2005

Improvements on handling design errors in communication protocols.

Lihua Duan
University of Windsor

Improvements on Handling Design Errors in Communication Protocols

by

Lihua Duan

A Thesis

Submitted to the Faculty of Graduate Studies and Research

through Computer Science

in Partial Fulfillment of the Requirements for

the Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

2005

Your file Votre référence
ISBN: 0-494-09791-4
Our file Notre référence
ISBN: 0-494-09791-4

Abstract

With the rapid development of the Internet and distributed systems, communication protocols play a more and more important role. The correctness of the design of these communication protocols becomes crucial especially when critical applications are concerned. Common logical design errors in communication protocols include deadlock states, unspecified receptions, channel overflow, non-executable transitions, etc. Such design errors can be removed via protocol synthesis, or be detected through reachability analysis. The former may introduce more states and transitions than needed and the latter suffers from state space explosion problem. Here we present an improvement on existing technique to transform a protocol design into a deadlock-free one where the number of introduced new states and transitions can be considerably reduced. We also propose a sound reduction technique on a class of protocol designs to significantly reduce their sizes in order to perform reachability analysis.

Keywords: Communication Protocols, Protocol Synthesis, Formal Verification, Protocol Design, Communicating Finite State Machines.

Acknowledgments

First and foremost, I would like to express my heartfelt thanks to my supervisor, Dr. Jessica Chen, for her invaluable guidance, extensive time, extreme patience, and enthusiastic encouragement during my entire master studies. Without her help, the work presented here would not have been possible. Also, as an international student, I would have had a much harder life without her support and advice. I appreciate her for all she has done for me very much and it will always remain with me held deeply in my memory.

I would like also to thank my committee members, Dr. Scott Goodwin, Dr. Tim Traynor, and Dr. Luis Rueda, for spending their precious time to read this thesis and providing their comments and suggestions to this thesis.

Furthermore, I would like to thank my friends and fellow graduate students in my group for their help and discussions.

My special thanks goes to the secretary at the School of Computer Science, Ms. Mary Mardegan, for her consistent help.

Last, but not least, I would like to thank all my family members for their support and encouragement for my studies.

Contents

List of Tables

List of Figures

1 Introduction

With the rapid development of the Internet and distributed systems, communications among individual processes play a more and more important role. Protocols are the core of the communication networks. A protocol is communication software that specifies the interactions among a set of communicating entities by exchanging messages over the channels. In other words, "A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event" [18]. These entities, in some literature, are also called communicating processes. Very often, each process is modeled as a *communicating finite state machine* (CFSM)[2, 3] and each channel between two processes is modeled as an error-free simplex FIFO queue.

A protocol design consisting of a set of CFSMs may suffer from logical errors. The common logical errors include deadlock states, channel overflow, nonexecutable transitions, etc. Once a faulty protocol is put into use, especially for critical applications, the loss can be enormous. Thus, it is crucial to construct an error-free protocol design and formally verify its correctness before the implementation. Typical approaches to achieve this goal are as follows.

(1) Protocol synthesis. Given a partially specified protocol design, a complete design is constructed formally and automatically so that the constructed design does not manifest any logical errors. In reverse engineering, a set of observations of an existing communication system can be regarded as a partially specified protocol design, and the recovery of the presumed design is the work of protocol synthesis.

(2) Formal verification. There are two directions in the literature: theorem proving and finite state based model checking. The former involves three aspects, namely, formal modeling of the design, formal modeling of the design proper-

ties, and the inference rules to prove the design satisfies the properties. For the latter, the design is interpreted as operational models, such as Finite States Machines (FSM) or Labeled Transition Systems (LTS), and the design properties are normally modeled by temporal logic languages. These approaches use state exploration techniques to check if the design conforms to the properties. In the context of this thesis, when we mention formal verification it means the finite state based model checking approach since the protocol is modeled as a set of CFSMs.

In a software life cycle, protocol synthesis is used in the design phase, and formal verification is used in the design verification. A software engineer may have different concerns under these two situations though he/she is always dealing with the logical errors in the protocol design. In the design phase, a major concern is to construct a minimum design that is error free and satisfies all the requirements of the customers. In the design verification, he/she will confront state explosion problem; i.e., the state space increases exponentially with the increase of the size of the protocol design, so one of the major concerns is to reduce the number of states meanwhile all the logical errors in the design can still be detected. This thesis is motivated by these concerns, and some improvements are made upon existing methods.

In the design phase, when a set of observations is given, we aim at constructing a deadlock-free design from it. The work in this thesis is motivated by [6]. In [6], three construction rules were proposed, and we want to improve the third rule, namely, $Rule_{neg}$, in a sense to add fewer states and transitions while the constructed design is still deadlock-free. Given a 2-process protocol design, the original rule adds the entire negation ("negation " means if a transition in the specification is a sending transition with the label $-m$, then the negation of that transition is a receiving transition with the label $+m$; vice versa.) of the specification of one process onto the specification of the other process. Actually, some of the newly added states and transitions make no contribution to the removal of deadlock states. Therefore, in

our improved $Rule'_{neg}$ we consider only the partial specification of one process that can really help to remove the deadlock states, and thus fewer states and transitions are added to the specification of the other process.

In the design verification, the method we propose aims at reducing the number of states of the specification of each process before the global reachability analysis. Actually, our method can be regarded as pre-processing of the global reachability analysis, and other reduction methods, such as partial order reduction, can be applied to our derived specifications. We develop two reduction rules to deal with a specific pattern of transitions in the protocol specification, that is, at some state, there is a choice about the execution order of two transitions; whatever option is chosen, after the execution of these two transitions, the same state is entered eventually. *Reduction Rule 1* deals with the choice of a sending transition and a receiving transition while *Reduction Rule 2* deals with the choice of two sending transitions to different processes. When the conditions of the rules are met, some transitions can be considered as redundant transitions for formal verification and are removed, thus, the search state space is reduced. Furthermore, we prove *Reduction Rule 1* preserves deadlock and unspecified reception states and channel overflow errors but may not preserve non-executable transition errors. When the error of non-executable transitions is concerned, *Reduction Rule 1* should not be used. We also prove *Reduction Rule 2* preserves all these four errors. In this thesis, we only discuss the verification of these four errors; the verification of other advanced properties is beyond the scope of this thesis. In the end, we discuss the efficiency of our reduction method with two examples. The drawback of this method is that if the protocol specification does not contain any pattern conforming to the conditions of the rules, the application of our method has no effect.

The rest of the thesis is organized as follows. Section 2 discusses the related work on the construction of protocol design and the reduction techniques of protocol design verification. Section 3 introduces the terminology and notation used

throughout the thesis. Section 4 shows some common errors of protocols and the advanced properties of protocols. In Section 5, we improve the third construction rule to construct a deadlock-free protocol design with adding fewer states and transitions when a set of observations is given. In Section 6, we propose a method with two reduction rules to reduce the protocol specifications during design verification. In the last section we address the conclusions of this thesis and future work.

2 Related Work

2.1 Related Work of Protocol Design Construction

2.1.1 Protocol Synthesis Approaches

The methods of protocol synthesis are divided into two categories: protocol-oriented synthesis methods and service-oriented synthesis methods. The former completes the protocol design with the given incomplete one; and the latter constructs the underlying protocol with the specification of the service that the protocol should provide. Saleh [25] gave an annotated bibliography on the synthesis of communication protocols. The method discussed in Section 5 belongs to the first category: we start from incomplete traces/observations, complete the design so that the interactions between its protocol entities proceed without manifesting any logical errors.

The related work we discuss here shares some common assumptions: 1) The channels are FIFO and reliable; 2) The formal model is communicating finite state machine (CFSM), which we will give formal definition in next section.

One of the most influential papers in protocol-oriented synthesis is presented by Zafiropulo *et al.* in [34], and it is referred to as the ZWRCB methodology. Given two CFSMs, which might be incomplete or erroneous, the methodology proceeds as follows:

(1) The designer adds one sending transition to one machine;

(2) The designer executes an algorithm that is based on the synthesis rules to add the corresponding receiving transitions to another machine. In the paper, three rules are developed to add the receive transitions.

This procedure continues until the termination condition is met, that is, the same state is reentered by the same receiving transition. Note, the designer should interact with the procedure whenever a designer's decision is needed.

Their approach is different from ours in the following aspects:

(1) Theirs is a forward engineering method while ours may be used as a reverse engineering method. Though the given design might be incomplete, in their approach, it is assumed that some information in the format of the design graph is given. Ours may start from a set of traces, and we have to construct the design from them.

(2) Theirs can deal with the protocol with cycles while ours can only deal with tree-like graphs. This is because the former can identify when and which states are repeated from the graph directly while ours cannot identify the states from the implementation.

(3) The ZWRCB method first flattens the graph of each machine into a tree and adds all the missing receiving transitions into the tree. Then use a "flooring" operation to merge identical states and edges. In contrast to that, we construct the design directly without an intermediary graph.

(4) The ZWRCB method only considers that one sending transition may have multiple responding receiving transitions; but actually, the sending transition may have multiple occurrences; i.e., for the sending transition and the corresponding receiving transition, a one-to-many relationship is possible, and a many-to-one relationship is possible, and a many-to-many relationship is also possible.

(5) The ZWRCB is semi-automatic while ours is automatic.

The work of Sidhu [26] synthesizes n-process synchronous protocol designs from the informal specification; e.g., English description or informal graphical representations. He developed the global specification as a reachability tree which is the same as the one in [34]. In particular, he formalized the states of the global specification as an $n \times n$ matrix which contains the information of each process and each channel. The author gave some suggestions for preserving the error-free property during

protocol synthesis but no rule is given to guarantee its error-freeness. Our approach starts from a set of formal — but maybe incomplete — traces and the approach results in an error-free design.

The work of Gouda and Yu [12] is limited to two CFSMs and is based on the assumption that protocols are asymmetric where one machine M has a higher priority than the other. The algorithm takes machine M and constructs two communicating machines M' and N' such that 1) M' is constructed from M by adding some receiving transitions to it, and 2) the communication between M' and N' is bounded and free from errors. They only add receiving transitions while we add both sending transitions and receiving transitions.

The work of Choi [8] is limited to two CFSMs. It starts from a set of well-formed protocol sequences, then applies a synthesizing algorithm to generate the CFSMs, and finally uses equivalence relations to reduce the CFSMs.

The work of Kakuda and Wakehara [16] synthesizes protocols for an unlimited number of processes. The key idea of their approach is components. First, the protocol specification of one process is split into many components according to which process it communicates with; second, it matches the components from 22 patterns developed by the paper to get a refined deadlock-free design; finally, it synthesizes the refined components to a protocol specification.

2.1.2 The Idea of Alur et al.

In [1], Alur et al. answers the following questions: does a given design exactly describe an implementation? That is, is the design realizable by an implementation? If it is realizable, does it contain any deadlock states? Furthermore, what are the conditions that guarantee a deadlock-free realizable design?

An example in [1] is given in Figure 1 to show the idea of Alur et al. It is the setting of a nuclear power plant. Two clients, P_1 and P_2, can perform remote updates on the processes which control the plant: process UR controls the amount

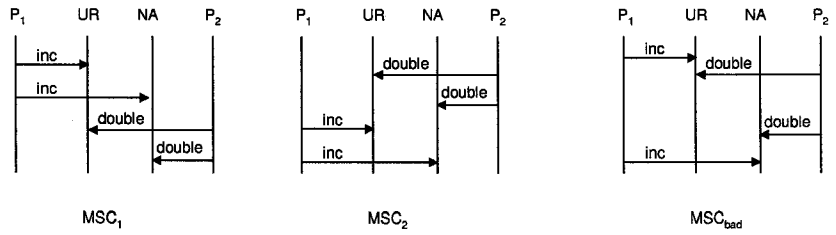


Figure 1: An Example to Show the Idea of Alur et al.

of Uranium fuel in the daily supply and process NA controls the amount of Nitric Acid in the daily supply. It is necessary that these amounts be equal in order to avoid a nuclear accident. The "inc" message denotes a request to increment the fuel amount by one unit, while the "double" message denotes a request to double the fuel amount. MSC_1 and MSC_2 are the design of this system. The authors argue that any system implemented according to this design will definitely have a trace like MSC_{bad} which will result in an incorrect fuel mix and furthermore an nuclear accident. They then conclude this system must be redesigned.

Their ideas share some similarities with ours:

- Analyze and construct a deadlock-free design from a given set of traces though they are in the form of message sequence charts in [1];
- Solve the similar problem; i.e., if we project these traces to the distributed components at first, then compose them to a global specification, more traces will be created than the previous ones.

We also have some differences:

- Our approach is to construct a deadlock-free design while their approach is to verify if a design is deadlock-free. Our approach adds transitions to the protocol specification directly according to our rules; and after one execution of our rules, we construct all the possible traces. Alur et al. propose two conditions for a deadlock-free design and thus two algorithms to check if these

two conditions are satisfied respectively; if the design is not deadlock-free, the run stops and one faulty trace is given. Once there is a faulty trace, the protocol of the system has to be redesigned.

- We use a different formalism, namely, CFSM, and it has strength and weakness. The strength is that the design is always realizable: we do not need to consider solving the realizability problem. This is because the finite state machine is an abstraction of the behavior of the corresponding real machine. On the other hand, message sequence charts they use can contain more information, for example, a message sequence chart implies all the partial order relations among the events while we cannot find this information from the known traces. Thus, their approach can find more traces than ours.

2.2 Related Work on Reduction During Protocol Verification

2.2.1 Partial Order Reduction (POR)

Partial order reduction [11, 10] is a series of state reduction techniques used in the verification tool SPIN. It avoids redundant interleaving of transitions by selective search. POR uses the concept of trace equivalence to partition the set of all the possible transitions. Two traces are equivalent if one can be transformed into the other by swapping adjacent independent transitions. One representative trace is selected from each equivalence class, thereby reducing the state space.

POR is different from our method proposed in Section 6 in the following aspects. First, it is applied during the global reachability analysis while our reduction rule can be applied on the protocol specifications before the global reachability analysis. In some sense, our method proposed in Section 6 can be considered as the pre-processing of POR. Second, though both methods try to find out what kind of interleavings are unnecessary for verification, POR focuses on the interleavings in

the global specification while our method focuses on the interleavings in the protocol specifications. Third, distinguishing the independent transitions is a huge and complicated task for partial order reduction since it needs to consider the partial order relations among different processes, but for our method, it is straightforward to distinguish if two transitions are independent within one process, namely, if sending messages or receiving messages are enabled at the same time but to or from different channels, they are independent. Fourth, the sleep set method in POR is similar to ours in a way that both deal with special patterns of independent transitions. But the sleep set method is based on the record of the tracing history so it is dynamic while our method is static. Fifth, POR is more efficient than ours as far as reduced states and transitions are concerned.

2.2.2 Simultaneous Reachability Analysis (SRA) and Blocking-based Simultaneous Reachability Analysis (BSRA)

Simultaneous Reachability Analysis (SRA) [22, 17] and Blocking-based Simultaneous Reachability Analysis (BSRA) [20, 21] are reduction techniques to tackle the state explosion problem during protocol reachability analysis in another direction. SRA allows multiple process to proceed simultaneously and merge all the involved transitions as one transition with multiple simultaneously executable actions. However, each process can execute at most one transition each time. BSRA is based on the idea of SRA and made some improvements in the sense that more transitions can be merged each time by introducing the notion of blocking points. In BSRA, transitions of receiving messages in different processes and transition sequences of sending messages to different processes are merged into one transition in the global network of the protocol. Its idea is to use local blocking points, which are states with one or more outgoing receiving transitions, or with no outgoing transitions at all. The merging must start and stop at local blocking points. The algorithm preserves deadlocks and non-executable transitions, but channel overflow and un-

specified receptions are not guaranteed to be preserved. However, the algorithm can be modified to preserve these two properties. For example, to verify the property of channel overflow of a channel c, the algorithm will delay the consumption of the messages in c as much as possible; the algorithm can only check one channel of one execution.

The reduction technique works in an eager semantics in a sense that when a message is sent to the channel, it will be consumed in the next transition during reachability analysis. Thus, the interleaving with delayed executions is reduced.

Note: They used a different definition of unspecified receptions from ours in that they do not consider the possibility of succeeding transitions consuming the messages already in the channels.

2.2.3 Prefix-based Techniques

[15, 5] proposed a prefix-based algorithm to solve state explosion problem during formal verification, and it is called reachability testing. One important concept is "race variant", which is the common prefix of some traces. All possible traces are partitioned according to race variants: traces with the same race variant as their prefix are in the same equivalence class. The idea of the algorithm is to control an execution up to a certain point (the end of race variant) and then exhaustively explore the possible paths after that point. In the next run, all the information of the previous run is abandoned. That is how it solves the state explosion problem. The most challenging task of this method is how to identify and compute the race variants. A partial order "happened before" relation is defined to help to identify the race variants.

It is a technique starting from application levels for verifying concurrent programs and it is implemented as prototype tool RichTest.

3 Preliminary

We consider an *n-process communication protocol* (or *n-process protocol*, *protocol* for short) as a fixed number of processes communicating with each other by sending and receiving messages over error-free simplex channels. Each process is a protocol entity which is represented by a *communicating finite state machine (CFSM)*, and each error-free simplex channel is represented by an unbounded FIFO queue. The following definition of CFSM is formalized as in [6].

Definition 3.1 (Communication Finite State Machines) *A communication finite state machine is a quadruple $(S,M,s^0,\rightarrow)$ where*

- *S is a finite set of states and $s^0 \in S$ is the initial state,*
- *M is a finite set of messages and*
- *$\rightarrow \subseteq S \times \{+m, -m | m \in M\} \times S$ is a set of transitions.*

A transition $(s, \mu, s') \in \rightarrow$ of a process, also denoted as $s \xrightarrow{\mu} s'$, intuitively, changes the state of the process from s to s' by event μ. We use $-m$ to denote the event of sending message m, and $+m$ to denote the event of receiving message m. Moreover, we will use E_M to denote the set of events of sending/receiving messages in M; i.e., $E_M = \{+, -\} \times M$. In this thesis, since we use graphs to represent the specifications of processes and protocols, we also call the events that cause the transitions the labels of the transitions, as we do in graph theory, for convenience.

