z, \sigma)) \in \mathcal{N}_{12}(V_{12}, V_{21})$ then $\psi_1(y_1, \sigma) = \psi_1(z_1, \sigma)$.

Proof. Using Lemma 2 and the fact that $(y, \sigma, \xi(y, \sigma)) \in V_{12}$ and $(z, \sigma, \xi(z, \sigma)) \in \mathcal{N}_{12}(V_{12}, V_{21})$, there exist $\tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^n \in \tilde{X}_1$ and $x^1, x^2, \ldots, x^n \in X$ such that $(x^1, x^2 \in \tilde{x}^1)$, $(x^2, x^3 \in \tilde{x}^2) \ldots (x^n, x \in \tilde{x}^n)$ where $y = x^1$ and $z = x$. Since $\tilde{R}_1$ is correct, y and x^2 must share the same first component, *i.e.*, $y_1 = x_1^2$. Similarly, $x_1^2 = x_1^3$, and so on, until finally $x_1^n = z_1$. Considering the transitive nature of $=$, it is apparent that $y_1 = z_1$ and hence $\psi_1(y_1, \sigma) = \psi_1(z_1, \sigma)$. □

Proposition 5. If $\tilde{R}_1$ is correct, then a disabled transition $(y, \sigma, \xi(y, \sigma)) \in C_{12}$ will only cause disabled transitions $(z, \sigma, \xi(z, \sigma))$ to be added for consistency. Likewise, an enabled transition in C_{12} will only cause enabled transitions to be added for consistency.

Proof. Follows from Proposition 4. □

Chapter 5

Effects of Reduction on Communication

5.1 Applying Supervisor Reduction

Using the concepts described in the previous chapters, a strategy for the further reduction of communication between distributed agents can now take shape. Given two supervisors, modify them through some known method (such as **simsup** in CTCT or through personal observation) so that the original agents are comparable to their reduced counterparts. Then transform the smaller supervisors into explicitly-defined agents using the explicit functions υ and υ^{μ} and determine their communication scheme using the algorithm given in [10]. The expectation is that in the worst case the number of event transmissions will not be increased, while in some situations the amount of communication required will be reduced. The sole task that remains is to formally state and prove the previous statement.

5.2 Formal Statement of Primary Conjecture

Conjecture. *Consider a plant* $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$ *and two sets of explicitly-defined distributed agents (*$\mathbf{R_1}$*,* $\mathbf{R_2}$*) and (*$\hat{\mathbf{R}}_\mathbf{1}$*,* $\hat{\mathbf{R}}_\mathbf{2}$*) where* $\mathbf{R_i} = (R_i, \psi_i)$*,* $i \in \{1, 2\}$*,* $R_i = (\Sigma, X_i, \xi_i, x_{i,0})$*, and each* *supervisor* $\mathbf{R_i}$ *is comparable to* $\mathbf{\hat{R}_i}$ *with respect to* $\mathbf{G}$*. Thus,*

$$\begin{aligned} L(\mathbf{R_1} \wedge \mathbf{R_2}/\mathbf{G}) &= L(\mathbf{\hat{R}_1} \wedge \mathbf{\hat{R}_2}/\mathbf{G}) \\ L_m(\mathbf{R_1} \wedge \mathbf{R_2}/\mathbf{G}) &= L_m(\mathbf{\hat{R}_1} \wedge \mathbf{\hat{R}_2}/\mathbf{G}) \end{aligned}$$

Let com^*_{ij} *and* $\widehat{com}^*_{ij}$*,* $i, j \in \{1, 2\}, i \neq j$*, be the valid, feasible and implementable communication mappings derived from the minimal communication algorithm given in [10] for (*R_1*,* R_2*) and (*$\hat{R}_1$*,* $\hat{R}_2$*), respectively. Then* $\forall\, s \in L(\mathbf{\hat{R}_1} \wedge \mathbf{\hat{R}_2}/\mathbf{G})$*,*

$$\widehat{com}^*_{ij}(s) \subseteq com^*_{ij}(s)$$

In the interests of clarity and organization, a particular strategy has been applied to the proof for this theorem. It is composed of three parts: the first examines the problem and defines additional parameters, the second considers communications made for correctness, and the third considers transmissions made for consistency. However, before launching immediately into these sections, a few important points should be made:

1. The focus in previous sections on comparability and reachable product may have seemed somewhat superfluous, but its importance will become evident here. Every transition that must be communicated from agent to agent is determined using the product of those agents, either directly (in the case or correctness) or indirectly (in the case of consistency, which requires additional operations to the product). Hence, much of this proof will focus on or refer to reachable product, especially with respect to comparability.

2. Although the theorem statement considers explicitly-defined supervisors $\mathbf{R_i} = (R_i, \psi_i)$, the communication algorithm (in its current form) is solely concerned with the behaviour of these supervisors, embodied in $R_i = (\Sigma, X_i, \xi_i, x_{i,0})$. As a result, the proof will also focus on R_i and temporarily discard ψ_i. The feedback map is employed once the algorithm has returned its results and disabled transitions must be removed from the communication lists.

5.3 Problem Analysis

With both pairs ($\mathbf{R_1}$, $\mathbf{R_2}$) and ($\hat{\mathbf{R}}_\mathbf{1}$, $\hat{\mathbf{R}}_\mathbf{2}$) as input to the minimal communication algorithm given in [10], the resulting communication schemes will be referred to as com^*_{ij} and $\widehat{com}^*_{ij}$, respectively, $i, j \in \{1, 2\}, i \neq j$. For all $s \in L(\hat{\mathbf{R}}_\mathbf{1} \wedge \hat{\mathbf{R}}_\mathbf{2}/\mathbf{G})$ and $\sigma \in \Sigma$,

$$\begin{aligned} \sigma \in \widehat{com}^*_{ij}(s) \quad &\Rightarrow \quad s\sigma \in L(\hat{\mathbf{R}}_\mathbf{1} \wedge \hat{\mathbf{R}}_\mathbf{2}/\mathbf{G}) \quad \Rightarrow \quad s\sigma \in L(\hat{R}_1 \wedge \hat{R}_2) \\ &\Rightarrow \quad s\sigma \in L(\mathbf{R_1} \wedge \mathbf{R_2}/\mathbf{G}) \quad \Rightarrow \quad s\sigma \in L(R_1 \wedge R_2) \end{aligned}$$

Let $R = (R_1 \times R_2) = (\Sigma, X, \xi, x_0, X_m)$ and $\hat{R} = (\hat{R}_1 \times \hat{R}_2) = (\Sigma, \hat{X}, \hat{\xi}, \hat{x}_0, \hat{X}_m)$. In addition, assign the following values to the end state of s in R and $\hat{R}$: $\xi(s, x_0) = (x_1, x_2) = x$ and $\hat{\xi}(s, \hat{x}_0) = (\hat{x}_1, \hat{x}_2) = \hat{x}$. For simplicity, transitions will be represented in the following form: $\tau = (x, \sigma, y)$ and $\hat{\tau} = (\hat{x}, \sigma, \hat{y})$ where $x, y \in X$, $\hat{x}, \hat{y} \in \hat{X}$ and $\sigma \in \Sigma$. Thus,

$$\begin{aligned} s\sigma \in L(\hat{R}) \quad &\Rightarrow \quad (\hat{x}, \sigma, \hat{y})! \quad \Rightarrow \quad \hat{\tau}! \\ s\sigma \in L(R) \quad &\Rightarrow \quad (x, \sigma, y)! \quad \Rightarrow \quad \tau! \end{aligned}$$

Taking the above into account,

$$\begin{aligned} \sigma \in \widehat{com}^*_{ij}(s) \quad &\Rightarrow \quad \hat{\tau} \in (\hat{C}_{ij} \cup \hat{N}^*_{ij}) \qquad &&(\text{Construction of } \widehat{com}^*_{ij}(s)) \\ &\Rightarrow \quad \hat{\tau} \in \hat{C}_{ij} \vee \hat{\tau} \in \hat{N}^*_{ij} &&(\hat{C}_{ij} \cap \hat{N}^*_{ij} = \emptyset) \end{aligned}$$

Thus, there are two primary cases: $\hat{\tau} \in \hat{C}_{ij}$ and $\hat{\tau} \in \hat{N}^*_{ij}$. In addition, each of these primary cases will have two subcases: $i = 1$, $j = 2$ and $i = 2$, $j = 1$. The proof for the first primary case, $\hat{C}_{ij}$, will be based on the proof of comparability from $\mathbf{R}$ to $\hat{\mathbf{R}}$ with respect to $\mathbf{G}$ given in Theorem 1. In contrast, the examination of $\hat{N}^*_{ij}$ will be far more discussion and example-based.

5.4 $\hat{C}_{ij}$

Case 1 ($\hat{\tau} \in \hat{C}_{21}$).

$$\begin{aligned}
\hat{\tau} \in \hat{C}_{21} &\Rightarrow \sigma \in (\Sigma_{2,o} - \Sigma_{1,o}) \wedge \hat{x}_1 \neq \hat{y}_1 && \text{(Definition of } \hat{C}_{21}\text{)} \\
&\Rightarrow \sigma \in (\Sigma_{2,o} - \Sigma_{1,o}) \wedge x_1 \neq y_1 && \text{(Contrapositive of } x_1 = y_1 \Rightarrow \mu_1(x_1) = \mu_1(y_1)\text{)} \\
&\Rightarrow \tau \in C_{21} && \text{(Definition of } C_{21}\text{)}
\end{aligned}$$

Case 2 ($\hat{\tau} \in \hat{C}_{12}$). This case is analogous to that of $\hat{\tau} \in \hat{C}_{21}$.

5.5 $\hat{N}^*_{ij}$

Unfortunately, at this juncture the proof of the primary conjecture is forced to come to a screeching halt due to the following counterexample to the consistency portion of the problem.

5.5.1 Communication Scheme

Example In the previous chapter, the running example made a full examination of communication between $\hat{\mathbf{R}}_\mathbf{1}$ and $\hat{\mathbf{R}}_\mathbf{2}$. This counterexample will focus on communication between $\mathbf{R_1}$ and $\mathbf{R_2}$, which will now be described in similar detail. To begin, line **1** of the *Main* function calculates the reachable product of the two agents as shown in Figure 5.1. From this, the transitions that must be communicated to maintain correctness (as determined on line **2** of *Main*) are listed in Table 5.1. Note that each $(x, \sigma, y) \in C_{ij}$ may have a "counterpart" $(\mu(x), \sigma, \mu(y))$ in $\hat{C}_{ij}$ or $\hat{N}^*_{ij}$ that is also communicated; these transitions are also listed in the table.

To review, the transitions communicated for correctness by reduced agents ($\hat{\mathbf{R}}_\mathbf{1}$, $\hat{\mathbf{R}}_\mathbf{2}$) can be found in Table 4.1. Both of these sets highlight two important points:

1. As proven in the previous section, for every state $\hat{x} \in \hat{X}$ in $\hat{\mathbf{R}}$ that communicates an event for correctness, each state $x \in \mu^{-1}(\hat{x})$ in $\mathbf{R}$ also communicates the same event for correctness. Specifically, consider $((4, iii), a_2, (2, iii)) \in \hat{C}_{21}$ where $\mu^{-1}((4, iii)) = \{(4, iii), (7, iii)\}$ (note that $(4, v)$ and $(7, v)$ are not reachable in $\mathbf{R}$). Both $((4, iii), a_2, (2, v))$ and $((7, iii), a_2, (2, v))$ are included in C_{21}.

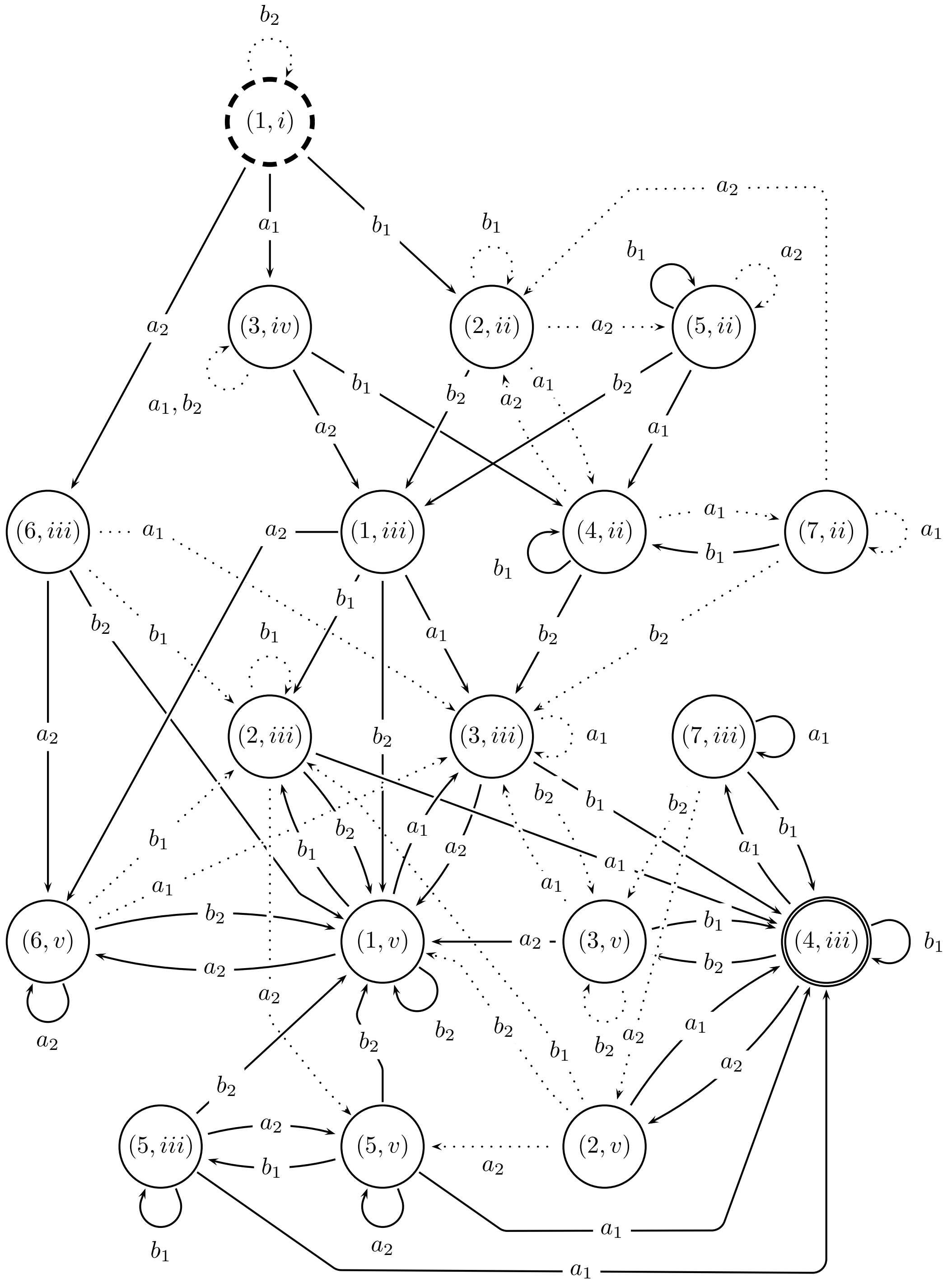


Figure 5.1: Running Example - Reachable product of $\mathbf{S_1}$ and $\mathbf{S_2}$ (Review of Figure 3.9)

C_{12}	**Counterpart**	C_{21}	**Counterpart**
$((1,i),a_1,(3,iv))$	Not Communicated	$((1,iii),a_2,(6,v))$	$((1,iii),a_2,(1,iii)) \in \hat{N}_{21}$
$((1,i),b_1,(2,ii))$	$((1,i),b_1,(2,ii)) \in \hat{C}_{12}$	$((1,v),a_2,(6,v))$	$((1,iii),a_2,(1,iii)) \in \hat{N}_{21}$
$((1,v),a_1,(3,iii))$	Not Communicated	$((2,ii),a_2,(5,ii))$	$((2,ii),a_2,(5,ii)) \in \hat{C}_{21}$
$((1,v),b_1,(2,iii))$	$((1,iii),b_1,(2,iii)) \in \hat{N}_{12}$	$((2,ii),b_2,(1,iii))$	$((2,ii),b_2,(1,iii)) \in \hat{C}_{21}$
$((2,v),a_1,(4,iii))$	Not Communicated	$((2,iii),a_2,(5,v))$	$((2,iii),a_2,(5,iii)) \in \hat{C}_{21}$
$((2,v),b_1,(2,iii))$	Not Communicated	$((2,iii),b_2,(1,v))$	$((2,iii),b_2,(1,iii)) \in \hat{C}_{21}$
$((3,iv),b_1,(4,ii))$	$((3,i),b_1,(4,ii)) \in \hat{C}_{12}$	$((2,v),a_2,(5,v))$	$((2,iii),a_2,(5,iii)) \in \hat{C}_{21}$
$((3,v),a_1,(3,iii))$	Not Communicated	$((2,v),b_2,(1,v))$	$((2,iii),b_2,(1,iii)) \in \hat{C}_{21}$
$((3,v),b_1,(4,iii))$	$((3,iii),b_1,(4,iii)) \in \hat{N}_{12}$	$((3,iii),a_2,(1,v))$	$((3,iii),a_2,(1,iii)) \in \hat{C}_{21}$
$((5,v),a_1,(4,iii))$	Not Communicated	$((3,iv),a_2,(1,iii))$	$((3,i),a_2,(1,iii)) \in \hat{C}_{21}$
$((5,v),b_1,(5,iii))$	Not Communicated	$((3,v),a_2,(1,v))$	$((3,iii),a_2,(1,iii)) \in \hat{C}_{21}$
$((6,v),a_1,(3,iii))$	Not Communicated	$((4,ii),a_2,(2,ii))$	$((4,ii),a_2,(2,ii)) \in \hat{C}_{21}$
$((6,v),b_1,(2,iii))$	$((1,iii),b_1,(2,iii)) \in \hat{N}_{12}$	$((4,ii),b_2,(3,iii))$	$((4,ii),b_2,(3,iii)) \in \hat{C}_{21}$
		$((4,iii),a_2,(2,v))$	$((4,iii),a_2,(2,iii)) \in \hat{C}_{21}$
		$((4,iii),b_2,(3,v))$	$((4,iii),b_2,(3,iii)) \in \hat{C}_{21}$
		$((5,ii),b_2,(1,iii))$	$((5,ii),b_2,(1,iii)) \in \hat{C}_{21}$
		$((5,iii),b_2,(1,v))$	$((5,iii),b_2,(1,iii)) \in \hat{C}_{21}$
		$((5,v),b_2,(1,v))$	$((5,iii),b_2,(1,iii)) \in \hat{C}_{21}$
		$((6,iii),b_2,(1,v))$	Not Communicated
		$((6,v),b_2,(1,v))$	Not Communicated
		$((7,ii),a_2,(2,ii))$	$((4,ii),a_2,(2,ii)) \in \hat{C}_{21}$
		$((7,ii),b_2,(3,iii))$	$((4,ii),b_2,(3,iii)) \in \hat{C}_{21}$
		$((7,iii),a_2,(2,v))$	$((4,iii),a_2,(2,iii)) \in \hat{C}_{21}$
		$((7,iii),b_2,(3,v))$	$((4,iii),b_2,(3,iii)) \in \hat{C}_{21}$

Table 5.1: Running Example - Correctness Sets C_{12} and C_{21}

2. Although most of the transmissions sent by $\mathbf{R_1}$ and $\mathbf{R_2}$ for correctness are also sent by $\hat{\mathbf{R}}_\mathbf{1}$ and $\hat{\mathbf{R}}_\mathbf{2}$ for correctness or consistency, several of them are not. Specifically, consider $((1,v),a_1,(3,iii)) \in C_{12}$ where $\mu((1,v)) = (1,ii)$. In $\hat{\mathbf{R}}$, a_2 is not communicated at $(1,ii)$ for either correctness or consistency.

