INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

On the Requirements for Protocol Conversion

Kirsten Beyers

A Thesis
Submitted to the School of Graduate Studies and Research of the University of Ottawa in Partial Fulfillment of the Requirements for the Degree of Masters in Computer Science[*]

School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario

[*] The Masters program in Computer Science is a joint program with Carleton University, administered by the Ottawa-Carleton Institute for Computer Science

© Kirsten Beyers, Ottawa Ontario, Canada. November 1998

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-38734-8

Canada

Abstract

With the number of different networks in existence, networks need to be interconnected via gateways to allow end users of one network to communicate with end users of other networks. The gateways that connect the networks must deal with heterogeneity in the protocols they support. One approach is to implement a *protocol converter* on the gateway to translate and synchronize the messages between different protocols in such a way that the required service of the common upper layer protocol is satisfied.

In this thesis we explore one of the main problems of heterogeneous network interconnection at a gateway: *the protocol conversion problem*. The protocol conversion problem deals with the rules for message exchanges and the behavioral concerns that deal with the reconciliation between different protocol message formats, their orderings and the service they provide. This thesis provides a formulation of the protocol conversion problem in a general framework.

Formal methods have been proposed for the protocol conversion problem based on a common specification model for protocols: the Communicating Finite State Machine (CFSM) model. These formal methods require that the conversion requirements be defined formally as well. There are two types of conversion requirements: protocol-level conversion requirements that specify the translation requirements and the synchronization requirements of the protocol converter in terms of peer protocol messages, and service-level conversion requirements that specify the service requirements in terms of the service primitives of the conversion system. Based on the presented formulation of the protocol conversion problem, in this thesis an approach to synthesize service-level conversion requirements is proposed. Furthermore, an approach to synthesize the protocol-level conversion requirements that capture the semantic relation between the two protocols as it is specified by the service-level conversion requirements is proposed. It is then shown that existing formal methods can be used to synthesize a protocol converter using the constructed protocol conversion requirements.

Acknowledgments

My supervisor, Dr. Hasan Ural, introduced me to the field of protocol engineering, and in particular, the protocol conversion problem. He has continuously challenged and motivated me. I would like to thank him for his encouragement, patience and support.

My graduate studies were funded in part by the following institutions: The National Sciences and Engineering Research Council of Canada, The Faculty of Graduate Studies and Research of Carleton University, The School of Computer Science of Carleton University, the School of Graduate Studies and Research of the University of Ottawa.

To Basilio.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CFSM	Communicating Finite State Machine
FIFO	First In First Out
FSM	Finite State Machine
OSI	Open Systems Interconnection
SIA	Service Interface Adapter
SCP	Submodule Construction Problem

Chapter One

Introduction

Communication between two end users in a network relies on the ability of the communicating entities to exchange information in an orderly fashion. Communication protocols provide the set of rules that govern the orderly exchange of information. With the number of different networks in existence, networks need to be interconnected via gateways to allow end users of one network to communicate with end users of other networks. This can complicate the communication since each constituent network may have its own communication protocols different from the other networks communication protocols. This is called heterogeneous network interconnection. The gateways that connect the networks must deal with this heterogeneity. One approach is to implement a protocol converter on the gateway to translate and synchronize the messages between different protocols in such a way that the required service of the upper layer protocols is satisfied. The end users on the different networks will still use the same access protocols to establish connection and transmit information; insertion of gateways and converters is transparent.

Heterogeneous network interconnection at a gateway involves solving two main problems: architectural problem and protocol conversion problem. The architectural problem deals with matching the layered architectures in two networks to identify the layers at which the interconnection will take place. The protocol conversion problem deals with the rules for message exchanges and the behavioral concerns that deal with the reconciliation between different protocol message formats, their orderings and the service they provide.

Interconnection typically can occur at any layer as long as the protocols are consistent in the endsystems above the interconnection layer. Two different approaches to protocol conversion can be considered: conversion at the service level (via a service interface adapter) or conversion at the protocol level (via a protocol converter).

The design of a communication protocol and, in particular, the design of a protocol converter are challenging problems. To address these problems formal methods for protocol design, and more recently, formal methods for synthesizing protocol converters, have been developed. In a formal design or synthesis method, a protocol is represented by a formal model. The communicating finite state machine (CFSM) model [BZ83] is a formal model widely used to specify communication protocols.

In the CFSM model, a protocol specification is represented by a network of protocol entities that communicate by exchanging messages. The behavior of each protocol entity is modeled by a CFSM. The communication between the protocol entities can be modeled by direct communication (synchronous or asynchronous over simplex channels) or indirect communication (via a communication medium also modeled by a CFSM). In the case of asynchronous communication, each channel is represented by an error free FIFO queue.

A formal method for protocol conversion requires that the conversion requirements be defined formally as well. There are various forms of conversion requirements found in the literature, however, they can be grouped by the type of conversion requirements they specify. Protocol-level conversion requirements specify the translation requirements and the synchronization requirements of the protocol converter in terms of the peer protocol messages of the two different protocols. The service-level conversion requirements specify the service requirements in terms of the service primitives of the conversion system. The approaches proposed in the literature synthesize protocol converters that satisfy either protocol-level conversion requirements or service-level conversion requirements, but not both.

Service-level conversion requirements are typically specified in the form of a required service specification of the conversion system, i.e., a CFSM that specifies the service that must be provided by the conversion system protocol in terms of the service primitives. Of

the various forms of protocol-level conversion requirements proposed in the literature, the conversion seed [Oku86][CL90d] is the most expressive in specifying the translation requirements and synchronization requirements of the converter. The conversion seed is a CFSM that declares the semantic relation between two dissimilar protocols in terms of their peer protocol messages. [Oku86] and [CL90d] each propose a solution for an instance of the protocol conversion problem that rely on protocol-level conversion requirements specified as a conversion seed. Of the approaches based on protocol-level conversion requirements, the conversion seed approaches [Oku86][CL90d] are the most formal and precise protocol-level approaches to the protocol conversion problem. If a suitable conversion seed can be found they provide efficient, algorithmic solutions to determine if a protocol converter exists and generate one if it does.

In the sequel, we provide a formulation of the protocol conversion problem in a general framework. We have observed that there is a gap between the starting point of the protocol conversion problem in our formulation and the starting point of each of the approaches proposed in the literature. None of the approaches based on service-level conversion requirements provides a mechanism to derive the required service of the conversion system. Furthermore, none of the approaches based on protocol-level conversion requirements, including the conversion seed approaches, provides a mechanism to derive protocol-level conversion requirements that capture known service-level requirements.

Based on our formulation of the protocol conversion problem, an approach to synthesize the required service specification for the conversion system is proposed. Furthermore, an approach to construct a conversion seed that captures the semantic relation between the two protocols as it is specified in the required service specification is proposed. Existing formal approaches can then be used to synthesize a protocol converter.

In Chapter 2, we discuss heterogeneous network interconnection. In Chapter 3, our formulation of the protocol conversion problem is presented. We also provide the necessary background to study the proposed approaches to protocol conversion found in the literature. In Chapter 4, we provide a survey of formal methods to protocol

conversion. In Chapter 5, we propose an approach for deriving the service-level requirements, and their equivalent protocol-level requirements, for protocol conversion. Examples of the application of our approach for the synchronous case and the asynchronous case are provided in Chapter 6 and Chapter 7, respectively. Chapter 8 concludes the thesis with a summary of contributions and directions for future research.

List of Contributions

- We provide a formulation of the protocol conversion problem, and a survey and analysis of the current literature on protocol conversion, restricted to the CFSM model for protocol specification.
- We provide a formal approach to synthesize the required service specification for the conversion system. This approach allows for the maximum safe solution.
- We provide a formal approach to synthesize a conversion seed that captures the service-level conversion requirements for safety as specified in a required service specification for the conversion system. The approach can be applied to both the synchronous model of communication and the asynchronous model of communication.
- We provide an improvement to our conversion seed synthesis approach. The improvement is an efficient reduction technique that can reduce the number of states and transitions in order to reduce the state space explosion that occurs when finding the product of CFSMs.
- We provide a second improvement to our conversion seed synthesis approach. The improvement uses the efficient protocol pruning technique [LNS95][KLNS93] to remove those parts of the input CFSMs that do not contribute to the service to be provided.
- Finally, when our synthesis approaches are used with an existing conversion seed based approach, we have a new formal approach for constructing a protocol converter. To our knowledge this is the first approach not based on the existence of an *SIA*, that addresses the safety property for the asynchronous model of communication.

Chapter Two

Heterogeneous Network Interconnection

A general network architecture has been described by the OSI reference model. The reference model has seven layers as shown in Figure 1 [Zim80].