Note: an erroneous channel, which is also called a noisy channel or an unreliable channel, distorts the messages passing through it. In the appendix of [34], Zafiropulo *et al.* used a CFSM to model the behavior of an erroneous simplex channel responding to one particular message x. Whenever a message x is received, the process representing the behavior of the channel moves to the state s^1 from the initial state. Then it arbitrarily chooses one of the following actions to return to the initial

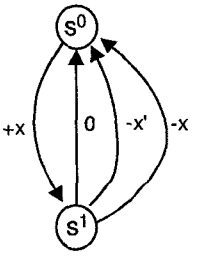


Figure 2: Model An Erroneous Channel Using CFSM

state from state s^1: 1) non-event, which models the behavior that the channel loses the message x; 2) sending x', which models the behavior that the channel corrupts x; or 3) sending x, which models the behavior that the channel lets the message go through unchanged. Figure 2 shows the process of an erroneous channel using CFSM. Furthermore, an erroneous channel may reorder the messages if it contains more than one message. Solving problems caused by erroneous channels, error recovery, numbering messages and synchronous hand-shaking techniques may be applied. It is another important topic of protocol design area, and some research work has been done upon it, such as [23, 7], etc. In this thesis, we focus on solving problems caused by concurrent actions rather than those of erroneous channels, so we restrict our discussion to error-free, FIFO channels, which means the channels always send what they have received in the same order.

In a protocol, we use binary relation (i, j) to denote the existence of a simplex channel from process P_i to process P_j, and we use $M_{i,j}$ to denote all messages that can be put onto it. For convenience, we assume that the messages in different channels are all distinct. This assumption is reasonable because the sender should know the destination of the message in order to send it out to the channel and the receiver should know the source of this message. In a sense, each message has a header to label its source and destination, and to make the messages in different channels disjoint. Now, we can give the definition of n-process communication protocols.

Definition 3.2 (n-process Communication Protocols) *An n-process communication protocol P is a triple (L, M, T) where*

- $L \subseteq \{(i,j)|i,j \in [1,n], i \neq j\}$ *for* $n \geq 2$,
- $M = \{M_{i,j}|(i,j) \in L\}$ *with* $M_{i,j} \cap M_{k,l} = \emptyset$ *if* $(i,j) \neq (k,l)$ *and*
- $T = \{(S_i, M_i, s_i^0, \rightarrow_i)|i \in [1,n],\ M_i = \bigcup_{j=1,\ldots,n, j\neq i}(M_{i,j} \cup M_{j,i})\}$ *is a set of CFSMs.*

In the context of this thesis, we use P_i to represent the i^{th} process of $n-$process protocol P, and use T_i to represent the CFSM of process P_i.

An n-process Communication Protocol is minimal if no reduction can be made to $(CFSM_i)_{i=1}^n$ using standard determinization and reduction algorithms in automata theory [14].

A global state of a protocol is composed of a local state of each process and a content of each channel. The content of the channel can be represented as a string ω of messages which might be empty. In this thesis, we may use "state" as a short for"global state" and "local state" will be used in an explicit way. We use an $n \times n$ matrix to represent a global state as in [31]. Let s_i denote the local state of process P_i, and let $\omega_{i,j}$ denote the content of simplex channel (i,j). Then we have a global s:

$$s = \begin{pmatrix} s_1 & \omega_{1,2} & \omega_{1,3} & \ldots & \omega_{1,n} \\ \omega_{2,1} & s_2 & \omega_{2,3} & \ldots & \omega_{2,n} \\ \omega_{3,1} & \omega_{3,2} & s_3 & \ldots & \omega_{3,n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ \omega_{n,1} & \omega_{n,2} & \omega_{n,3} & \ldots & s_n \end{pmatrix}$$

The evolution of a protocol is described in terms of the transitions from one global state to another. Such transitions are built up on the basis of the transitions

of each process, taking into account their effect on the contents of related channels. We use $|\omega|$ to denote the length of a string ω and considering it is a FIFO channel, we will assume the following functions on strings:

- $del(\omega)$ ($|\omega| \geq 1$) returns the string resulting after the first element of ω is deleted;
- $ins(\omega, m)$ returns the string resulting after m is appended at the end of ω;
- $\omega[i]$ ($|\omega| \geq 1$) returns the ith element of ω.

The initial global state is characterized in this way: 1) each process P_i, $i \in [1, n]$, is at its initial state s_i^0; 2) all channels are empty; i.e., for any $i, j \in [1, n], i \neq j$, $\omega_{i,j} = \varepsilon$. In this thesis, we assume protocols are periodic; i.e., after an execution of the protocol, it returns to the initial global state; therefore, we do not have any additional final global state, and the initial global state is regarded as the only final global state.

If each process of a protocol cannot execute its next sending transition until it receives the reply to the previous sent message, it is a synchronous protocol. Otherwise, it is an asynchronous protocol. In this thesis, we discuss methods of general protocols but especially suitable for asynchronous protocols.

Definition 3.3 (Network of a Protocol) *The network N of protocol $P = (L, \{M_{i,j} \mid (i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ is a quadruple $(S, M, s^0, \rightarrow)$, where*

- $S = \prod_{i=1,\ldots,n} S_i \times \prod_{(i,j)\in L} M_{i,j}^*$;
- $M = \bigcup_{i=1,\ldots,n} M_i$;
- $s^0 = \prod_{i=1,\ldots,n} s_i^0 \times \prod_{(i,j)\in L} \varepsilon$;
- $\rightarrow \subseteq S \times E_M \times S$ *is the set of transitions defined as follows:* $\forall s, s' \in S$, $m \in M_{k,l}$ *where* $s = \prod_{i=1,\ldots,n} s_i \times \prod_{(i,j)\in L} \omega_{i,j}$, $(\omega_{i,j} \in M_{i,j}^*)$,
  $s \xrightarrow{+m} s'$ *iff* $\exists (k,l) \in L$, $m \in M_{k,l}$, *and* $s_l' \in S_l$ *such that*

- $\omega_{k,l}[1] = m$,
- $s_l \xrightarrow{+m}_l s'_l$,
- $s' = \prod_{i=1,...,l-1,l+1,...,n} s_i \times s'_l \times \prod_{(i,j)\in L \wedge (i,j)\neq(k,l)} \omega_{i,j} \times del(\omega_{k,l})$;

$s \xrightarrow{-m} s'$ *iff* $\exists (k,l) \in L$, $m \in M_{k,l}$, *and* $s'_k \in S_k$ *such that*

- $s_k \xrightarrow{-m}_k s'_k$,
- $s' = \prod_{i=1,...,k-1,k+1,...,n} s_i \times s'_k \times \prod_{(i,j)\in L \wedge (i,j)\neq(k,l)} \omega_{i,j} \times ins(\omega_{k,l}, m)$;

The network of a protocol is actually a directed graph in which global states are denoted as nodes and transitions are denoted as arcs. In this thesis, we use the network of a protocol and the global specification of a protocol interchangeably; we use the protocol specifications of a protocol and CFSMs of a protocol interchangeably. Note: $M^*_{i,j}$ means a string over alphabet $M_{i,j}$ and it is possibly an empty string.

Definition 3.4 (Reachable States) *Let* $N = (S, M, s^0, \rightarrow)$ *be a network of protocol* P. *A state* $s \in S$ *is reachable if* $s = s^0$, *or for some* $r \geq 1$, $\exists s^1, ..., s^r \in S$, $\exists \mu_1, ..., \mu_r \in E_M$ *such that* $s^r = s$ *and* $s^{i-1} \xrightarrow{\mu_i} s^i$ *for* $i = 1, ..., r$.

Definition 3.5 (Traces) *Let* $N = (S, M, s^0, \rightarrow)$ *be a network of protocol* P *and* $s \in S$ *is a reachable state. If* $s = s^0$, *then the trace* ρ *of* s *is* ε. *Otherwise, for some* $r \geq 1$, $\exists s^1, ..., s^r \in S$, $\exists \mu_1, ..., \mu_r \in E_M$, $s^{i-1} \xrightarrow{\mu_i} s^i$ *for* $i = 1, ..., r$, *such that* $s^r = s$. *Let* $\rho = \mu_1 \mu_2 ... \mu_r$, *then* $\rho \in E^*_M$ *is a trace of* s, *and we have* $s^0 \xrightarrow{\rho} s$.

Intuitively, a state s is reachable iff there exists a trace ρ leading to it from the initial state. R will denote the reachable global state space of a protocol P. Clearly, $R \subseteq S$ since R contains only those states in S that are reachable. We also use $\rho[i]$ to denote the ith element of ρ and $|\rho|$ to denote the length of the ρ.

A trace $\rho \in E^*_M$ is well-formed if all receiving events have their corresponding sending events; a well-formed trace $\rho \in E^*_M$ is complete if all sending events have their corresponding receiving events.

Definition 3.6 (Well-formed Trace) *$\forall e \in E_M, \varrho \in E_M^*$, e is **possible after** a trace ϱ if $\exists m \in M$, such that either e is a sending event $-m$ or e is a receiving event $+m$ only when the number of $-m$ of ϱ is more than the number of $+m$ of ϱ. A trace $\rho \in E_M^*$ is well-formed if for every prefix ϱ of ρ, the next event e is **possible after** ϱ.*

Definition 3.7 (Complete Trace) *A well-formed trace $\rho \in E_M^*$ is complete if for all $m \in M$, the number of $-m$ of ρ is equal to the number of $+m$ of ρ.*

Definition 3.8 (Deterministic Network) *Let $N = (S, M, s^0, \rightarrow)$ be a network of protocol P, and $\rho \in E_M^*$ is a trace. s^0 **after** ρ is a set of states defined as $\{s | s^0 \xrightarrow{\rho} s\}$. N is deterministic if for all $\rho \in E_M^*$, s^0 **after** ρ has at most one element; otherwise, it is nondeterministic.*

In this paper, if not specified, the network of a protocol is deterministic.

Unlike a trace, we define the fragment of a trace ρ as a path σ which can start from any state, and we have $\sigma \in E_M^*$. Moreover, for any $s \in R$, we have the following expressions of the formulas:

- $s^1 \xrightarrow{\mu_1 \mu_2 \cdots \mu_r} s^{r+1}$ if $\exists s^2, ..., s^r \in S$ such that $s^i \xrightarrow{\mu_i} s^{i+1}$ for $i = 1, ..., r$;
- $s \xrightarrow{\sigma}$ where $\sigma \in E_M^*$ if $\exists s'$ such that $s \xrightarrow{\sigma} s'$;
- $s \xrightarrow{\sigma} \!\!\!\!\!\!\not\;\;$ where $\sigma \in E_M^*$ if $\nexists s'$ such that $s \xrightarrow{\sigma} s'$.
- $\sigma_1.\sigma_2$ is the concatenation of two paths σ_1 and σ_2, and we have $s \xrightarrow{\sigma_1.\sigma_2} s''$, where $\sigma_1, \sigma_2 \in E_M^*$, if $\exists s, s', s'' \in S$, such that $(s \xrightarrow{\sigma_1} s') \wedge (s' \xrightarrow{\sigma_1} s'')$.

4 Properties of Protocols

4.1 Common Errors of Protocols

In this thesis, we assume the channels between protocol entities are reliable, and the receiver can eventually get the message that is sent to it if it waits long enough; i.e., we do not discuss the protocols with explicit time constraints such as time-out. Within this framework, we discuss some properties of protocols that people are concerned with, i.e., potential design errors, namely, deadlock state, nonexecutable transitions, state ambiguities, and channel overflow. The reason why we call them potential design errors is because some of these errors are designed on purpose by the designers. For example, the designers may intend to terminate the protocol at a deadlock state when its function is complete rather than returning to the initial state. However, it is always useful to identify these potential errors.

4.1.1 Deadlock States

A deadlock is a reachable global state s where all channels are empty and no process can send a message.

Definition 4.1 (Deadlock States) *Let* $P = (L, \{M_{i,j} | (i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ *be a protocol and* $N = (S, M, s_0, \rightarrow)$ *a network of* P. *A state* s *is called deadlock if it is reachable and* $s = \prod_{i=1,\ldots,n} s_i \times \prod_{(i,j)\in L} \varepsilon$ *and* $\nexists i \in [1,n], m \in M_i$, *such that* $s_i \xrightarrow{-m}$.

Figure 3, which is a modification of [34], shows an example of various potential design errors. At the beginning of the interaction, both process P_1 and P_2 are at their initial states. When P_1 enters state r^1 by sending the message x , P_2 enters state t^1 by sending the message z at the same time. Then P_1 receives the message z and enters state r^2, while P_2 receives the message x and enters state t^2. Now both process are waiting to receive messages while all channels are empty and no process

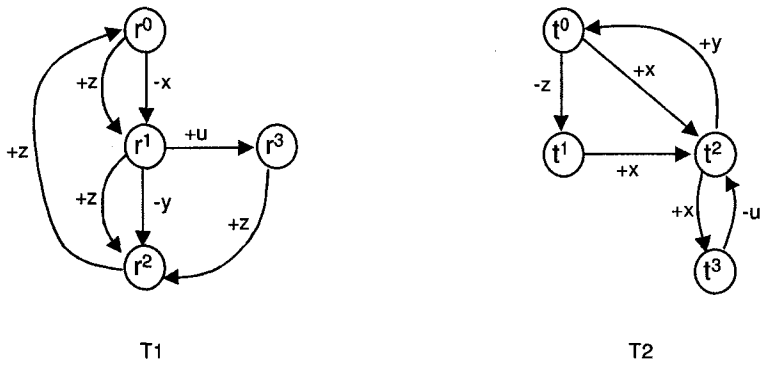


Figure 3: An Example of Two-process Protocol Containing Various Design Errors

can possibly send a message. Thus, the global state containing r^2 and t^2 as its local states for process P_1 and P_2 respectively is a deadlock state in the network of the protocol.

4.1.2 Unspecified Reception States

An unspecified reception is a reachable global state s where the head of an incoming channel cannot be consumed by the related process at s and at all reachable global states succeeding s.

Definition 4.2 (Unspecified Reception States) *Let* $P = (L, \{M_{i,j} | (i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ *be a protocol and* $N = (S, M, s_0, \rightarrow)$ *a network of* P. *A state* s *is called an unspecified reception state if it is reachable and* $s = \prod_{i=1,\ldots,n} s_i \times \prod_{(i,j)\in L} \omega_{i,j}$, *where* $\omega_{i,j} \in M_{i,j}^*$, *such that* $\exists (k,l) \in L$, $\omega_{k,l} \neq \varepsilon$, $s \xrightarrow{+\omega_{k,l}[1]}$, *and* $\forall t \in S$, $\sigma \in E_M^*$, *such that* $s \xrightarrow{\sigma} t : t \xrightarrow{+\omega_{k,l}[1]}$.

Our definitions of deadlock states and unspecified reception states conform to the notions in most literature, such as [34, 6], etc.. However, in some literature, such as [1], the definitions of deadlock states and unspecified receptions are combined together as deadlock states since both the deadlock states and the unspecified reception states share the common problem; i.e., they both get stuck and there is

no outgoing transition from them in the network of the protocol, therefore, we will use error state for a shorthand of deadlock state or unspecified reception state.

In Figure 3, there is an unspecified reception error. Let us look at the following scenario. Both process P_1 and P_2 are at their initial states. The process P_2 sends the message z and enters the state t^1. The process P_1 receives the message z and enters the state r^1, then P_1 continues to send the message y. Now it is stuck. Both processes are waiting for receiving messages while the head of channel $(1,2)$, namely y, cannot be consumed by P_2. Thus, the global state containing these two local states is an unspecified reception state in the network of the protocol.

4.1.3 Channel Overflow

A channel overflow error may occur if: 1) the length of the content of a channel may be infinite, since no channel in the real world can have an infinite length to contain the content; 2) the length of the content of a channel is finite but it is greater than the physical limitation of the channel.

Definition 4.3 (Channel Overflow Error) *Let* $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ *be a protocol,* $N = (S, M, s_0, \rightarrow)$ *the network of* P. *For any* $s \in S$, $s = \prod_{i=1,\ldots,n} s_i \times \prod_{(i,j)\in L} \omega_{i,j}$, *where* $\omega_{i,j} \in M_{i,j}^*$, *and the physical limitation of the channel* (i,j) *is* $k_{i,j}$. *a channel overflow error occurs over channel* (i,j) *if* $\exists s \in S, \rho \in E_M^*$ *such that* $s^0 \xrightarrow{\rho} s$ *and* $k_{i,j} < |\omega_{i,j}|$.

Definition 4.4 (Bounded Protocol) *Let* $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ *be a protocol and* $N = (S, M, s_0, \rightarrow)$ *a network of* P. *If* $\exists k$, k *is an integer, for any state* $s = \prod_{i=1,\ldots,n} s_i \times \prod_{(i,j)\in L} \omega_{i,j}$, *where* $\omega_{i,j} \in M_{i,j}^*$, *such that* $|\omega_{i,j}| \leq k$, *then the protocol* P *is a bounded protocol. Otherwise, it is unbounded. If* k *is the least integer satisfying condition* $|\omega_{i,j}| \leq k$, *we say the protocol is* $k-$*bounded.*

Figure 4 shows a simple example of unbounded protocol: if process P_2 delays receiving the messages in channel $(1,2)$, the length of the content may be infinite

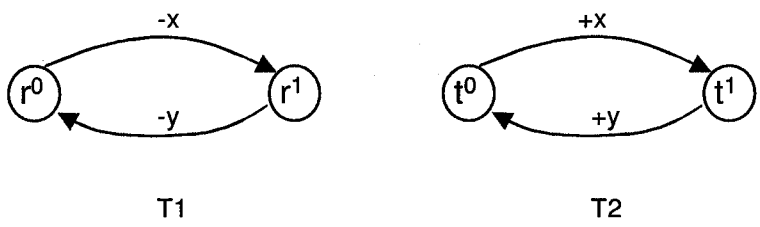


Figure 4: An Simple Example of An Unbounded Protocol

and we cannot find an integer always greater than it. The example in Figure 3 is a 2-bounded protocol. All synchronous protocols are 1-bounded protocols.