Given the results from the first few steps of the method, line **3** calculates the "viewpoint" automaton for agent $\mathbf{R_1}$ as shown in Figure 5.2 and determines the associated consistency transitions. There are three composite states in $\tilde{R}_1$ that contain more than one state from $\hat{\mathbf{R}}$: $\{(1,iii),(1,v)\}$,

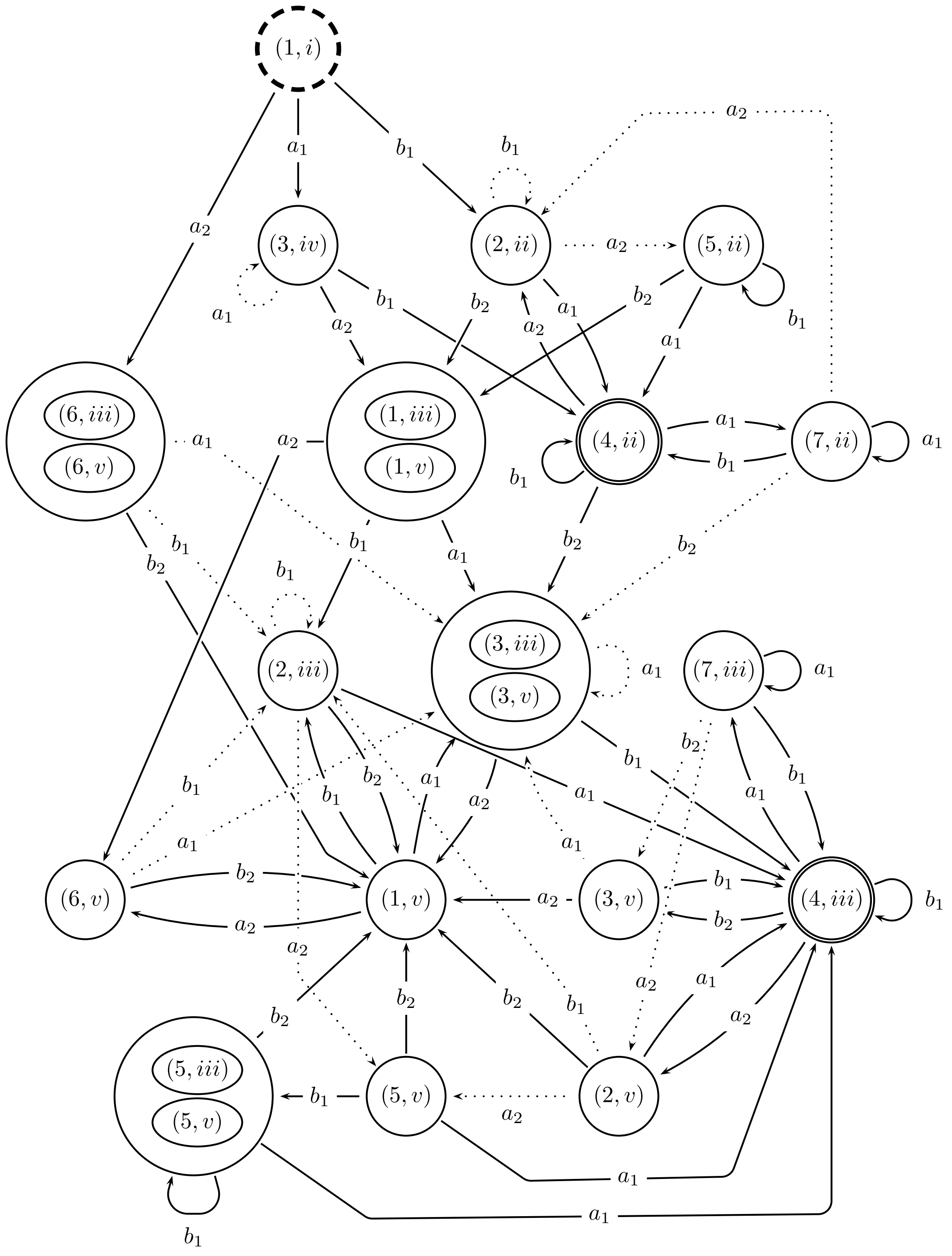


Figure 5.2: Running Example - $\tilde{R}_1$ Generated by $\mathcal{N}_{12}(C_{12}, C_{21})$

$\{(3, iii), (3, v)\}$, $\{(5, iii), (5, v)\}$ and $\{(6, iii), (6, v)\}$. As such, the consistency results shown in Table 5.2 are restricted to these states.

N_{12}^{max}		
Transition	**Cause**	**State in $\tilde{R}_1$**
$((1, iii), a_1, (3, iii))$	$((1, v), a_1, (3, iii)) \in C_{12}$	$\{(1, iii), (1, v)\}$
$((1, iii), b_1, (2, iii))$	$((1, v), b_1, (2, iii)) \in C_{12}$	$\{(1, iii), (1, v)\}$
$((3, iii), a_1, (3, iii))$	$((3, v), a_1, (3, iii)) \in C_{12}$	$\{(3, iii), (3, v)\}$
$((3, iii), b_1, (4, iii))$	$((3, v), b_1, (4, iii)) \in C_{12}$	$\{(3, iii), (3, v)\}$
$((5, iii), a_1, (4, iii))$	$((5, v), a_1, (4, iii)) \in C_{12}$	$\{(5, iii), (5, v)\}$
$((5, iii), b_1, (5, iii))$	$((5, v), b_1, (5, iii)) \in C_{12}$	$\{(5, iii), (5, v)\}$
$((6, iii), a_1, (3, iii))$	$((6, v), a_1, (3, iii)) \in C_{12}$	$\{(6, iii), (6, v)\}$
$((6, iii), b_1, (2, iii))$	$((6, v), b_1, (2, iii)) \in C_{12}$	$\{(6, iii), (6, v)\}$

Table 5.2: Running Example - Consistency Set N_{12}^{max}

At line **4**, it is agent $\mathbf{R_2}$'s turn to establish its observations of the system and transmissions required for consistency; Figure 5.3 represents the viewpoint automaton. There are a great deal more composite states in $\tilde{R}_2$ than in $\tilde{R}_1$. However, this does not result in a large increase in the number of transmissions to maintain consistency, as shown in Table 5.3.

N_{21}^{*}		
Transition	**Cause**	**State in $\tilde{R}_2$**
$((1, v), b_2, (1, v))$	$((3, v), a_2, (1, v)) \in C_{21}$	$\{(1, v), (3, v)\}$
$((3, v), b_2, (3, v))$	$((1, v), b_2, (1, v)) \in N_{21}^{*}$	$\{(1, v), (3, v)\}$
$((5, ii), a_2, (5, ii))$	$((2, ii), a_2, (5, ii)) \in C_{21}$	$\{(2, ii), (4, ii), (5, ii), (7, ii)\}$
$((5, iii), a_2, (5, v))$	$((2, iii), a_2, (5, v)) \in C_{21}$	$\{(2, iii), (4, iii), (5, iii), (7, iii)\}$
$((5, v), a_2, (5, v))$	$((2, v), a_2, (5, v)) \in C_{21}$	$\{(2, v), (5, v)\}$
$((6, v), a_2, (6, v))$	$((1, v), a_2, (6, v)) \in C_{21}$	$\{(1, v), (6, v)\}$

Table 5.3: Running Example - Consistency Set N_{21}^{*}

Once N_{21}^{*} has been determined, a recalculation of agent $\mathbf{R_1}$'s (potentially) minimal communications for consistency can take place at line **5** using the automaton given in Figure 5.4. Interestingly, the composite states for N_{12}^{min} are nearly identical to those for N_{12}^{max} with one notable exception:

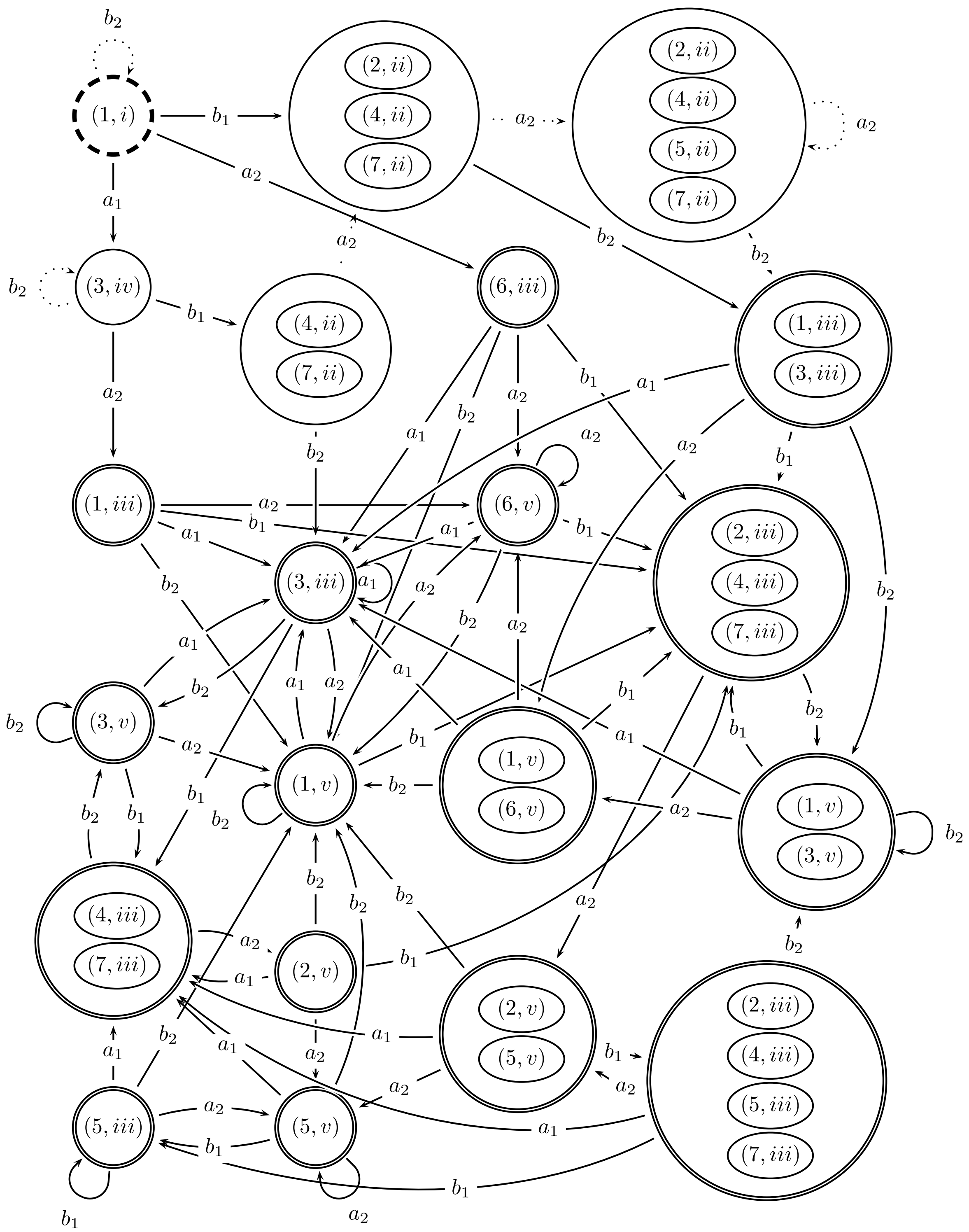


Figure 5.3: Running Example - $\tilde{R}_2$ Generated by $\mathcal{N}_{21}(C_{21}, C_{12} \cup N_{12}^{max})$

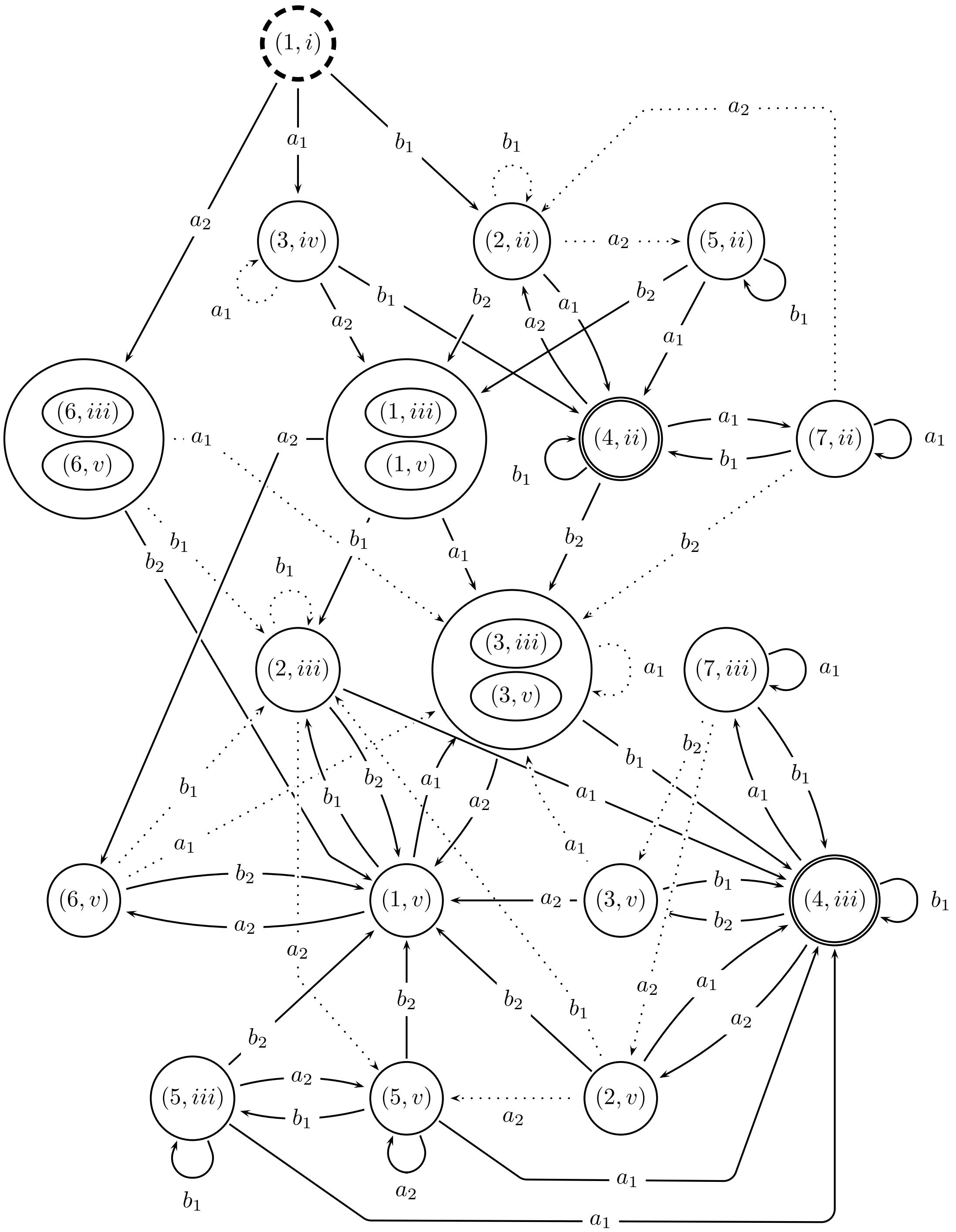


Figure 5.4: Running Example - $\tilde{R}_1$ Generated by $\mathcal{N}_{12}(C_{12}, C_{21} \cup N^*_{21})$

$\{(5, iii), (5, v)\}$ is missing, due to $((5, iii), a_2, (5, v)) \in N^*_{21}$ eliminating the ϵ-transition between them. This results in the exclusion of transmissions for state $(5, iii)$ that were in N^{max}_{12}. The members of N^{min}_{12} are listed in Table 5.4.

N^{min}_{12}		
Transition	**Cause**	**State in $\tilde{R}_1$**
$((1, iii), a_1, (3, iii))$	$((1, v), a_1, (3, iii)) \in C_{12}$	$\{(1, iii), (1, v)\}$
$((1, iii), b_1, (2, iii))$	$((1, v), b_1, (2, iii)) \in C_{12}$	$\{(1, iii), (1, v)\}$
$((3, iii), a_1, (3, iii))$	$((3, v), a_1, (3, iii)) \in C_{12}$	$\{(3, iii), (3, v)\}$
$((3, iii), b_1, (4, iii))$	$((3, v), b_1, (4, iii)) \in C_{12}$	$\{(3, iii), (3, v)\}$
$((6, iii), a_1, (3, iii))$	$((6, v), a_1, (3, iii)) \in C_{12}$	$\{(6, iii), (6, v)\}$
$((6, iii), b_1, (2, iii))$	$((6, v), b_1, (2, iii)) \in C_{12}$	$\{(6, iii), (6, v)\}$

Table 5.4: Running Example - Consistency Set N^{min}_{12}

At line **7**, all possible transitions sets within the range bounded by N^{max}_{12} and N^{min}_{12} that do *not* violate consistency with C_{12}, C_{21} and N^*_{21} are flagged. As seen previously in the example with the reduced set of supervisors, there are four possible sets:

$N_1 = N^{min}_{12}$

$N_2 = N^{min}_{12} \cup \{((5, iii), a_1, (4, iii))\}$

$N_3 = N^{min}_{12} \cup \{((5, iii), b_1, (5, iii))\}$

$N_4 = N^{max}_{12}$

Unlike the reduced supervisors, however, it is not N^{max}_{12} that is the minimal set but N^{min}_{12}. Again, in the interests of brevity, all the automata and operations associated with this decision will not be included. Instead, the following informal discussion will highlight the most important points.

Consider $\mathcal{N}_{21}(C_{21}, C_{12} \cup N^{min}_{12})$ first with respect to Figure 5.3. The exclusion of $((5, iii), a_1, (4, iii))$ and $((5, iii), b_1, (5, iii))$ from N^{min}_{12} reduces the events on transitions $((5, iii), a_1, \{(4, iii), (7, iii)\})$ and $((5, iii), b_1, (5, iii))$ to ϵ. Although the self-loop does not affect the structure of $\tilde{R}_2$ in the slightest, the other transition does: $(5, iii)$ becomes composite state $\{(4, iii), (5, iii), (7, iii)\}$. However, this is a subset of an existing state $\{(2, iii), (4, iii), (5, iii), (7, iii)\}$, no other new states are introduced and all existing states are preserved. Thus, $\mathcal{N}_{21}(C_{21}, C_{12} \cup N^{min}_{12}) = N^*_{21}$.

Now consider $\mathcal{N}_{12}(C_{12} \cup N_{12}^{min}, C_{21} \cup N_{21}^{*})$ with reference to Figure 5.2. Both $((5,iii), a_1, (4,iii))$ and $((5,iii), b_1, (5,iii))$ were included to be consistent with transmissions made by $(5,v)$ due to composite state $\{(5,iii),(5,v)\}$. This state was generated solely due to the reduction of $((5,iii), a_2, (5,v))$ to ϵ as the transition is not necessary for $\mathbf{R_i}$'s correctness. However, $((5,iii), a_2, (5,v)) \in N_{21}^{*}$ which results in the elimination of the ϵ transition, the composite state and the need to communicate at $(5,iii)$ for consistency purposes. This also does not affect any composite states in the automaton. Thus, $\mathcal{N}_{12}(C_{12} \cup N_{12}^{min}, C_{21} \cup N_{21}^{*}) = \emptyset$.

As a result of the calculations made above, every set N_k, $1 \leq k \leq 4$ is flagged. The decision at line **7** is made simple as there is no doubt which set has the fewest elements: N_{12}^{min}. All that remains for lines **8** through **10** is to create R_1^* and R_2^* (which are extremely similar to Figures 5.4 and 5.3) and determine the communication scheme $(com_{12}^{*}, com_{21}^{*})$ using state-based mappings (ϕ_{12}, ϕ_{21}) as given in Table 5.5.

5.5.2 Examination of Counterexample

It is rather remarkable that, considering the complexity and length of this example, demonstrating what makes it a counterexample requires only one event. Consider the string $s = a_2$; $s \in L(\mathbf{R_1} \wedge \mathbf{R_2}/\mathbf{G})$ and through comparability, $s \in L(\hat{\mathbf{R}}_\mathbf{1} \wedge \hat{\mathbf{R}}_\mathbf{2}/\mathbf{G})$. In $\hat{R}_2^*$ (similar to Figure 4.5),

$$\tilde{\hat{\xi}}_2(s, \tilde{\hat{x}}_{2,0}) = \{(1,iii),(3,iii)\}$$

and in R_2^* (similar to Figure 5.3),

$$\tilde{\xi}_2(s, \tilde{x}_{2,0}) = (6,iii)$$

From the communication maps established for the running example,

$$\begin{array}{rcccl} \widehat{com}_{21}^{*}(s) & = & \phi_{21}(\{(1,iii),(3,iii)\}) & = & \{a_2\} \\ com_{21}^{*}(s) & = & \phi_{21}((6,iii)) & = & \{b_2\} \end{array}$$

Therefore, in this instance, $\widehat{com}_{ij}^{*}(s) \nsubseteq com_{ij}^{*}(s)$ and the primary conjecture is proven to be false.