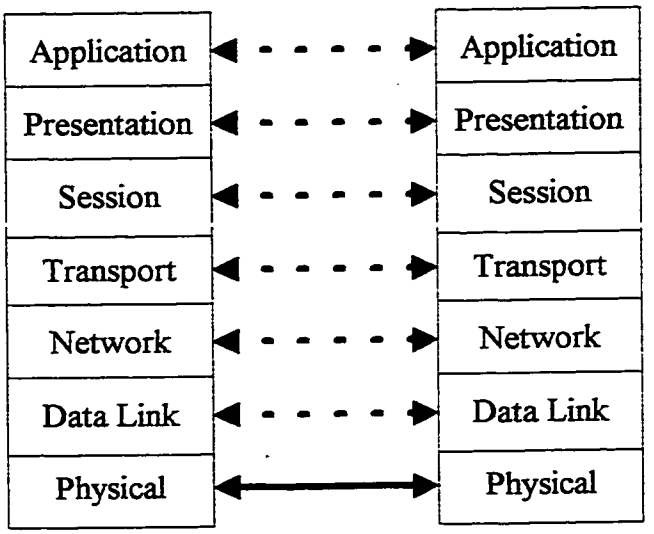

Figure 1: Network architecture based on the OSI reference model

The first layer, the physical layer, provides a virtual link for transmitting a sequence of bits between any pair of nodes joined by a physical communication channel. The second layer, the data link control layer, converts the unreliable bit pipe at the physical layer into a higher level, virtual communication link for sending packets asynchronously but error-free in both directions over the link. The third layer, the network layer, provides a virtual link for end-to-end packets; routing, flow control and congestion control are implemented at this layer. The fourth layer, the transport layer, provides a virtual link for end-to-end messages; it provides multiplexing services, and handles important issues such as naming and addressing, connection establishment and termination, error recovery, and

synchronization. The fifth layer, the session layer, provides a virtual session; it handles the interactions between the two end-points in setting up a session. The presentation layer provides a virtual network service to the final layer, the application layer. The main functions of the presentation layer are data encryption, data compression, and code conversion. The application layer does the work specific to the particular application to provide services to the users of the network.

Although many existing protocol architectures do not follow exactly the layered structure of the OSI reference model, the layered approach has become essentially universal and has been widely adopted in many computer networks.

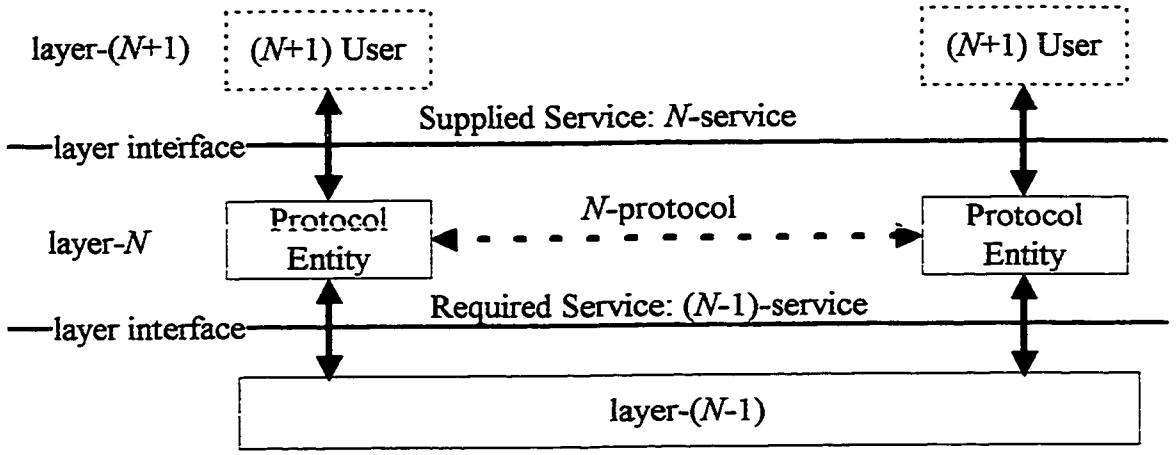

Figure 2: Layered network model

In any layered network model, the concepts of *protocol* and *service* are crucial (Figure 2). Each layer consists of a collection of protocol entities that are distributed over different locations. The protocol entities that are in the same layer are called peer protocol entities. The peer protocol entities of layer-N provide the communication service, called N-service, to layer-(N+1) users. The N-service is accessed by the user entities through a layer interface. Likewise, the protocol entities of layer-N access the communication service called (N-1)-service, provided by layer-(N-1) through another layer interface. The entities of layer-N use the (N-1)-service for exchanging messages. The rules that govern the exchange of these messages among the entities are collectively called an N-protocol. Therefore, for each layer-N, there exist two levels of abstraction: N-service and N-protocol. At the higher level of abstraction, the N-service defines the functionality to be

provided to the (N+1)-entities. At the lower level of abstraction, the N-protocol defines the behaviour of the N-entities inside layer-N.

In reality, no data or messages are transmitted horizontally from one entity to another except in the lowest layer. Each layer passes data down to the layer immediately below it; however, each entity is able to conceptually think of its communication as being horizontal.

In [Sun90] network interconnection is defined to be "any technique which enables systems on one network to communicate with or make use of services on another network". The interconnection is accomplished at a particular node, called a gateway, connecting two or more networks to form an internetwork. Protocol conversion is associated with network interconnection because of the need to interconnect networks of different architectures (i.e., implementing different protocols). Protocol conversion, if necessary, is usually realized in the gateway, although this is not necessarily always the case. The distinction between the gateway function and the converter function was addressed in [Flu88].

gateway function: a system that represents a particular point of data transit between a number of physical or logical networks, whether they use the same protocols or not.

converter function: a system that realizes a mapping at a given layer between two different communication protocols. It can be realized either within an endsystem or within a gateway.

All three possible combinations of the two functions exist. A gateway where no conversion is necessary because the protocols used in the interconnected networks are the same; a converter that is implemented within an endsystem; or a converter that is implemented within a gateway that interconnects two networks with different network architectures. We will be considering heterogeneous network interconnection within a gateway where conversion is necessary.

2.1 Heterogeneous Network Interconnection Architectures

Consider the interconnection of two networks at a gateway. The ideal solution to the interoperability would be to use common protocols for all layers above the network layer [Gro86]; however, this is not currently feasible. Since conversion is a reality, the optimum solution would be to perform conversion at all layers within a gateway, which would make full interoperation through a converter possible. However, the higher protocol layers (i.e., above the transport layer) are so diverse that an all-embracing protocol layer conversion that retains the end-to-end significance of every layer is not feasible.

The particular node and the layer of interconnection are two of the major architectural issues in internetworking. The nature of the application plus economic and system structure considerations [Gre86] suggests the choice of the node and the choice of the protocol layers at which the interconnection is to be made. Interconnection typically occurs at the physical layer (repeaters), at the data link layer (bridges) or at the network layer (routers, usually in wide area networks). There are cases, however, where interconnection occurs at the transport layer or application layer.

Figures 3 and 4 show two heterogeneous interconnection architectures. The process indicated by "═══" in each figure represents where interconnection with conversion is necessary.

In Figure 3, two subnets, U and V, have to be interconnected to allow for communication between endsystems S and D. The two subnets being interconnected form an internet. In this example there are three different network architectures, A, B and C. The interconnection and conversion is implemented at layer-N of network A and layer-M of network B by a gateway.