A possible solution to tackle the channel overflow errors for an unbounded protocol is to modify the protocol itself either considering the recovery mechanism or making the protocol bounded. To tackle the channel overflow error for a $k-$bounded protocol, we can set the physical limitation of channels greater than k or modify the protocol itself.

The definition of the bound of a protocol P comes from the network of P which may suffer from the state explosion problem, but it is hard to find a better way. Next, we will show that it is not easy to find the bound of P from other specification, such as CFSMs of P. Let us start with the following claim.

Claim: Given a protocol $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i,\ M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$, P is unbounded if in any $CFSM_i$, $\exists \sigma \in E_{M_i}^*$, where $\sigma = \mu_1 ... \mu_r$, such that for any $l \in [1,r]$, μ_l is in set $\{-\} \times M_{i,j}$ for some $j \in [1,n]$ and $s_i^1 \xrightarrow{\mu_1} s_i^2, ..., s_i^r \xrightarrow{\mu_r} s_i^1$.

PROOF: The condition of the claim actually guarantees there is a cycle of sending transitions in a CFSM. With this condition it is obvious that the process can always send the messages to the channel. Thus, it is possible there exists a trace with an infinite length. □

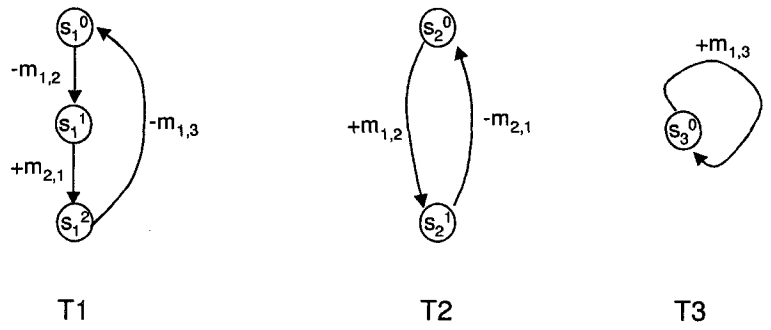


Figure 5: Counterexample of *Invalid Claim 1*

Intuitively, a protocol is unbounded if any of its CFSMs has a sending transition cycle. This cycle is called a livelock in some literature. Actually, the above condition is sufficient but not necessary: the reverse is invalid. Figure 5 shows a counterexample: There is no sending transition cycle in any of the CFSMs, but the communication between P_1 and P_3 is not bounded when P_1 and P_2 keep talking and P_3 is not going to receive any messages from the channel $(1,3)$.

For a similar reason, both directions of the following claim are invalid.

Invalid Claim: P is $k-$bounded iff $\exists j \in [1,n], \sigma \in E^*_{M_i}$ of $CFSM_i$, where $\sigma = \mu_1 ... \mu_r$, such that $\forall l \in [1, r-1]$, μ_l is in set $\{-\} \times M_{i,j}$, μ_r is in set $\{+\} \times M_{i,j}$, $s_i^1 \xrightarrow{\mu_1} s_i^2, ..., s_i^r \xrightarrow{\mu_r} s_i^{r+1}$, and k is the maximum number of μ_l that go to the same channel i, j for some j.

In fact, if we can find such k as that in the above claim, we can only conclude that the protocol is at least $k-$bounded. Thus, to decide the bound of a protocol, knowing only the information of each $CFSM_i$ of P is not enough, we have to know the composition of them, i.e., the network N of P.

4.1.4 Nonexecutable Transitions

Since the network N of a protocol P is the composition of all the process entities, it is straightforward that transitions in N involving process P_i is a subset of the

transitions defined in the corresponding CFSM of the process P_i. On the other hand, the transitions defined in the CFSM of the process P_i may not be a subset of transitions in N involving the corresponding process P_i. In the CFSM of the process P_i, some states may never be reached and some transitions may never be executed under normal conditions. We are more concerned about nonexecutable transitions because whenever these transitions are identified, we can trace and find the unreachable states.

Definition 4.5 (Nonexecutable Transitions) *Let* $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ *be a protocol,* $N = (S, M, s_0, \rightarrow)$ *the network of* P*, and* $\mu \in E_{M_i}$ *an event of process* P_i*. A transition* $s_i \xrightarrow{\mu} s_i'$ *of process* P_i *is nonexecutable if for all* $s \in S$ *that are reachable and contain* s_i *as its local state of process* P_i*,* $\nexists s' \in S$*, such that* $s \xrightarrow{\mu} s'$*, where* s_i' *is the local state of process* P_i *at* s'*.*

In Figure 3, no normal transition sequence can cause state t^2 of P_2 to receive message x, hence t^3 is not entered and message u cannot be sent. Consequently, state r^3 of process P_1 cannot be reached and the receiving transition with label $+z$ is nonexecutable.

4.1.5 Stable States and State Ambiguities

A tuple $(s_1, ..., s_n)$ of states is stable when all the channels between them are empty. Identifying stable-state tuples is useful for detecting loss of synchronization if the protocol is intended to be a synchronous protocol.

Definition 4.6 (Stable-state Tuples) *Let* $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ *be a protocol. If* $\exists s \in S$*, such that* $\forall (i,j) \in L, \omega_{i,j} = \epsilon$*, then* $(s_1, ..., s_n)$ *is a stable-state tuple, where* s_i *is the local state of process* P_i *at* s*.*

For 2-process protocol, a pair (s_1, s_2) of process P_1 and P_2 is stable when the channels between them are empty.

A state ambiguity exists when more than one state in one process can coexist stably with the exact same state of the other process.

Definition 4.7 (State Ambiguity) *Let* $P = (\{(1,2),(2,1)\}, \{M_{1,2}, M_{2,1}\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^2)$ *be a 2-process protocol.* (s_1, s_2) *is a stable-state pair at* s*, where* $s \in S$*,* s_1, s_2 *is the local state of process* P_1 *and* P_2 *at* s *respectively. A state ambiguity exists when* $\exists s' \in S$ *such that* (s'_1, s'_2) *is a stable state pair with either* $(s_1 = s'_1)$ *or* $(s_2 = s'_2)$.

In Figure 3, when process P_1 sends messages x and y and enters state r^2 eventually, P_2 receives the message x and y and returns to its initial state eventually. (r^0, t^0) and (r^2, t^0) are both stable-state pairs. Hence, t^0 of P_2 can coexist with two states of P_1, namely, r^0 and r^2, and it has a state ambiguity.

Note: The notion of state ambiguity is useful when 2-process protocols are concerned. State ambiguity is not necessarily an error, and it depends on the designer's intention. In this thesis, we do not discuss the potential error of state ambiguity.

4.2 Advanced Properties of Protocols

In the previous subsection, we discussed several simple traditional properties of protocol design, namely, deadlock states, nonexecutable transitions, channel overflow, and state ambiguity. With the development of the IT industry, more advanced properties have come into our concern. Among them, properties most used are divided into two categories: safety properties and liveness properties. As described in [19], intuitively, a safety property insists that "bad things" do not happen, a liveness property insists that "good things" do eventually happen. For example, the property described in English could be "for all the paths from initial state, a global state containing the local state s_i of process P_i must eventually happen"; "there exists a path in which all the global states do not containing the local state s_i of process P_i"; etc. Actually, all the design errors we discussed in the previous subsection

are in the family of the safety properties. In the literature, protocol properties are flexibly specified as temporal formulas using some temporal logic languages, such as linear-time propositional temporal logic (LPTL), computation tree logic (CTL), etc. How to verify these properties is a problem of model checking. Since model checking is not our focus, in this thesis, we only give a general procedure to verify a general property of a protocol.

As we know many properties can be represented as LPTL formulas. In [33], it is proved that it is possible to build a *Büchi automaton* [4] that accepts exactly the infinite words satisfying the temporal formula. A construction of a Büchi automaton from a formula can be found in [32, 27]. The following procedure can be found in [33, 30]:

(1) Build a Büchi automaton for the negation of the specified formula f which represents the required liveness property, and the resulting automaton A_f accepts all sequences of states that violate the formula f;

(2) Compose A_f and $(CFSM_i)_{i=1}^n$ of the protocol P as product A;

(3) Check if A is empty. If A is empty, it means P satisfies the property f, otherwise, P does not satisfy the property f.

In this thesis, we do not discuss the advanced properties of protocols. The reduction rules in Section 6 cannot be generalized for the detection of these properties.

5 Construction Rules During Protocol Design

In this section, we discuss the following problems: Given a set of observations, do they correspond to a deadlock-free design? If they do not correspond to a deadlock-free design, can we construct a deadlock-free design containing all these observations with reasonable construction rules? Is this design minimum? If we cannot construct such a design, what else do we need to do? Can more observations be helpful?

Our work of the construction of protocol designs is an improvement of the work of Chen and Ural in [6]. We use the first two construction rules, namely, $Rule_{det}$ and $Rule_{red}$, and four examples in [6]. The third rule $Rule_{neg}$ of the paper solves the deadlock state problems by mirroring the specification of one process onto the specification of the other process; however, we prove this rule adds more states and transitions than needed, and in this thesis, we improve $Rule_{neg}$ in the sense that the added states and transitions are no more than and often less than those added by original $Rule_{neg}$ while it is still guaranteed that no deadlock states are left.

5.1 Previous Work

The proposed method in [6] is presented in the context of reverse engineering. The method is used to recover a deadlock-free design when a set of global observations of an existing implementation is given.

Definition 5.1 (Global observations) *A global observation is a well-formed and complete trace that starts from and ends at the global initial state without passing through the local initial state of any process twice.*

We do not require that the given set of global observations is complete; i.e., all the possible traces are included or all the transitions of presumed design are executed at least once since these assumptions are too strong for reverse engineering. Actually, the classical algorithms in automata theory, such as the subset algorithm,

minimization algorithm in [14], which were proposed decades years ago, can construct the design for the first case. However, when a set of observations is given, we can assume the observation is conducted by some experienced people who know what functions of the system are of the user's concern and can recognize the start and the end of the observation; it is possible that the concerned functions are only part of all the functions of the existing system. We cannot construct any design beyond the observations. Under the following assumptions, our solution guarantees that the constructed design is deadlock-free.

- The functionality of the implementation of a protocol design is periodic; i.e., all the observations start from and end at the initial state.

- The presumed design has no other cycles when the protocol specifications are expressed as CFSM graphs except those starting from and ending at the initial state. This is because other cycles can not be recognized if the states in the graph cannot be identified; i.e., an observer cannot realize whether some state has already been repeated during an execution. In the literature, some techniques have been developed to identify the states such as the D-method based on distinguishing sequences in [13], the U-method based on UIO sequences in [24], the W-method based on characterization sets in [9], etc. However, all these methods require the knowledge of the protocol design. In reverse engineering, the implementation is a black box and no design is available, so far, there is no complete solution to the state identification problem in reverse engineering although partial solution has been proposed in [28].

- Both the protocol specifications and the network of the protocol are deterministic.

- More than one trace is observed. If only one trace can be observed then it is the correct design and no further work is necessary.

The method has two steps:

(1) Project each observation to the individual process, and construct the draft design $CD_0(O)$, in which the projections of each process start from and end at the local initial states;

(2) Apply construction rules to the draft design. There are two directions: 1) Apply $Rule_{det}$ first and then $Rule_{red}$, this direction does not change the semantics of the draft design, and some deadlock states can be removed, but the constructed design may not be deadlock-free; 2) Apply $Rule_{det}$ first and then $Rule_{neg}$. This method will add new semantics to the draft design, all the deadlock states are removed and the constructed design is deadlock free.

5.1.1 Projection of Global Observations and Initial Design Construction

In this subsection, we discuss the first step of the method. We use the same projection derivation rule as in [6]. Given a trace ρ, we can derive $proj(\rho, i)$, the projection of o on process P_i:

$$proj(\rho,i) = \begin{cases} \epsilon & \text{if } \rho = \epsilon \\ -m_{i,j}proj(\rho',i) & \text{if } \rho = -m_{i,j}\rho' \text{ for } j \in [1,n] \\ +m_{j,i}proj(\rho',i) & \text{if } \rho = +m_{j,i}\rho' \text{ for } j \in [1,n] \\ proj(\rho',i) & \text{if } \rho = -m_{l,j}\rho' \text{ or } +m_{j,l}\rho' \text{ for } l,j \in [1,n], l \neq i \end{cases}$$

$proj(\rho, i)$ reflects the sequence of events within the trace ρ that are related to process P_i. Since ρ starts from and ends at the global initial state without going over it, $proj(\rho, i)$ starts from and ends at the local initial state without going over it.

Given a set of traces $\Sigma = \{\rho_1, ..., \rho_r\}$, we can derive a set of projections on process P_i, denoted as $proj(\Sigma, i)$, where $proj(\Sigma, i) = \{pro(\rho_j, i) | j \in [1, r], proj(\rho_j \neq \epsilon)\}$. We use $proj(\Sigma, i)$ over Σ for process P_i, for $i = 1, ..., n$, to construct $CFSM_i$.

Definition 5.2 (Generated CFSM) *Given a set of projections* $proj(\Sigma, i) = \{b_1, ..., b_k\}$ *over* Σ *on process* P_i, *where* $b_l \neq \epsilon$ *for* $l = 1, ..., k$, $k \geq 1$. *The CFSM over* $proj(\Sigma, i)$ *is* $(S, M_i, s^0, \rightarrow)$ *where:*

- $S = \{s^{l,j} | l \in [1, k], j \in [1, |b_l| - 1]\} \cup \{s^0\}$;
- $s^0 \in S$ *is the initial state;*
- $M_i = \bigcup_{j=1,...,n, j \neq i}(M_{i,j} \cup M_{j,i})$;
- $\rightarrow \subset S \times E_{M_i} \times S$ *is the least relation satisfying*
  - $s^0 \xrightarrow{b_l[1]} s^{l,1}$, *for* $l \in [1, k]$;
  - $s^{l,j-1} \xrightarrow{b_l[j]} s^{l,j}$, *for* $l \in [1, k], j \in [2, |b_l| - 1]$;
  - $s^{l,j-1} \xrightarrow{b_l[j]} s^0$, *for* $l \in [1, k], j = |b_l|, j \geq 2$; *and*
  - $s^0 \xrightarrow{b_l[1]} s^0$ *for* $l \in [1, k], |b_l| = 1$.

In the following, we use $CD_0(\Sigma)$ to denote the constructed design $(L, M, \{T_i\}_{i=1}^n)$, where T_i is the $CFSM$ over $proj(\Sigma, i)$ as defined in the above definition.

5.1.2 Construction Rules $Rule_{det}$ and $Rule_{red}$

In this subsection, we review the first two construction rules, namely, $Rule_{det}$ and $Rule_{red}$ to construct protocol design in [6]. Note, these two rules are applicable to n−process protocols.

When two processes send messages concurrently, we call this phenomenon a collision. The incomplete observations can cause deadlock states or unspecified receptions only when there are collisions in the presumed design. We have the following proposition (See [6] for proof).

Proposition 5.1 *If the presumed design of the set of observations is deadlock free, unspecified reception free, and free from collisions, and the constructed design* $CD(O)$ *is deterministic, then* $CD(O)$ *is free from deadlocks and unspecified receptions.*

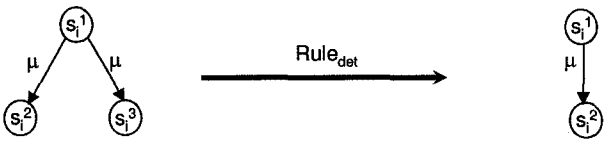


Figure 6: Demonstration to $Rule_{det}$ for Protocol Design

Let $T_i = (S_i, M_i, s_i^0, \rightarrow_i)$ be the $CFSM$ of process P_i in a given $CD_0(O)$.

$Rule_{det}$: If $\exists s_i^1, s_i^2, s_i^3 \in S_i$, $s_i^2 \neq s_i^3$, $\exists \mu \in E_M$, such that $s_i^1 \xrightarrow{\mu} s_i^2$ and $s_i^1 \xrightarrow{\mu} s_i^3$, then

(1) Remove s_i^3 from S_i;

(2) Remove $s_i^1 \xrightarrow{\mu} s_i^3$ from $\rightarrow_i$;

(3) $\forall \mu' \in E_M, s_i^4 \in S_i$, such that $s_i^3 \xrightarrow{\mu'} s_i^4$, substitute $s_i^3 \xrightarrow{\mu'} s_i^4$ by $s_i^2 \xrightarrow{\mu'} s_i^4$ in $\rightarrow_i$.

Note, this rule is $Rule_{det}$ in [6]. How this rule is applied is shown in figure 6.

$Rule_{red}$: If $\exists s_i^1, s_i^2 \in S_i$, $s_i^1 \neq s_i^2$, $\forall \mu \in E_M, s_i \in S_i$, such that $s_i^1 \xrightarrow{\mu} s_i$ iff $s_i^2 \xrightarrow{\mu} s_i$, then

(1) Remove $s_i^2 from S_i$;

(2) Remove $s_i^2 \xrightarrow{\mu} s_i$ from $\rightarrow_i$ for any $s_i \in S_i$ and $\mu \in E_M$;

(3) $\forall \mu \in E_M, s_i \in S_i$, such that $s_i \xrightarrow{\mu} s_i^2$, substitute $s_i \xrightarrow{\mu} s_i^2$ by $s_i \xrightarrow{\mu} s_i^1$ in $\rightarrow_i$.

Note, this rule is $Rule_{red}$ in [6]. How this rule is applied is shown in Figure 7.

$Rule_{det}$ can remove the deadlocks and unspecified receptions caused by nondeterminism and $Rule_{red}$ can help to reduce the design. Meanwhile, these two rules do not introduce any new deadlocks or unspecified receptions.

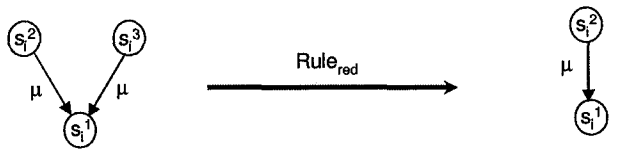


Figure 7: Demonstration to $Rule_{red}$ for Protocol Design

Theorem 5.2 *$Rule_{det}$ does not introduce any new deadlock or unspecified reception to tree-like constructed designs.*

Theorem 5.3 *$Rule_{red}$ does not contribute to the removal of any deadlock or unspecified reception, nor does it introduce new errors to the constructed design.*

Besides the properties we discussed, $Rule_{det}$ preserves trace equivalence. First, we given some useful definitions in the set theory.