In this particular example, the relationship characterized by comparability did not guarantee that $(6,iii) \in X$ and $\mu((6,iii)) = (1,iii) \in \hat{X}$ shared the structural properties that would result

$\tilde{x}_1 \in \tilde{X}_1$	$\phi_{12}(\tilde{x}_1)$	$\tilde{x}_2 \in \tilde{X}_2$	$\phi_{21}(\tilde{x}_2)$
$\{(1,iii),(1,v)\}$	$\{a_1,b_1\}$	$\{(2,ii),(4,ii),(7,ii)\}$	$\{a_2,b_2\}$
$\{(3,iii),(3,v)\}$	$\{a_1,b_1\}$	$\{(2,ii),(4,ii),(5,ii),(7,ii)\}$	$\{a_2,b_2\}$
$\{(6,iii),(6,v)\}$	$\{a_1,b_1\}$	$\{(4,ii),(7,ii)\}$	$\{a_2,b_2\}$
$(1,i)$	$\{a_1,b_1\}$	$\{(1,iii),(3,iii)\}$	$\{a_2\}$
$(1,v)$	$\{a_1,b_1\}$	$\{(2,iii),(4,iii),(7,iii)\}$	$\{a_2,b_2\}$
$(2,ii)$	$\emptyset$	$\{(2,iii),(4,iii),(5,iii),(7,iii)\}$	$\{a_2,b_2\}$
$(2,iii)$	$\emptyset$	$\{(4,iii),(5,iii),(7,iii)\}$	$\{a_2,b_2\}$
$(2,v)$	$\{a_1,b_1\}$	$\{(4,iii),(7,iii)\}$	$\{a_2,b_2\}$
$(3,iv)$	$\{b_1\}$	$\{(1,v),(3,v)\}$	$\{a_2,b_2\}$
$(3,v)$	$\{a_1,b_1\}$	$\{(1,v),(6,v)\}$	$\{a_2,b_2\}$
$(4,ii)$	$\emptyset$	$\{(2,v),(5,v)\}$	$\{a_2,b_2\}$
$(4,iii)$	$\emptyset$	$(1,i)$	$\{a_2\}$
$(5,ii)$	$\emptyset$	$(1,iii)$	$\{a_2\}$
$(5,iii)$	$\emptyset$	$(3,iii)$	$\{a_2\}$
$(5,v)$	$\{a_1,b_1\}$	$(6,iii)$	$\{b_2\}$
$(6,v)$	$\{a_1,b_1\}$	$(3,iv)$	$\{a_2\}$
$(7,ii)$	$\emptyset$	$(1,v)$	$\{a_2,b_2\}$
$(7,iii)$	$\emptyset$	$(2,v)$	$\{a_2,b_2\}$
		$(3,v)$	$\{a_2,b_2\}$
		$(5,v)$	$\{a_2,b_2\}$
		$(6,v)$	$\{a_2,b_2\}$

Table 5.5: Running Example - State-Based Communication Mappings ϕ_{12} and ϕ_{21} for R_1^* and R_2^*

in an identical or reduced communication scheme for $(1,iii)$. The obvious question is why did this happen in this case?

To begin, examine the operations performed for ($\hat{\mathbf{R}}_\mathbf{1}$, $\hat{\mathbf{R}}_\mathbf{2}$). When determining $\hat{N}_{12}^{max}$, agent $\hat{\mathbf{R}}_\mathbf{1}$ does not communicate $((1,iii),a_1,(3,iii))$ to $\hat{\mathbf{R}}_\mathbf{2}$ because it is not required for correctness (since $iii = iii$) or consistency (since although $(1,iii)$ is in a composite state with $(1,i)$, a_1 is not communicated at state $(1,i)$), as shown in Figure 4.3. Then during the calculations for $\hat{N}_{21}^*$, agent $\hat{\mathbf{R}}_\mathbf{2}$ cannot distinguish $(1,iii)$ from $(3,iii)$ as $((1,iii),a_1,(3,iii))$ is not communicated, generating composite state $\{(1,iii),(3,iii)\}$, as illustrated in Figure 4.5. Since $((3,iii),a_2,(1,iii))$ is communicated for correctness, $((1,iii),a_2,(1,iii))$ is now communicated for consistency.

Now consider ($\mathbf{R}_\mathbf{1}$, $\mathbf{R}_\mathbf{2}$). When determining N_{12}^{max}, agent $\mathbf{R}_\mathbf{1}$ generates composite state

$\{(6, iii), (6, v)\}$ because $((6, iii), a_2, (6, v))$ is not communicated for correctness (since $6 = 6$), as given in Figure 5.2. As a result, both $((6, iii), a_1, (3, iii))$ and $((6, iii), b_1, (2, iii))$ are communicated to be consistent with similar transmissions made at state $(6, v)$. Then during the calculations of N^*_{21}, $(6, iii)$ is not made part of a composite state in $\tilde{R}_2$ because the only transitions in or out of that state are either observed by $\mathbf{R_2}$ or communicated by $\mathbf{R_1}$, as shown in Figure 5.3. Since a_2 is not transmitted at $(6, iii)$ for correctness and there is no reason to transmit it for consistency, the result is that a_2 is not communicated at state $(6, iii)$ of R^*_2.

It is important to note that even though the primary conjecture is incorrect in its formal expression, it may be informally correct in some cases. In this example, ($\mathbf{\hat{R}_1}$, $\mathbf{\hat{R}_2}$) does communicate "no worse or less" than ($\mathbf{R_1}$, $\mathbf{R_2}$), referring to the overall *number* of transmissions. Consider that for these two pairs of agents, $((6, iii)), a_2, (6, v))$ and $((1, iii), a_2, (1, iii)) \in N^*_{21}$ is the only case where a transmission for an event σ on a reduced state $\hat{x}$ is not also made for σ on x where $\mu(x) = \hat{x}$. Beyond this, numerous additional communications are made by ($\mathbf{R_1}$, $\mathbf{R_2}$) that are not made by ($\mathbf{\hat{R}_1}$, $\mathbf{\hat{R}_2}$). For example, C_{12} contains an additional eight transitions with no counterparts in $\hat{C}_{12}$; this results in eight additional transmissions that will be made by ($\mathbf{R_1}$, $\mathbf{R_2}$) for correctness that will not be made by ($\mathbf{\hat{R}_1}$, $\mathbf{\hat{R}_2}$).

Given that the running example proved to be a counterexample, the obvious question is whether it is possible for a ($\mathbf{R_1}$, $\mathbf{R_2}$) and ($\mathbf{\hat{R}_1}$, $\mathbf{\hat{R}_2}$) to satisfy the primary conjecture. Thankfully, it is possible and the example which proves it is very simple in comparison to the one examined thus far.

5.6 Proof of Concept Example

Consider Figures 5.5 through 5.7 representing a plant $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$ and implicitly-defined supervisors T_1 and T_2 where $T_i = (\Sigma, X_i, \xi_i, x_{i,0})$, $i \in \{1, 2\}$. As in the running example, supervisor T_i can control all events in Σ, but may only observe $\{a_i, b_i\}$.

Given the similar structure shared by the plant and the supervisors, it is not surprising that a state reduction with respect to the plant is possible. The implicitly-defined agents $\hat{T}_1$ and $\hat{T}_2$, given in Figures 5.8 and 5.9, respectively, are the proposed replacements where $\hat{T}_i = (\Sigma, \hat{X}_i, \hat{\xi}_i, \hat{x}_{i,0})$.

Supervisor T_i is language-equivalent to $\hat{T}_i$ with respect to $\mathbf{G}$, $i \in \{1, 2\}$. Define state mappings μ_1 and μ_2 from agent T_i to $\hat{T}_i$ as given in Table 5.6.

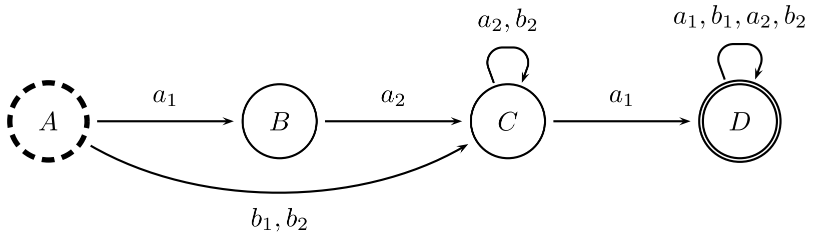


Figure 5.5: Proof of Concept Example - Plant **G**

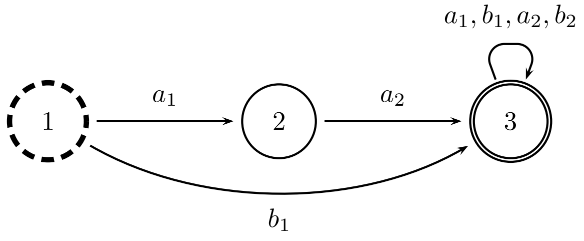


Figure 5.6: Proof of Concept Example - Supervisor T_1

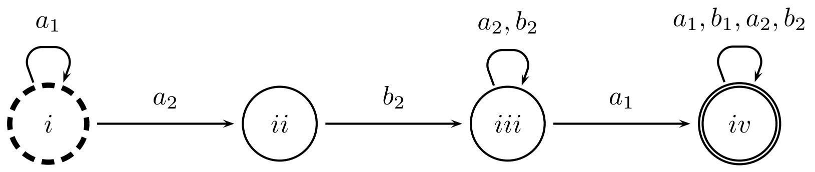


Figure 5.7: Proof of Concept Example - Supervisor T_2

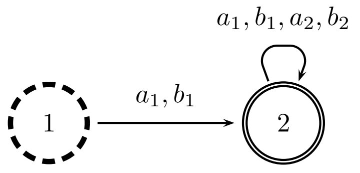


Figure 5.8: Proof of Concept Example - Reduced Supervisor $\hat{T}_1$

The information given thus far, not surprisingly, culminates in the declaration that T_i is comparable to $\hat{T}_i$ with respect to **G** via the state mapping μ_i. The behaviour of **G** when these agents enforce their control policies is shown in Figures 5.10 and 5.11.

Before submitting these agents to the minimal communication algorithm, they must first be made

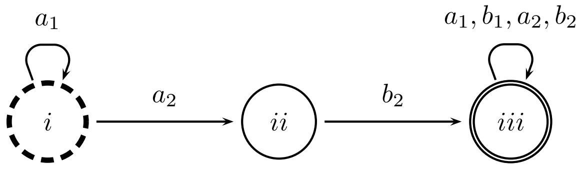


Figure 5.9: Proof of Concept Example - Reduced Supervisor $\hat{T}_2$

$x \in X_1$	$\mu_1(x)$	$x \in X_2$	$\mu_2(x)$
1	1	i	i
2	2	ii	ii
3	3	iii	iii
		iv	iii

Table 5.6: Proof of Concept Example - State Mappings μ_1 and μ_2

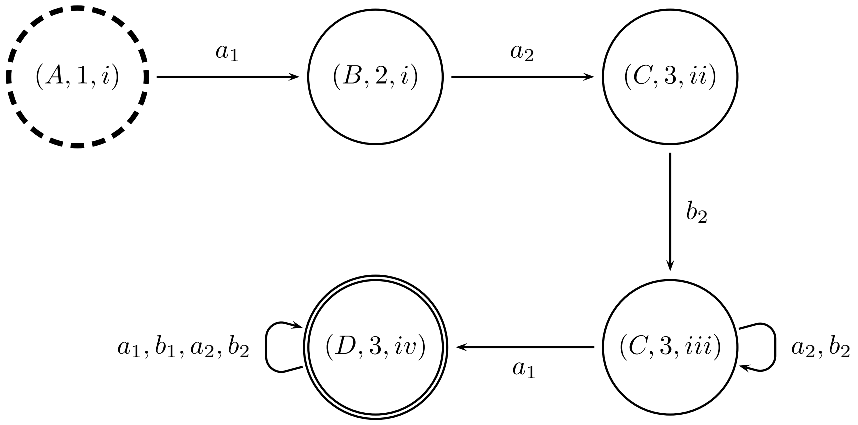


Figure 5.10: Proof of Concept Example - Result of Supervisors $T_1 \wedge T_2$ acting on Plant $\mathbf{G}$

into fully-defined supervisors using the explicit functions υ for T_i and υ^{μ} for $\hat{T}_i$, $i \in \{1, 2\}$. This process yields $(\mathbf{T_1}, \mathbf{T_2})$ and $(\hat{\mathbf{T}}_\mathbf{1}, \hat{\mathbf{T}}_\mathbf{2})$ where $\mathbf{T_i} = (T_i, \phi_i)$. Each of these supervisors is equivalent to its implicitly-defined counterpart. The only difference is in appearance; structurally, they are identical to the automata given in Figures 5.6 through 5.9, but each state now has a disabled self-loop for every event that was implicitly disabled.

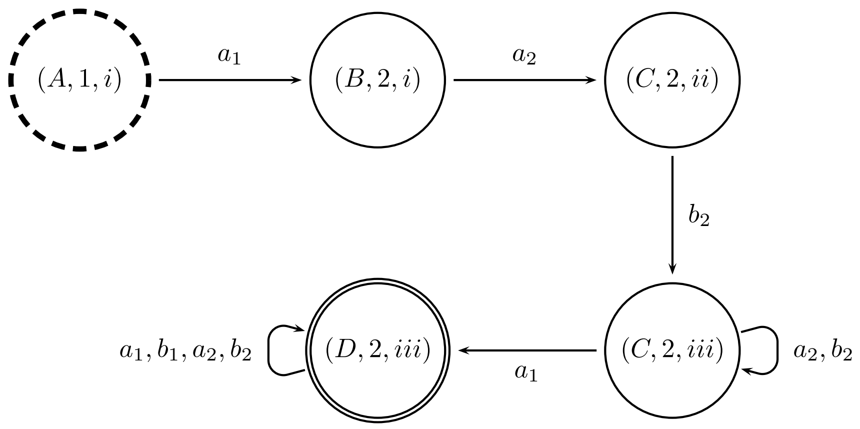


Figure 5.11: Proof of Concept Example - Result of Reduced Supervisors $\hat{T}_1 \wedge \hat{T}_2$ Acting on Plant $\mathbf{G}$

The first few steps of the *Main* function are concerned with calculating the reachable product of the supervisors and determining the transitions that must be communicated for correctness. The diagram given in Figure 5.12 and the sets C_{12} and C_{21} in Table 5.7 are the results of that process for $(\mathbf{T_1}, \mathbf{T_2})$.

C_{12}	C_{21}
$((1,iii),a_1(2,iv))$	$((2,i),a_2,(3,ii))$
$((3,iii),a_1,(3,iv))$	$((2,ii),a_2,(3,ii))$
$((2,iii),a_1,(2,iv))$	$((2,iii),a_2,(3,iii))$
	$((2,iv),a_2,(3,iv)$

Table 5.7: Proof of Concept Example - Correctness Sets C_{12} and C_{21}

Using these sets, the next step is to determine $\tilde{R}_1$ (as illustrated in Figure 5.13) and from that, N_{12}^{max} (as given in Table 5.8). Since all the states in $\tilde{R}_1$ are composite and C_{12} is nonempty, there are several transitions that must be communicated for consistency.

The next task for *Main* is to calculate $\tilde{R}_2$ (as shown in Figure 5.14) and N_{21}^{*} (as given in Table 5.9). Similar to $\tilde{R}_1$ and N_{12}^{max}, there are many composite states in $\tilde{R}_2$ and C_{21} is nonempty, leading to the addition of many communications for consistency.

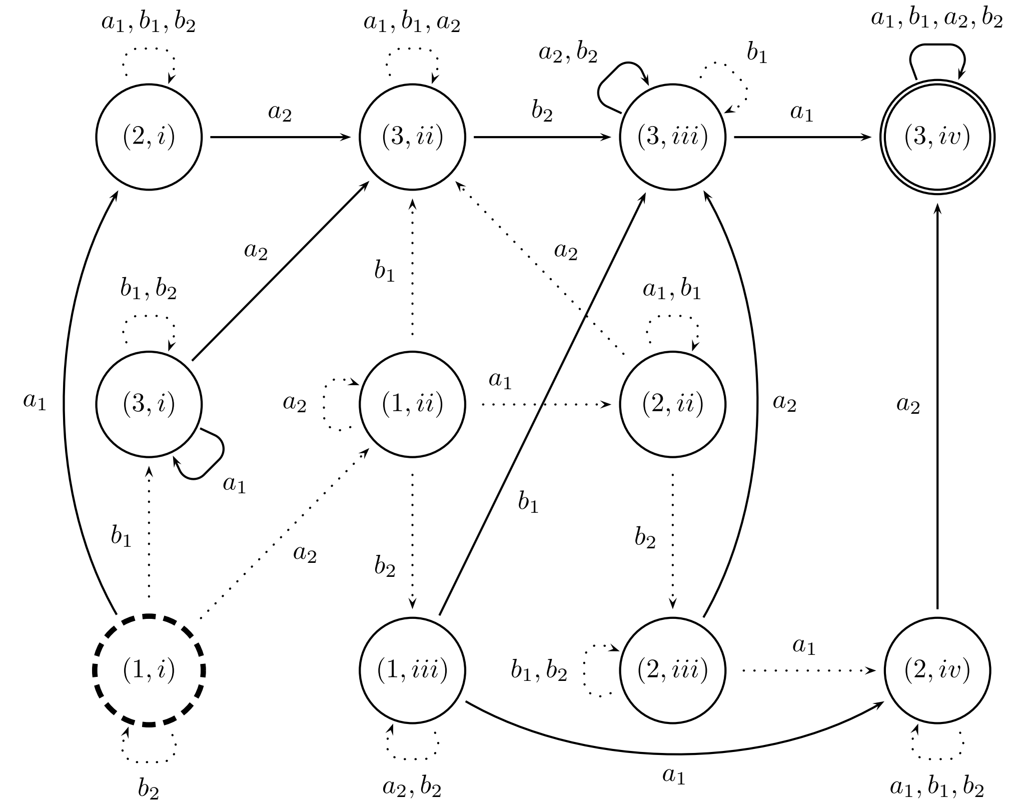


Figure 5.12: Proof of Concept Example - Supervisor $\mathbf{R} = (\mathbf{T_1} \times \mathbf{T_2})$

N_{12}^{max}		
Transition	**Cause**	**State in $\tilde{R}_1$**
$((1,i), a_1, (2,i))$	$((1,iii), a_1(2,iv)) \in C_{21}$	$\{(1,i),(1,ii),(1,iii)\}$
$((1,ii), a_1, (2,ii))$	$((1,iii), a_1(2,iv)) \in C_{21}$	$\{(1,i),(1,ii),(1,iii)\}$
$((2,i), a_1, (2,i))$	$((2,iii), a_1, (2,iv)) \in C_{21}$	$\{(2,i),(2,ii),(2,iii),(2,iv)\}$
$((2,ii), a_1, (2,ii))$	$((2,iii), a_1, (2,iv)) \in C_{21}$	$\{(2,i),(2,ii),(2,iii),(2,iv)\}$
$((2,iv), a_1, (2,iv))$	$((2,iii), a_1, (2,iv)) \in C_{21}$	$\{(2,i),(2,ii),(2,iii),(2,iv)\}$
$((3,i), a_1, (3,i))$	$((3,iii), a_1, (3,iv)) \in C_{21}$	$\{(3,i),(3,ii),(3,iii),(3,iv)\}$
$((3,ii), a_1, (3,ii))$	$((3,iii), a_1, (3,iv)) \in C_{21}$	$\{(3,i),(3,ii),(3,iii),(3,iv)\}$
$((3,iv), a_1, (3,iv))$	$((3,iii), a_1, (3,iv)) \in C_{21}$	$\{(3,i),(3,ii),(3,iii),(3,iv)\}$

Table 5.8: Proof of Concept Example - Consistency Set N_{12}^{max}

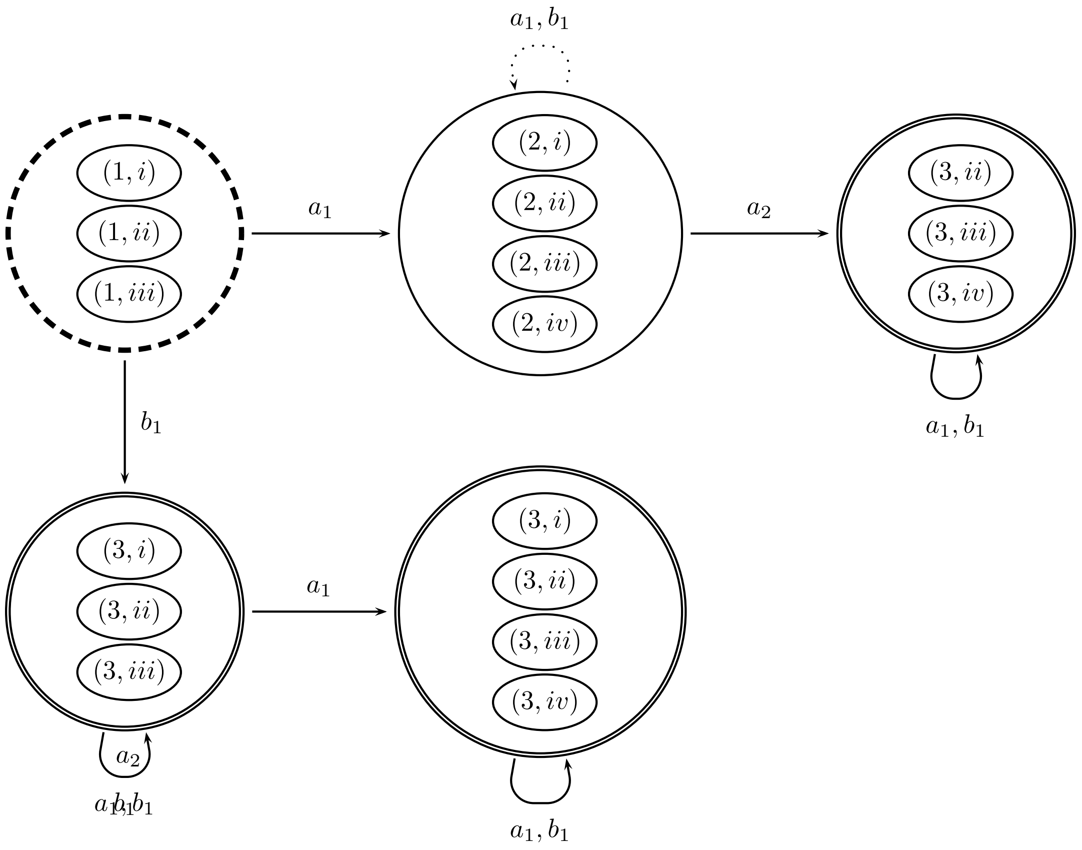


Figure 5.13: Proof of Concept Example - $\tilde{R}_1$ Generated by $\mathcal{N}_{12}(C_{12}, C_{21})$