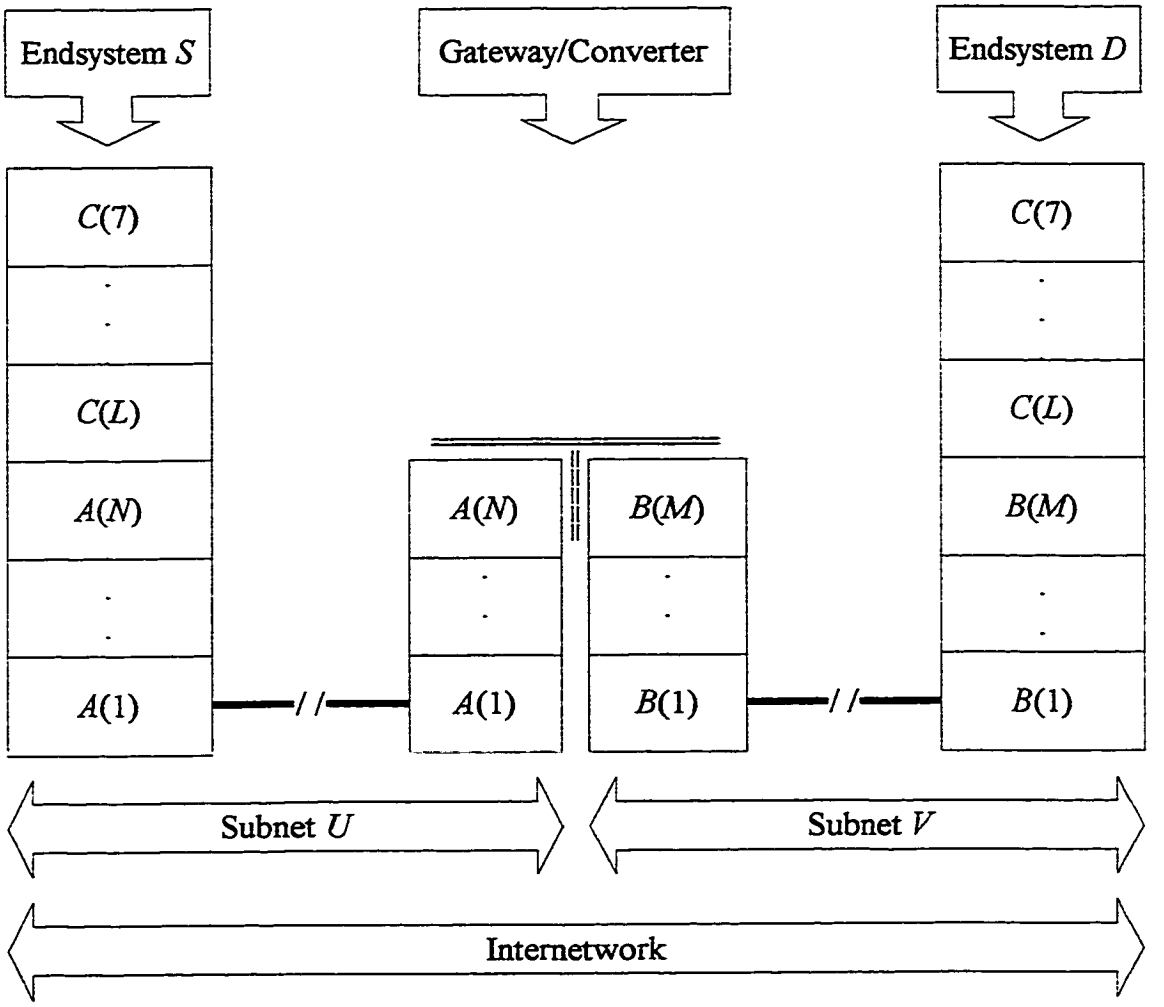

Figure 3: Heterogeneous network interconnection, complete conversion

This example demonstrates *complete conversion* between architectures A and B, up to layer N and M, respectively. The A header information and protocols are absent along the path to the right of the point of the conversion, and, accordingly, the B header information and protocols are absent along the path to the left of the conversion. Above the interconnection layer, the protocols must be consistent in the endsystems. It is assumed that the N-service of network architecture A, and the M-service of network architecture B each satisfy the required service of layer-L of network architecture C.

In Figure 4, a "classical" protocol conversion situation is shown. The network architecture of both endsystems are compatible, i.e., they both implement network architecture A at all layers. However, the actual network connection is heterogeneous

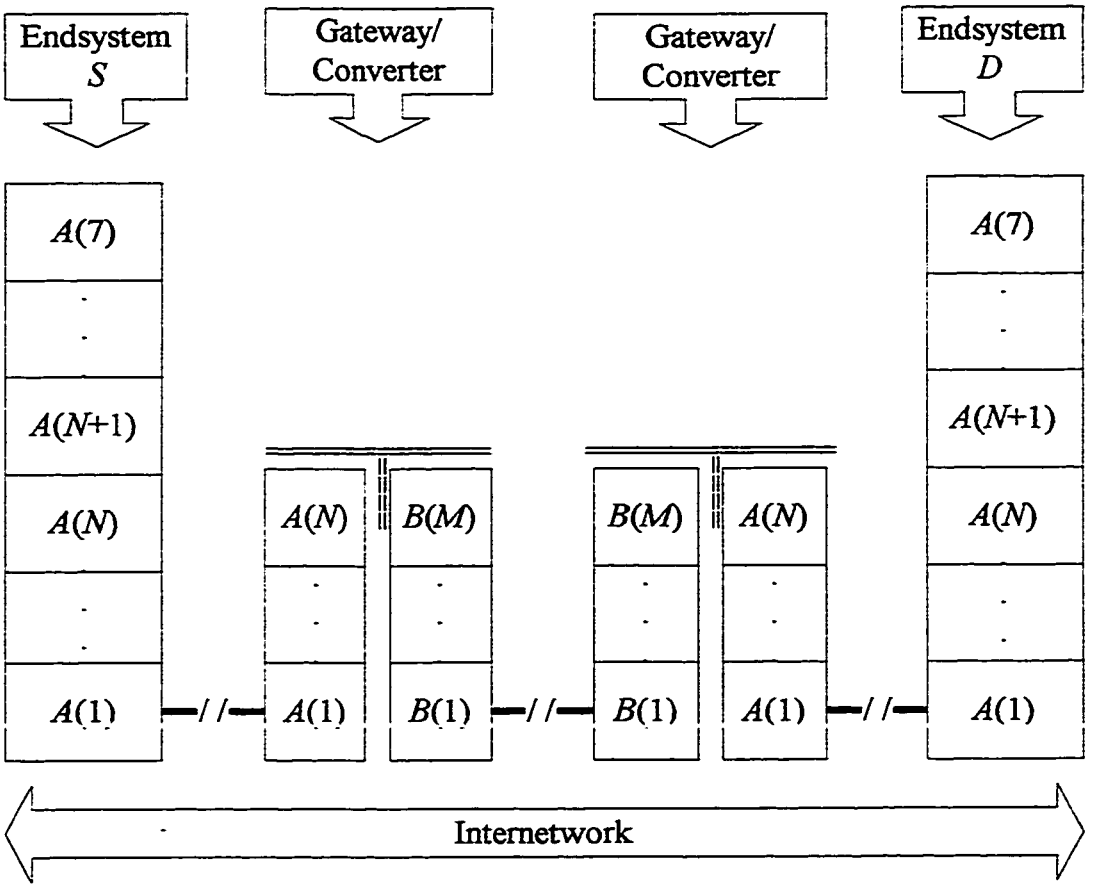

Figure 4: Heterogeneous network interconnection, partial conversion

because it passes through the segment in the center which obeys network architecture *B*. The two gateways implement conversion between layer-*N* of network architecture *A* and layer-*M* of network architecture *B*. In this case only layers 1 through *M* of network architecture *B* are used.

This example demonstrates *partial conversion* between network architectures *A* and *B*, so long as the conversion takes place at a lower layer than the application layer, i.e., there are portions of the access path over which the headers of network architecture *A* are encapsulated within those of *B*.

2.2 Protocol Mismatches

Assume the node has been chosen for the interconnection that will be performed between layer-N of architecture A and layer-M of architecture B.

The N-service of architecture A and the M-service of architecture B must be analyzed to determine if there is a usable large common subset of functionality. This can be done by decomposing the N-protocol of architecture A into a set of *atomic protocol functions* [Gre86], or *basic components* [Bie89], that the layer performs during the steady-state phase as a service to layer-(N+1) of architecture A. (For example, segmenting, concatenating, resequencing, error control, flow control, synchronous vs. asynchronous, class of service, etc.) The M-protocol should be decomposed into its atomic protocol functions as well. The degree of overlap between these sets indicates the extent to which a conversion can be made.

There are a number of points that must be taken into consideration when two sets of functions are analyzed to identify the common subset of functionality between N-service and M-service. These are listed below:

I. Three sets of protocols must be considered: peer protocols, control protocols and interface protocols.

An entity engages in a set of interface protocols with the next higher layer in the same node, peer protocols with its peer in another node and executes control protocols with a control entity within the same node. An interface protocol will not provide the correct services if there are significant mismatches involving the other protocols.

The internal details of each layer entity will vary among implementations. What must match are the externally visible properties of a protocol: syntax rules (commands, responses and parameters) and semantics (allowed sequences of exchanges including time dependencies). Each atomic protocol function in each of layer-N and layer-M should be broken down into its peer, control and interface protocols, expressed in terms of their syntax and semantics. There must be a reasonable match between all three protocols if the conversion system will be able to support that atomic protocol function.

II. Both the transient and steady state aspects of a protocol must be considered.
The transient aspects of a protocol (i.e., connecting, disconnecting, test and recovery) must be analyzed when studying a possible mismatch along with the steady-state (time interval during which the access-path does useful work for the end users) aspects of a protocol.

III. Long term effects must be considered.
It is insufficient to think of mapping single events at layer-*N* to single events at layer-*M*. The mapping must take into account both the transient and steady state aspects as sequences of events over a span of time.

Once the atomic protocol functions are determined for each of layer-*N* and layer-*M*, and they are each decomposed into their peer, interface and control protocols, including both transient and steady-state aspects, the common subset of functionality is analyzed to determine if it is effective enough to satisfy the required service of the common upper layer.

A *protocol mismatch* [Gre86] at the point of conversion is the set of disagreements that exists within the common subset of atomic protocol functions. A *hard mismatch* is a degree of mismatch that renders the system inoperable. For example, a hard mismatch occurs if a required atomic protocol function is not included within the common subset of functions. A hard mismatch also occurs if the mismatch between the protocols for the same atomic protocol function is so severe that the requirements can not be met. A *soft mismatch* is a degree of mismatch that is not severe enough to prevent the network from being used for the particular application intended. Of course, it is possible that there is *no mismatch* between the protocols for an atomic protocol function, i.e., the protocols have the same syntax and semantics.

2.3 Approaches to Interconnection

Given the node where the interconnection will be implemented, and the layer of architecture *A*, layer-*N*, and the layer of architecture *B*, layer-*M*, how can we interconnect

the protocols at these layers so that the required service of the common upper layer is satisfied?