Definition 5.3

- *A* **relation** *is a set of ordered pairs.*
- *A relation R in a set X is* **reflexive** *if $(\forall x \in X)(x, x) \in R$.*
- *A relation R in a set X is* **antisymmetric** *if $(x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$.*
- *A relation R in a set X is* **transitive** *if $(x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$.*
- *A relation R in a set X is an* **equivalence relation** *if R is reflexive, symmetric and transitive.*
- *A relation R in a set X is a* **partial order** *if R is reflexive, antisymmetric, and transitive.*
- *A* **partially ordered set** *is an ordered pair $(X, \preceq)$ in which X is a set and $\preceq$ is a partial order in X.*

- *If* $\forall x, y \in X, (x \preceq y) \vee (y \preceq x)$, *then* $\preceq$ *is called a* **totally ordered set** *or a* **chain**.

Definition 5.4 (Trace Equivalence) *[29] Let* $Tr(p)$ *denote the set of traces of* p. *Two processes* p *and* q *are trace equivalent, notation* $p =_{Tr} q$, *if* $Tr(p) = Tr(q)$. *In trace semantics, two processes are identified iff they are trace equivalent.*

Theorem 5.4 *Given a tree-like CFSM,* $Rule_{det}$ *preserves trace equivalence.*

PROOF: We first prove one transformation of $Rule_{det}$ and multiple transformations can be induced. Let $\forall \rho \in Tr(T_i)$, $\rho = \mu_1\mu_2...\mu_r$, and $\mathcal{F}_1$ is the transformation function of $Rule_{det}$. We have three conditions: 1) $\exists \mu_l, l \in [1, r]$, such that $\mu_l = \mu$ and ρ passes s_i^1 and s_i^3; 2) $\exists \mu_l, l \in [1, r]$, such that $\mu_l = \mu$ but ρ does not pass both s_i^1 and s_i^3; i.e., ρ passes s_i^1 and s_i^2; 3) $\nexists \mu_l, l \in [1, r]$, such that $\mu_l = \mu$.

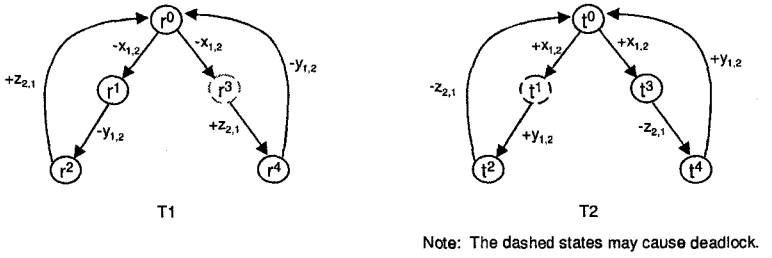
For case 1), though the transition $s_i^1 \xrightarrow{\mu} s_i^3$ is removed, this transition is replaced by $s_i^1 \xrightarrow{\mu} s_i^2$ and all the transitions starting from s_i^3 has been moved to s_i^2. Also because the graph is tree-like, no other transition can enter s_i^3 and the trace passing s_i^3 but not passing s_i^1 does not exist. Hence, ρ is still in the set of $Tr(\mathcal{F}_1(T_i))$.

For case 2) and 3), ρ is not changed at all, so ρ is still in the set $Tr(\mathcal{F}_1(T_i))$.

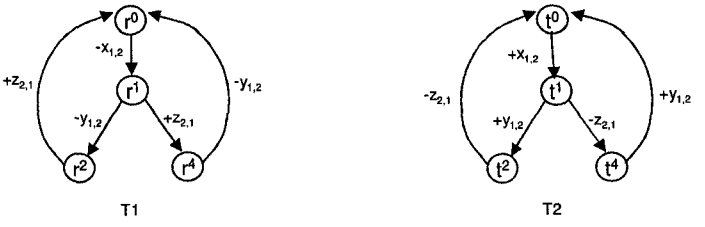
Thus, we have $Tr(T_i) \subseteq Tr(\mathcal{F}_1(T_i))$. On the other hand, since $Rule_{det}$ removes the transitions with consideration of replacement of affected traces, no new trace is created and every trace in $Tr(\mathcal{F}_1(T_i))$ should also be in $Tr(T_i)$. Hence, we also have $Tr(\mathcal{F}_1(T_i)) \subseteq Tr(T_i)$. Therefore, $Tr(T_i) = Tr(\mathcal{F}_1(T_i))$. $Rule_{det}$ preserves the trace equivalence. □

5.2 Examples and Deadlocks in the Constructed Design

In this subsection, we give some examples to show that constructed designs may contain deadlock state errors and unspecified reception errors and how $Rule_{det}$ can remove the errors of Example 5.1-5.2, while leaving some errors of Example 5.3-5.4 unremoved, which can be removed by using $Rule_{neg}$.



(A) Construction Design $CD_0(O)$ of Example 5.1



(B) Construction Design $Rule_{det}(CD_0(O))$ of Example 5.1

Figure 8: The Constructed Designs of Example 5.1

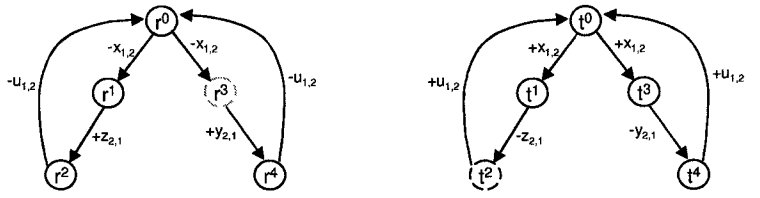
Example 5.1 (from [6]) Consider a protocol with two processes and two channels between them. We have a set of traces as:

$$O = \{-x_{1,2} + x_{1,2} - z_{2,1} + z_{2,1} - y_{1,2} + y_{1,2}, -x_{1,2} + x_{1,2} - y_{1,2} + y_{1,2} - z_{2,1} + z_{2,1}\}$$

Figure 8 (A) shows the constructed design $CD_0(O)$. T_1 and T_2 in $CD_0(O)$ are the $CFSMs$ over $proj(O,1)$ and $proj(O,2)$ respectively. Let us look at the following scenario: T_1 sends $x_{1,2}$ and enters r^3, then T_2 receives $x_{1,2}$ and enters t^1. Now it reaches a global state

$$\begin{pmatrix} r^3 & \epsilon \\ \epsilon & t^1 \end{pmatrix}$$

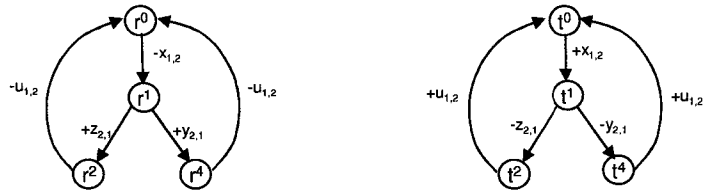
in the network of this design, and it is a deadlock state because both T_1 and T_2 are expecting to receiving messages while both channels are empty. Let us apply $Rule_{det}$ to $CD_0(O)$ and the deadlock states are removed. The specification after the construction is shown in Figure 8(B).



T1 T2

Note: The dashed states may cause unspecified receptions.

(A) Construction Design $CD_0(O)$ of Example 5.2



T1 T2

(B) Construction Design $Rule_{det}$ ($CD_0(O)$) of Example 5.2

T1 T2

(C) Construction Design $Rule_{red}$ ($Rule_{det}$ ($CD_0(O)$)) of Example 5.2

Figure 9: The Constructed Designs of Example 5.2

Example 5.2 (from [6]) Consider a protocol with two processes and two channels between them. We have a set of traces as:

$$O = \{-x_{1,2} + x_{1,2} - z_{2,1} + z_{2,1} - u_{2,1} + u_{2,1}, -x_{1,2} + x_{1,2} - y_{2,1} + y_{2,1} - u_{1,2} + u_{1,2}\}$$

Figure 9 (A) shows the constructed design $CD_0(O)$. T_1 and T_2 in $CD_0(O)$ are the $CFSMs$ over $proj(O,1)$ and $proj(O,2)$ respectively. Let us look at the following scenario: T_1 sends $x_{1,2}$ and enters r^3, and T_2 receives $x_{1,2}$ and enters t^1, then sends

$z_{2,1}$ and enters t^2. Now it reaches a global state

$$\begin{pmatrix} r^3 & \epsilon \\ z_{2,1} & t^2 \end{pmatrix}$$

in the network of this design, and it is an unspecified reception state because T_1 is expecting to receive $y_{2,1}$ while only $z_{2,1}$ is available in the channel. This example is suitable for applying $Rule_{det}$ and $Rule_{red}$. After applying these two rules, the deadlock states are removed and the specification is minimized. The specification after the construction is shown in Figure 9(B) and (C).

Example 5.3 (from [6]) Consider a protocol with two processes and two channels between them. We have a set of traces as:

$$O = \{-x_{1,2} + x_{1,2} - u_{1,2} - y_{2,1} + u_{1,2} + y_{2,1} - z_{2,1} + z_{2,1},$$

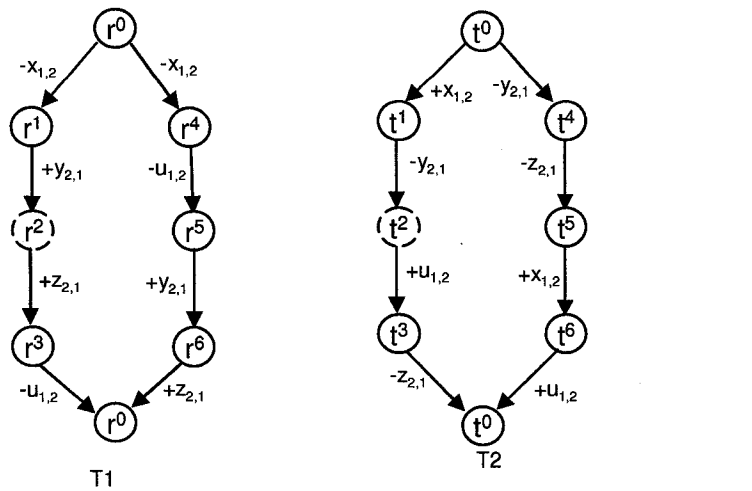
$$-y_{2,1} - x_{1,2} + y_{2,1} - z_{2,1} + x_{1,2} + z_{2,1} - u_{1,2} + u_{1,2}\}$$

Figure 10 shows the constructed design $CD_0(O)$. T_1 and T_2 in $CD_0(O)$ are the $CFSMs$ over $proj(O,1)$ and $proj(O,2)$ respectively. Let us look at the following scenario: T_1 sends $x_{1,2}$ and enters r^1, then T_2 receives $x_{1,2}$ and enters t^1, after that, T_2 sends $y_{2,1}$ and enters t^2, then T_1 receives $y_{2,1}$ and enters r^2. Now it reaches a global state

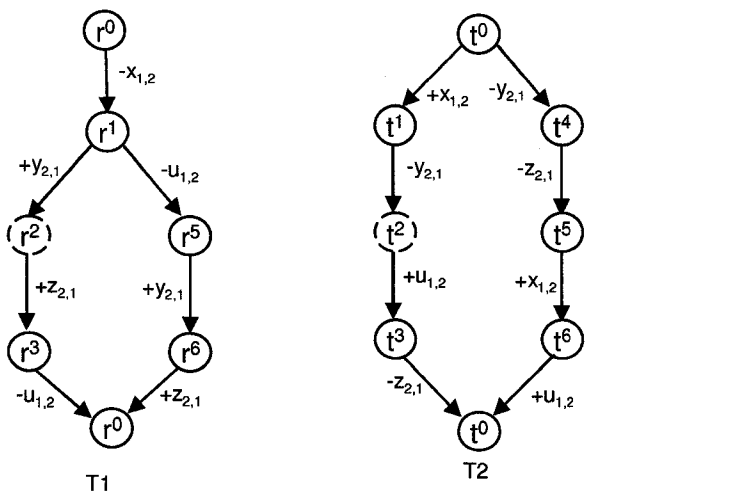
$$\begin{pmatrix} r^2 & \epsilon \\ \epsilon & t^2 \end{pmatrix}$$

in the network of this design, and it is a deadlock state because both T_1 and T_2 are expecting to receive messages while both channels are empty. However, this example is a little complex, and after $Rule_{det}$ is applied, we can still observe that potential deadlock state in Figure 10(B).

Example 5.4 Consider a protocol with two processes and two channels between them.

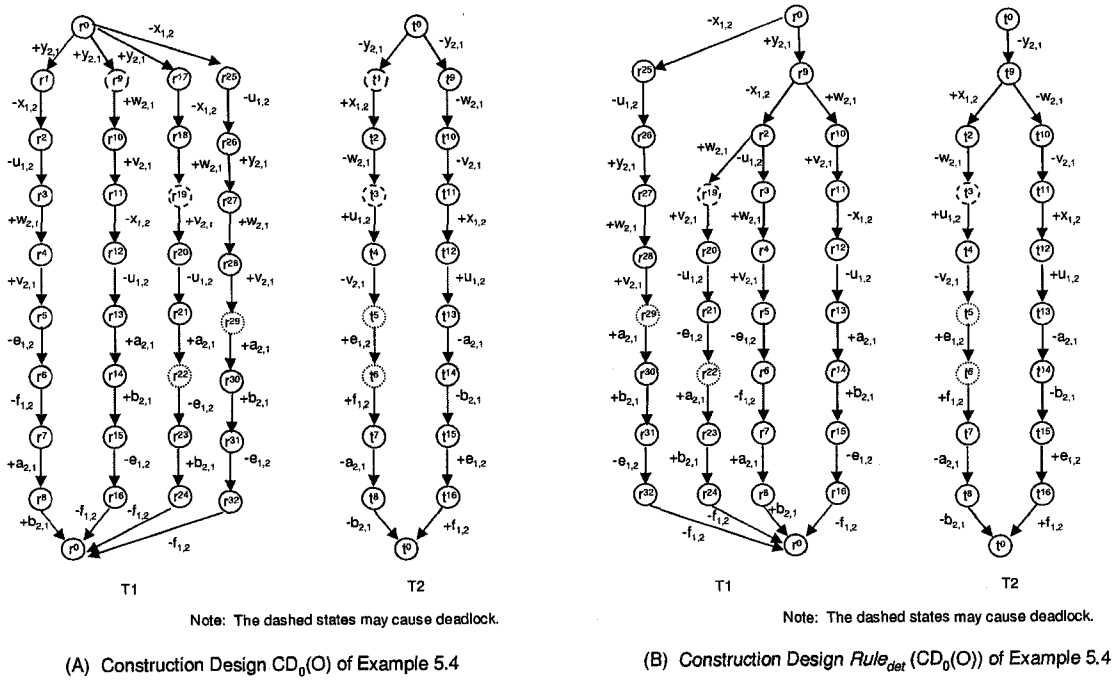


(A) Construction Design $CD_0(O)$ of Example 5.3



(B) Construction Design $Rule_{det}(CD_0(O))$ of Example 5.3

Figure 10: The Constructed Designs of Example 5.3



(A) Construction Design $CD_0(O)$ of Example 5.4

(B) Construction Design $Rule_{det}(CD_0(O))$ of Example 5.4

Figure 11: The Constructed Designs of Example 5.4

We have a set of traces as:

$$O = \{-y_{2,1} + y_{2,1} - x_{1,2} + x_{1,2} - w_{2,1} - u_{1,2} + u_{1,2} + w_{2,1}$$

$$-v_{2,1} + v_{2,1} - e_{1,2} + e_{1,2} - f_{1,2} + f_{1,2} - a_{2,1} + a_{2,1} - b_{2,1} + b_{2,1},$$

$$-y_{2,1} + y_{2,1} - w_{2,1} + w_{2,1} - v_{2,1} + v_{2,1} - x_{1,2} + x_{1,2}$$

$$-u_{1,2} + u_{1,2} - a_{2,1} + a_{2,1} - b_{2,1} + b_{2,1} - e_{1,2} + e_{1,2} - f_{1,2} + f_{1,2},$$

$$-y_{2,1} + y_{2,1} - x_{1,2} - w_{2,1} + w_{2,1} - v_{2,1} + v_{2,1} + x_{1,2}$$

$$-u_{1,2} + u_{1,2} - e_{1,2} - a_{2,1} + a_{2,1} - b_{2,1} + b_{2,1} + e_{1,2} - f_{1,2} + f_{1,2},$$

$$-x_{1,2} - y_{2,1} - u_{1,2} + y_{2,1} - w_{2,1} + w_{2,1} - v_{2,1} + v_{2,1}$$

$$+x_{1,2} + u_{1,2} - e_{1,2} - a_{2,1} + a_{2,1} - b_{2,1} + b_{2,1} + e_{1,2} - f_{1,2} + f_{1,2}\}$$

Figure 11(A) shows the constructed design $CD_0(O)$. T_1 and T_2 in $CD_0(O)$ are the $CFSMs$ over $proj(O,1)$ and $proj(O,2)$ respectively. It contains four deadlock states in the network of this design, namely,

$$\begin{pmatrix} r^9 & \epsilon \\ \epsilon & t^1 \end{pmatrix} \begin{pmatrix} r^{19} & \epsilon \\ \epsilon & t^3 \end{pmatrix} \begin{pmatrix} r^{29} & \epsilon \\ \epsilon & t^5 \end{pmatrix} \begin{pmatrix} r^{22} & \epsilon \\ \epsilon & t^6 \end{pmatrix}.$$

The first deadlock state is due to nondeterminism, and the rest are due to the rules of composing the network of a protocol, which allow at some global state, the next transition can be any available transition from the protocol specifications. With the rules, more traces than the given set of observations may be created. For Example 5.4, the application of $Rule_{det}$ removes the first deadlock state as shown in Figure 11(B), but the other two deadlock states remain.

The reasons of occurrences of the deadlock states and unspecified receptions include nondeterminism as shown in Example 5.1-5.2 and incompleteness of observations as shown in Example 5.3-5.4. In the next subsection, we give construction rule $Rule'_{neg}$ to remove the deadlock states caused by incomplete observations for the 2-process protocol.