N^*_{21}		
Transition	**Cause**	**State in $\tilde{R}_2$**
$((3,i), a_2, (3,ii))$	$((2,i), a_2, (3,ii)) \in C_{12}$	$\{(2,i), (3,i)\}$
$((3,ii), a_2, (3,ii))$	$((2,ii), a_2, (3,ii)) \in C_{12}$	$\{(2,ii), (3,ii)\}$
$((3,iii), a_2, (3,iii))$	$((2,iii), a_2, (3,iii)) \in C_{12}$	$\{(2,iii), (3,iii)\}$
$((3,iv), a_2, (3,iv))$	$((2,iv), a_2, (3,iv)) \in C_{12}$	$\{(2,iv), (3,iv)\}$
$((1,i), a_2, (1,ii))$,	$((3,i), a_2, (3,ii)) \in N_{21}$	$\{(1,i), (3,i)\}$
$((1,ii), a_2, (1,ii))$	$((3,ii), a_2, (3,ii)) \in N_{21}$	$\{(1,ii), (3,ii)\}$
$((1,iii), a_2, (1,iii))$	$((3,iii), a_2, (3,iii)) \in N_{21}$	$\{(1,iii), (3,iii)\}$

Table 5.9: Proof of Concept Example - Consistency Set N^*_{21}

The next step involves recalculating $\tilde{R}_1$ for N^{min}_{12}, as given in Figure 5.15 and Table 5.10, respectively. Since there are fewer composite states in this version of $\tilde{R}_1$ than that for N^{max}_{12}, there is

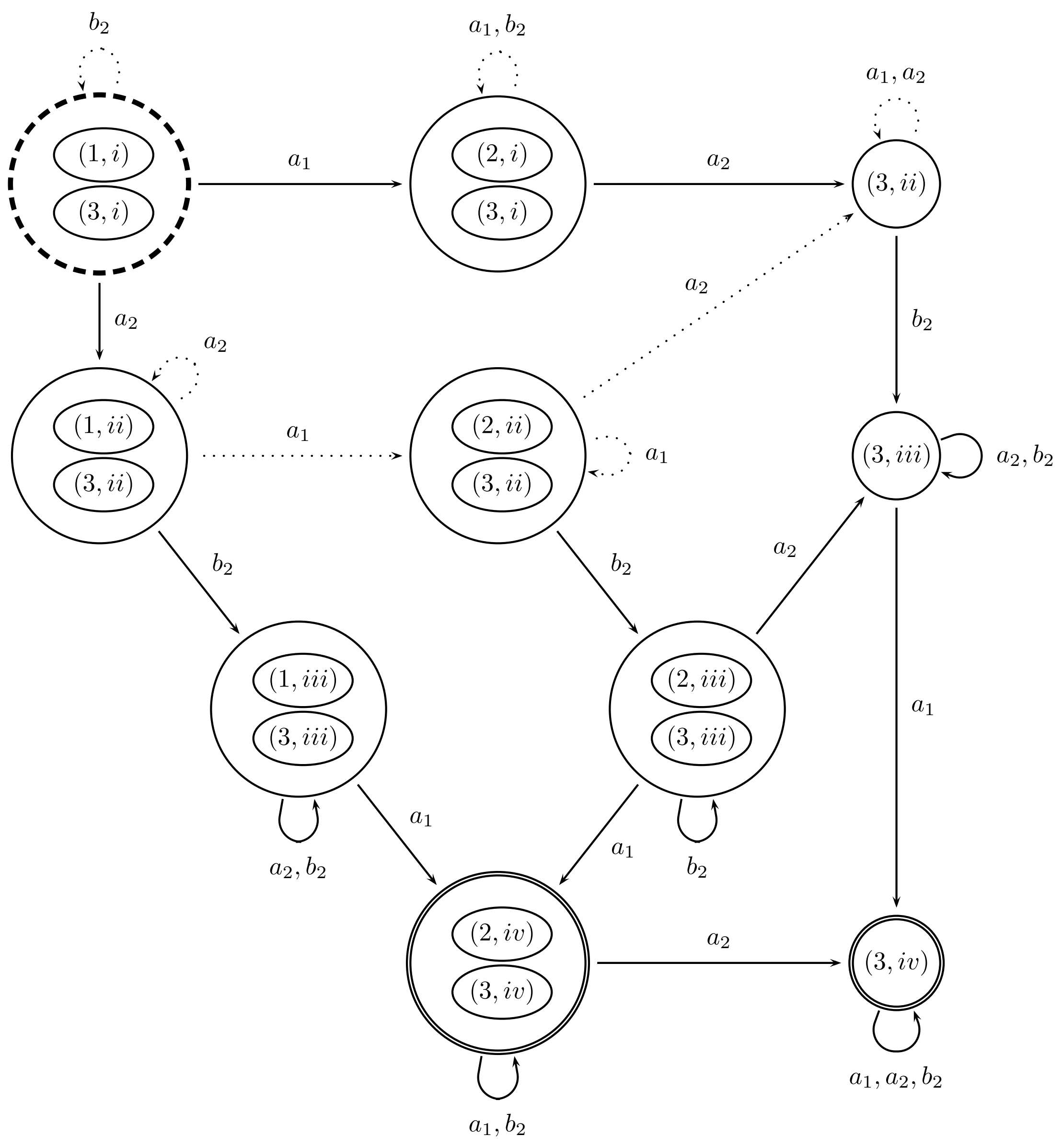


Figure 5.14: Proof of Concept Example - $\tilde{R}_2$ Generated by $\mathcal{N}_{21}(C_{21}, C_{12} \cup N_{12}^{max})$

a reduction in the number of transitions that must be communicated for consistency.

The next task is to determine N_{12}^{*} using N_{12}^{max} and N_{12}^{min} as the boundaries. To be succinct, it suffices to state informally that N_{12}^{min} is chosen because:

- $\mathcal{N}_{12}(C_{12} \cup N_{12}^{min}, C_{21} \cup N_{21}^{*}) = \emptyset$ since N_{12}^{min} is the result of $\mathcal{N}_{12}(C_{12}, C_{21} \cup N_{21}^{*})$
- $\mathcal{N}_{21}(C_{21}, C_{12} \cup N_{12}^{min}) = N_{21}^{*}$ since the $\tilde{R}_2$ generated by this operation has amalgamated states

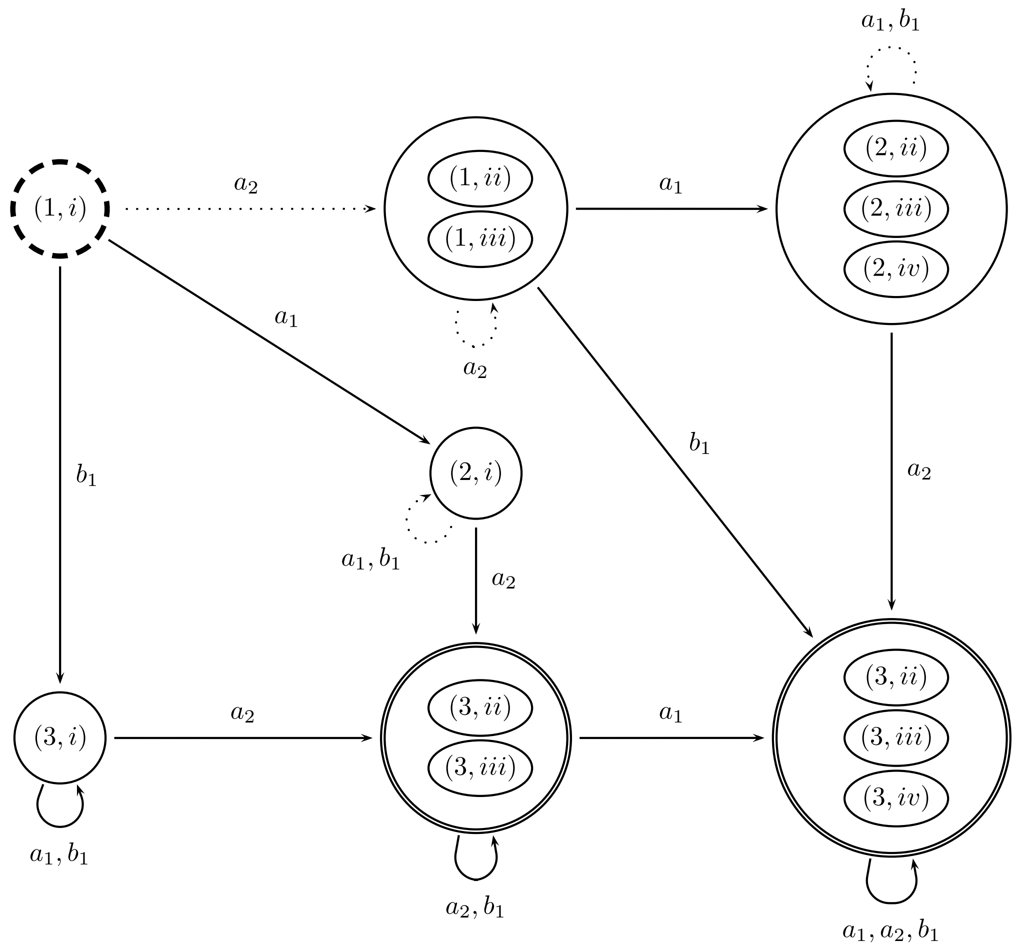


Figure 5.15: Proof of Concept Example - $\tilde{R}_1$ Generated by $\mathcal{N}_{12}(C_{12}, C_{21} \cup N^*_{21})$

N_{12}^{min}		
Transition	**Cause**	**State in $\tilde{R}_1$**
$((1,ii),a_1,(2,ii))$	$((1,iii),a_1(2,iv)) \in C_{21}$	$\{(1,ii),(1,iii)\}$
$((2,ii),a_1,(2,ii))$	$((2,iii),a_1,(2,iv)) \in C_{21}$	$\{(2,ii),(2,iii),(2,iv)\}$
$((2,iv),a_1,(2,iv))$	$((2,iii),a_1,(2,iv)) \in C_{21}$	$\{(2,ii),(2,iii),(2,iv)\}$
$((3,ii),a_1,(3,ii))$	$((3,iii),a_1,(3,iv)) \in C_{21}$	$\{(3,ii),(3,iii),(3,iv)\}$
$((3,iv),a_1,(3,iv))$	$((3,iii),a_1,(3,iv)) \in C_{21}$	$\{(3,ii),(3,iii),(3,iv)\}$

Table 5.10: Proof of Concept Example - Consistency Set N_{12}^{min}

$\{(1,i),(2,i)\}$ and $\{(2,i),(3,i)\}$ shown in Figure 5.14 into a single state $\{(1,i),(2,i),(3,i)\}$, which does not affect the outcome of the consistency calculations

Thus, $N^*_{12} = N^{min}_{12}$.

The next to last task is to determine the state-based communication mappings given C_{12}, C_{21}, N^*_{12} and N^*_{21}. The results appear in Table 5.11. Note that the final R^*_1 and R^*_2 are nearly identical to Figures 5.15 and 5.14, respectively.

$\tilde{x}_1 \in \tilde{X}_1$	$\phi_{12}(\tilde{x}_1)$	$\tilde{x}_2 \in \tilde{X}_2$	$\phi_{21}(\tilde{x}_2)$
$\{(1,ii),(1,iii)\}$	$\{a_1\}$	$\{(1,i),(2,i),(3,i)\}$	$\{a_2\}$
$\{(3,ii),(3,iii)\}$	$\{a_1\}$	$\{(1,ii),(3,ii)\}$	$\{a_2\}$
$\{(2,ii),(2,iii),(2,iv)\}$	$\{a_1\}$	$\{(2,ii),(3,ii)\}$	$\{a_2\}$
$\{(3,ii),(3,iii),(3,iv)\}$	$\{a_1\}$	$\{(1,iii),(3,iii)\}$	$\{a_2\}$
$(1,i)$	$\emptyset$	$\{(2,iii),(3,iii)\}$	$\{a_2\}$
$(2,i)$	$\emptyset$	$\{(2,iv),(3,iv)\}$	$\{a_2\}$
$(3,i)$	$\emptyset$	$(3,ii)$	$\{a_2\}$
		$(3,iii)$	$\{a_2\}$
		$(3,iv)$	$\{a_2\}$

Table 5.11: Proof of Concept Example - State-Based Communication Mappings ϕ_{12} and ϕ_{21}

The string-based mappings (com^*_{12}, com^*_{21}) are easily derived from these state-based ones. For example, given strings $s = a_1$ and $s' = a_1a_2$ where $s, s' \in L(T_1 \wedge T_2/\mathbf{G})$, $com^*_{12}(s) = \phi_{21}(\{(1,ii),(3,ii)\}) = \{a_2\}$. Hence, for the legal sequence s', agent R^*_2 would transmit a_2 to R^*_1.

In contrast, there is far less communication that occurs between $\hat{T}_1$ and $\hat{T}_2$. Figure 5.16 represents the results of *Main* determining their reachable product. Interestingly, neither $\hat{C}_{12}$ nor $\hat{C}_{21}$ contains any elements; $\hat{C}_{12} = \hat{C}_{21} = \emptyset$. This is due to the fact that state changes in $\hat{\mathbf{T}}_\mathbf{i}$ are only precipitated by events that agent i can observe.

With empty correctness sets, there is no need to explore the calculations associated with determining the transmissions for consistency; the result is that $\hat{N}^*_{12} = \emptyset$ and $\hat{N}^*_{21} = \emptyset$. This example demonstrates that not only is it possible for a reduction in state space to result in fewer communications, it is also possible to completely remove the need for communication altogether.

Note that both $(\mathbf{T_1}, \mathbf{T_2})$ and $(\hat{\mathbf{T}}_\mathbf{1}, \hat{\mathbf{T}}_\mathbf{2})$ are trim with respect to the plant $\mathbf{G}$. It is possible that

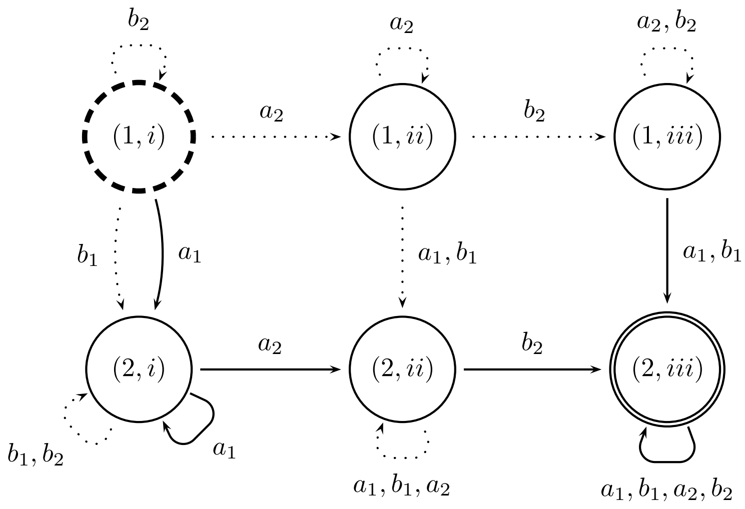


Figure 5.16: Proof of Concept Example - Supervisor $\hat{\mathbf{R}} = (\hat{\mathbf{T}}_\mathbf{1} \times \hat{\mathbf{T}}_\mathbf{2})$

the results for either pair of agents might have been much different if they had included synthesis-inaccessible states which "required" transmissions for correctness from the point of view of *Main*. The last example of this chapter will examine just such a scenario.

5.7 Trim vs. Untrim Example

Untrim? Why would a system designer ever use an agent that contains synthesis-inaccessible states? Does that not smack of sloppy, inefficient design? Well, the answer is both yes and no. Consider the following quandary: a designer has a rather simple "generic" pair of supervisors $(\mathbf{S_1}, \mathbf{S_2})$ that enforce a particular behaviour and a series of complex plants $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_n$ in need of control. Is it feasible to perform potentially large calculations to trim $(\mathbf{S_1}, \mathbf{S_2})$ n times for each plant $\mathbf{G}_i, 1 \leq i \leq n$, when perhaps only a few states of $(\mathbf{S_1}, \mathbf{S_2})$ will never be visited?

This situation takes on added weight if $(\mathbf{S_1}, \mathbf{S_2})$ must communicate. The *Main* function is designed to determine the transmissions necessary to distinguish states; *all* states, including those that are synthesis-inaccessible. Intuition hints that flagging transitions on inaccessible states for communication would result in additional transitions on accessible states being communicated for consistency where they would not have otherwise. Is it possible that the computation time saved by not trimming $(\mathbf{S_1}, \mathbf{S_2})$ will be outweighed by a potentially inefficient communication system? Conversely, is it possible that trimming $(\mathbf{S_1}, \mathbf{S_2})$ may not be worth the effort if no additional communication is engendered?

This example is essentially a "proof of concept" of the latter point, *i.e.*, a synthesis-inaccessible state may not cause superfluous transmissions. It will use a plant and supervisors identical or similar to those seen in the running example. As such, the focus of this example will not be on the process of determining their minimal communication schemes. Instead, greater weight will be given to a comparison of their transmissions with respect to the synthesized legal language.

The following plant $\mathbf{G} = (\Sigma, Q, \delta, q_0, Q_m)$ and supervisors $(\hat{\mathbf{W}}_\mathbf{1},\ \hat{\mathbf{W}}_\mathbf{2})$ where $\hat{\mathbf{W}}_\mathbf{i} = (\hat{W}_i, \hat{\phi}_i)$, $\hat{W}_i = (\Sigma, \hat{X}_i, \hat{\xi}_i, \hat{x}_{i,0})$ and $\hat{\phi}_i : \Sigma \times \hat{X}_i \rightarrow \Psi_i$ for $i \in \{1, 2\}$ appeared previously in this dissertation and are given in Figures 5.17, 5.18 and 5.19 for review. Although these agents are defined as

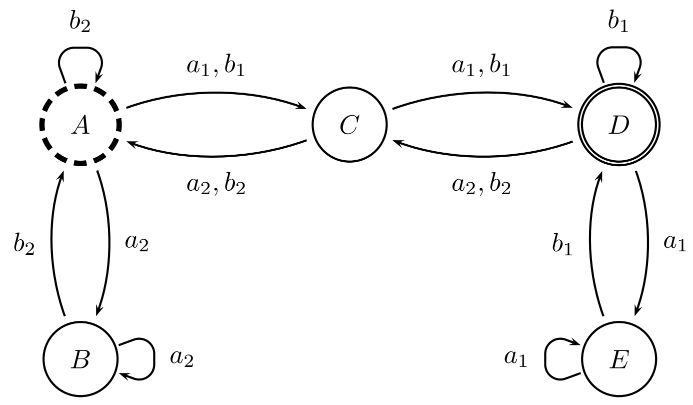


Figure 5.17: Trim vs. Untrim Example - Plant $\mathbf{G}$ (Review of Figure 2.1)

the "reduced" agents, the original supervisors do not differ greatly. In fact, $\mathbf{W_2}$ is not reduced; $\mathbf{W_2} = \hat{\mathbf{W}}_\mathbf{2}$ as shown in Figure 5.19. In addition, $\mathbf{W_1} = (W_1, \phi_1)$ (given in Figure 5.20) is simply an untrimmed version of its reduced counterpart; the only difference is synthesis-inaccessible state 5. It will come as no surprise that $\mathbf{W_i}$ is language-equivalent and comparable to $\hat{\mathbf{W}}_\mathbf{i}$ with respect to $\mathbf{G}$ via the state mappings given in Table 5.12.