There are a number of approaches to interconnection at a gateway. The selection of an approach depends on the degree of mismatch between the protocols (none, soft or hard) and the required service of the upper layer. If possible, we could modify the entities to provide the required service to the upper layer and make the protocols compatible. This is a very expensive choice because of the number of nodes that would have to be changed. Approaches that are more practical include interconnection at the service level, interconnection at the protocol level, and protocol complementation.

2.3.1 Service level interconnection

In [BMM90] there is a good discussion on the interconnection of services at the service level. It claims that most compatibility issues can be discussed in terms of the compatibility of services and no need to consider the actual protocol implementation. In this case, only the interface protocols are considered to implement the mapping between layer-*N* and layer-*M* since the interconnection is performed above these layers.

I. Service concatenation or service relay

Service concatenation (Figure 5) may be used when interconnecting identical services at a gateway where an interaction of one interface can be mapped onto a corresponding interaction of the other interface. Obviously, these protocols must have no protocol mismatch. Note that not all protocols without a mismatch can simply be concatenated.

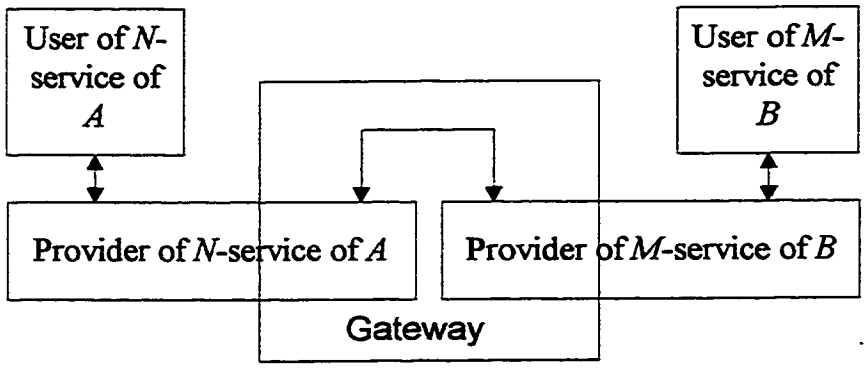

Figure 5: Service concatenation

The *N*-service of *A* and the *M*-service of *B* can be concatenated if the following conditions hold:

- Interactions initiated by the provider of the *N*-service of *A* (resp. *M*-service of *B*), can be mapped onto interactions initiated by a user of the *M*-service of *B* (resp. *N*-service of *A*); this should include a mapping of corresponding parameters of the mapped interactions.
- Legal sequences of interactions initiated by the provider of the *N*-service of *A* (resp. *M*-service of *B*) are consistent with the legal sequences of interactions that could be initiated by a user of the *M*-service of *B* (resp. *N*-service of *A*).

II. Service Interface Adaptation

Service interface adaptation (Figure 6) may be used when interconnecting possibly different services at a gateway that are abstractly equivalent, i.e., there is no semantic mismatch. This implies that although they have different interfaces they provide the same abstract service and a mapping between the two interfaces can be defined.

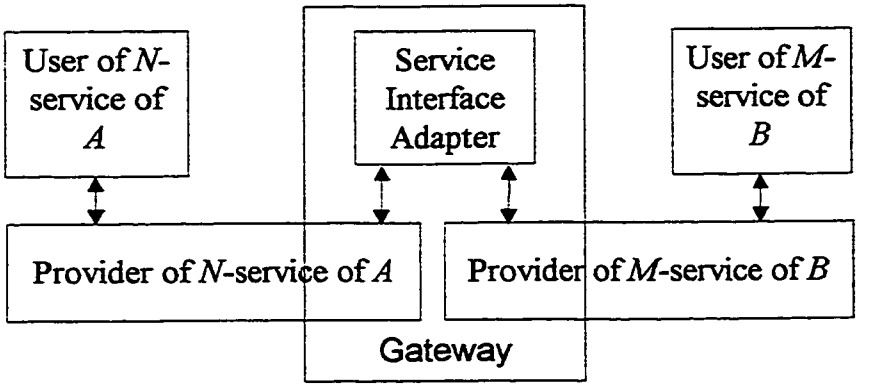

Figure 6: Service interface adaptation

Semantically equivalent services can be interconnected through an interface adapter module if the following conditions hold:

- abstract interactions can be "extracted" from the two interface specifications; and
- the conditions defined for service concatenation above hold for this set of abstract interactions.

III. Restricted Service Interface Adaptation

Service interface adaptation (Figure 7) may be used to provide interconnection at a gateway over the common subset of functionality if the services to be concatenated are not equivalent. This would require that both protocols be restricted to the functionality of this common service subset. This solution is only realistic if this common service subset satisfies the required service of the common upper layer, i.e., a soft mismatch exists between the protocols.

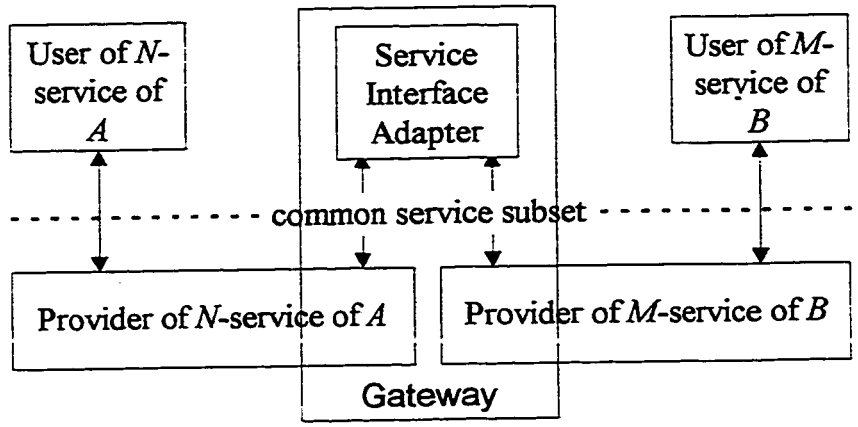

Figure 7: Restricted service interface adaptation

The easiest approach to interconnection at a gateway is to use a common service boundary since it is not concerned with the details of the peer protocols and the control protocols. The conversion is done between layer-*N* of one network architecture, and layer-*M* of the other, assuming that the protocols above those layers in the two networks are compatible. The advantage is that existing protocol implementations can be used within the gateway (Figure 8).

The service level interconnection at a gateway raises valid concerns about the end-to-end significance of the original protocols, the concatenation of reliable and unreliable services, etc., since not all the peer and control protocol's functions will surface at the service level and this may be relevant to the required service. Clearly, the *M*-protocol and *N*-protocol terminate at the gateway thus any end-to-end synchronization capability of the existing services will be lost.

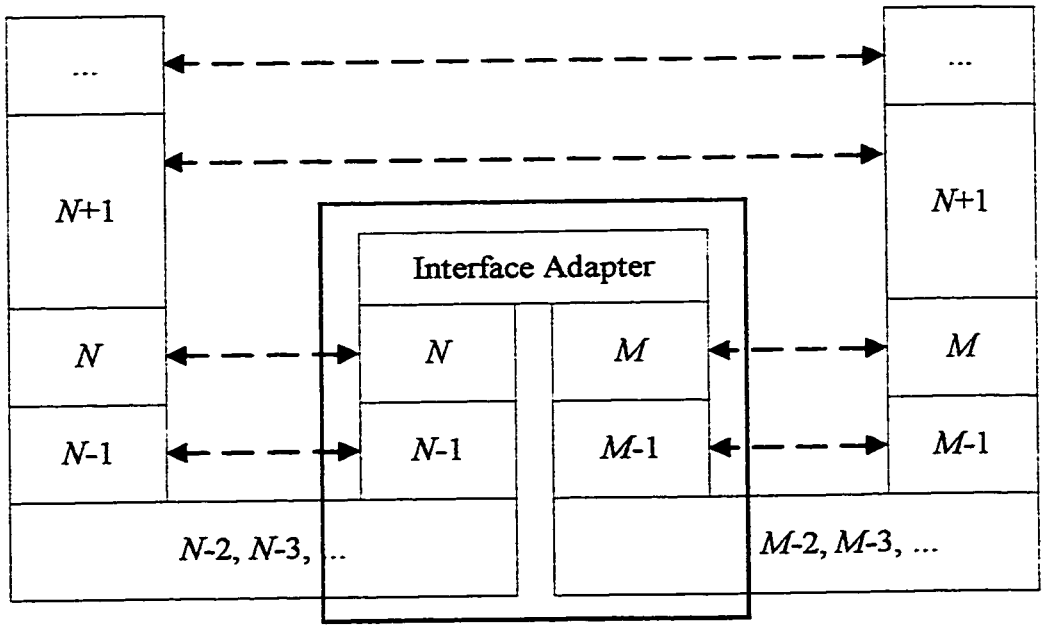

Figure 8: Service level interconnection at a gateway

Sunshine [Sun90] addresses this major issue of interconnection: step-wise versus end-to-end service. Bochmann et al. [BMM90] also addresses it under the term concatenation invariance. A global property of a communication service is *concatenation invariant* [BMM90] if it remains satisfied on an end-to-end basis over several concatenated communication services and it is satisfied by each service individually. Most global properties of communication services are concatenation invariant according to [BMM90], however, if certain required global properties are not concatenation invariant then service level interconnection is not sufficient.