5.3 Improvement on Construction Rule $Rule'_{neg}$

From the point of view of software engineering, the desirable properties of a protocol design include: 1) it meets all the requirements of the customers; 2) it does not manifest any logical error; 3) the design is minimum so that after design phase, programming and testing can be more efficient.

As we mentioned in the previous subsection 5.2, we introduce the improved $Rule'_{neg}$ to remove the deadlock state caused by incomplete observations while $Rule'_{neg}$ adds no more states or transitions to the design than $Rule_{neg}$. Note, both $Rule_{neg}$ and $Rule'_{neg}$ can only be applied to 2-process protocols. Also, without special indication, $Rule'_{neg}$ means the improved $Rule'_{neg}$, not the original $Rule_{neg}$.

First, we give the definition of negation of a trace.

Definition 5.5 (Negation of Traces) *Let σ be a path, the negation of σ, denoted by $\bar{\sigma}$, is defined as*

$$\begin{cases} \epsilon & \sigma = \epsilon \\ -m\bar{\sigma'} & \sigma = +m\sigma' \\ +m\bar{\sigma'} & \sigma = -m\sigma' \end{cases}$$

Let $pos(\omega)$ denote the sequence of all the events of receiving messages in ω in the same order, and $neg(\omega)$ the sequence of all the vents of sending messages in ω in the same order, we have

$$pos(\omega)\begin{cases} \epsilon & \omega = \epsilon \\ +m\,pos(\omega') & \omega = +m\omega' \\ pos(\omega') & \omega = -m\omega' \end{cases}$$

$$neg(\omega)\begin{cases} \epsilon & \omega = \epsilon \\ -m\,neg(\omega') & \omega = -m\omega' \\ neg(\omega') & \omega = +m\omega' \end{cases}$$

Definition 5.6 *Let P be a 2-process protocol. σ is a path from the local initial state in T_1, and σ' is a path from the local initial state in T_2. σ and σ' can form a trace in the network of P if we can execute all the transitions in σ and σ' without encountering any deadlock state or unspecified reception.*

We apply $Rule'_{neg}$ to tree-like CFSMs. For each tree-like CFSM, the sequence of events between s^0 and s is unique without passing s^0 twice. We use $neg(s)$ and $pos(s)$ to denote the sequence of sending and receiving events respectively from s^0 to s. Note, CD_0 and $Rule_{det}(CD_0(O))$ contain only tree-like CFSMs.

Let P be a given deterministic protocol with two CFSMs T_1 and T_2. We assume there is no unspecified reception error in the constructed protocol design. $Rule'_{neg}$ aims at removing the error of deadlock states.

$Rule'_{neg}$:

(1) In T_2, while $\exists s_2^1 \in S_2, s_2^0 \xrightarrow{\sigma} s_2^1$, but for all $m_{2,1} \in M_{2,1}$ $\nexists s_2^2 \in S_2$ such that $s_2^1 \xrightarrow{-m_{2,1}} s_2^2$, do:

in T_1, if $\exists s_1^1 \in S_1$, $s_1^0 \xrightarrow{\sigma'} s_1^1$, such that $pos(s_1^1) = pos(\bar{s_2^1})$, $neg(s_1^1) = neg(\bar{s_2^1})$, and σ and σ' can form a trace in the network of P, but $\nexists s_1^2$ for all $m_{1,2} \in M_{1,2}$ such that $s_1^1 \xrightarrow{-m_{1,2}} s_1^2$, then

(a) select one path ς from s_1^1 to the local initial state s_1^0;

(b) append $\bar{\varsigma}$ to s_2^1 in T_2; i.e., add the new states of $\bar{\varsigma}$ into S_2, and add the new transitions of $\bar{\varsigma}$ into $\rightarrow_2$;

(2) If $\exists s^1, s^2 \in S_2 (s^1 \neq s^2)$ such that $pos(s^1) = pos(s^2)$ and $neg(s^1) = neg(s^2)$, then

(a) remove s^2 from S_2;

(b) $\forall \mu \in E_M$, $s \in S_2$, such that $s^2 \xrightarrow{\mu} s$, substitute $s^2 \xrightarrow{\mu} s$ by $s^1 \xrightarrow{\mu} s$ in $\rightarrow_2$;

(c) $\forall \mu \in E_M$, $s \in S_2$, such that $s \xrightarrow{\mu} s^2$, substitute $s \xrightarrow{\mu} s^2$ by $s \xrightarrow{\mu} s^1$ in $\rightarrow_2$.

A straightforward algorithm that applies the first step of $Rule'_{neg}$ can be written by adapting Depth-First-Search Algorithm (DFS). For a 2-process protocol P, suppose u is the maximum number of states among CFSMs, and v is the maximum number of transitions among CFSMs, the time complexity of the adapted algorithm is $O((u+v)^2)$. For step 2, in [6], an algorithm is proposed with the time complexity $O(uv)$.

$Rule'_{neg}$ is applied to the design after the application of $Rule_{det}$, so it works on a deterministic design with the tree-like structure, and the structure is preserved after the execution of step (2). Step (1) of $Rule'_{neg}$ is to find the potential deadlock states and append a path with a sending event as its first event to each potential deadlock state. Since the sending transitions can always be executed, the potential deadlock states will not be deadlock states any more. Also, it is sufficient to check only one

protocol specification of P, that is, either T_1 or T_2, and all its potential deadlock states are removed. This is because when no state in one machine is a potential deadlock state, the global states containing these states as local states cannot be deadlock states.

Note that the application of $Rule'_{neg}$ adds new semantics to the specification of P, however, since a good design very often includes the mirroring of the other protocol specification in one protocol specification, the design constructed by this rule is quite reasonable.

Also, $pos(s_1^1) = pos(\bar{s_2^1})$ and $neg(s_1^1) = neg(\bar{s_2^1})$ does not necessarily mean the path reaching s_2^1 and the path reaching s_1^1 can form a trace in the network of P, that is, there exists a trace in the network of P such that the projections of this trace on P_1 and P_2 are exactly those two paths. To avoid adding undesired transitions, we add the condition that σ in T_2 and σ' in T_1 can form a trace in the network of P and reach some global state which contains both s_2^1 and s_1^1 as its local states. For example, we have two paths, which are the fragments of two observations: $\rho_1 = -x_{1,2} - y_{2,1} + y_{2,1} - z_{2,1} + z_{2,1} - u_{1,2} + x_{1,2} + u_{1,2}$, and $\rho_2 = -x_{1,2} + x_{1,2} - y_{2,1} + y_{2,1} - u_{1,2} + u_{1,2} - z_{2,1} + z_{2,1}$. Project these two observations to T_1 and T_2, we have $\omega_1^1 = -x_{1,2} + y_{2,1} + z_{2,1} - u_{1,2}$ and $\omega_1^2 = -x_{1,2} + y_{2,1} - u_{1,2} + z_{2,1}$ in T_1, and $\omega_2^1 = -y_{2,1} - z_{2,1} + x_{1,2} + u_{1,2}$ and $\omega_2^2 = +x_{1,2} - y_{2,1} + u_{1,2} - z_{2,1}$ in T_2. Thus, we can see $pos(\omega_1^1) = pos(\bar{\omega_2^2}) = +y_{2,1} + z_{2,1}$ and $neg(\omega_1^1) = neg(\bar{\omega_2^2}) = -x_{1,2} - u_{1,2}$, but these two paths cannot form a trace in the network of P. Furthermore, we can see none of these states in the P is a deadlock state so far.

Since T_2 is deterministic and tree-like, σ is unique. However, the paths that can match σ in T_1 might be more than one. Since we assume there is no unspecified reception, there are two cases for those matching paths: 1) paths that match σ and do not cause deadlock at s_2^1; 2) paths that match σ but cause deadlock at s_2^1. For case 1), since there is no deadlock and unspecified reception error, it means there exists a sending transition among next transitions at s_1^1 in T_1. For case 2), since

there is a deadlock but not unspecified reception error, it means the next transitions at s_1^1 are all receiving transitions. If we choose one of the paths starting from s_1^1 and ending at the initial state s_1^0 in the latter case, whose first event is the label of a receiving transition, appending the negation of the path to state s_2^1, the potential deadlock will be removed since a sending transition is always executable. Because of a similar reason, we can see although more than one state in T_1 might cause deadlock states with s_2^1, we only need to consider one such state and append the negation of one of its followed paths to the initial state to s_2^1, all potential deadlock states are removed.

Proposition 5.5 *For any constructed design P with only tree-like deterministic CFSMs, $\exists s_2^1 \in S_2$, $s_2^0 \xrightarrow{\sigma} s_2^1$, $\forall m_{2,1} \in M_{2,1}$, $\nexists s_2^2 \in S_2$ such that $s_2^1 \xrightarrow{-m_{2,1}} s_2^2$. If there is a deadlock state in the network of P, σ' exists in T_1 such that σ and σ' can form a trace in the network of P.*

PROOF: From the definition of deadlock states, we know when the deadlock occurs, both of the channels are empty. It means every sending transition has the matching receiving transitions, and vice versa. Suppose s^1 is a deadlock state containing s_2^1 as its local state of process P_2, it means there must exist σ' in T_1 such that σ' includes the events that receive all the messages sent to the channel $(2,1)$ by the events in σ and meanwhile σ includes the events that receive all the messages sent to the channel $(1,2)$ by the events in σ'. Thus, σ and σ' form a trace in the network of P. □

Proposition 5.6 *For any constructed design P with only tree-like CFSMs, the design after the application of $Rule'_{neg}$ is deadlock free.*

PROOF: When we apply $Rule'_{neg}$ on T_2, all the risky states, i.e., all the states that have no outgoing sending transitions, will be examined. The risky states may be real deadlock states, and may be not. There are three cases: 1) s_2^1 cannot find a

matching state s_1^1 in T_1; 2) there are matching states s_1^1 for s_2^1 but for each s_1^1 there is at least one outgoing sending transition at s_1^1; 3) s_2^1 find a matching state s_1^1 and there is no outgoing sending transition at s_1^1, we say s_2^1 is a potential deadlock state.

For case 1), from Proposition 5.6, we know s_2^1 is not a deadlock state.

For case 2), because a sending transition is always executable, the global state containing both s_1^1 and s_2^1 cannot be a deadlock state.

For case 3), $Rule'_{neg}$ deals with the potential deadlock states. The application of $Rule'_{neg}$ will add a sending transition to every potential deadlock state, thus, the potential deadlock is removed. Furthermore, the appended path is the negation of the path in T_1 and it means the appended path will not introduce any new deadlock state. □

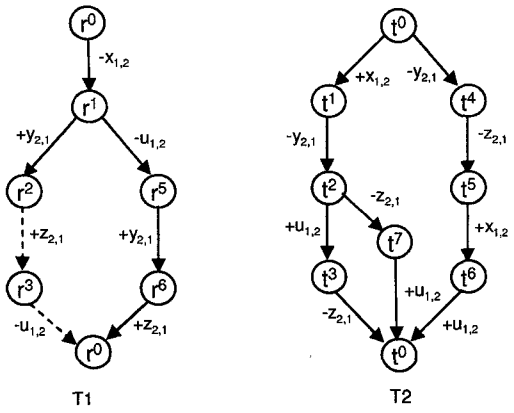
Proposition 5.7 *$Rule'_{neg}$ adds no more states and transitions than the original $Rule_{neg}$.*

PROOF: The original $Rule_{neg}$ adds the entire negation of T_1 onto T_2, while the goal of the improved $Rule'_{neg}$ is to add the transitions that really help to remove deadlock states, which are part of the negation of T_1. In the worst case, the improved $Rule'_{neg}$ adds the entire negation of T_1 as the original $Rule_{neg}$. □

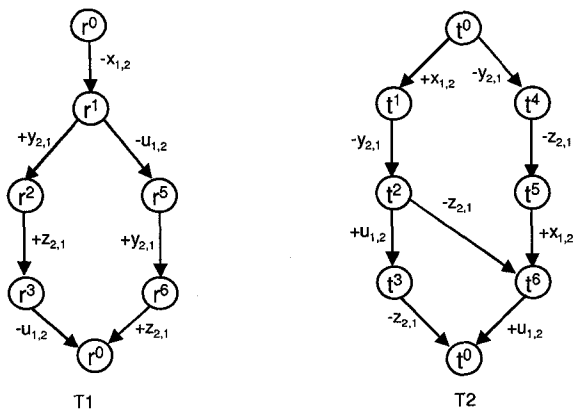
Now we can use $Rule'_{neg}$ to remove the deadlock in the Example 5.3. Figure 12 shows that the application of $Rule'_{neg}$ has two steps: first, it identifies t^2 as a potential deadlock state and appends the negation of path $+z_{2,1} - u_{1,2}$ to t^2; second, it merges state t^6 and state t^7 because $pos(t^6) = pos(t^7)$ and $neg(t^6) = neg(t^7)$.

Figure 13 shows how $Rule'_{neg}$ works on Example 5.4. The application of this example is a bit different from Example 5.3 in a way that the first step of $Rule'_{neg}$ has been applied twice before moving to the second step. This is because we put a "while"-statement in the rule condition.

Unlike $Rule_{det}$ and $Rule_{red}$, $Rule'_{neg}$ can only be applied to 2-process protocols. This is because for $n-$process protocols (when $n \geq 3$), the specification of one



(A) Construction Design $Rule_{neg}'(Rule_{det}(CD_0(O)))$ of Example 5.3 – Step 1



(B) Construction Design $Rule_{neg}'(Rule_{det}(CD_0(O)))$ of Example 5.3 – Step 2

Figure 12: The Constructed Designs of Example 5.3 Using the Improved $Rule'_{neg}$

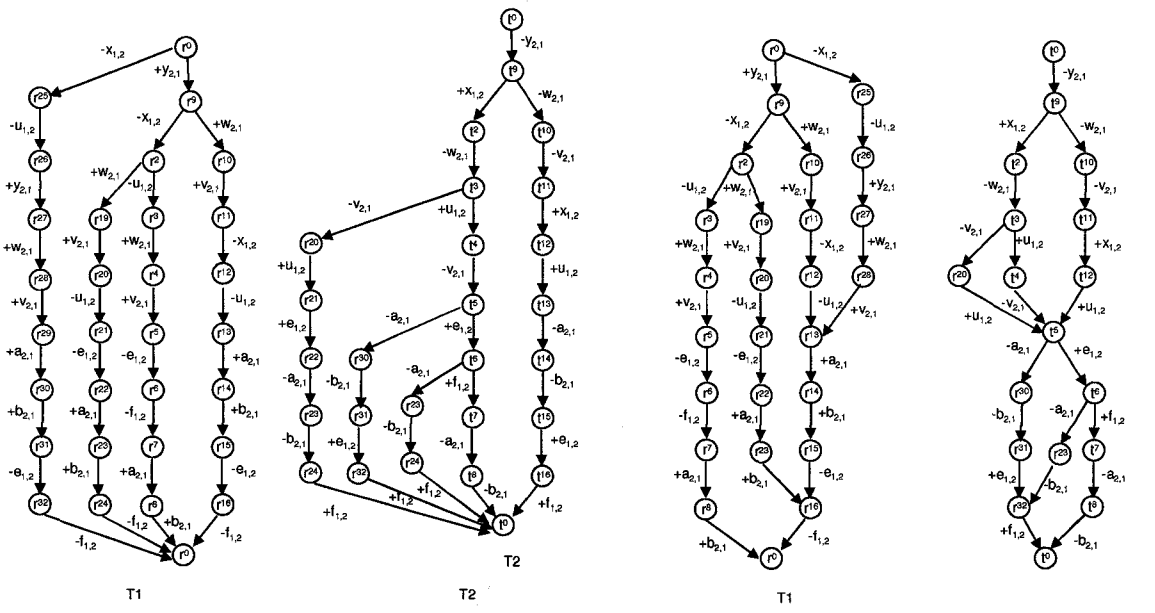


Figure 13: The Constructed Designs of Example 5.4 Using the Improved $Rule'_{neg}$

process involves the interaction with other processes (usually more than one process), and the mirroring of it will also involve more than one process, which will make the semantics of the mirroring specification ambiguous. Furthermore, $Rule'_{neg}$ assumes there is no unspecified reception error and aims at removing all the deadlock states without composition of the network of P.

5.4 Summary and Comparison

The work in this section is inspired by the work in [6] and now we compare them. The idea of removing the deadlock states without the composition of network of P is exciting because the composition of the network of P may cause state explosion problem. Our $Rule'_{neg}$ improves the original $Rule_{neg}$ in the sense that it removes all the deadlock states with less augmentation of transitions. Disadvantage of the improved $Rule'_{neg}$ comparing to the original one is that the complexity of the algorithm is increased because of the selection of relevant paths while the original rule just copies the entire negation of T_1 on to T_2.

Figure 14 shows the constructed design using the original $Rule_{neg}$ in [6]. We can see the added path $+u_{1,2} - y_{2,1}$ from t^1 to t^3 has no contribution to the removal of deadlock, which is not added as shown in Figure 12. For example 5.4, Figure 15 shows the constructed design using the original $Rule_{neg}$. Comparing it with the constructed design using the improved $Rule'_{neg}$ in Figure 13, we can see 3 more states and 5 more transitions are added, similarly, they do not contribute to the removal of the deadlock states.

Comparing $Rule'_{neg}$ with $Rule_{neg}$ for these two examples, we evaluate the reduction efficiency as in Table 1.