As must be familiar to the reader by now, the first few steps of the *Main* function are concerned with taking the reachable product of the given supervisors and determining the transitions that

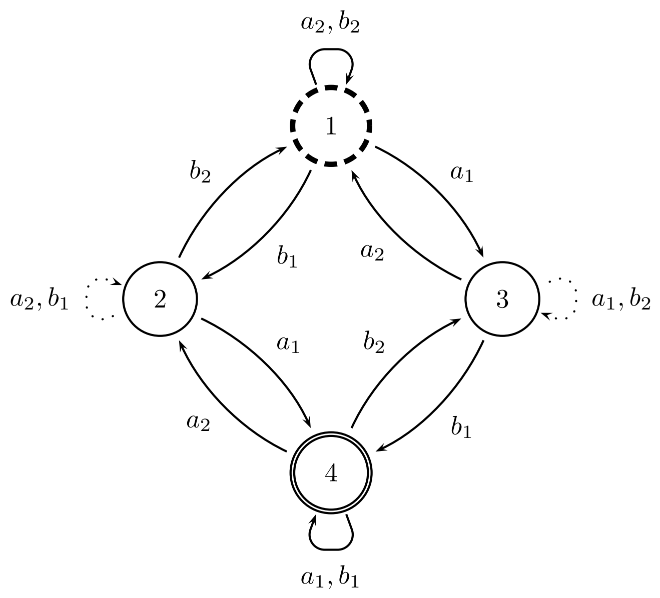


Figure 5.18: Trim vs. Untrim Example - Supervisor $\hat{\mathbf{W}}_{\mathbf{1}}$ (Review of Figure 2.2)

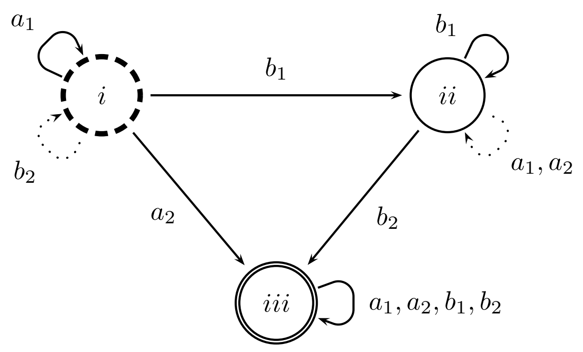


Figure 5.19: Trim vs. Untrim Example - Supervisor $\hat{\mathbf{W}}_{\mathbf{2}}$ (Review of Figure 2.5)

must be communicated for correctness. The results of these operations for ($\hat{\mathbf{W}}_{\mathbf{1}}$, $\hat{\mathbf{W}}_{\mathbf{2}}$) and ($\mathbf{W_1}$, $\mathbf{W_2}$) are shown in Figures 5.21 and 5.22, and Tables 5.13 and 5.14, respectively.

The remainder of *Main* is concerned with determining the transitions that must be communicated for consistency and calculating the "point of view" automata for each agent. Figures 5.23 through 5.26 and Tables 5.15 and 5.16 represent the results of those operations.

While R_1^* ($\hat{R}_1^*$) and R_2^* ($\hat{R}_2^*$) each give the observed behaviour of the original agents $\mathbf{W_1}$ ($\hat{\mathbf{W}}_{\mathbf{1}}$) and

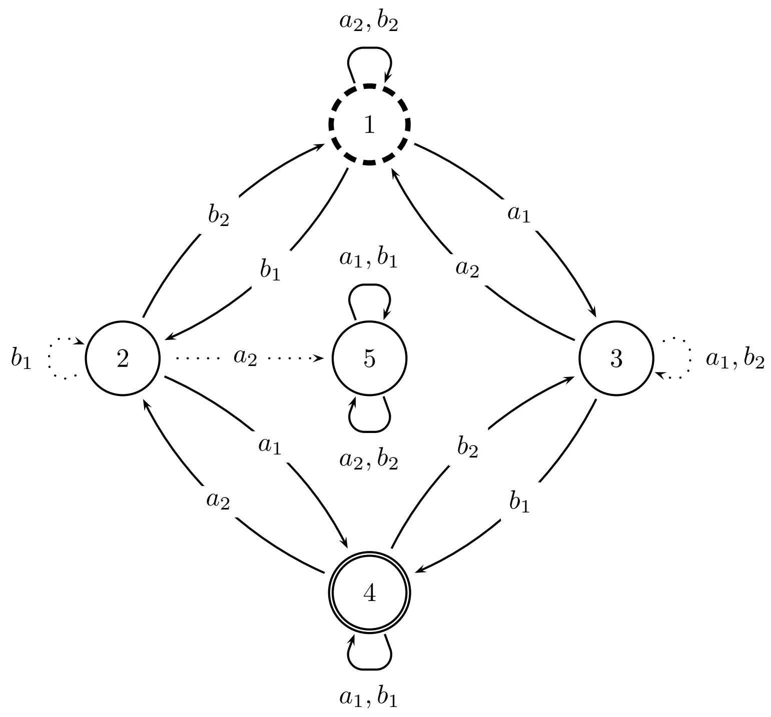


Figure 5.20: Trim vs. Untrim Example - Supervisor $\mathbf{W_1}$

$x \in X_1$	$\mu_1(x)$	$x \in X_2$	$\mu_2(x)$
1	1	i	i
2	2	ii	ii
3	3	iii	iii
4	4		
5	2		

Table 5.12: Trim vs. Untrim Example - State Mappings μ_1 and μ_2

$\mathbf{W_2}$ ($\hat{\mathbf{W}}_{\mathbf{2}}$), it is the combined behaviour of these agents that will synthesize the plant's legal language. Thus, the conjunction of these "viewpoint" agents must be taken. Figures 5.27 and 5.28 represent the conjunctions $\hat{R}^* = (\hat{R}_1^* \wedge \hat{R}_2^*)$ and $R^* = (R_1^* \wedge R_2^*)$, respectively. Note that a new naming convention is introduced; the combined state names such as $((1, iii), \{(1, iii), (2, iii), (3, iii), (4, iii), (5, iii)\})$ are simply too long to use in the diagram. A list mapping the short-form representations to their long-form names is given in Tables 5.17 and 5.18.

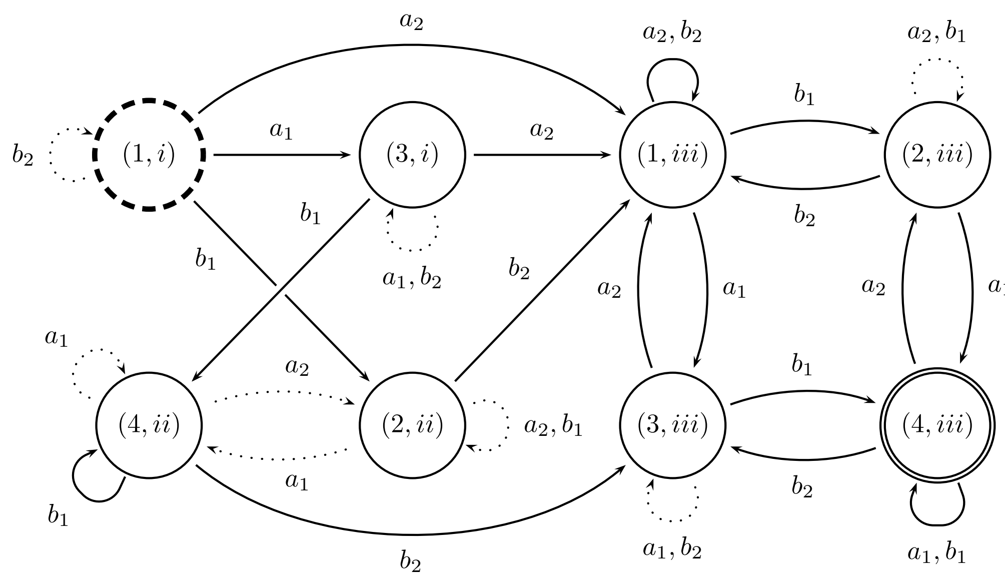


Figure 5.21: Trim vs. Untrim Example - Supervisor $\hat{\mathbf{R}} = (\hat{\mathbf{W}}_{\mathbf{1}} \times \hat{\mathbf{W}}_{\mathbf{2}})$ (Review of Figure 2.6)

$\hat{C}_{12}$	$\hat{C}_{21}$	
$((1, i), b_1, (2, ii))$	$((3, i), a_2, (1, iii))$	$((2, iii), b_2, (1, iii))$
$((3, i), b_1, (4, ii))$	$((2, ii), b_2, (1, iii))$	$((3, iii), a_2, (1, iii))$
	$((4, ii), a_2, (2, ii))$	$((4, iii), a_2, (2, iii))$
	$((4, ii), b_2, (3, iii))$	$((4, iii), b_2, (3, iii))$

Table 5.13: Trim vs. Untrim Example - Correctness Sets $\hat{C}_{12}$ and $\hat{C}_{21}$

C_{12}	C_{21}		
$((1, i), b_1, (2, ii))$	$((3, i), a_2, (1, iii))$	$((4, ii), b_2, (3, iii))$	$((4, iii), b_2, (3, iii))$
$((3, i), b_1, (4, ii))$	$((2, ii), a_2, (5, ii))$	$((2, iii), a_2, (5, iii))$	$((4, iii), a_2, (2, iii))$
	$((2, ii), b_2, (1, iii))$	$((2, iii), b_2, (1, iii))$	
	$((4, ii), a_2, (2, ii))$	$((3, iii), a_2, (1, iii))$	

Table 5.14: Trim vs. Untrim Example - Correctness Sets C_{12} and C_{21}

It is interesting to note that while $\hat{R}^* = (\hat{R}_1^* \wedge \hat{R}_2^*)$ is isomorphic to the implicitly-defined conjunction of $\hat{\mathbf{W}}_{\mathbf{1}}$ and $\hat{\mathbf{W}}_{\mathbf{2}}$ (as seen by viewing only the solid transitions in Figure 5.21), $R^* =$

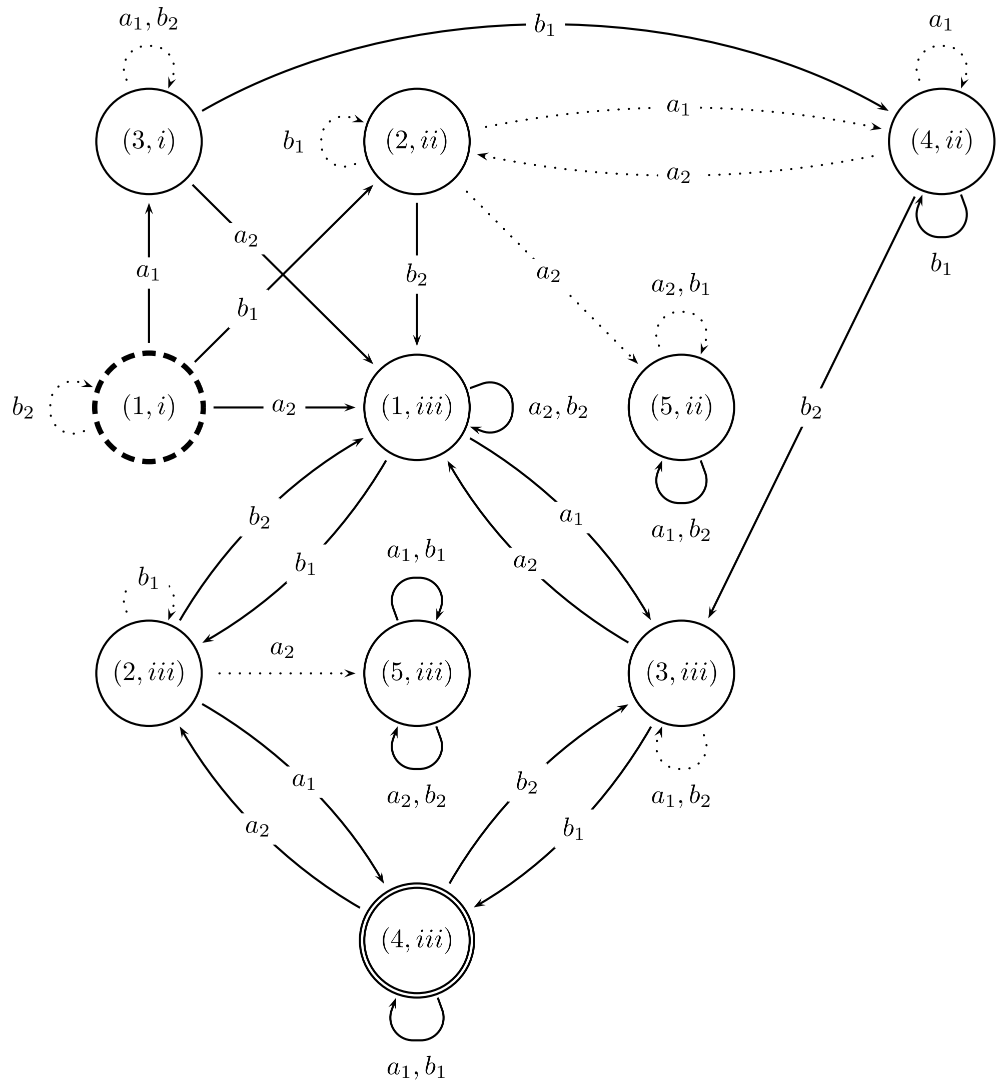


Figure 5.22: Trim vs. Untrim Example - Supervisor $\mathbf{R} = (\mathbf{W_1} \times \mathbf{W_2})$

$(R_1^* \wedge R_2^*)$ is not isomorphic to the implicitly-defined conjunction of $\mathbf{W_1}$ and $\mathbf{W_2}$ (represented by the states and enabled transitions in Figure 5.22). This is despite the fact that the implicitly-defined agents corresponding to $(\hat{\mathbf{W}}_\mathbf{1} \wedge \hat{\mathbf{W}}_\mathbf{2})$ and $(\mathbf{W_1} \wedge \mathbf{W_2})$ are isomorphic themselves. However, $R^* = (R_1^* \wedge R_2^*)$ is identical to $\hat{R}^* = (\hat{R}_1^* \wedge \hat{R}_2^*)$ with respect to supervision, differing structurally by four superfluous states: *Neuf*, *Dix*, *Onze* and *Douze*, which mirror *Cinq*, *Six*, *Sept* and *Huit*, respectively.

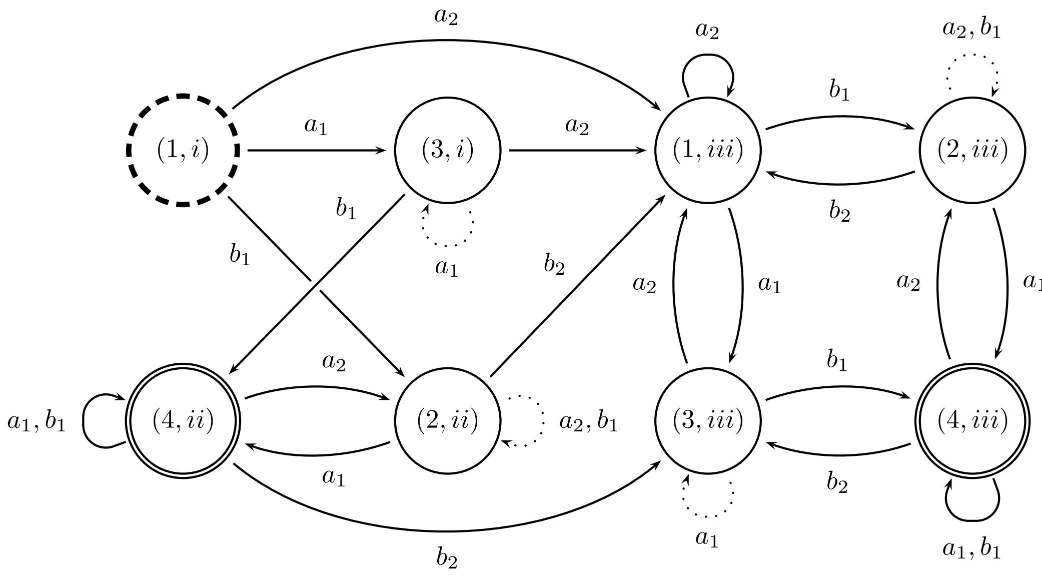


Figure 5.23: Trim vs. Untrim Example - $\hat{R}_1^*$ Resulting From $\hat{C}_{21}$ and $\hat{N}_{21}^*$

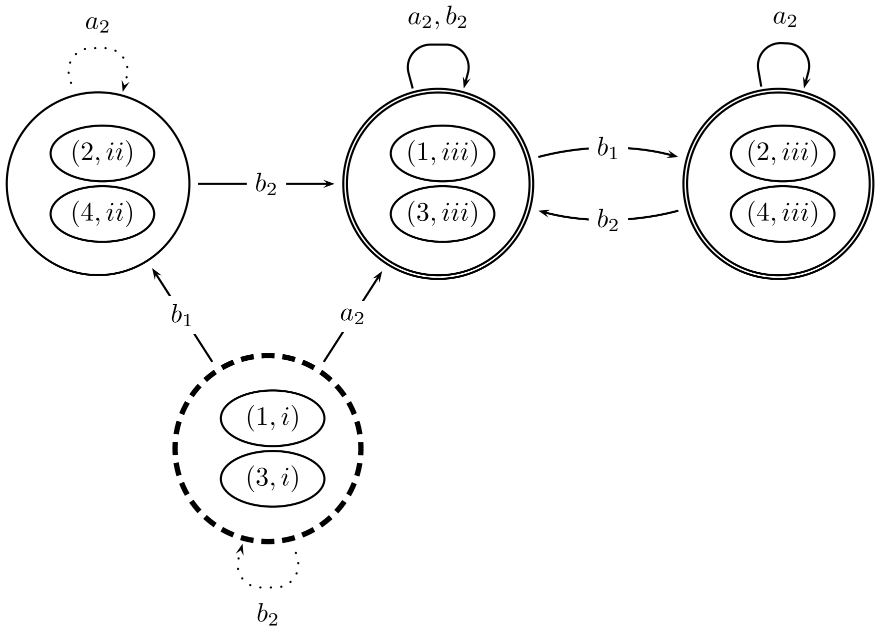


Figure 5.24: Trim vs. Untrim Example - $\hat{R}_2^*$ Resulting From $\hat{C}_{12}$ and $\hat{N}_{12}^*$

With the definitions of implicit agents $\hat{R}^*$ and R^* in place, it is possible to construct the automata that represent the result of $(\hat{R}_1^*, \hat{R}_2^*)$ and (R_1^*, R_2^*) acting on the plant $\mathbf{G}$, as shown in Figures 5.29

$\hat{N}^*_{12}$		
Transition	**Cause**	**State in $\hat{R}^*_i$**
$((1,iii),b_1,(2,iii))$	$((1,i),b_1,(2,ii)) \in \hat{C}_{12}$	$\{(1,i),(1,iii)\}$
$((3,iii),b_1,(4,iii))$	$((3,i),b_1,(4,ii)) \in \hat{C}_{12}$	$\{(3,i),(3,iii)\}$
$\hat{N}^*_{21}$		
$((1,i),a_2,(1,iii))$	$((3,i),a_2,(1,iii)) \in \hat{C}_{21}$	$\{(1,i),(3,i)\}$
$((2,ii),a_2,(2,ii))$	$((4,ii),a_2,(2,ii)) \in \hat{C}_{21}$	$\{(2,ii),(4,ii)\}$
$((1,iii),a_2,(1,iii))$	$((3,iii),a_2,(1,iii)) \in \hat{C}_{21}$	$\{(1,iii),(3,iii)\}$
$((2,iii),a_2,(2,iii))$	$((4,iii),a_2,(2,iii)) \in \hat{C}_{21}$	$\{(2,iii),(4,iii)\}$

Table 5.15: Trim vs. Untrim Example - Consistency Sets $\hat{N}^*_{12}$ and $\hat{N}^*_{21}$

$N^*_{12} = \emptyset$		
N^*_{21}		
Transition	**Cause**	**State in R^*_2**
$((5,ii),a_2,(5,ii))$	$((2,ii),a_2,(5,ii)) \in C_{21}$	$\{(2,ii),(4,ii),(5,ii)\}$
$((5,ii),b_2,(5,ii))$	$((2,ii),b_2,(1,iii)) \in C_{21}$	$\{(2,ii),(4,ii),(5,ii)\}$
$((1,iii),a_2,(1,iii))$	$((3,iii),a_2,(1,iii)) \in C_{21}$	$\{(1,iii),(3,iii)\}$
$((1,iii),b_2,(1,iii))$	$((5,iii),b_2,(5,iii)) \in N^*_{21}$	$\{(1,iii),(3,iii),(5,iii)\}$
$((3,iii),b_2,(3,iii))$	$((5,iii),b_2,(5,iii)) \in N^*_{21}$	$\{(1,iii),(3,iii),(5,iii)\}$
$((5,iii),a_2,(5,iii))$	$((2,iii),a_2,(5,iii)) \in C_{21}$	$\{(2,iii),(4,iii),(5,iii)\}$
$((5,iii),b_2,(5,iii))$	$((2,iii),b_2,(1,iii)) \in C_{21}$	$\{(2,iii),(4,iii),(5,iii)\}$