Consider the example of delivery confirmation. Local delivery confirmation is a confirmation that is issued after the data is received by the network. Remote delivery confirmation is a confirmation that is issued after the data is received by the end node and sent to the receiving user. User-to-user delivery confirmation is a confirmation that is issued after an explicit acknowledgment of receipt is issued by the receiving user. Local and user-to-user delivery confirmation are concatenation invariant, however, remote delivery confirmation is not concatenation invariant.

Note that quantitative (i.e. performance considerations) are not concatenation invariant. Delay tends to be additive under concatenation and throughput follows the minimum limit.

2.3.2 Protocol level interconnection

A more complex but more efficient approach to interconnection at a gateway requires replacing layer-*N* and layer-*M* in the gateway with a converter that maps the layer-*N* protocol to the layer-*M* protocol and vice-versa. The converter maps between sequences of events of one protocol into sequences of events of the other protocol. The converter function is defined explicitly in terms of the peer protocol messages exchanged within the two interconnected networks according to their respective protocols. Protocol level conversion attempts to provide conversions for all the protocol level common functionality. This can make the converter completely transparent in some cases (Figure 9).

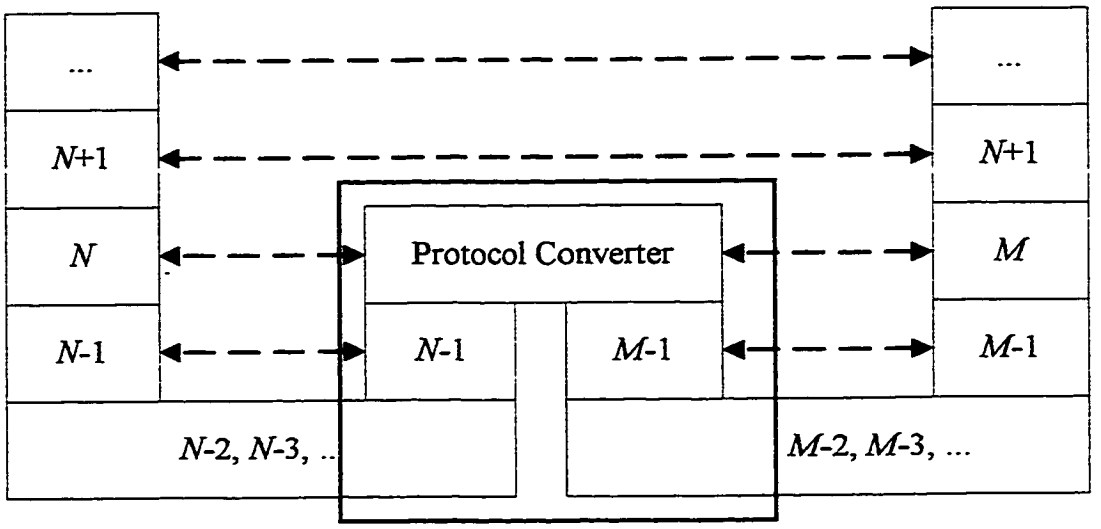

Figure 9: Protocol level interconnection at a gateway

Maintenance of the end-to-end semantics with a peer protocol converter is essential, at least within the common subset of functionality required by the application. Protocol level interconnection overcomes some of the problems with service level interconnection because it enables end-to-end synchronization at the conversion level. As an example one can refer to [Bie89] which describes three approaches for the protocol level interconnection of error control mechanisms:

- no interconnection of the mechanism:

If the error control mechanism is not included in the common subset of functionality that is required by the application then it is not necessary to interconnect the mechanisms of the two protocols. For example, the converter generates an acknowledgment as soon as it receives a message from an entity. Therefore, the error control is between the entity and the converter only and the end-to-end significance is lost.

- partial interconnection of the mechanism:

  If the error control mechanism is included in the common subset of functionality required by the application and it has end-to-end significance, then it is essential that it be preserved. With a partial interconnection of the mechanism, the converter performs the error control according to the protocols and interconnects the control mechanism for acknowledgments of messages and indications of non-recoverable errors. For example, the converter does not acknowledge the receipt of a message before it receives the acknowledgment from the other entity. This preserves the end-to-end significance but increases the delay.

- complete interconnection:

  If the error control mechanism must be interconnected and the mechanisms of the protocols have the same semantics then complete interconnection can be used. In this case, the converter does not perform any error control. The converter simply passes the protocol information onto the endsystem. Therefore, any end-to-end significance is maintained assuming the two protocol mechanisms have no mismatch.

2.3.3 Protocol complementation

Where a hard mismatch occurs, the service level interconnection and the protocol level interconnection cannot provide a service that meets the requirements of the application. A method to handle this problem [Sun90] is to add a convergence sublayer protocol to the system. Complementing is used to augment the *N*-service or the *M*-service or both, to enlarge the common subset of atomic protocol functions of layer-*N* and layer-*M*. The

disadvantage is the fact that the internal nodes may have to be changed that run the uncomplemented protocols fine.

A complementation protocol can be built on top of the less powerful protocol to upgrade the lower level service to the level of the other one. Thus, an entity would have to reside on all systems within the less powerful network to realize this protocol. A complementation protocol may also be introduced at the end nodes of the system, on top of the minimal service subset, and may be used to provide services not provided by any of the interconnected components. The endpoint approach guarantees a full service with common attributes at both ends by requiring implementation of a common protocol in both endpoint nodes. It makes use of simpler services on the individual networks along the way and hence allows use of simpler gateways. Most errors along the path will be corrected by the endpoint mechanisms. Finally, a complementation protocol can be introduced throughout the entire system, i.e., at the gateway and end nodes.

Chapter Three

Protocol Conversion

Heterogeneous network interconnection at a gateway involves solving two main problems: architectural problem and protocol conversion problem. The architectural problem deals with matching the layered architectures in the two networks to identify the layers at which the interconnection will take place. The protocol conversion problem deals with the rules for message exchanges and the behavioural concerns that deal with the reconciliation between different protocol message formats, their orderings and the service they provide.

The analysis of the protocol conversion problem that began with [Gre86] was reviewed in Chapter 2. [Gre86] provided an overview of the protocol internetworking architectures along with a discussion of the different circumstances under which conversion has to take place between different protocols, and possible approaches to the protocol conversion problem. [Gre86] also pointed out the lack of theoretical study of the protocol conversion problem and need for the development of formal methods. Two different approaches to protocol conversion can be considered: conversion at the service level (via a service interface adapter) or conversion at the protocol level (via a protocol converter). We are considering the problem of finding a protocol converter using a formal approach (although finding the service interface adapter may be part of the solution).

A formal approach to synthesize a protocol converter must provide a mechanism for specifying the requirements of the conversion. It is likely, when considering the interconnection of two different protocols, that there exists a required service (of the upper layer protocol) that must be satisfied. Furthermore, it is also likely that one has protocol-level requirements that must be satisfied (i.e., with respect to delay, end-to-end

significance, degree of mismatch) that would dictate whether complete, partial or no interconnection of an atomic protocol function was desirable.

Furthermore, with each conversion requirement there must be rules about what it means to satisfy that conversion requirement. In this chapter we provide a description of the protocol conversion problem, some preliminary models and notation that is necessary for the following discussion, and formalize the protocol conversion problem. In Chapter 4, we provide a survey of formal methods to protocol conversion.

3.1 The Protocol Conversion Problem

Consider two entities, $E0$ and $E1$ that communicate with each other by exchanging messages. The communication between $E0$ and $E1$ can be modeled in one of two ways: *direct communication* or *indirect communication*.

Figure 10: Direct communication

We say that entities $E0$ and $E1$ communicate directly if there is no explicit representation of the communication medium in the model. Implicitly, their communication is via a pair of perfect FIFO queues. The queues may be bounded by some integer value which represents the maximum number of messages that may be in transit in each direction, or unbounded (i.e., the maximum number of messages in transit is infinite). If the queues are unbounded or the bound on the capacity of the queues is greater than zero then the communication is *asynchronous*. In asynchronous communication, a message transmission corresponds to the message being placed at the tail of a queue and a message reception corresponds to the removal of a message from the head of a queue. If the bound on the capacity of both queues is zero then this yields *synchronous* communication which implies that the transmission of a message from one entity is directly coupled with the reception of that message by the other entity.

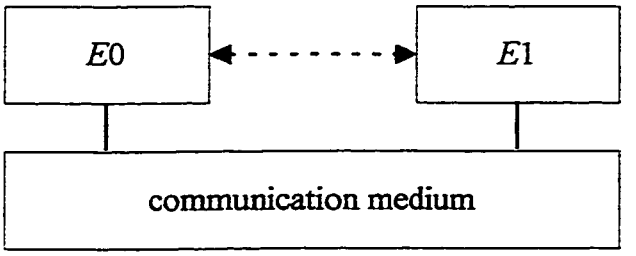

Figure 11: Indirect communication

We say that entities $E0$ and $E1$ communicate indirectly if there is an explicit communication medium in the model. The transmission of a message from one entity is directly coupled with the reception of that message by the communication medium. In turn, the transmission of a message from the communication medium is directly coupled with the reception of that message by the other entity. Thus, when the communication between the entities is indirect, the communication between an entity and the communication medium is synchronous. Note that the explicit communication medium may or may not model an imperfect medium where messages may be lost, duplicated, delayed or reordered.