Note that $Rule'_{neg}$ can work independently without applying $Rule_{det}$ before it, that is, $Rule'_{neg}$ can apply on $CD_0(O)$ directly while Proposition 5.5-5.7 still hold. We apply $Rule_{det}$ before $Rule'_{neg}$ because of two reasons: 1) we want to remove the states and transitions that are unnecessary to take into account for $Rule'_{neg}$

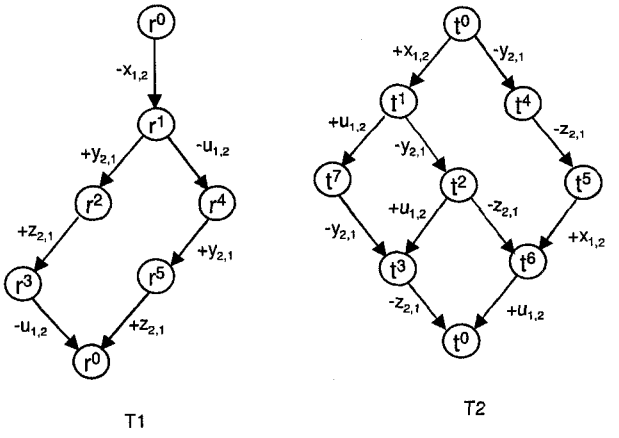


Figure 14: The Constructed Designs with Rules in [6] of Example 5.3

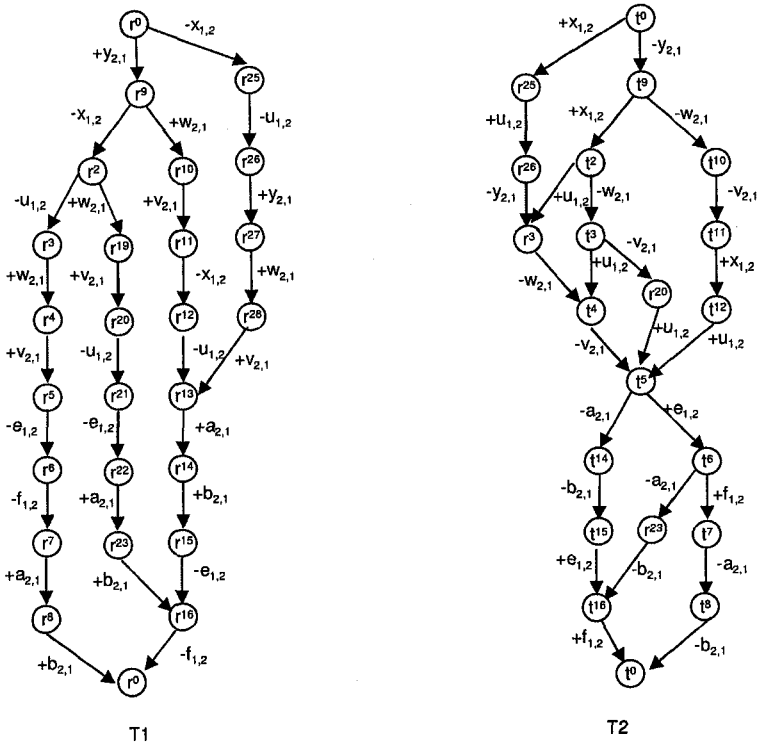


Figure 15: The Constructed Designs with Rules in [6] of Example 5.4

	$Rule_{neg}$ adds		$Rule'_{neg}$ adds		Reduction Efficiency	
	states	*transitions*	*states*	*transitions*	*states*	*transitions*
T2 in Example 3	1	3	0	1	100%	67%
T2 in Example 4	5	9	2	4	50%	56%

Table 1: The Reduction Efficiency of $Rule'_{neg}$ Compared with $Rule_{neg}$

and minimize the specification first; 2) $Rule_{det}$ may reduce the number of potential deadlock states, thus, the times of appending the negation of some paths in T_1 are reduced. For example, in Figure 11(A), t^1 in T_2 is a potential deadlock state, and $Rule'_{neg}$ will be applied to this state; but after applying $Rule_{det}$, this state is merged to t^9 and it is not a potential deadlock state any longer, which is shown in Figure 11(B), then $Rule'_{neg}$ will not be applied to this state.

The improved $Rule'_{neg}$ assumes there is no unspecified reception in the constructed design. Actually, resolving unspecified reception error is an open problem. The following example shows unspecified reception is hard to tackle with incomplete observations.

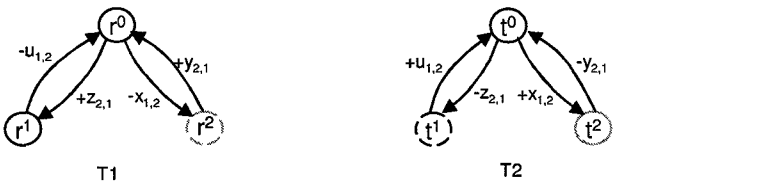
Example 5.5 (from [6]) Consider a protocol with two processes and two channels between them. We have a set of traces as:

$$O = \{-x_{1,2} + x_{1,2} - y_{2,1} + y_{2,1}, -z_{2,1} + z_{2,1} - u_{1,2} + u_{1,2}\}$$

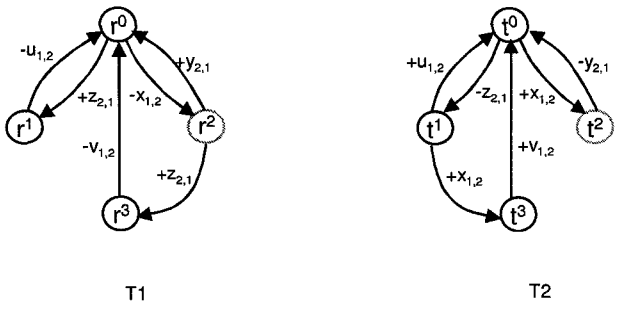
Figure 16(A) shows the constructed design $CD_0(O)$. T_1 and T_2 in $CD_0(O)$ are the $CFSMs$ over $proj(O,1)$ and $proj(O,2)$ respectively. Let us look at the following scenario: T_1 sends $x_{1,2}$ and enters r^2 while T_2 sends $z_{2,1}$ and enters t^1. Now it reaches a global state

$$\begin{pmatrix} r^2 & x_{1,2} \\ z_{2,1} & t^1 \end{pmatrix}$$

in the network of this design, and it is a deadlock state because T_1 is expecting to receive $y_{2,1}$ while channel $(2,1)$ contains only $z_{2,1}$ and T_2 is expecting to receiving



(A) Construction Design $CD_0(O)$ of Example 5.5



(B) Presumed Design of Example 5.5

Figure 16: The Constructed Designs of Example 5.5

$u_{1,2}$ while channel $(1,2)$ contains only $x_{1,2}$. We can see neither of our rules can be applied to this example to remove the deadlock states.

6 Reduction Rules During Design Verification

To verify a concurrent system using the state exploration technique, the state explosion problem is always a big concern. The solution is to reduce the possible exploration space. In the literature, the researchers usually focus on the state reduction techniques during the global reachability analysis, such as [11], etc. It is also possible to conduct the reduction on the protocol specifications right before the global reachability analysis.

6.1 Reduction Rules

In this subsection, we discuss two reduction rules. We call CFSMs specifying each process of a protocol as protocol specifications comparing to the notion of the global specification during reachability analysis. We assume the protocol specifications are minimal and deterministic. Within the framework, we give the following two rules to reduce the protocol specifications.

Reduction Rule 1: Given an n-process protocol $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$, if for any process P_i, $\exists s_i^1, s_i^2, s_i^3, s_i^4 \in S_i$, $m_{i,k}, m_{l,i} \in M_i$, where $k, l \in [1,n]$, such that

- $(s_i^1 \xrightarrow{-m_{i,k}} s_i^2) \wedge (s_i^2 \xrightarrow{+m_{l,i}} s_i^4) \wedge (s_i^1 \xrightarrow{+m_{l,i}} s_i^3) \wedge (s_i^3 \xrightarrow{-m_{i,k}} s_i^4)$, and
- $\nexists s_i' \in S, m_{u,i}, m_{i,v} \in M_i$, where $u, v \in [1,n]$, such that $(s_i' \neq s_i^1) \wedge ((s_i' \xrightarrow{+m_{u,i}} s_i^3) \vee (s_i' \xrightarrow{-m_{i,v}} s_i^3))$ or $(s_i' \neq s_i^4) \wedge ((s_i^3 \xrightarrow{+m_{u,i}} s_i') \vee (s_i^3 \xrightarrow{-m_{i,v}} s_i'))$;

then remove state s_i^3 and transitions $s_i^1 \xrightarrow{+m_{l,i}} s_i^3$ and $s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ from the specification of process P_i.

Reduction Rule 2: Given an n-process protocol $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$, if for any process P_i, $\exists s_i^1, s_i^2, s_i^3, s_i^4 \in S_i$, $m_{i,k}, m_{i,l} \in M_i$, where $k, l \in$

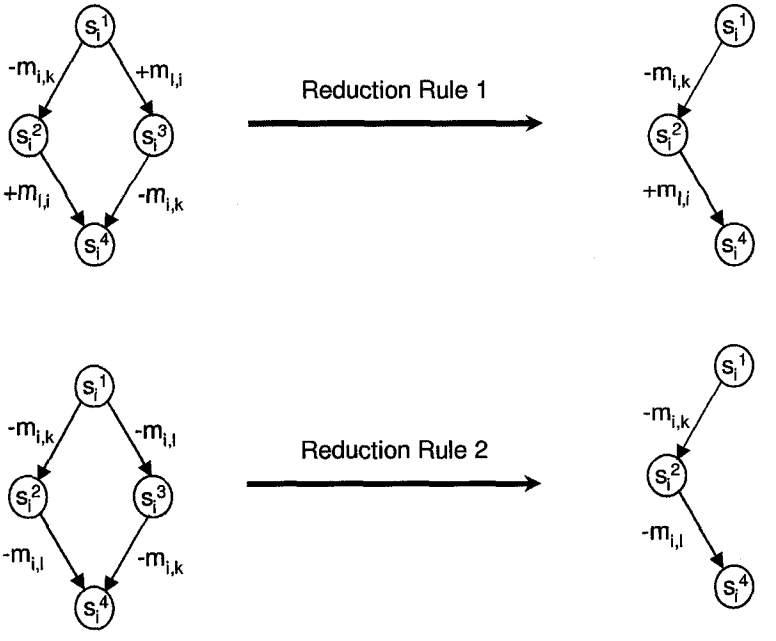


Figure 17: Demonstration to the Reduction Rules for Deadlock Verification

$[1, n], k \neq l$, such that

- $(s_i^1 \xrightarrow{-m_{i,k}} s_i^2) \wedge (s_i^2 \xrightarrow{-m_{i,l}} s_i^4) \wedge (s_i^1 \xrightarrow{-m_{i,l}} s_i^3) \wedge (s_i^3 \xrightarrow{-m_{i,k}} s_i^4)$, and
- $\nexists s_i' \in S, m_{u,i}, m_{i,v} \in M_i$, where $u, v \in [1, n]$, such that $(s_i' \neq s_i^1) \wedge ((s_i' \xrightarrow{+m_{u,i}} s_i^3) \vee (s_i' \xrightarrow{-m_{i,v}} s_i^3))$ or $(s_i' \neq s_i^4) \wedge ((s_i^3 \xrightarrow{+m_{u,i}} s_i') \vee (s_i^3 \xrightarrow{-m_{i,v}} s_i'))$;

then remove state s_i^3 and transitions $s_i^1 \xrightarrow{-m_{i,l}} s_i^3$ and $s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ from the specification of process P_i.

For *Rule 1*, the first condition requires there is a specific pattern of transitions in the protocol specification, that is, at some state, there is a choice about the execution order of a sending transition and a receiving transition: 1) execute the sending transition first, then the receiving transition; 2) execute the receiving transition first, then the sending transition. Whatever option is chosen, after the execution of these two transitions, the same state is entered eventually. The second condition

requires the states and the transitions to be removed should not have any other states or transitions involved. The sent message and the received message in *Rule 1* can be exchanged with the same process or different processes. *Rule 2* considers a similar choice pattern but both of the transitions are sending transitions. For *Rule 2*, we require that these two messages must go to different channels. If these two messages go to the same channel, because the channel is FIFO, the choice will have different impact on the content of the channel, that is, if the orders of the messages are different, the contents of the channel are different even though the messages are same. Furthermore, different states in the global specification are entered after the execution of these transitions. Thus, no state or transition can be removed. The second condition of *Rule 2* also requires the states and the transitions to be removed should not have any other states or transitions involved. How to apply *Rule 1* and *Rule 2* is shown in Figure 17.

A straightforward algorithm that applies *Rule 1* can be written by adapting Depth-First-Search Algorithm (DFS). For an $n-$process protocol P, suppose u is the maximum number of states among CFSMs, and v is the maximum number of transitions among CFSMs, the time complexity of the algorithm applying *Rule 1* is $O(n(u+v))$. Similarly, the time complexity of the algorithm applying *Rule 2* is $O(n(u+v))$.

The reduction efficiency of our rules depends on the protocol design itself. If the protocol is a synchronous protocol, then our rule is of no use. Actually, no reduction techniques are useful in that case. If the protocol has some concurrent transitions, the reduction will be useful.

Next, we prove the application of the *Rule 1* preserves the properties of error states, and channel overflow while *Rule 2* preserves the properties of error states, nonexecutable transitions, and channel overflow. First, we prove that the application of the *Rule 1* and *Rule 2* preserves the property of error states of a protocol design.

6.2 Preserving Error States Property

As we mentioned in Section 4, we use error state for a shorthand of deadlock state or unspecified reception state since at either of these states the execution gets stuck and no outgoing transition is possible.

Our rules preserve a strict error states property; i.e., during reachability analysis, the error states should remain erroneous in the global specification after the applications of the rules. A looser requirement can be: if the global specification has an error state, then after the application of the rules, the global specification still has an error state. It does not bound the error with the state. We start the proof with *Rule 1*. Intuitively, the form of this kind of pattern is an interleaving of two enabled transitions at the same state s_i^1. At this state, process P_i can either get a message from one channel, or send a message to some other channel. Both of these transitions can be executed separately. Now we check the behavior of these four states during reachability analysis. Due to the presence of the existence of sending transition $s_i^1 \xrightarrow{-m_{i,k}} s_i^2$, any global state containing state s_i^1 as its local state for the process P_i is not an error state. Because of a similar reason, neither are the states containing state s_i^3. We can check transition $s_i^2 \xrightarrow{+m_{l,i}} s_i^4$ to see if this receiving transition with label $+m_{l,i}$ may introduce an error state. Thus checking path $s_i^1 \xrightarrow{+m_{l,i}} s_i^3 \wedge s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ becomes unnecessary.

We introduce some notations: T_i denotes the CFSM of process P_i, T_i' denotes the CFSM after applying *Rule 1* once, $\mathcal{F}_1$ denotes the mapping of *Rule 1* from T_i to T_i', then we have $\mathcal{F}_1 : T_i \rightarrow T_i'$. Furthermore, we overload the notation of $\mathcal{F}_1$ to the network of P. Let N be the network of P before *Rule 1*, and $\mathcal{F}_1(N)$ is the network of P after applying *Rule 1* to some process P_i once. If we can prove one application of *Rule 1* preserves the error state property of a protocol, then multiple applications are implied.

Proposition 6.1 *If a global state in N is reachable but not an error state, then*

either this global state presents in $\mathcal{F}_1(N)$ which implies it is reachable but not an error state or this global state is removed.

PROOF: Because of the removal of s_i^3, the states that have the state s_i^3 as their local state for process P_i are removed. Then we discuss the states that remain. Suppose $s \in S$ is a global state in N that is reachable but not a error state. Since s is a reachable state, there exists a trace $\varrho \in E_M^*$ from the initial state, $\varrho = \mu_1\mu_2...\mu_r$, such that $s_0 \xrightarrow{\mu_1} s_1, ..., s_{r-1} \xrightarrow{\mu_r} s_r$ and $s_r = s$. Since s is not an error state, there is a path $\sigma \in E_M^*$, which starts from s and ends at some error state or the initial state(in this case, all the states on the trace are not error states). Thus $\exists t$, t is an integer, $\sigma = \mu_{r+1}, ..., \mu_t$, such that $s \xrightarrow{\mu_{r+1}} s_{r+1}, ..., s_{t-1} \xrightarrow{\mu_t} s_t$ and either s_t is an error state or $s_t = s_0$. Then we have $\rho = \varrho.\sigma$ and it is a trace that passes on s. Let $\mu_e = +m_{l,i}$, and $\mu_f = -m_{i,k}$, where $e, f \in [1, t], e < f$. Then μ_e and μ_f represent the labels of transition $s_i^1 \xrightarrow{+m_{l,i}} s_i^3$ and transition $s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ respectively.

Trace ρ must contain both μ_e and μ_f. We prove it by contradiction. First, we suppose ρ contains μ_e and does not contain μ_f. Since μ_f is an event to send a message and it can always be executed, and there is no other transition that can leave from state s_i^3, ρ will contain μ_f too. Second, we suppose ρ contains μ_f and does not contain μ_e. Since there is no other transition entering state s_i^3 except the transition with label μ_e, such ρ does not exist. Hence, there are two cases to be considered: 1) ρ does not contain μ_e and μ_f; 2) ρ contains both μ_e and μ_f.

For case 1), the application of *rule 1* will not affect ρ of s. Thus, s is not an error state in $\mathcal{F}_1(N)$.