Table 5.16: Trim vs. Untrim Example - Consistency Sets N^*_{12} and N^*_{21}

$\tilde{\tilde{x}} \in \tilde{\hat{X}}$	**Short-Form**	$\tilde{\tilde{x}} \in \tilde{\hat{X}}$	**Short-Form**
Un	$((1,i),\{(1,i),(3,i)\})$	*Cinq*	$((1,iii),\{(1,iii),(3,iii)\})$
Deux	$((2,ii),\{(2,ii),(4,ii)\})$	*Six*	$((3,iii),\{(1,iii),(3,iii)\}$
Trois	$((3,i),\{(1,i),(3,i)\})$	*Sept*	$((2,iii),\{(2,iii),(4,iii)\}$
Quatre	$((4,ii),\{(2,ii),(4,ii)\})$	*Huit*	$((4,iii),\{(2,iii),(4,iii)\}$

Table 5.17: Trim vs. Untrim Example - Short-Form Names for the States of $\hat{R}^* = (\hat{R}^*_1 \wedge \hat{R}^*_2)$

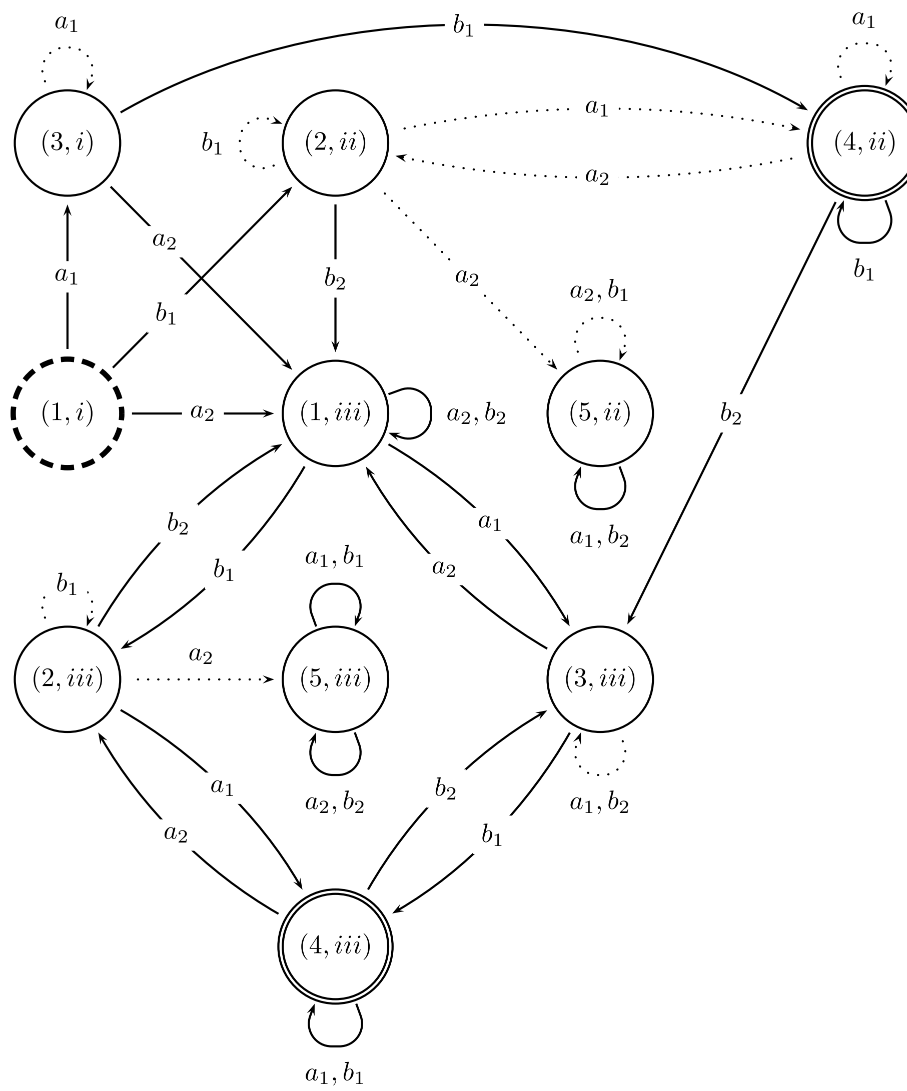


Figure 5.25: Trim vs. Untrim Example - R_1^* Resulting From C_{21} and N_{21}^*

and 5.30, respectively. Then, using the sets $\hat{C}_{12}$, $\hat{C}_{21}$, $\hat{N}_{12}^*$ and $\hat{N}_{21}^*$ (and similarly for C_{12}, C_{21}, N_{12}^* and N_{21}^*), the transmissions that will take place at each state of synthesis can be determined for both pairs of agents and are given in Tables 5.19 and 5.20.

There are several important points that are highlighted in this result. First, it was stated previously that states *Neuf*, *Dix*, *Onze* and *Douze* in R^* mirror *Cinq*, *Six*, *Sept* and *Huit*, respectively. Not surprisingly, the communications associated with each superfluous state mirror the transmissions for its essential counterpart.

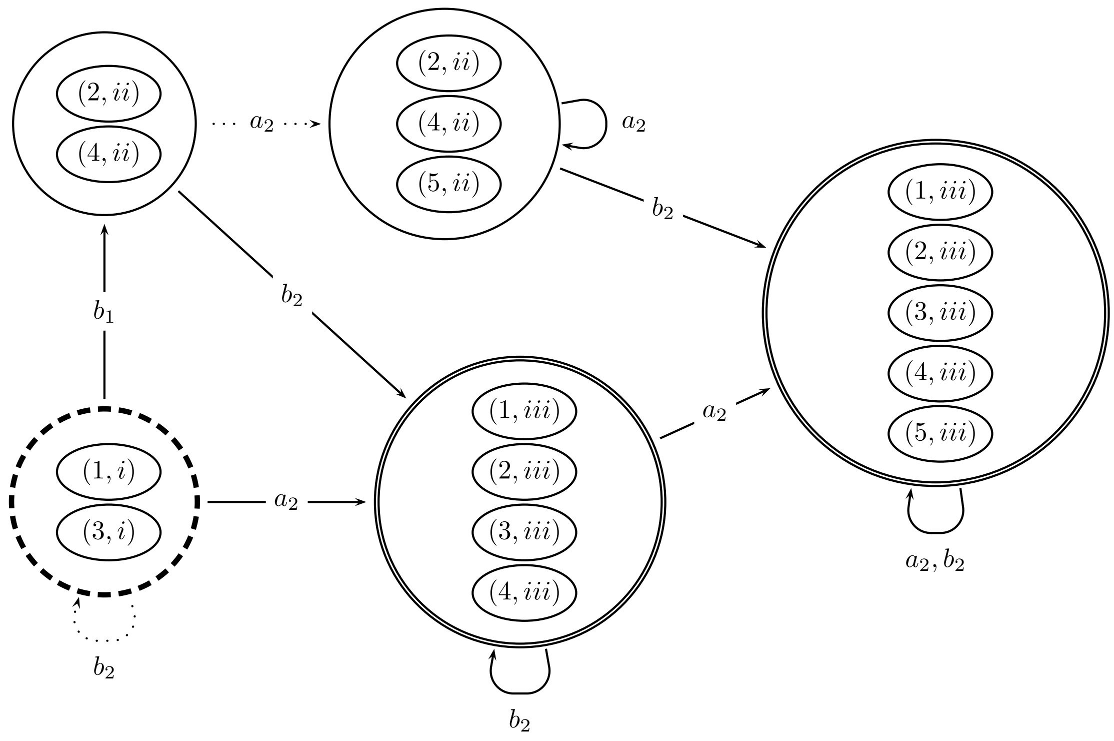


Figure 5.26: Trim vs. Untrim Example - R_2^* Resulting From C_{12} and N_{12}^*

$\tilde{x} \in \tilde{X}$	**Short-Form**	$\tilde{x} \in \tilde{X}$	**Short-Form**
Un	$((1,i),\{(1,i),(3,i)\})$	*Cinq*	$((1,iii),\{(1,iii),(2,iii),(3,iii),(4,iii)\})$
Deux	$((2,ii),\{(2,ii),(4,ii)\})$	*Six*	$((3,iii),\{(1,iii),(2,iii),(3,iii),(4,iii)\}$
Trois	$((3,i),\{(1,i),(3,i)\})$	*Sept*	$((2,iii),\{(1,iii),(2,iii),(3,iii),(4,iii)\}$
Quatre	$((4,ii),\{(2,ii),(4,ii)\})$	*Huit*	$((4,iii),\{(1,iii),(2,iii),(3,iii),(4,iii)\}$
		Neuf	$((1,iii),\{(1,iii),(2,iii),(3,iii),(4,iii),(5,iii)\})$
		Dix	$((3,iii),\{(1,iii),(2,iii),(3,iii),(4,iii),(5,iii)\}$
		Onze	$((2,iii),\{(1,iii),(2,iii),(3,iii),(4,iii),(5,iii)\}$
		Douze	$((4,iii),\{(1,iii),(2,iii),(3,iii),(4,iii),(5,iii)\}$

Table 5.18: Trim vs. Untrim Example - Short-Form Names for the States of $R^* = (R_1^* \wedge R_2^*)$

Second, the communication schemes for $\hat{R}^*$ and R^* do not satisfy the primary conjecture of this thesis, *i.e.*, $\hat{R}^*$ does not communicate a strict subset of the events transmitted by R^*. For example, as shown in Table 5.19 at state $(A,Cinq)$ in $\hat{R}^*$, a_2 or b_1 will be transmitted if either event occurs. In

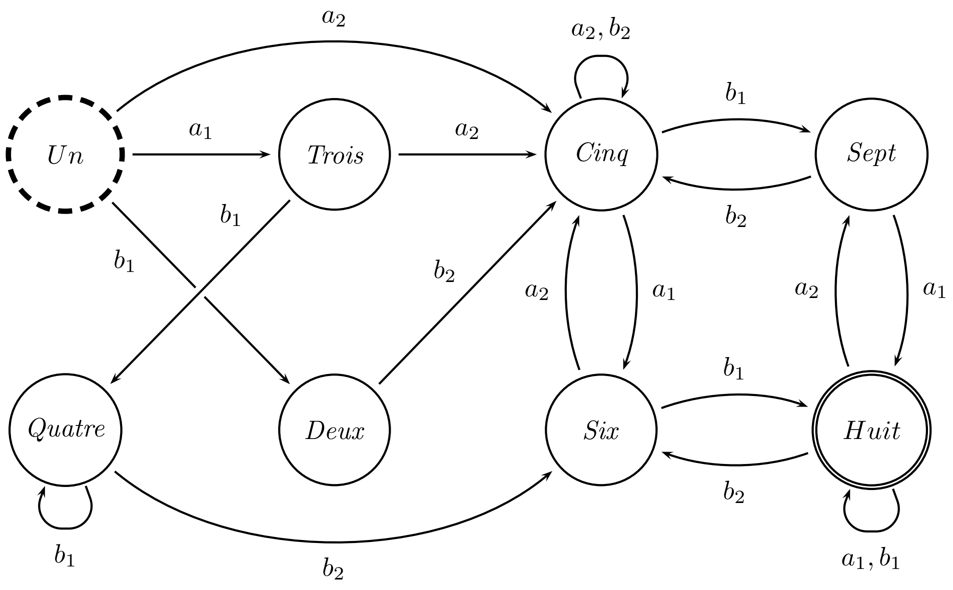


Figure 5.27: Trim vs. Untrim Example - Implicit Supervisor $\hat{R}^* = (\hat{R}_1^* \wedge \hat{R}_2^*)$

$\tilde{\hat{x}} \in \tilde{\hat{X}}$	**Communications**	$\tilde{\hat{x}} \in \tilde{\hat{X}}$	**Communications**
(A,Un)	$\{a_2, b_1\}$	(C,Six)	$\{a_2, b_1\}$
$(A,Cinq)$	$\{a_2, b_1\}$	$(C,Sept)$	$\{b_2\}$
$(B,Cinq)$	$\{a_2\}$	$(D,Quatre)$	$\{b_2\}$
$(C,Deux)$	$\{b_2\}$	$(D,Huit)$	$\{a_2, b_2\}$
$(C,Trois)$	$\{a_2, b_1\}$	$(E,Huit)$	$\emptyset$

Table 5.19: Trim vs. Untrim Example - Events Communicated When $\hat{R}_1^*$ and $\hat{R}_2^*$ Act on $\mathbf{G}$

contrast, as shown in Table 5.20 at state $(A,Cinq)$ in R^*, the potential communications are $\{a_2, b_2\}$.

Third, the communication scheme for R^* is equivalent to that for $\hat{R}^*$ with respect to the total *number* of potential event transmissions. In fact, the state-based communication mappings are identical with the exceptions (highlighted in bold font) given in Table 5.21.

Although $\hat{R}^*$ transmits an additional b_1 at (C,Six) that R^* does not, R^* communicates b_2 at $(B,Cinq)$ (and $(B,Neuf)$) where $\hat{R}^*$ does not. The only other difference is that $\hat{R}^*$ will communicate b_1 if it occurs at $(A,Cinq)$ while $\hat{R}^*$ will transmit b_2 instead at $(A,Cinq)$ and $(A,Neuf)$. Overall, the number of potential transmissions for both sets of agents is identical.

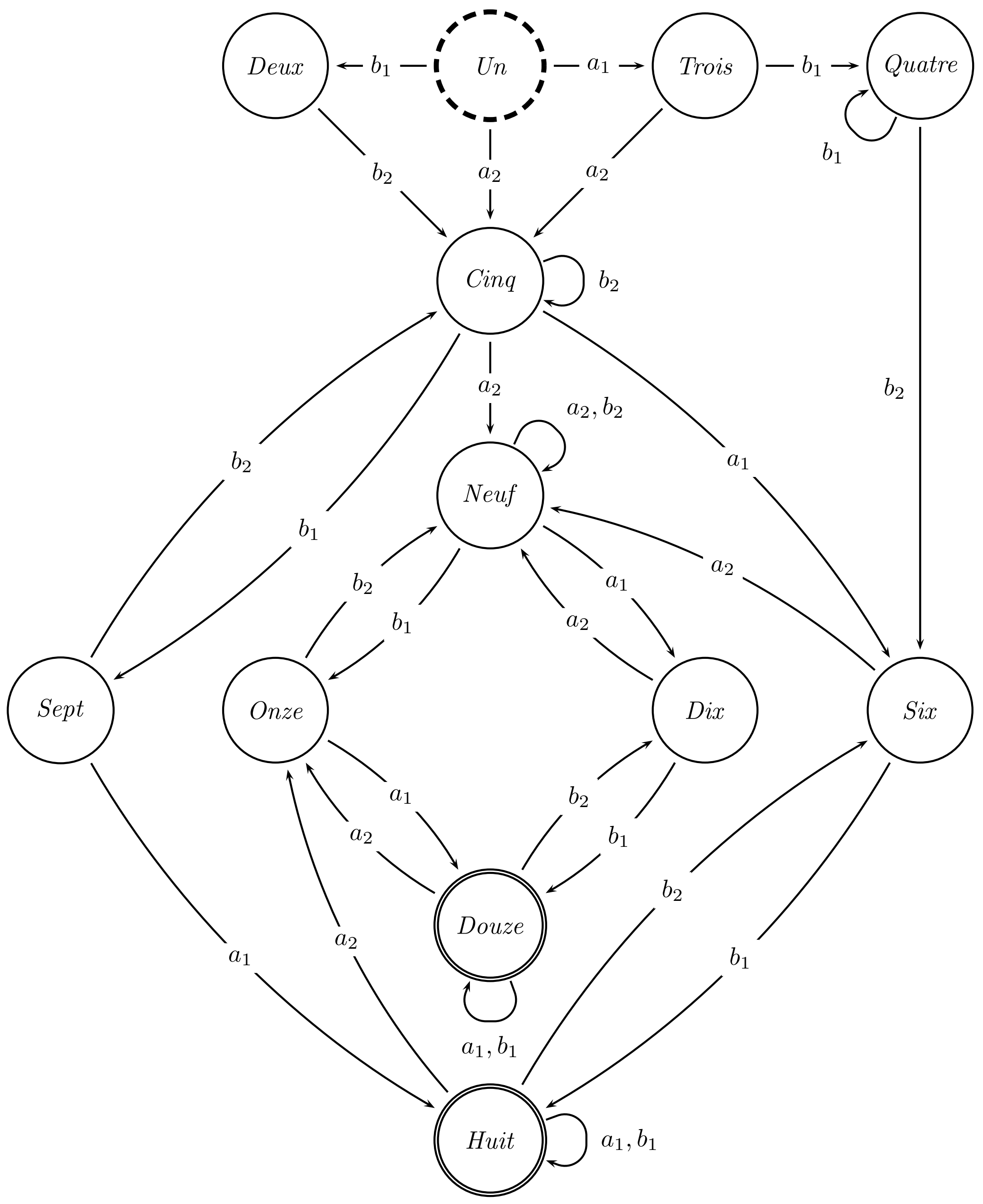


Figure 5.28: Trim vs. Untrim Example - Implicit Supervisor $R^* = (R_1^* \wedge R_2^*)$

Lastly, although the synthesis-inaccessible state in R^* resulted in communication differences with respect to $\hat{R}^*$, it did not adversely affect the number of transmissions required. Although it is certainly reasonable to expect that compensating for synthesis-inaccessible states would result

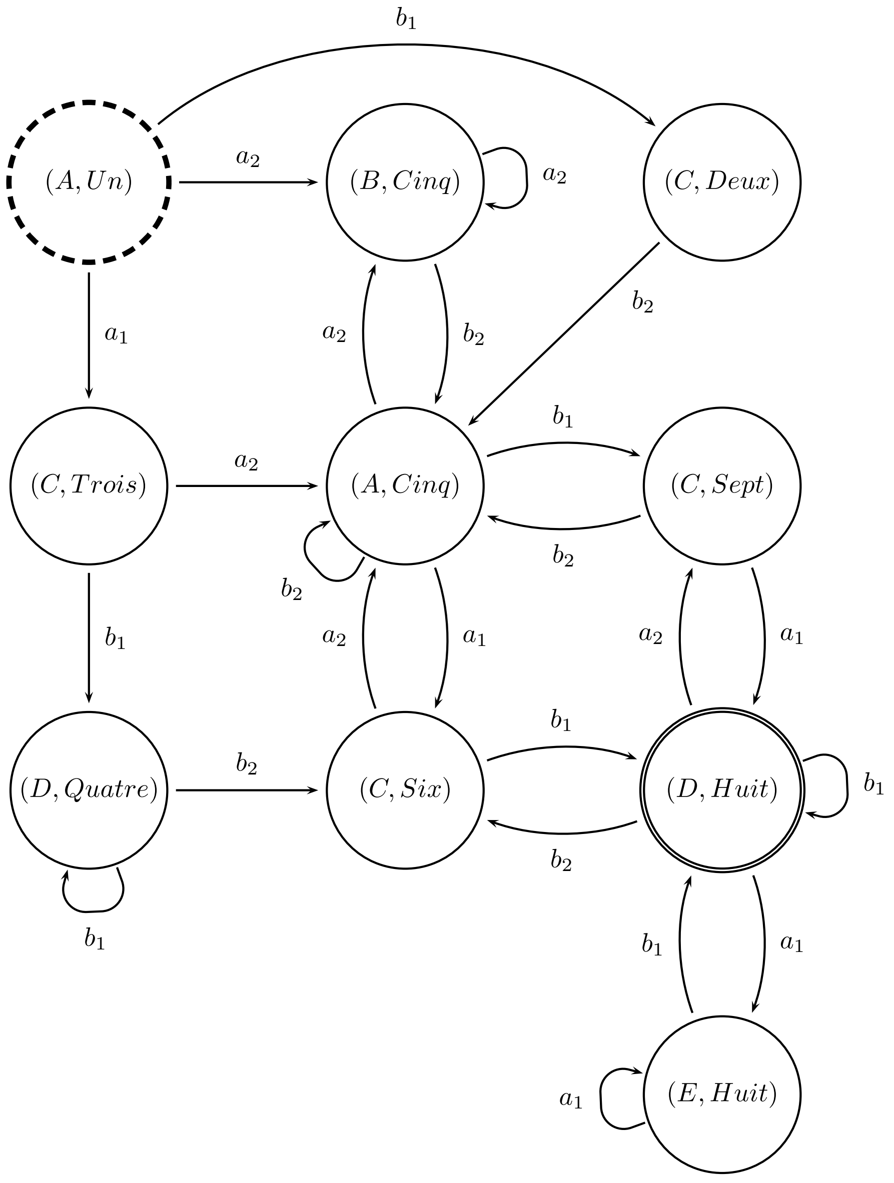


Figure 5.29: Trim vs. Untrim Example - Supervisors $\hat{R}_1^*$ and $\hat{R}_2^*$ Acting on Plant $\mathbf{G}$

in some unnecessary communication, this example demonstrates that that may not be the case. This implies that trimming a supervisor with respect to the plant is not as essential in terms of communication as one might think.