The formulation of a protocol conversion problem between network X and network Y is based on the following assumptions:

1. Layer M protocol (called protocol P) of network X and layer N protocol (called protocol Q) of network Y are two different protocols.
2. There is a common peer protocol U which is layer M+1 protocol of network X and layer N+1 protocol of network Y. Thus, protocol U is common to both networks X and Y.
3. The required service (RS_U) for the common peer protocol U is defined.

In this discussion, protocols involving two communicating peer protocol entities, $E0$ and $E1$, will be considered for the protocol conversion problem. The peer protocol entities either communicate indirectly through a communication medium providing the required service for the protocol; or communicate directly. If they communicate indirectly then the communication type between the entities and the communication medium is synchronous.

If they communicate directly then the communication type between the entities may be synchronous (i.e., the bound on the capacity of each FIFO queue is zero) or asynchronous (i.e., the bound on the capacity of each FIFO queue is greater than zero).

The peer protocol entities also communicate with the upper layer of the network by exchanging messages. The communication between each protocol entity and the upper layer is modeled by direct synchronous communication.

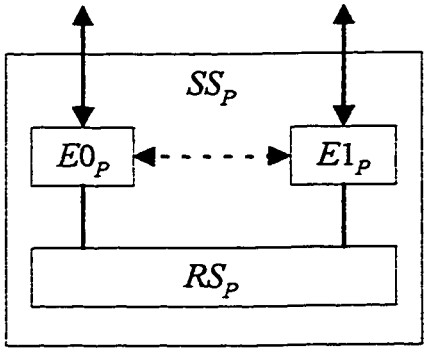

Figure 12: Layer M of network X

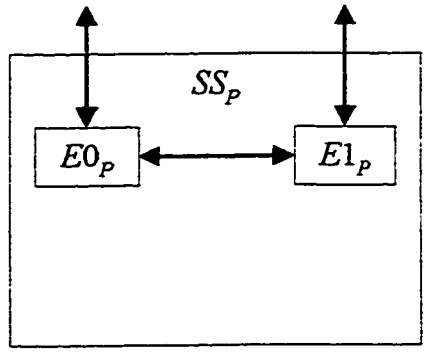

Figure 13: Layer M of network X'

In layer M of network X (Figure 12), $E0_P$ and $E1_P$ are two peer protocol entities that communicate indirectly through the communication medium supplying the required service (RS_P) of protocol P to provide the supplied service (SS_P) of protocol P. In layer M of network X' (Figure 13), $E0_P$ and $E1_P$ communicate directly (synchronously or asynchronously) to provide SS_P.

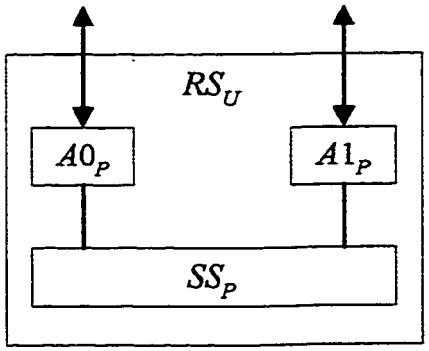

Figure 14: $SS_P \neq RS_U$ in network X or X'

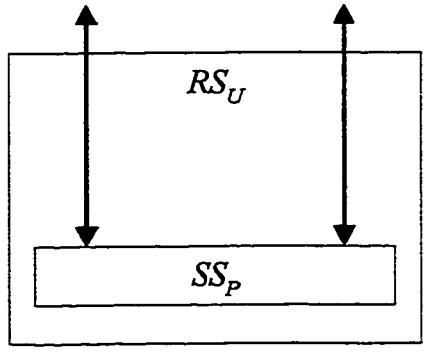

Figure 15: $SS_P = RS_U$ in network X or X'

If there is a gap between SS_P and RS_U, two interface adapters ($A0_P$ and $A1_P$) can be used to supply the required service (RS_U) for protocol U (Figure 14). Otherwise, SS_P provides RS_U (Figure 15).

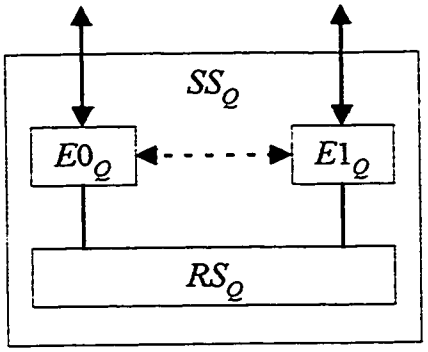

Figure 16: Layer N of network Y

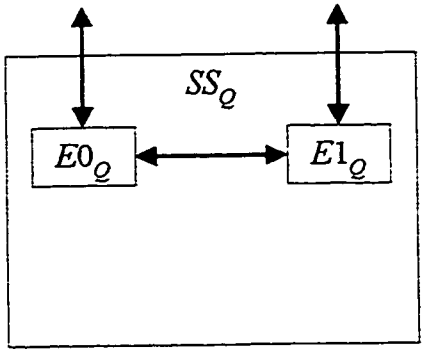

Figure 17: Layer N of network Y'

Similarly in layer N of network Y (Figure 16), $E0_Q$ and $E1_Q$ are two peer protocol entities that communicate indirectly through the communication medium supplying the required service (RS_Q) of protocol Q to provide the supplied service (SS_Q) of protocol Q. In layer N of network Y' (Figure 17), $E0_Q$ and $E1_Q$ communicate directly (synchronously or asynchronously) to provide SS_Q.

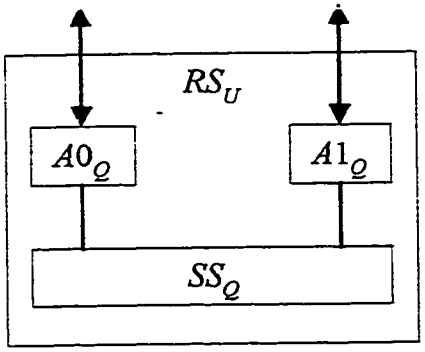

Figure 18: $SS_Q \neq RS_U$ in network Y or Y'

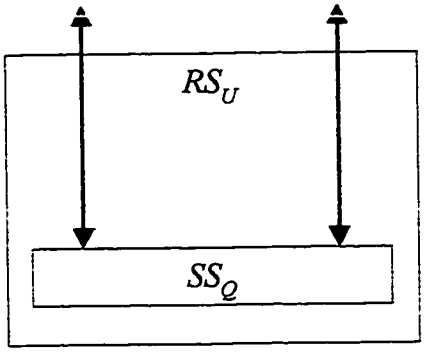

Figure 19: $SS_Q = RS_U$ in network Y or Y'

If there is a gap between SS_Q and RS_U, two interface adapters ($A0_Q$ and $A1_Q$) can be used to supply the required service (RS_U) for protocol U (Figure 18). Otherwise, SS_Q provides RS_U (Figure 19).

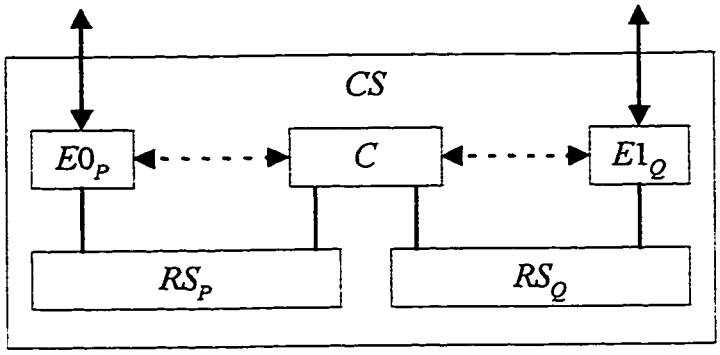

Figure 20: Network Z

Network Z (Figure 20) shows the conversion system for network X (Figure 12) and network Y (Figure 16). The entities $E0_P$ and C communicate indirectly through the communication medium supplying RS_P, and the entities C and $E1_Q$ communicate indirectly through the communication medium supplying RS_Q. The three entities $E0_P$, C, and $E1_Q$, together with the two communication media provide a new protocol whose service specification is CS.

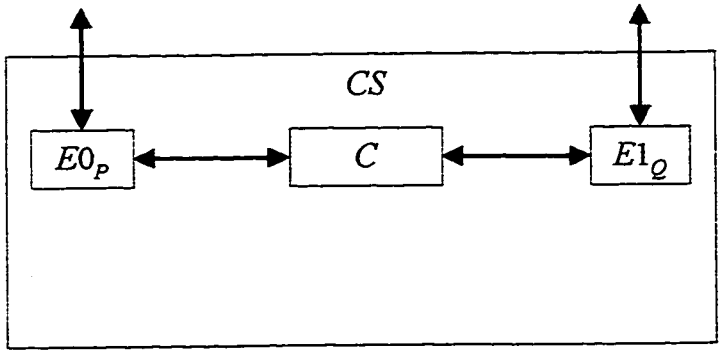

Figure 21: Network Z'

Network Z' (Figure 21) shows the conversion system for network X' (Figure 13) and network Y' (Figure 17). The entities $E0_P$ and C communicate directly, and the entities C and $E1_Q$ communicate directly. The three entities $E0_P$, C, and $E1_Q$ together provide a new protocol whose service specification is CS.