For case 2), let s^{e-1} be the last global state that contains s_i^1 as its local state for process P_i, s^e and s^{f-1} the first and the last global state that contains s_i^3 as its local state for process P_i respectively, and s^f the first global state that contains s_i^4 as it local states for process P_i. Since ρ contains both μ_e and μ_f, ρ can be decomposed as $\rho = \varrho'.\mu_e.\gamma.\mu_f.\sigma'$, where $s^{e-1} \xrightarrow{\mu_e} s^e$ and $s^{f-1} \xrightarrow{\mu_f} s^f$. If s^e and s^{f-1} are the same state, $\gamma = \epsilon$. It is obvious that the transition with label $-m_{k,i}$ is in ϱ' and that the

transition with label $+m_{i,j}$ is in σ'. The transitions in γ are from other processes other than process P_i and independent from either the transition with label μ_e or the transition with label μ_f; otherwise, transitions in γ cannot be inserted between them because of the specification of process P_i. Let $\rho' = \varrho'.\mu_f.\gamma.\mu_e.\sigma'$ which is obtained by exchanging the positions of μ_e and μ_f. Because of the existence of path $s_i^1 \xrightarrow{-m_{i,j}} s_i^2 \wedge s_i^2 \xrightarrow{+m_{k,i}} s_i^4$ in the protocol specification of process P_i, trace ρ' must exist in $\mathcal{F}_1(N)$. Thus, s is not an error state in $\mathcal{F}_1(N)$. □

Proposition 6.2 *If a global state in $\mathcal{F}_1(N)$ is reachable but not an error state, then this global state in N is reachable but not an error state.*

PROOF: Trivial. Similar proof to the proof of Proposition 6.1. □

Proposition 6.3 *A global state in N is an error state iff this global state in $\mathcal{F}_1(N)$ is an error state.*

PROOF: Trivial. Similar proof to the proof of Proposition 6.1. □

Theorem 6.4 *$\mathcal{F}_1$ preserves the property of error states of a protocol P.*

PROOF: From Proposition 6.1-6.3, we know there are fewer states in $\mathcal{F}_1(N)$, but those states in N but not in $\mathcal{F}_1(N)$ are not error states. For those states in both N and $\mathcal{F}_1(N)$, we have that they are not error states in N iff they are not error states in $\mathcal{F}_1(N)$ and that they are error states in N iff they are error states in $\mathcal{F}_1(N)$. Thus, $\mathcal{F}_1$ preserves the property of error states of a protocol P. □

Similarly, we introduce some notations for *Rule 2*: $\mathcal{F}_2$ denotes the mapping of *Rule 2* from T_i to T_i', then we have $\mathcal{F}_2 : T_i \rightarrow T_i'$. Furthermore, we overload the notation of $\mathcal{F}_2$ to the network of P. N is the network of P before the application of *Rule 2*, and $\mathcal{F}_2(N)$ is the network of P after applying *Rule 2* to some process P_i once.

Theorem 6.5 *$\mathcal{F}_2$ preserves the property of error states of a protocol P.*

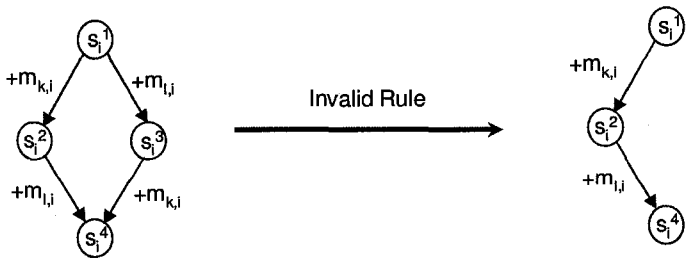


Figure 18: Demonstration to the Invalid Reduction for Error State Verification

PROOF: Similar to the proof of theorem 6.4. □

Now, we show the following rule is not valid in terms of preserving error states. How the rule works is shown in Figure 18.

Invalid Rule: Given an n-process protocol $P = (L, \{M_{i,j}|(i,j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$, for any process P_i, $\exists s_i^1, s_i^2, s_i^3, s_i^4 \in S_i$, $m_{k,i}, m_{l,i} \in M_i$, where $k, l \in [1,n], k \neq l$, if

- $(s_i^1 \xrightarrow{+m_{k,i}} s_i^2) \wedge (s_i^2 \xrightarrow{+m_{l,i}} s_i^4) \wedge (s_i^1 \xrightarrow{+m_{l,i}} s_i^3) \wedge (s_i^3 \xrightarrow{+m_{k,i}} s_i^4)$, and
- $\nexists s_i' \in S, m_{u,i}, m_{i,v} \in M_i$, where $u, v \in [1,n]$, such that $(s_i' \neq s_i^1) \wedge ((s_i' \xrightarrow{+m_{u,i}} s_i^3) \vee (s_i' \xrightarrow{-m_{i,v}} s_i^3))$ or $(s_i' \neq s_i^4) \wedge ((s_i^3 \xrightarrow{+m_{u,i}} s_i') \vee (s_i^3 \xrightarrow{-m_{i,v}} s_i'))$;

then remove the state s_i^3 and the transitions $s_i^1 \xrightarrow{+m_{k,i}} s_i^3$ and $s_i^3 \xrightarrow{+m_{j,i}} s_i^4$ from the specification of process P_i.

We give a counterexample in Figure 19 to show this rule is invalid. It is a 3-process protocol. In the reachability analysis, the global states containing s_3^1 are not error states since the receiving transition $s_3^1 \xrightarrow{+m_{1,3}} s_3^3$ can be executed. The global states containing s_3^3 may be error states in N if event sequence is chosen as $+m_{1,2} - m_{2,1}$ rather than $+m_{1,2} - m_{2,3} - m_{2,1}$ for T_2. But after applying *Invalid*

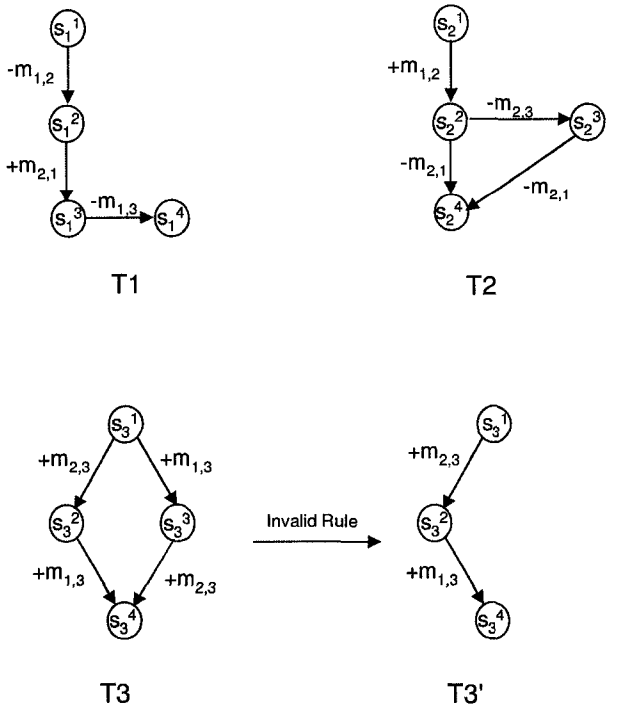


Figure 19: A Counterexample of the *Invalid Rule* for Error States Verification

Rule, the states containing s_3^1 become error states and the states containing s_3^3 are removed. Thus, the error states property of the reachability analysis are changed in terms of our strict definition of preserving error states.

6.3 Preserving Channel Overflow Property

Channel overflow occurs because the physical limitation of a channel is less than the length of content of possible traces. We want to prove after the transformation of *Rule 1* and *Rule 2*, channel overflow properties is preserved; i.e., for each channel $(i, j) \in L$ the network N of P has a channel overflow error iff $\mathcal{F}(N)$ has a channel overflow error where $\mathcal{F}$ is the mapping of *Rule 1* or *Rule 2*. We have the following theorem.

Theorem 6.6 *$\mathcal{F}_1$ preserves the property of channel overflow of a protocol P.*

PROOF:

We prove the following equivalent statement: $\forall(i,j) \in L$, there exists a trace in $\mathcal{F}_1(N)$ that have equal length of the content to or longer length of the content than that of the removed traces. The statement is equivalent to the above because if the removed traces are not the only traces that have the longest length of the content of the channel (i,j), the overflow of the channel (i,j) will still occur after those traces are removed.

We divide the set of all traces of N into two families: traces affected by the reduction $\mathcal{F}_1$ and traces not affected by the reduction $\mathcal{F}_1$. For those traces that are not affected by the reduction $\mathcal{F}_1$, the traces are remained as they were and the contents of the channels are not changed. Hence, for any $(i,j) \in L$, if the trace causes a channel overflow in N, then it also causes a channel overflow in $\mathcal{F}_1(N)$.

For the traces affected by the reduction, we have two cases: 1) traces that only contain $+m_{l,i}$; 2) traces that contain both $+m_{l,i}$ and $-m_{i,k}$. As shown in the proof of Proposition 6.1, the removed traces that only contain $-m_{i,k}$ do not exist because there are no other incoming transitions of s_i^1 except $s_i^1 \xrightarrow{+m_{l,i}} s_i^3$ according to the requirement of *rule 1.*

For case 1), traces in N that contain only $+m_{l,i}$ can be decomposed as $\rho = \varrho.+m_{l,i}.\gamma$, where ϱ is a trace from the global initial state to the last global state that contains s_i^1 as the local state for process P_i, γ is a path starting from the first global state that contains s_i^3 as the local state for process P_i but not surpass the last global state that contains s_i^3 as the local state for process P_i. γ might be empty. After $\mathcal{F}_1$, $+m_{l,i}.\gamma$ is removed from N. For channel (l,i), suppose $|\omega_{l,i}| = len$ at the last global state containing s_i^1. After the receiving transition $s_i^1 \xrightarrow{+m_{l,i}} s_i^3$, we have operation $del(\omega_{l,i})$ to channel (l,i), and $|\omega_{l,i}|$ is $len - 1$. According to the specification of T_i, we know that between $s_i^1 \xrightarrow{+m_{l,i}} s_i^3$ and $s_i^3 \xrightarrow{-m_{i,k}} s_i^4$, it is impossible to have other receiving transitions in T_i. However, γ may have sending transitions from process P_l to P_i. Let the number of such transitions be inc, then $|\omega_{l,i}| = len - 1 + inc$.

With the specification of T_i, because the sending transition with label $-m_{i,k}$ can always be executed and the transition in γ is not relevant to it, we have the trace $\rho' = \varrho.-m_{i,k}.\gamma$ in $\mathcal{F}_1(N)$. At the ending state of trace ρ', in channel (l,i), the length of the content is $len + inc$, which is obviously greater than that of ρ; for channel (i,k), the length is increased by 1 because of the sending transition $s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ and we have operation $ins(\omega_{i,k}, m_{i,k})$ to channel (i,k); for any other channel except (l,i) and (i,k), ρ has the same length as ρ'.

For case 2), as we have discussed in the proof of Proposition 4.1, for each ρ of N that contains both $+m_{l,i}$ and $-m_{i,k}$, there exists ρ' in both N and N' such that $\rho = \varrho.+m_{l,i}.\gamma.-m_{i,k}.\sigma$ and $\rho' = \varrho.-m_{i,k}.\gamma.+m_{l,i}.\sigma$, where ϱ is a trace, γ and σ are paths and both may be empty. The transitions with labels in the γ are not relevant to the transitions we removed. It is straightforward that at the ending state of ρ, $\forall(i,j) \in L$, ρ has the same length as ρ'.

Hence, the reduction $\mathcal{F}_1$ does not change the overflow property in this case. □

Theorem 6.7 *$\mathcal{F}_2$ preserves the property of Channel Overflow of a protocol P.*

PROOF: All the proof of other traces except those containing only $-m_{i,l}$ is similar to the proof of Theorem 6.7, so we only discuss the traces containing only $-m_{i,l}$ here. We decompose those traces as $\rho = \varrho. - m_{i,l}.\gamma$, where ϱ is a trace from the global initial state to the last global state that contains s_i^1 as the local state for process P_i. γ is a path from the first global state that contains s_i^3 as the local state for process P_i but not surpass the last global state that contains s_i^3 as the local state for process P_i. γ might be empty. After $\mathcal{F}_2$, $-m_{i,l}.\gamma$ is removed from N.

From the specification of T_i, we know that both transitions of sending message $-m_{i,k}$ and $-m_{i,l}$ are enabled at s_i^1. This means that the cause, which has the partial order relationship with these two transitions and should happen before these two transitions, have happened before s_i^1. Thus, the cause must be in ϱ and must not in γ. Because of the rules of composing the network of P, there exists a trace

$\rho' = \varrho. - m_{i,k}. - m_{i,l}.\gamma$ in both N and N'. At the ending state of ρ', for channel (i,k), the length is greater by 1 than that of ρ. For any other channel, the length is the same as that in ρ. Hence, channel overflow property is preserved. □

Now we show *InvalidRule* may not preserve channel overflow property. The problematic traces are those containing only $+m_{l,i}$. We can decompose each trace as $\rho = \varrho.+m_{l,i}.\gamma$, but we cannot always find a trace that has longer length of content for all channels. For example, $\rho' = \varrho$ are not suitable because γ may have some sending transitions from process P_k or process P_l to process P_i; $\rho' = \varrho. + m_{k,i}. + m_{l,i}.\gamma$ is not suitable because channel (k,i) will have less length.

6.4 Preserving Nonexecutable Transitions Property

When we discuss deadlock states or channel overflow, we talk about the properties of executable transitions of P. If the transitions are nonexecutable, actually, the reduction will not give us relief on the state explosion problem since those transitions will not be composed into the network of P. For *Rule 1*, if some transitions are removed, we will not know if they are executable by comparing T_i and the projection of N on T_i. However, *Rule 2* can preserve the property of nonexecutable transitions by inference from the remained transitions.

Proposition 6.8 *For* Rule 2, *transitions* $s_i^1 \xrightarrow{-m_{i,k}} s_i^2 \wedge s_i^2 \xrightarrow{-m_{i,l}} s_i^4$ *are executable iff transitions* $s_i^1 \xrightarrow{-m_{i,l}} s_i^3 \wedge s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ *are executable.*

PROOF:

$\Rightarrow$) Suppose transitions $s_i^1 \xrightarrow{-m_{i,k}} s_i^2 \wedge s_i^2 \xrightarrow{-m_{i,l}} s_i^4$ are executable, then at least one global state containing s_i^1 as its local state for the process P_i can be reached. $s_i^1 \xrightarrow{-m_{i,l}} s_i^3 \wedge s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ are sending transitions, so they can always be executed whenever that global state is reached.

$\Leftarrow$) Similar proof as the other direction. □

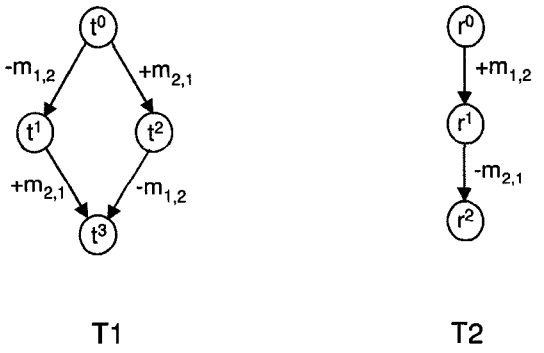


Figure 20: An Example to Show *Rule 1* Does Not Preserve the Property of Nonexecutable Transitions

Theorem 6.9 *$\mathcal{F}_2$ preserves the property of nonexecutable transitions of a protocol P.*

PROOF: From the above proposition, during the global analysis, if we find $s_i^1 \xrightarrow{-m_{i,k}} s_i^2 \wedge s_i^2 \xrightarrow{-m_{i,l}} s_i^4$ are executable, so are $s_i^1 \xrightarrow{-m_{i,l}} s_i^3 \wedge s_i^3 \xrightarrow{-m_{i,k}} s_i^4$; otherwise, $s_i^1 \xrightarrow{-m_{i,l}} s_i^3 \wedge s_i^3 \xrightarrow{-m_{i,k}} s_i^4$ are nonexecutable. □

As far as nonexecutable transitions are concerned, *Rule 1* and *Invalid Rule* may not preserve the property since there always exists a chance that the receiving transition are nonexecutable. An example in Figure 20 shows *Rule 1* does not preserve the property of nonexecutable transitions. For T_2, it must receive a message from T_1 before it can send a message to T_1. Hence, the transitions $t^0 \xrightarrow{+m_{2,1}} t^2 \wedge t^2 \xrightarrow{-m_{1,2}} t^4$ are nonexecutable. If they are removed, there is no way to recover this information.

6.5 Summary and Examples

Now we summarize the reduction rules and their properties preservation of deadlock states, channel overflow and nonexecutable transitions in Table 2.

Here we give a two-process protocol design which has neither deadlock state nor nonexecutable transitions and show how the *Rule 1* works. Figure 21 is the CFSMs

	Rule 1	*Rule 2*	*Invalid Rule*
Deadlock States	√	√	×
Unspecified Receptions	√	√	√
Channel Overflow	√	√	×
Nonexecutable Transitions	×	√	×

Table 2: The Properties Preservation Table of Reduction Rules

	Original number		*Reduced number*		*Reduction Efficiency*	
	states	*transitions*	*states*	*transitions*	*states*	*transitions*
T1	7	8	1	2	14%	25%
T2	9	12	2	4	22%	33%
Global	36	71	5	13	14%	18%

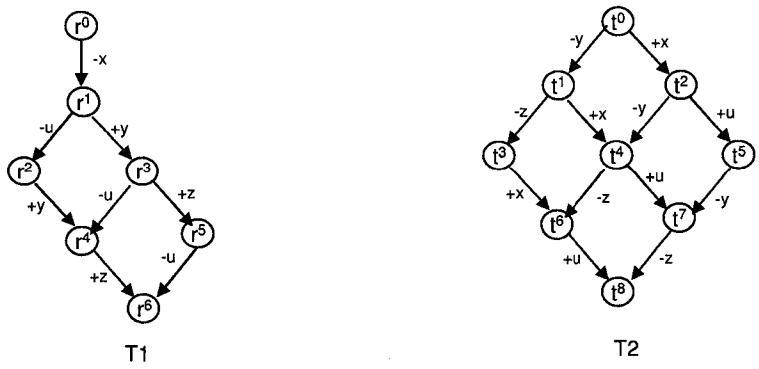
Table 3: The Efficiency Table of Reduction Rule for Example 1

of the protocol P and Figure 22 is the generated global specification. Because there are only two processes involved, *Rule 2* is not suitable to be applied.

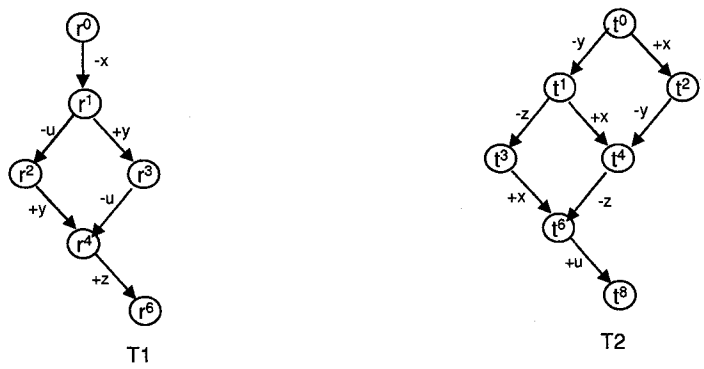
For this example, we evaluate the reduction efficiency of our method as in Table 3.