This example also serves as a demonstration of how results from the minimal communication

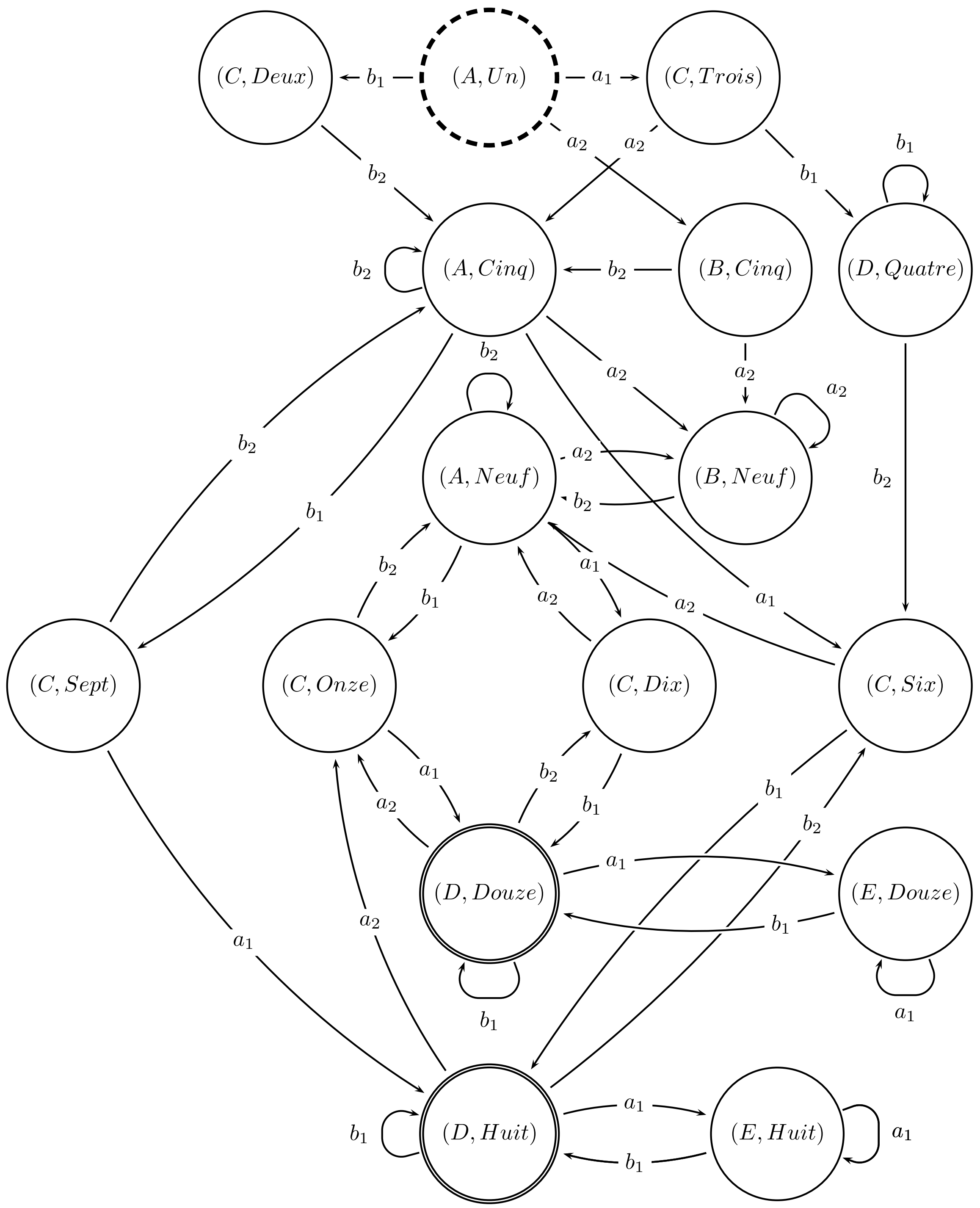


Figure 5.30: Trim vs. Untrim Example - Supervisors R_1^* and R_2^* Acting on Plant **G**

$\tilde{x} \in \tilde{X}$	**Communications**	$\tilde{x} \in \tilde{X}$	**Communications**
(A, Un)	$\{a_2, b_1\}$	$(C, Sept)$	$\{b_2\}$
$(A, Cinq)$	$\{a_2, b_2\}$	(C, Dix)	$\{a_2\}$
$(A, Neuf)$	$\{a_2, b_2\}$	$(C, Onze)$	$\{b_2\}$
$(B, Cinq)$	$\{a_2, b_2\}$	$(D, Quatre)$	$\{b_2\}$
$(B, Neuf)$	$\{a_2, b_2\}$	$(D, Huit)$	$\{a_2, b_2\}$
$(C, Deux)$	$\{b_2\}$	$(D, Douze)$	$\{a_2, b_2\}$
$(C, Trois)$	$\{a_2, b_1\}$	$(E, Huit)$	$\emptyset$
(C, Six)	$\{a_2\}$	$(E, Douze)$	$\emptyset$

Table 5.20: Trim vs. Untrim Example - Events Communicated When R_1^* and R_2^* Act on $\mathbf{G}$

$\tilde{\hat{x}} \in \tilde{\hat{X}}$	**Communications**	$\tilde{x} \in \tilde{X}$	**Communications**	$\tilde{x} \in \tilde{X}$	**Communications**
$(A, Cinq)$	$\{a_2, \mathbf{b_1}\}$	$(A, Cinq)$	$\{a_2, \mathbf{b_2}\}$	$(A, Neuf)$	$\{a_2, \mathbf{b_2}\}$
$(B, Cinq)$	$\{a_2\}$	$(B, Cinq)$	$\{a_2, \mathbf{b_2}\}$	$(B, Neuf)$	$\{a_2, \mathbf{b_2}\}$
(C, Six)	$\{a_2, \mathbf{b_1}\}$	(C, Six)	$\{a_2\}$	(C, Dix)	$\{a_2\}$

Table 5.21: Trim vs. Untrim Example - Communication Differences Between $\hat{R}^*$ and R^* During Synthesis

algorithm are applied to a pair of supervisors and utilized during synthesis. Although the process is certainly not trivial, the results speak for themselves: agents that enforce the required behaviour but communicate as little as possible.

Chapter 6

Conclusions

The research documented here describes an effort to combine known manipulations of distributed discrete-event supervisors (in this case, full definition and state-size reduction) in an effort to make the minimal communication algorithm as given in [10] more accessible and further minimize the results. Specifically, the primary conjecture stated that a pair of reduced agents will communicate a subset of the events transmitted by the original agents if the latter pair is comparable to the former. Care was taken to not make the definition of the explicit function so complicated or comparability so restrictive that it would be unrealistic for a system designer to use. However, enforcing fewer requirements on the structure of the agents resulted in cases where the supervisors are comparable, but their minimal communication schemes are not; that is, not in the strict manner described in the primary conjecture. The counterexample and proof-of-concept examples illustrated both possible outcomes.

Hindsight and discussion offer a few opinions that foresight seems to have missed. The first is that in amalgamating states in general, an agent may find its ability to make decisions diminished. For example, consider a reduction method (other than comparability) that combines two states that differ on control policy. This will result in the agent erring on the side of caution and disabling events where it would not have previously. If that disablement is unacceptable, the agent will need some other information upon which it will base its control decision. That information may have to be transmitted from another agent, resulting in communication where perhaps there was none before.[1] If the two states are totally redundant, which is possible, then these issues are not applicable. However, even the most inoccuous structural change may have unintended results.

The second thought is that the minimal communication algorithm only has knowledge about the structure of the two agents which it is given as input. The function does not take into account disabled events, inaccessible states or the system as whole. If these factors were considered in the calculations, the algorithm might be more efficient in terms of running time and might yield different results. However, this is not the case and as a result, the differences between two pairs of comparable agents that are dismissed during synthesis cannot be dismissed by the algorithm. It assumes that everything allowed by the structure of the supervisors is permitted for a reason, and thus must be considered when calculating the communication scheme. These discrepancies between the original and reduced agents may then result in discrepancies in their event transmissions. Perhaps if a desired control behaviour is the goal, an algorithm should be designed with that in mind.

Despite the considerations just given, the results of this research are hardly insignificant. Considering the difficulties that present themselves when determining communication, any option that will produce a reasonable result would be an asset. Moreover, although the counterexample did not strictly satisfy the conjecture, it demonstrates the case where the original supervisors communicate more than the reduced supervisors in terms of the *number* of events. The existence of a counterexample also does not imply the non-existance of satisfactory cases, *i.e.*, cases where the reduced agents communicate a subset of the events transmitted by the original supervisors.

Although the work documented here made at least some inroads into this topic, there is still far more that can be explored. Ideally, this would culminate in an algorithm that, given a system and a desired behaviour, produces the most efficient agents possible in terms of state size, supervisory action and communication. For the immediate future, it is more realistic to implement consideration of control policy and system operation when designing a transmission scheme. State amalgamation is also worth a second look, but based on different criteria such as user-specified options, *e.g.*, a requirement stating that x and y must be distinguishable until α occurs, after which they need not be.

It is apparent that when it comes to minimizing communication, there is one major pitfall: when

[1]This particular case was suggested by Dr. Kai Salomaa during personal discussions about this research.

attempting to reduce a given transmission scheme, any changes to one agent's policy means that the policies of every other agent must be checked, as their view of the system may be affected. The potential for infinite iteration makes this problem inherently complex to solve. Even the definition of "minimal" may be subject to the designer's preferences, *i.e.*, number of transmissions vs. message size. However, until delay-free and fault-free networks become available, it is a topic whose complexity should not detract from the need for further research and eventual resolution.

Bibliography

[1] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, MA, 1999.

[2] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260, March 1988.

[3] W. M. Wonham et al. CTCT, last updated in 2005. Discrete-event systems software which implements the background theory included in this dissertation; can be downloaded at http://www.control.toronto.edu/people/profs/wonham/wonham.html.

[4] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[5] Ying Huang. Minimal communication software, 2002-2003. A Java implementation of the algorithm given in [10] employed to determine/confirm the results for each of the examples.

[6] P. Kozák and W. M. Wonham. Fully decentralized solutions of supervisory control problems. *IEEE Transactions on Automatic Control*, 40(12):2094–2097, Dec. 1995.

[7] F. Lin and W. M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions on Automatic Control*, 35(12):1330–1337, Dec. 1990.

[8] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, Jan. 1987.

[9] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1), Jan. 1989.

[10] K. Rudie, S. Lafortune, and F. Lin. Minimal communication in a distributed discrete-event system. *IEEE Transactions on Automatic Control*, 48(6):957–975, June 2003. Also appears as Control Group Report No. CGR-00-06, College of Engineering, University of Michigan, 2000.

[11] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.

[12] Karen Rudie. The current state of decentralized discrete-event control systems. In *Proceedings of the 10th Mediterranean Conference on Control and Automation*, July 2002.

[13] R. Su and W. M. Wonham. Supervisor reduction for discrete-event systems. *Discrete-Event Dynamic Systems: Theory and Applications*, 14:31–53, 2004.

[14] Shigemasa Takai. Minimizing the set of local supervisors in fully decentralized supervision. *IEEE Transactions on Automatic Control*, 44(7):1441–1444, July 1996.

[15] J. G. Thistle. Supervisory control of discrete event systems. *Mathematical Computing and Modelling*, 23(11/12):25–53, 1996.

[16] A. F. Vaz and W. M. Wonham. On supervisor reduction in discrete-event systems. *International Journal of Control*, 44(2):475–491, Dec. 1986.

[17] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems*, 1:13–30, 1988.

Appendix A

Supervisor Reduction: A Quick Reference Guide

Quotients [8]

Definition: Generator/Plant [8, p. 207].

We define a *generator* to be a 5-tuple

$$\mathcal{G} = (Q, \Sigma, \delta, q_0, Q_m)$$

Footnote 1: The terms *generator* and *marker state* are nonstandard, but better suited to our interpretation than, for example "automaton" and "final state" . . . it will play the role of "plant" in the sense of control theory.

The automaton $\mathcal{G}$ is initially defined in the paper as a generator, with a footnote to clarify the authors' position. It is only later when controllable and uncontrollable events are introduced that $\mathcal{G}$ is referred to as a plant.

Definition: Projection [8, p. 219].

Let $\mathcal{S} = (S, \phi)$ and $\hat{\mathcal{S}} = (\hat{S}, \hat{\phi})$ each be supervisors for $\mathcal{G}$, where as usual

$$\begin{array}{ll} S = (X, \Sigma, \xi, x_0, X_m) & \phi : X \to \{0,1\}^{\Sigma} \\ \hat{S} = (\hat{X}, \Sigma, \hat{\xi}, \hat{x}_0, \hat{X}_m) & \hat{\phi} : \hat{X} \to \{0,1\}^{\Sigma} \end{array}$$

We shall say that a (total) function $\pi : X \to \hat{X}$ is a *projection* from $\mathcal{S}$ to $\hat{\mathcal{S}}$ and write $\pi : \mathcal{S} \to \hat{\mathcal{S}}$, provided

(i) $\pi : X \to \hat{X}$ is surjective,

(ii) $\pi(x_0) = \hat{x}_0$ and $X_m = \pi^{-1}(\hat{X}_m)$,

(iii) $\hat{\xi} \circ (\mathrm{id}_\Sigma \times \pi)(\sigma, x) = \pi \circ \xi(\sigma, x)$ for all (σ, x) where $\xi(\sigma, x)$ is defined,

(iv) $\hat{\phi} \circ \pi = \phi$.

Under these conditions we shall refer to $\hat{\mathcal{S}}$ as the *quotient* of $\mathcal{S}$ under π.

Loosely put, a projection is a mapping of states that preserves the transition function, the feedback map, and the initial and final states. Of vital note is the condition in (iii) that only requires $\hat{\xi}$ to be consistent with ξ where $\xi(\sigma, x)$ is defined. This means that states x and x' may be amalgamated even if there exists σ such that $\xi(\sigma, x)$ is defined and $\xi(\sigma, x')$ is not.
A footnote defined on (iii) clarifies that $\mathrm{id}_\Sigma \times \pi : \Sigma \times X \to \Sigma \times \hat{X} : (\sigma, x) \mapsto (\sigma, \pi(x))$; in simpler terms, (iii) translates to $\pi \circ \xi(\sigma, x) = \hat{\xi}(\sigma, \pi(x))$.

Proposition 8.1 [8, p. 219].

Let $\mathcal{S}$ be complete with respect to $\mathcal{G}$, and let $\pi : \mathcal{S} \to \hat{\mathcal{S}}$ be a projection. Then,

(i) π is unique,

(ii) $(L_m, L_c, L)(\mathcal{S}/\mathcal{G}) = (L_m, L_c, L)(\hat{\mathcal{S}}/\mathcal{G})$,

(iii) $\hat{\mathcal{S}}$ is complete with respect to $\mathcal{G}$,

(iv) $\mathcal{S}$ is nonblocking (respectively, nonrejecting, nonproper) iff $\hat{\mathcal{S}}$ is nonblocking (respectively, nonrejecting, nonproper)

This is an extremely important result in that any quotient supervisor is guaranteed to have the same desirable properties of the original with respect to completeness and nonblocking. It provides further incentive to attempt supervisor reduction.

Definition: K-equivalence [8, p. 222].

Let $\equiv (\mathrm{mod} K), K \subset \Sigma^*$ denote K-equivalence on $\Sigma^* : s \equiv s'(\mathrm{mod} K)$ if for all $t \in \Sigma^*$, $st \in K$ iff $s't \in K$.

It is also noted in the paper that $\equiv (\text{mod} K)$ is a right-congruence.

Definition: K-reduced and K-trim [8, p. 222, Paraphrased]. Let $\mathbf{S} = (S, \phi)$ be an automaton such that $L(\mathbf{S})$ includes the language K. Thus, $\mathbf{S}$ is *K-reduced* if for $s, s' \in K$ and $s \equiv s'(\text{mod} K)$ then

$$\xi(s, x_0) = \xi(s', x_0)$$

$\mathbf{S}$ is also *K-trim* if every state of $\mathbf{S}$ is visited by a word in K; that is, for every $x \in X$ there is $s \in K$ such that $\xi(s, x_0) = x$.

These two properties are used to define the *efficient* supervisor $\mathcal{S}$ in Theorem 10.1 where any other supervisor that enforces identical control on $\mathcal{G}$ must have a state size as large or larger than that of $\mathcal{S}$.

Theorem 10.1 [8, pp. 222-223].

> **Quotient structure theorem.** Let $\mathcal{S} = (S, \phi)$ be a complete supervisor for $\mathcal{G}$. Write $K_1 = L_m(\mathcal{S}/\mathcal{G})$, $K_3 = L(\mathcal{S}/\mathcal{G})$ and assume that S is K_3-reduced and K_3-trim. . . . Finally let
>
> $$\hat{S}^0 = (X^0, \Sigma, \xi^0, x_0^0, X^0)$$
>
> be a trim recognizer for K_3. Subject to the forgoing hypotheses, there exists a subset $X_m^0 \subset X^0$ and a state feedback map $\phi^0 : X^0 \to \{0,1\}^\Sigma$ with the following properties:
>
> (i) The supervisor
>
> $$\mathcal{S}^0 = (S^0, \phi^0) \qquad S^0 = (X^0, \Sigma, \xi^0, x_0^0, X_m^0)$$
>
> is a complete supervisor for $\mathcal{G}$ with
>
> $$L_m(\mathcal{S}^0/\mathcal{G}) = K_1, \qquad L(\mathcal{S}^0/\mathcal{G}) = K_3$$
>
> (ii) There is a projection $\pi : \mathcal{S}^0 \to \mathcal{S}$.
>
> (iii) If $\mathcal{S}$ is proper then so is $\mathcal{S}^0$.

This theorem essentially proves that "any efficiently constructed supervisor is a quotient (high-level, or lumped, model) of the desired closed-loop behaviour". In other words, a supervisor that recognizes and marks the same language as the efficient supervisor $\mathcal{S}$ must have a unique projection onto $\mathcal{S}$.

Covers [16]

Definition: $\equiv$ for strings [16, pp. 476-477].

> If $K \subseteq \Sigma^*$ is a language and $s, t \in \Sigma^*$ are arbitrary strings, then $s \equiv t \bmod K$ means that s and t belong to the same Nerode equivalence class of K, namely for all $w \in \Sigma^*$, $sw \in K$ iff $tw \in K$.

This definition is identical to that for K-equivalence in [8]. The term "Nerode class" is also used numerous times in the paper.

Definition: Standard supervisor [16, p. 477].

> Write $K := L(\mathbf{S}/\mathbf{G})$, $L := L(\mathbf{G})$. Following Ramadge and Wonham (1983), we say that a supervisor $\mathbf{S} = (S, \psi)$ for $\mathbf{G}$ is *complete* if, for all $s \in K$, the two conditions $s\sigma \in L$ and $\psi(\sigma, x) = 1$ with $x = \xi(s, x_0)$ together imply $\xi(\sigma, x)!$, namely $s\sigma \in K$. Also, a supervisor $\mathbf{S} = (S, \psi)$ will be called *normal* if the control law ψ is defined with as much flexibility as possible, namely enablement (1) or disablement (0) is assigned to a pair $(\sigma, x), \sigma \in \Sigma_c$, only when necessary. ...
> Thirdly, S will be called *strongly K-accessible* if for every $x \in X$ there exists $s \in K$ such that $\xi(s, x_0) = x$ (i.e. S is K-accessible) and, for all $\sigma \in \Sigma$ and $x \in X$, $\xi(\sigma, x)!$ only if there is $s \in K$ such that $\xi(s, x_0) = x$) and $s\sigma \in K$. Finally, we shall say that a supervisor S is *standard* if it is complete, normal and strongly K-accessible. It is straightforward to assume that any complete supervisor may be replaced by a standard version without changing the control action with respect to the behaviour of $\mathbf{G}$.

The term "standard" is defined here to encapsulate several requirements on $\mathbf{S}$, all of which are needed to prove the theorems to come in the paper. None of the above conditions are considered unusual or overly stringent; in fact, they are all viewed as good practice in control theory. Loosely put, they enforce the following with respect to all $s \in K$

1. (Complete) $\mathbf{S}$ accepts any σ that follows s if $s\sigma$ is accepted by $\mathbf{G}$ and σ is not explicitly disabled by ψ
2. (Normal) $\psi(s, \sigma)$ has a value of 0 if $s\sigma$ is accepted by $\mathbf{G}$ but not $\mathbf{S}$; 1 if it is accepted by both; and dc if it is not accepted by $\mathbf{G}$
3. (Strongly K-Accessible) $\mathbf{S}$ only contains states that can be reached by some string *and* only contains transitions that can follow some string in K

Definition: Cover [16, p. 477].