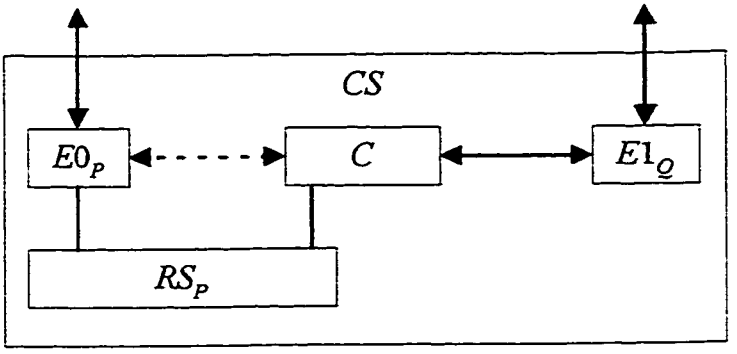

Figure 22: Network Z''

Network Z'' (Figure 22) shows the conversion system for network X (Figure 12) and network Y' (Figure 17). The entities $E0_P$ and C communicate indirectly through the communication medium supplying RS_P, and the entities C and $E1_Q$ communicate directly. The three entities $E0_P$, C, and $E1_Q$, together with the communication medium provide a new protocol whose service specification is CS.

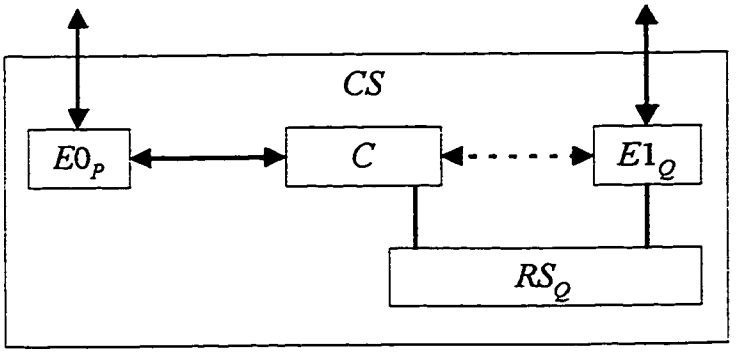

Figure 23: Network Z'''

Of course the dual of Figure 22 is also possible. Network Z''' (Figure 23) shows the conversion system for network X' (Figure 13) and network Y (Figure 16). The entities $E0_P$ and C communicate directly, and the entities C and $E1_Q$ communicate indirectly through the communication medium supplying RS_Q. The three entities $E0_P$, C, and $E1_Q$, together with the communication medium provide a new protocol whose service specification is CS.

Within a conversion system, the fact that $E0_P$ ($E1_Q$) is communicating with a converter C, as opposed to the peer protocol entity $E1_P$ ($E0_Q$), should be transparent to $E0_P$ ($E1_Q$). Therefore the type of communication, synchronous, asynchronous or indirect, between

entity $E0_P$ ($E1_Q$) and entity C is assumed to be the same as the type of communication between $E0_P$ ($E1_Q$) and $E1_P$ ($E0_Q$). This is apparent in Figure 20: Network Z, Figure 21: Network Z', Figure 22: Network Z'' and Figure 23: Network Z'''.

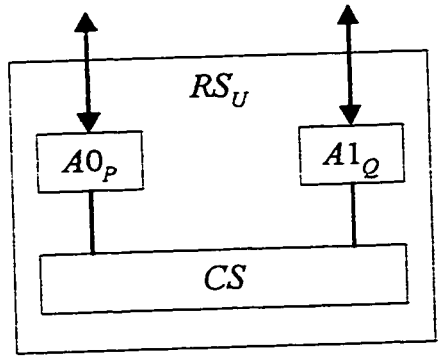

Figure 24: $SS_P \neq RS_U$ in network X or X', $SS_Q \neq RS_U$ in network Y or Y'

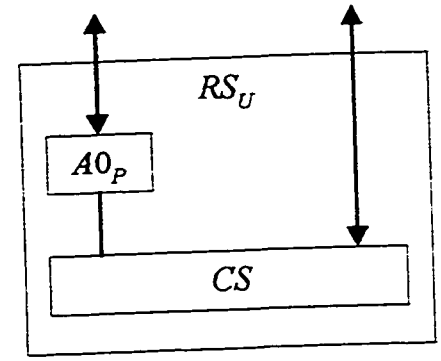

Figure 25: $SS_P \neq RS_U$ in network X or X', $SS_Q = RS_U$ in network Y or Y'

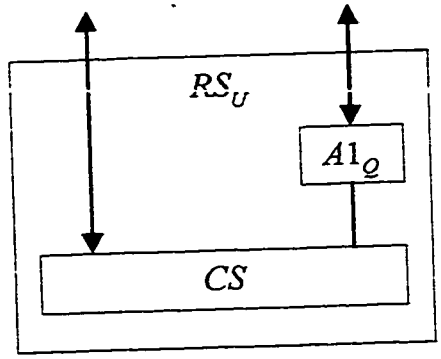

Figure 26: $SS_P = RS_U$ in network X or X', $SS_Q \neq RS_U$ in network Y or Y'

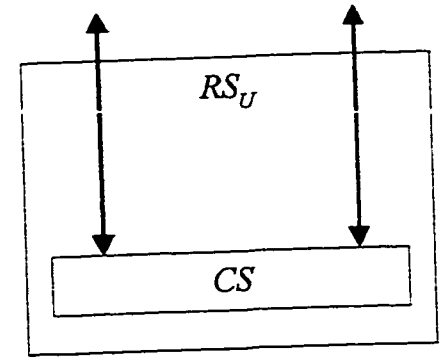

Figure 27: $SS_P = RS_U$ in network X or X', $SS_Q = RS_U$ in network Y or Y'

In all four cases the new protocol with service specification CS together with the existing adapters can supply the required service (RS_U) for the common peer protocol U. If there is a gap between SS_P and RS_U, then interface adapter $A0_P$ can be used to fill the gap between CS and RS_U (Figure 24, Figure 25). Accordingly, if there is a gap between SS_Q and RS_U, then interface adapter $A1_Q$ can be used to fill the gap between CS and RS_U (Figure 24, Figure 26). If both SS_P and SS_Q provide RS_U then CS provides RS_U (Figure 27).

3.2 Preliminaries

This section provides a brief description of the models used by the approaches proposed in the literature for protocol specification and the terms that are commonly used when considering the protocol conversion problem.

A specification of a protocol includes the specification of the peer protocol entities, and in the case of indirect communication, the specification of the communication medium through which the peer entities communicate. These will be collectively referred to as the protocol entities.

In all specification models used by the approaches referenced in the sequel, a protocol entity is specified as a *communicating finite state machine*. A communicating finite state machine (CFSM) is defined by a finite set of states, including a designated initial state; a finite set of actions; and a partial deterministic state transition function which specifies how the entity changes state through the execution of an action in its action set. In some models there is also a designated subset of the state set called the final state set, although usually this is not the case since the behaviour of a protocol entity is expected to continue forever.

The behaviour of a CFSM E is modeled by all possible sequences of state transitions and their associated actions from the initial state. We can regard a CFSM as a directed graph with nodes representing the states and directed edges representing the transitions. The behaviour of a CFSM E corresponds to the paths in its directed graph starting from the initial state. A sequence of actions is *accepted* by a CFSM E if and only if there is a sequence of state transitions (or path) from the initial state with the same sequence of associated actions. We denote the set of all sequences of actions accepted by a CFSM E as $L(E)$. The *projection* of a CFSM E over a subset of its action set Δ is denoted by $\text{Proj}_{\Delta}(E)$ [Chapter 5].

A *protocol* is a set of communicating finite state machines modeling its protocol entities. A protocol also includes an indication of how the protocol entities communicate. There are three different models of communication used in the literature.

I. *synchronous symmetric communication*

We have already introduced the concept of synchronous communication; i.e., the transmission of a message from one entity is directly coupled with the reception of that message by the other entity. If the communication model is synchronous *symmetric* then the causality of the message exchange is not considered. Message transmissions and message receptions are treated similarly. In fact, the directly coupled actions of two protocol entities are recognized by common action names. The entities synchronize on the directly coupled actions. There is no indication of which entity sends the message and which entity receives the message.

II. *synchronous asymmetric communication*

This model, like the previous one, assumes synchronous communication between the protocol entities; i.e., the transmission of a message from one entity is directly coupled with the reception of that message by the other entity. However, if the communication model is synchronous *asymmetric* then the causality of the message exchange is considered. The action set of an entity will contain messages prefixed with a '+', indicating a message reception action, and messages prefixed with a '–' indicating a message transmission action. In this case, the directly coupled actions of two protocol entities are recognized by common message names with opposite signs, i.e., $(+m, -m)$. The entities synchronize on the directly coupled actions, however, it is explicit which entity sends the message and which entity receives the message.