The second example has a deadlock state and two nonexecutable transitions, which is represented by dashed lines in the protocol specification in Figure 23. Figure 24 is the generated global specification. We see after the application of *Rule 1*, the deadlock state is still in the global specification and two nonexecutable transitions are not in the global specification.

For this example, we evaluate the reduction efficiency of our method as in Table 4.



(A) The Original Protocol Specifications of The Example



(B) The Reduced Protocol Specifications of The Example

Figure 21: The Application of the *Rule 1* upon Example 1

	Original number		*Reduced number*		*Reduction Efficiency*	
	states	*transitions*	*states*	*transitions*	*states*	*transitions*
T1	5	5	0	0	0%	0%
T2	7	8	1	2	14%	25%
Global	19	28	2	5	11%	18%

Table 4: The Efficiency Table of Reduction Rule for Example 2

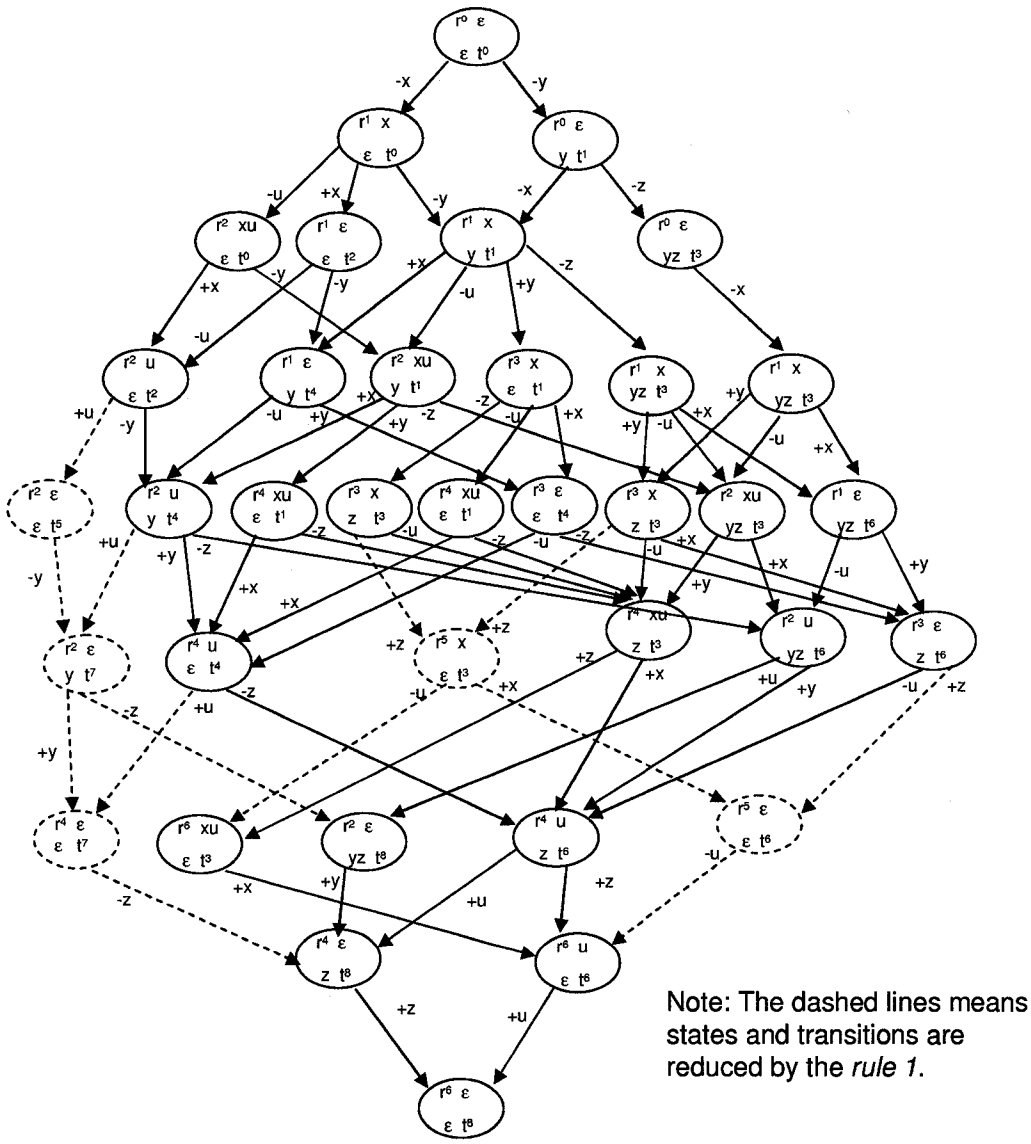
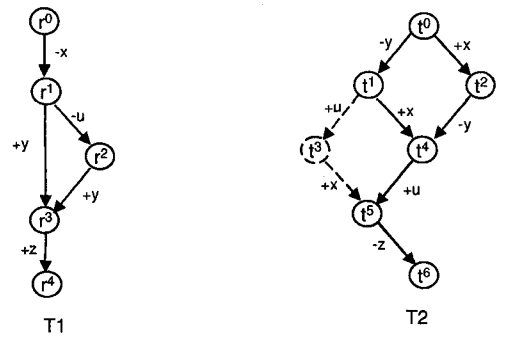
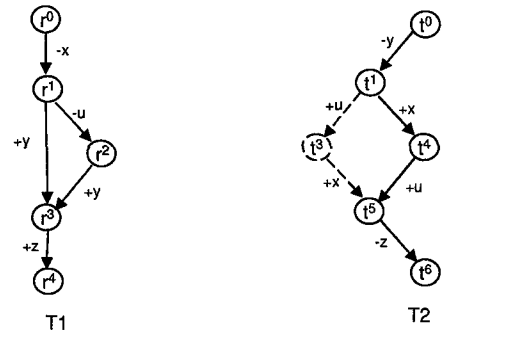


Figure 22: The Global Specification of Example 1



(A) The Original Protocol Specifications of Example 2



(B) The Reduced Protocol Specifications of Example 2

Note: The dashed lines present non-executable transitions and states.

Figure 23: The Application of the *Rule 1* upon Example 2

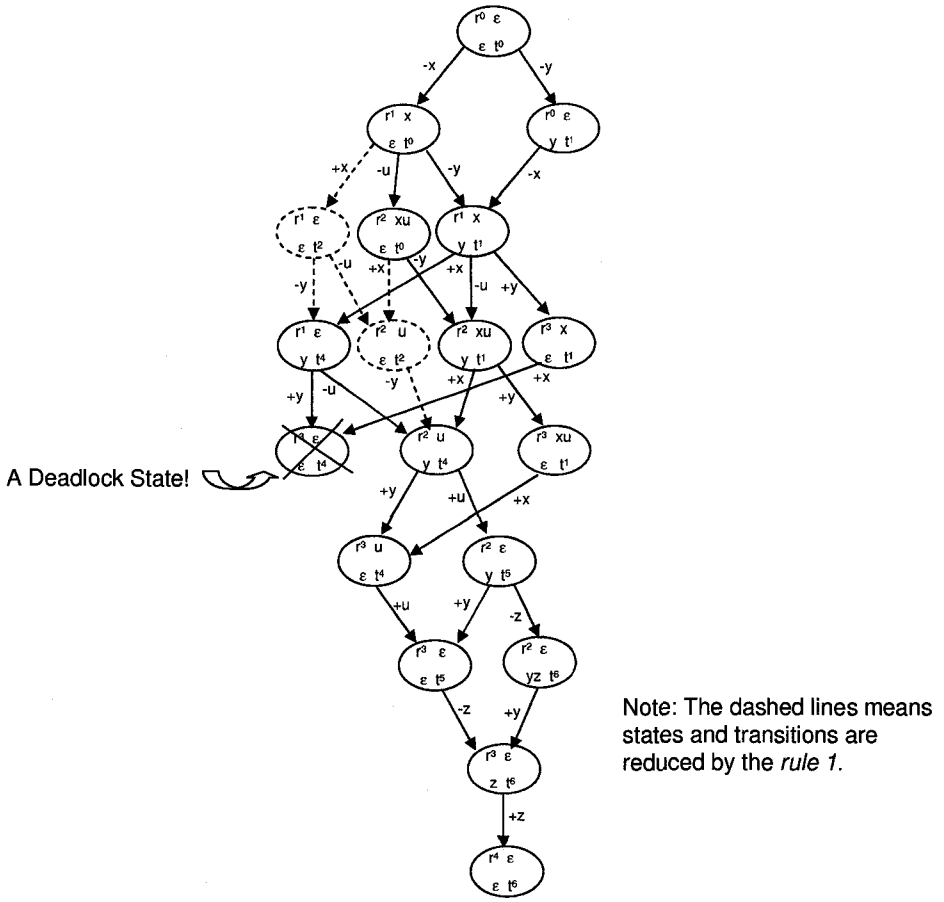


Figure 24: The Global Specification of Example 2

6.6 A Discussion

In the literature, reduction techniques are performed during the global reachability analysis, but the reduction method proposed in this section is performed before the global reachability analysis. The former is dynamic and the latter is static.

As we mentioned in Section 1, in model checking based formal verification, the design is interpreted as operational models, such as Finite States Machines (FSM) or Labeled Transition Systems (LTS), and the design properties are normally modeled by temporal logic languages. From the view of model checking, our method conducts transformation on the protocol design before checking if it satisfies the property. For an $n-$process protocol P, we denote the specification of process P_i as T_i, the performed transformation as $\mathcal{F}$, the desired design property as ψ. $||$ is a "parallel" operator which composes the processes of a protocol (that is, processes will be executed together to form the network of that protocol), and $\vdash$ is an binary relation operator which means "satisfies". Ideally, $(T_1|| T_2|| ...|| T_n) \vdash \psi$ iff $(\mathcal{F}(T_1)|| \mathcal{F}(T_2)|| ...|| \mathcal{F}(T_n)) \vdash \psi$, where ψ is a general property expressed by some temporal logic language.

However, the work of this section is not that general. The limitation of our work is in two aspects: 1) limited applicability of the transformation rules, that is, only two special patterns are considered; 2) the properties we can preserve is not so general that any existing temporal language can express more properties than those we discussed and the transformation of applying our rules may violate other properties. For example, when the desired property is "there exists a trace such that the event receiving message $m_{l,i}$ followed by the event sending message $m_{i,k}$", but these transitions are removed in T_i by applying *Reduction Rule 1*, and such trace will never be found in the global reachability analysis. In this case, the desired property is violated. So our work may just be a start.

7 Conclusion and Future Work

In reverse engineering, when a set of observations is given, we aim at constructing a deadlock-free design from it. In [6], three construction rules were proposed to achieve this goal. In this thesis, we have improved the third rule $Rule_{neg}$. With improved rule $Rule'_{neg}$ we take into account the contribution of the mirror of T_1, and only select the ones that can really help to remove the deadlock states. By this improvement, fewer states and transitions may be added to T_2. However, the improved $Rule'_{neg}$ may not be an optimal solution in the sense of constructing a minimum design without deadlock errors, in the future, we will continue our study to answer the question like "Does there exist an optimal solution for this problem?" or "If such solution exists, how to achieve it?". This method only works for 2-process protocols, and how to remove the deadlock states in $n-$process protocols is left as future work. Furthermore, the problem of removing unspecified receptions remains open.

In this thesis, we have also proposed a method to reduce the number of states of protocol design right before global reachability analysis during protocol design verification. We developed two reduction rules to deal with a specific pattern of transitions in the protocol specification. *Reduction Rule 1* deals with a choice of a sending transition and a receiving transition while *Reduction Rule 2* deals with a choice of two sending transitions to different processes. When the conditions of the rules are met, some transitions can be considered as redundant transitions for the formal verification and are removed, thus, the search state space is reduced. Furthermore, *Reduction Rule 1* preserves deadlock and channel overflow errors but may not preserve non-executable transition errors. *Reduction Rule 2* preserves all these three errors. In the end, we discussed the efficiency of our reduction method with two examples. The drawback of this method is if the protocol specification does not contain any pattern conforming to the conditions of the rules, the application of our method is not effective. In this thesis, we only discuss four logical errors, namely,

deadlock states, unspecified receptions, channel overflow, and non-executable transitions, and other advanced properties verification is left for future work. Also, we only discuss the pattern that two paths have two same events but of a different order, discussion of paths with multiple events is left for future work.

Other possible future work include: (1) study the possibility of the present methods in the context where the communication channels are not FIFO; (2) study the possibility of the present methods in the context where the communication channels are not error-free.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proc. of the 22nd International Conference on Software Engineering (ICSE)*, pages 304–313, 2000.

[2] G. Bochmann. Finite state descriptions of communication protocols. In *Comp. Networks*, number 2, pages 361–372, 1978.

[3] D. Brand and P. Zafiropulo. On communicating finite state machines. In *J. ACM*, number 30, pages 323–342, 1983.

[4] J. Buchi. On a decision method in restricted second order arithmetic. In *Proc. International Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, 1962.

[5] R. Carver and Y. Lei. A general mdoel for reachability testing of concurrent programs. In *Proc. of ICFEM 2004, LNCS 3308*, pages 76–98, 2004.

[6] J. Chen and H. Ural. Construction of deadlok-free designs of communication protocols from observations. In *The Computer Journal*, volume 45, 2002.

[7] T. Cheung and A. Ghedamsi. A petri-net-based synthesis algorithm for network protocol design. In *TR-90-30, University of Ottawa, Dept. of Computer Science*, June 1990.

[8] T. Choi. A sequence method for protocol construction. In *Proc. 6th IFIP International Workshop on Protocol Specification, Tesing and Verification*, number 9, pages 1–18, June 1986.

[9] T. Chow. Testing software design modeled by finite-state machines. In *IEEE Trans. Software Eng.*, volume SE-4, pages 178–197, 1978.

[10] E. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order reduction. In *Intl Journal of Software Tools for Technology Transfer*, number 2, pages 279–287, 1999.

[11] P. Godefroid. Partial-order methods for the verification of concurrent systems: An approach to the state-explosion problem. In *Lecture Notes in Computer Science*, 1996.

[12] M. Gouda and Y. Yu. Synthesis of communicating finite-state mechines with guaranteed progress. In *IEEE Trans. on Commun., COM-32*, pages 779–788, July 1984.

[13] F. Hennie. Fault detecting experiments for sequential circuits. In *Proc. 5th Ann. Symp. Switching Circuit Theory and Logical Design*, pages 95–110, 1964.

[14] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[15] G. Hwang, K. Tai, and T. Huang. Reachability testing: An approach to testing concurrent software. In *International Journal of Software Engineering and Knowledge Engineering*, volume 5, pages 493–510, 1995.

[16] Y. Kakuda and Y. Wakahara. Component-based synthesis of protocols for unlimited number of processes. In *Proc. COMPSAC'87*, pages 721–730., 1988.

[17] B. Karacali and K. Tai. Model checking based on simultaneous reachability analysis. In *Proc. 7th Intl. SPIN Workshop on Model Checking of Software*, pages 315–324, 2000.

[18] J. F. Kurose and K. W. Ross. *Computer networking: a top-down approach featuring the Internet.* Addison Wesley, 2 edition, 2002.

[19] L. Lamport. Proving the correctness of multiprocess programs. In *IEEE Transactions on Software Engineering*, SE-3(2), pages 125–143, 1977.

[20] Y. Lei and K.-C.Tai. Blocking-based simultaneous reachability analysis. In *Proc. 13th IEEE Intl. Symp. on Software Reliability Engineering*, pages 316–326, 2002.

[21] Y. Lei, D. Kung, and Q. Ye. A blocking-based approach to protocol validation. In *Department of Computer Science and Engineering, The University of Texas at Arlington*, CSE-2005-2, 2005.

[22] K. Ozdemir and H. Ural. Protocol validation by simultaneous reachability analysis. In *Computer Communications*, number 20, pages 772–888, 1997.

[23] C. Ramamoorthy, Y. Yaw, R. Aggarwal, J. Song, and W. Tsai. Synthesis of two-party error-recoverable protocols. In *ACM SIGCOMM'86 Symp.*, pages 227–235, 1986.

[24] K. Sabnani and A. Dahbura. A protocol test generation procedure. In *Computer Networks and ISDN Syst.*, volume 15, pages 285–297, 1988.

[25] K. Saleh. Synthesis of communications protocols: an annotated bibliography. In *ACM SIGCOMM Computer Communication Review*, volume 26 of *5*, pages 40–59, Oct. 1996.

[26] D. Sidhu. Rules for synthesizing correct communications protocols. In *ACM SIGCOMM Computer Communication Review*, volume 12, pages 35–51, Jan 1982.

[27] A. Thayse and et al. *From modal logic to deductive databases: Introducing a logic based approach to Artificial Intelligence.* Wiley, 1989.

[28] H. Ural and H. Yenigun. Towards design recovery from observations. In *Proc. of IFIP FORTE'04*, volume LNCS 3235, pages 133–149, Sept 2004.

[29] R. van Glabbeek. The linear time – branching time spectrum II: The semantics of sequential systems with silent moves. In *Concur 93, Springer LNCS 715*, pages 66–81, 1993.

[30] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symposium on Logic in Computer Science*, pages 322–331, June 1986.

[31] C. West. General technique for communications protocol validation. In *IBM J. Res. Develop.*, volume 22, pages 393–404, July 1978.

[32] P. Wolper. On the relation of programs and computations to models of temporal logic. In *Proc. Temporal Logic in Specification, LNCS*, volume 398, pages 75–123, 1989.

[33] P. Wolper, M. Vardi, and A. Sistla. Reasoning about infinite computation paths. In *Proc. 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, 1983.

[34] P. Zafiropulo, C. West, H. Rudin, D. Cowan, and D. Brand. Towards analyzing and synthesizing protocols. In *IEEE Trans. Commun.*, volume COM-28, pages 651–661, Apr. 1980.

Vita Auctoris

Lihua Duan was born in 1976 in Taiyuan, China. She graduated from Beijing University of Posts and Telecommunications, Beijing, China, 1999, where she received a Bachelor's degree in Electrical and Electronic Engineering. She is currently a Master's candidate in the School of Computer Science at the University of Windsor and expects to graduate in summer, 2005.