To develop the main results we define a *cover* of $\mathbf{S}$ to be a family $\mathbf{C} = \{X_i \mid i \in I\}$ of subsets of X with the following properties (cf. Zeiger 1968):

$$\begin{aligned}
\forall\, i, &\quad X_i \neq \emptyset; \\
\text{For a subset } I_m \subset I, &\quad X_m = \cup\{X_i \mid i \in I_m\}, \\
&\quad X - X_m = \cup\{X_i \mid i \in I - I_m\}; \\
(\forall\, i, \sigma) : (\exists\, y \in X_i),\ \xi(\sigma, y)! &\quad \Rightarrow (\exists\, j)(\forall\, x \in X_i) \cdot \xi(\sigma, x)! \Rightarrow \xi(\sigma, x) \in X_j \\
&\quad \text{(for brevity, we write this property as } (\forall\, i, \sigma)(\exists\, j) \\
&\quad \xi(\sigma, X_i) \subset X_j) \\
(\forall\, i, \sigma)(\forall\, x, y \in X_i), &\quad \psi(\sigma, x) \neq dc \neq \psi(\sigma, y) \\
&\quad \Rightarrow \psi(\sigma, x) = \psi(\sigma, y)
\end{aligned}$$

Thus a cover of $\mathbf{S} = (S, \psi)$ is simply a covering of the state set X of S by non-empty subsets, such that the marked states (X_m) are covered separately from the unmarked states $(X - X_m)$, the cover elements behave consistently under the action of the transition function ξ, and a cover element exhibits uniform control action at those states where control matters.

Fix $i \in I, \sigma \in \Sigma$. If $\xi(\sigma, x)!$ for some $x \in X_i$, select $j \in I$ such that $\xi(\sigma, x) \in X_j$ for all such x. A triple (i, σ, j) as just described will be called a *cover triple*.

The definition of a cover has several interesting properties beyond those described in the preceeding papragraph.

1. Each subset $X_i \in I_m$ must be composed entirely of marked states.

2. As with projections, the transition function for the cover need only be consistent where transitions are defined in $\mathbf{S}$; $\forall\, i \in I, x \in X_i, \sigma \in \Sigma,\ \xi(\sigma, x)! \Rightarrow (\exists\, j \in I)\, \xi(X_i, \sigma) \subset X_j \wedge \xi(\sigma, x) \in X_j$ but it is possible that $\exists\, x \in X_i, \neg\xi(\sigma, x)!$.

3. No two states in a subset $X_i \in I$ have conflicting control actions for any event; the "don't care" action dc does not conflict with 0 or 1 since it is overridden by either of these.

Definition: $\overline{\mathbf{S}}$ based on $\mathbf{C}$ [16, p. 478].

Given a supervisor $\mathbf{S} = (S, \psi)$ and corresponding cover $\mathbf{C} = \{X_i \mid i \in I\}$ where $\mid \mathbf{C} \mid<\mid X \mid$, we define a reduced supervisor $\overline{\mathbf{S}} = (\overline{S}, \overline{\psi})$ as follows:

```
$\overline{S} = (\Sigma, I, \overline{\xi}, i_0, I_m)$
    Select $i_0 \in I$ such that $x_0 \in X_{i_0}$;
    Define $\overline{\xi} : \Sigma \times I \to I$ (pfn) as follows:
      For $\sigma \in \Sigma, i \in I$ select $j \in I$ such that $(i, \sigma, j)$
      is a cover triple and let $\overline{\xi}(\sigma, i) = j$;
    Define $\overline{\psi} : (\Sigma \times I) \to \{0, 1, dc\}$ as follows:
      For $\sigma \in \Sigma, i \in I$ if there exists $x \in X_i$ such that
      $\psi(\sigma, x) \neq dc$ then let $\overline{\psi}_i(\sigma) = \psi(\sigma, x)$;
      otherwise let $\overline{\psi}_i(\sigma) = dc$.
```

We shall say that $\overline{S}$ is *based on* $\mathbf{C}$. In general the reduced supervisor is determined by selections that in part are arbitrary, but this fact is of no account in what follows.

What follows is a lemma, corollary and theorem, all of which ultimately prove that $\overline{\mathbf{S}}$ is complete and induces the same behaviour on $\mathbf{G}$ as $\mathbf{S}$.

Definition: Weakly $L(\mathbf{S}/\mathbf{G})$-reduced [16, p. 480].

$\mathbf{S}$ will be called *weakly* $L(\mathbf{S}/\mathbf{G})$*-reduced* if, whenever $s, s' \in L(\mathbf{S}/\mathbf{G})$ are equivalent mod $L(\mathbf{S}/\mathbf{G})$, and for some $\sigma \in \Sigma$, $s\sigma \in L(\mathbf{G}) - L(\mathbf{S}/\mathbf{G})$, then $\xi(s, x_0) = \xi(s', x_0)$. In other words, the weakened condition is satisfied if the condition of $L(\mathbf{S}/\mathbf{G})$-reduction is restricted to those Nerode classes of $L(\mathbf{S}/\mathbf{G})$ with the property that some string s' of the class has an extension $s'\sigma$ that belongs to $L(\mathbf{G})$, although not to $L(\mathbf{S}/\mathbf{G})$. Strings that belong to a Nerode class without this property are not required to lead to some state of S. The states of S that appear in the weakened condition are precisely those where some $\sigma \in \Sigma_c$ must be disabled.

Theorem 2 [16, p. 481].

First generalized quotient structure theorem. Assume that $\mathbf{S}$ is a supervisor for $\mathbf{G}$ and that (for simplicity) $X_m = X$. Let $\hat{S}$ be a recognizer for $L(\mathbf{S}/\mathbf{G})$. Assume finally that $\mathbf{S}$ is standard and weakly $L(\mathbf{S}/\mathbf{G})$-reduced. ... Then there exists a control law

$$\hat{\psi} : (\Sigma \times \hat{X}) \to \{0, 1, dc\}$$

such that $\hat{\mathbf{S}} = (\hat{S}, \hat{\psi})$ is a supervisor for $\mathbf{G} = (\Sigma, \Sigma_u, \Sigma_c, Q, \delta, q_0, Q_m)$ with the property

$$L(\hat{\mathbf{S}}/\mathbf{G}) = L(\mathbf{S}/\mathbf{G}) \qquad L_m(\hat{\mathbf{S}}/\mathbf{G}) = L_m(\mathbf{S}/\mathbf{G})$$

> Furthermore, there is a cover $\hat{\mathbf{C}}$ of $\hat{\mathbf{S}}$ such that the corresponding reduced supervisior (based on $\hat{\mathbf{C}}$) is isomorphic to $\mathbf{S}$.

This theorem is essentially an alternate version of the *quotient structure theorem* in [8] from a cover perspective. The vital difference here is that instead of a unique projection between $\mathbf{S}$ and $\hat{\mathbf{S}}$ there is a cover $\hat{\mathbf{C}}$ that produces a supervisor isomorphic to $\mathbf{S}$. It is also less restrictive in that $\mathbf{S}$ is only weakly $L(\mathbf{S}/\mathbf{G})$-reduced, versus fully $L(\mathbf{S}/\mathbf{G})$-reduced.

Theorem 2.5 [16, p. 486].

> **Second generalized quotient structure theorem**. For our second generalization of the quotient structure theorem of Ramadge and Wonham (1983), we shall drop the condition on the given supervisor $\mathbf{S}$ that it be weakly $L(\mathbf{S}/\mathbf{G})$-reduced, at the expense of increasing the complexity of the 'canonical' supervisor $\hat{\mathbf{S}}$ from which $\mathbf{S}$ is to be obtained by an appropriate cover.

This section of the paper contains no formal statement of the theorem; rather, it contains a somewhat informal proof. In this instance, the recognizer $\hat{S}$ is defined for $L(\mathbf{S}/\mathbf{G})$ using a complex construction based a recognizer for $(L(\mathbf{G}) - L(\mathbf{S}/\mathbf{G})) \cup (L(\mathbf{S}/\mathbf{G}) - \overline{(L(\mathbf{G}) - L(\mathbf{S}/\mathbf{G}))})$. The end result is that $\mathbf{S}$ does not have to be reduced in any of the manners outlined previously for a cover to exist between it and $\hat{\mathbf{S}}$.

Control Covers [13]

Definition: G, SPEC and SIMSUP[13, p. 33].

> In supervisory control theory, a discrete-event system to be controlled (or plant) is modelled as a discrete transition structure
>
> $$\mathbf{G} = (Y, \Sigma, \eta, y_0, Y_m)$$
>
> ... A nonempty language $SPEC \subseteq \Sigma^*$ is represented by a DES **SPEC** if **SPEC** is reachable and
>
> $$L(\mathbf{SPEC}) = \overline{SPEC} \qquad L_m(\mathbf{SPEC}) = SPEC$$
>
> where $\overline{SPEC}$ is the prefix-closure of $SPEC$. Let $K \subseteq L_m(\mathbf{G}) \cap SPEC$ be the supremal

controllable sublanguage of $L_m(\mathbf{G})$ with respect to $SPEC$ (see, Wonham, 2002)[1]:

$$K = sup\,\mathcal{C}\,(L_m(\mathbf{G}) \cap SPEC)$$

In case $K \neq \emptyset$, a supremal supervisor for $\mathbf{G}$ (with respect to $SPEC$) is a DES **SUPER** which represents K. ... Write $\mid \mathbf{G} \mid$ for the state size of $\mathbf{G}$, etc. Then **SUPER** can be readily computed (e.g. using the operator **supcon** in Wonham, 2002) in time a polynomial function of $\mid \mathbf{G} \mid$, $\mid \mathbf{SPEC} \mid$

$$\mid \mathbf{SUPER} \mid \leq \mid \mathbf{G} \mid\mid \mathbf{SPEC} \mid$$

Here the conservative bound is typically representative of the actual size of **SUPER**. In applications, engineering intuition may (or may not) suggest that there might exist a DES, say **SIMSUP**, such that

$$\mid \mathbf{SIMSUP} \mid \ll \mid \mathbf{SUPER} \mid$$

and the following properties hold,

$$L(\mathbf{G}) \cap L(\mathbf{SIMSUP}) = L(\mathbf{SUPER}) \tag{1a}$$
$$L_m(\mathbf{G}) \cap L_m(\mathbf{SIMSUP}) = L_m(\mathbf{SUPER}) \tag{1b}$$

Definition: Control Equivalent [13, p. 33].

Any DES **SIMSUP** that satisfies (1a,b) is control equivalent to **SUPER** with respect to **G**.

Definition: $\mathcal{R}$ [13, p. 35].

Suppose **SUPER** $= (X, \Sigma, \xi, x_0, X_m)$. Let $E : X \to 2^{\Sigma}$ with

$$x \mapsto E(x) := \{\sigma \in \Sigma \mid \xi(x,\sigma)!\}$$

be the **SUPER**-enabled event set at state $x \in X$. Let $D : X \to 2^{\Sigma}$ with

$$x \mapsto D(x) := \{\sigma \in \Sigma \mid \neg\xi(x,\sigma)! \wedge (\exists\, s \in \Sigma^*)[\xi(x_0, s) = x \wedge \eta(y_0, s\sigma)!]\}$$

[1]The bibliographic entry given in [13] for this reference appears as follows:
Wonham, W. M. 2002. *Note on Control of Discrete-Event Systems: ECE 1636F/1637S 2002-2003.* Systems Control Group, Dept. of ECE, University of Toronto. URL: www.control.utoronto.ca/people/profs/wonham/wonham.html

be the **SUPER**-disabled event set at state $x \in X$. Let $M : X \to \{true, false\}$ with

$$x \mapsto M(x) := true \text{ if } x \in X_m$$

Finally, let $T : X \to \{true, false\}$ with

$$x \mapsto T(x) := true \text{ if } (\exists\, s \in \Sigma^*)\xi(x_0, s) = x \wedge \eta(y_0, s) \in Y_m$$

Let $\mathcal{R} \subseteq X \times X$ be the binary relation such that for a pair $x, x' \in X$, $(x, x') \in \mathcal{R}$ iff

1. $E(x) \cap D(x') = E(x') \cap D(x) = \emptyset$
2. $T(x) = T(x') \Rightarrow M(x) = M(x')$

Condition 1 says that for a pair of states (x, x') in $\mathcal{R}$, the associated enable/disable control actions should be consistent, namely no event is enabled at x but disabled at x'. Condition 2 requires that states x, x' in $\mathcal{R}$ be consistently marked either $true$ or $false$ in **SUPER** if they are reachable by some strings s, s' in $L_m(\mathbf{G})$ (in case $T(x) = T(x') = true$), or else if neither is reachable by strings in $L_m(\mathbf{G})$ (i.e., $T(x) = T(x') = false$). In the latter case (1b) shows that in fact $M(x) = M(x') = false$. Clearly $\mathcal{R}$ is reflexive and symmetric, but it need not be transitive, and so it is generally not an equivalence relation.

Of note here is that unlike the supervisor definition in [16], this supervisor is implicitly defined; there is no explicit feedback map. Hence, E and D are both determined by whether or not transitions are defined, versus map values of 0, 1 or dc.

Definition 2.1 [13, p. 35].

Recall that a cover of a set X is a family of subsets of X whose union is X. Let I be an index set. A *cover* $\mathbf{C} = \{X_i \mid i \in I\}$ of X is a control cover on **SUPER** if

1. $(\forall\, i \in I)\, X_i \neq \emptyset \wedge (\forall\, x, x' \in X_i)\, (x, x') \in \mathcal{R}$
2. $(\forall\, i \in I)(\forall\, \sigma \in \Sigma)(\exists\, j \in I)[(\forall\, x \in X_i)\, \xi(x, \sigma)! \Rightarrow \xi(x, \sigma) \in X_j]$

The subsets X_i are the cells of $\mathbf{C}$. A control cover $\mathbf{C}$ is a *control congruence* if $\mathbf{C}$ is a partition on X, namely all X_i are pairwise disjoint.

Condition 1 requires that each cell of $\mathbf{C}$ be nonempty and each pair of states in the same cell should belong to $\mathcal{R}$, namely their associated control action and marked status should be consistent. Condition 2 states that for each $X_i \in \mathbf{C}$ and each event $\sigma \in \Sigma$, the set of states that can be reached from any states in X_i by a one-step transition σ is covered by some $X_j \in \mathbf{C}$. Note that the cells of a control cover may overlap: $x \in X$ may belong to more than one cell X_i.

This definition is nearly identical to that of [16] with two significant exceptions:

1. The explicit requirement for marked states in [16] is handled here by $\mathcal{R}$, which is actually more rigourous: not only must all states $x \in X_i$ where $i \in I_m$ be marked in the supervisor, but there must also be a string that leads to x *and* a marked state in $\mathbf{G}$.
2. Since **SUPER** is implicitly defined, the enablement/disablement policy outlined in feedback map for [16] is again handled here by $\mathcal{R}$.

Definition: Induced supervisor [13, p. 36].

Given a control cover $\mathbf{C} = \{X_i \mid i \in I\}$ on **SUPER**, we construct an induced supervisor

$$\mathbf{J} = (I, \Sigma, \kappa, i_0, I_m)$$

as follows. Let

$$i_0 = \text{ some } i \in I \text{ with } x_0 \in X_i$$

$$I_m = \{i \in I \mid X_i \cap X_m \neq \emptyset\}$$

$$\kappa : I \times \Sigma \to I \text{ (pfn)}$$

with $\kappa(i, \sigma) = j$ provided, for some choice of $j \in I$,

$$(\exists\, x \in X_i)\, \xi(x, \sigma) \in X_j \wedge (\forall\, x' \in X)\, [\xi(x', \sigma)! \Rightarrow \xi(x', \sigma) \in X_j]$$

Because of overlapping, i_0 and κ may not be uniquely determined; here a fixed but arbitrary instance of $\mathbf{J}$ is intended. If $\mathbf{C}$ is a control congruence then $\mathbf{J}$ is uniquely determined by $\mathbf{C}$.

The supervisor induced by a control cover is nearly identical to that induced by a standard cover in [16]. Of particular note is the fact that the selection of i_0 is arbitrary in both cases, yet the induced agents remain control-equivalent to the original supervisors. Although it may appear that the induced supervisor here is not as rigorous in its requirement for marked states ($X_i \cap X_m \neq \emptyset$) as in [16], remember that all states $x, x' \in X_i$ must also exist as pairs in $\mathcal{R}$. Hence, all subsets in I_m are only composed of those states in **SUPER** that have identical marking actions.

Proposition 2.1 [13, p. 36].

J is control equivalent to **SUPER** with respect to **G**.

Definition: Normal supervisors [13, p. 36].

A DES **SIMSUP** $= (Z, \Sigma, \zeta, z_0, Z_m)$ is normal with respect to **SUPER** if

1. $(\forall\, z \in Z)(\exists\, s \in L(\mathbf{SUPER}))\, \zeta(z_0, s) = z$
2. $(\forall\, z, z' \in Z)(\forall\, \sigma \in \Sigma)\, [\zeta(z, \sigma) = z' \Rightarrow (\exists\, s \in \Sigma^*)\, [s\sigma \in L(\mathbf{SUPER}) \wedge \zeta(z_0, s) = z]]$
3. $(\forall\, z \in Z_m)(\exists\, s \in L_m(\mathbf{SUPER}))\, \zeta(z_0, s) = z$

Thus, **SIMSUP** is normal if (1) its states are all reachable and (2) its transitions [sic] all taken, under strings in $L(\mathbf{SUPER})$, namely no state or transition in **SIMSUP** is superfluous; (3) each marked state in **SIMSUP** is reachable by at least one string that is marked by **SUPER**, namely no state in **SIMSUP** is marked unnecessarily. If **SIMSUP** is control equivalent to **SUPER** with respect to **G** but not normal, then simply deleting all superfluous states and transitions, and converting all unnecessarily marked states into unmarked states in **SIMSUP**, will render **SIMSUP** normal, while preserving the control equivalence between **SIMSUP** and **SUPER**. Normalizing **SIMSUP** cannot increase its state size, so a minimal normal supervisor must also be a minimal supervisor. From now on we therefore consider only reduction to normal supervisors.

Requirements (1) and (3) are simply applications of *K-accessibility* from [16] to **SIMSUP** using $L(\mathbf{SUPER})$ and $L_m(\mathbf{SUPER})$. Requirement (2) is a more formal statement and application of *strong K-accessibility* from the same paper; namely that every state and transition in **SIMSUP** must be visited or taken by some string in $L(\mathbf{SUPER})$. While (2) may seem somewhat redundant in the face of (1) and (3), consider that a state may have self-transitions defined or that two states may have multiple transitions for different events between them.

In general, the definitions of "normal" here and in [16] differ greatly in that

1. **SIMSUP** and **SUPER** are implicitly defined, and thus have no feedback map.
2. A normal supervisor in [16] is defined relative to the plant, whereas a normal supervisor here is defined relative to a control equivalent supervisor.

Definition 2.3 [13, p. 37].

Let

$$\mathbf{G}_A = (X_A, \Sigma, \xi_A, A_{x,0}, A_{X,m})$$

$$\mathbf{G}_B = (X_B, \Sigma, \xi_B, B_{x,0}, B_{X,m})$$

$\mathbf{G}_B$ is a DES-epimorphic image of $\mathbf{G}_A$ under DES-epimorphism $\theta : X_A \to X_B$ if

1. $\theta : X_A \to X_B$ is surjective
2. $\theta(A_{x,0}) = B_{x,0}$ and $\theta(A_{X,m}) = B_{X,m}$
3. $(\forall\, x \in X_A)(\forall\, \sigma \in \Sigma)\, \xi_A(x,\sigma)! \Rightarrow [\xi_B(\theta(x),\sigma)! \wedge \xi_B(\theta(x),\sigma) = \theta(\xi_A(x,\sigma))]$
4. $(\forall\, x \in X_B)(\forall\, \sigma \in \Sigma)\, \xi_B(x,\sigma)! \Rightarrow [(\exists\, x' \in X_A)\, \xi_A(x',\sigma)! \wedge \theta(x') = x]$

In particular, $\mathbf{G}_B$ is DES-isomorphic to $\mathbf{G}_A$ if $\theta : X_A \to X_B$ is bijective.

Of interest here is the striking similarity between the definitions of DES-epimorphism and projections in [8]. In fact, the first two requirements are identical. However, the manner in which θ is preserved across the transition functions is different from π; while [8] simply requires that $\hat{\xi}(\sigma, \pi(x)) = \pi \circ \xi(\sigma, x)$ where $\xi(\sigma, x)!$, the definition here looks at it from both directions. Specifically, (3) states that any transition (x, σ, x') in $\mathbf{G}_A$ must be defined in $\mathbf{G}_B$ as $(\theta(x), \sigma, \theta(x'))$. In the other direction, (4) requires that any transition (y, σ, y') in $\mathbf{G}_B$ must be defined at least once in $\mathbf{G}_A$ as (x, σ, x') where $\theta(x) = y$, but not necessarily for all such x that map to y.

Theorem 2.1 [13, p. 37].

Generalized quotient theorem. Let **SUPER** be a supremal supervisor for **G** and let **SIMSUP** be any normal supervisor with respect to **SUPER** that is control-equivalent to **SUPER** with respect to **G**. Then there exists a control cover **C** on **SUPER** for which some induced supervisor **J** is DES-isomorphic to **SIMSUP**.