III. *asynchronous communication*

We have already introduced the concept of asynchronous communication, i.e., the protocol entities communicate via a pair of *channels* modeled by FIFO queues with length greater than zero. In this case, the communication must be asymmetric. The action set of an entity will contain messages prefixed with a '+', indicating a message reception action, and messages prefixed with a '–', indicating a message transmission action. A message transmission corresponds to the message being placed at the tail of a queue and a message reception corresponds to the removal of a message from the head of a queue. The content of a channel is represented by a sequence of zero or more messages. There can be a bound

on the content of a channel specified as a positive integer. If a bound is not specified then we say that the channel is unbounded.

The synchronous model of communication is appropriate where the delay in the communication between entities is negligible. The communication between protocol entities and their upper layer is always modeled by synchronous communication. In addition, in the case of direct communication, the communication between peer protocol entities may be modeled by synchronous communication. In the case of indirect communication, the communication between each peer protocol entity and the communication medium is always modeled by synchronous communication.

Whether the synchronous communication is modeled symmetrically or asymmetrically depends on whether the causality of a message exchange needs to be explicitly included in the specification.

The asynchronous model of communication is appropriate where there is some delay in a message being delivered. In the case of direct communication, the communication between peer protocol entities may be modeled by asynchronous communication.

There are different types of actions for which a transition in a protocol entity can be defined.

1. A *protocol interaction* is an action that corresponds to an interaction between the entity and another protocol entity.

   In the case of synchronous symmetric communication a protocol interaction is an action denoted by a message identifier, say m. A transition labeled by a protocol interaction semantically corresponds to the entity participating in a message exchange with exactly one other entity. The other entity must have the action m defined in its action set.

In the case of synchronous asymmetric communication or asynchronous communication, the protocol interactions can be further divided into two sets. A *message transmission* is an action denoted by a message identifier, say m, prefixed with a '–', i.e., $-m$. A transition labeled by a message transmission semantically corresponds to the entity sending message m to exactly one other protocol entity, or placing the message at the tail of a queue to exactly one other protocol entity. The other entity must have the action $+m$ defined in its action set. A *message reception* is an action denoted by a message identifier, say m, prefixed with a '+', i.e., $+m$. A transition labeled by a message reception semantically corresponds to the entity receiving message m from exactly one other protocol entity, or removing the message from the head of a queue from exactly one other protocol entity. The other entity must have the action $-m$ defined in its action set.

2. A *service primitive* is an action denoted by a service primitive identifier, say m, and, in the asymmetric case, is also prefixed by a '+' or '–', i.e., $+m$ or $-m$. However, an action m (symmetric case) or $+m$ or $-m$ (asymmetric case) will not appear in the action set of any other entity. A transition labeled with a service primitive action semantically corresponds to a message exchange (transmission or reception) between the entity specified as CFSM E and the environment.

3. An *internal action* is a designated action, τ (symmetric case), or an action with no sign, i.e., the action identifier is not prefixed by a '+' or '–' (asymmetric case). A transition labeled by an internal action semantically corresponds to an action that the entity specified as CFSM E can perform without any interaction with the environment or other CFSMs.

Some of the models in the sequel restrict service primitive actions and internal actions from the specification of the protocol entities.

The behaviour of a protocol can be modeled by a global state machine. A global state is a tuple consisting of the local state of each protocol entity (and the contents of each channel, if any exists). In the case of asynchronous communication, a transition in the

global state machine could represent one of its component CFSMs sending to or receiving from a channel. In the case of synchronous communication, a transition in the global state machine could represent two of its component CFSMs executing a synchronous message exchange. In both cases, a transition in the global state machine could represent one of its component CFSMs executing an internal action or a service primitive action.

Let G and H be global states. Define $G \rightarrow H$ if and only if H can be obtained from G by executing a single transition. Let $\rightarrow^*$ denote the transitive and reflexive closure of $\rightarrow$. We say that H is reachable from G if and only if $G \rightarrow^* H$. H is a reachable global state if and only if it is reachable from the initial global state.

A sequence of actions α accepted by a CFSM E is *executable* in a protocol P if there exists a sequence of actions accepted by the global state machine of P such that its subsequence restricted to actions defined in CFSM E is α.

A reachable global state of a protocol is a *deadlock state* if and only if there are no executable transitions defined at that state in the global state machine (and all channels are empty).

A protocol is *deadlock free* if and only if no reachable global state of the protocol is a deadlock state.

The following definitions relating to livelock applies to models in which the CFSM definition includes a designated set of states called final states.

A reachable global state of a protocol is a *livelock state* if and only if the state is not a deadlock state and no global final state is reachable from the state.

A protocol is *livelock free* if and only if no reachable global state of the protocol is a livelock state.

The following definitions relating to unspecified receptions apply to the synchronous asymmetric and asynchronous communication models.

A reachable global state of a protocol is an *unspecified reception (strong definition) state* if and only if

a) (asynchronous communication) there exists a message at the head of a queue and the corresponding message reception transition is not defined; or

b) (synchronous communication) there exists a message transmission transition at the corresponding local state of a protocol entity but the global transition corresponding to the synchronous exchange of that message is not defined at the global state of the protocol.

A reachable global state of a protocol is an *unspecified reception (weak definition) state* if and only if it is an unspecified reception (strong definition) state with respect to the reception of message *m* and all global states reachable from this state are also unspecified reception (strong definition) states with respect to the reception of the same message *m*.

A protocol is *unspecified reception free* if and only if no reachable global state of the protocol is an unspecified reception state.

The following definitions relating to channel overflow apply to the asynchronous communication model only.

A reachable global state of a protocol is a *channel overflow state* if and only if the content of a channel exceed the specified bound on the channel.

A protocol is *channel overflow free* if and only if no reachable global state of the protocol is a channel overflow state.

The *service specification* of a protocol is specified by a communicating finite state machine describing the service the protocol entities collectively provide to their environment. Thus, the action set of the service specification consists of the service primitive actions defined in its protocol entities.

3.3 Protocol Conversion Problem: Starting Point and Statement

This section formalizes the statement of the Protocol Conversion Problem. There are a number of characteristics of the problem statement: the properties of the given protocols P and Q, the specification of the conversion requirements, and the meaning of satisfying the conversion requirements.

Protocol Conversion Problem: Given two distinct protocols, P and Q, where the communication model for both protocols is synchronous symmetric, synchronous

asymmetric, or asynchronous; find a protocol converter C for the conversion system PQ such that the conversion system satisfies the given conversion requirements.

3.3.1 Protocols *P* and *Q*

We are given two distinct protocols, protocol P and protocol Q, modeled by sets of protocol entities, where each entity is specified by a CFSM:

case P1: $P = \{E0_P, RS_P, E1_P\}$ (Figure 12). The communication model among $E0_P$ and $E1_P$ is indirect and the communication model among $E0_P$ ($E1_P$) and RS_P is direct: synchronous symmetric or synchronous asymmetric. Protocol P provides the service specified by SS_P.

case P2: $P = \{E0_P, E1_P\}$ (Figure 13). The communication model among $E0_P$ and $E1_P$ is direct: synchronous symmetric, synchronous asymmetric or asynchronous. Protocol P provides the service specified by SS_P.

case Q1: $Q = \{E0_Q, RS_Q, E1_Q\}$ (Figure 16). The communication model among $E0_Q$ and $E1_Q$ is indirect and the communication model among $E0_Q$ ($E1_Q$) and RS_Q is direct: synchronous symmetric or synchronous asymmetric. Protocol Q provides the service specified by SS_Q.

case Q2: Q = $\{E0_Q, E1_Q\}$ (Figure 17). The communication model among $E0_Q$ and $E1_Q$ is direct: synchronous symmetric, synchronous asymmetric or asynchronous. Protocol Q provides the service specified by SS_Q.

To simplify the discussion we can consider the case of direct communication only. If protocol P is given as in **case P1**, or protocol Q is given as in **case Q1**, where the protocol entities communicate indirectly through a communication medium that provides the required service for the protocol, the protocol can be transformed into a protocol communicating directly as in **case P2** or **case Q2**, respectively.

For two protocol entities, E_1 and E_2, among which the communication model is synchronous symmetric, we can build a CFSM $E_1 \times E_2$ that corresponds to the joint behaviour E_1 and E_2, where $\times$ is the synchronous symmetric product [Chapter 5]. For two protocol entities, E_1 and E_2, among which the communication model is synchronous asymmetric, we can build a CFSM $E_1 \otimes E_2$ that corresponds to the joint behaviour E_1 and

E_2, where $\otimes$ is the synchronous asymmetric product [Chapter 5]. Accordingly, if $P = \{E0_P, RS_P, E1_P\}$ then $P' = \{E0'_P, E1_P\}$ where $E0'_P = E0_P \times RS_P$ (symmetric case) or $E0'_P = E0_P \otimes RS_P$ (asymmetric case). If $Q = \{E0_Q, RS_Q, E1_Q\}$ then $Q' = \{E0_Q, E1'_Q\}$, where $E1'_Q = E1_Q \times RS_Q$ (symmetric case) or $E1'_Q = E1_Q \otimes RS_Q$ (asymmetric case).

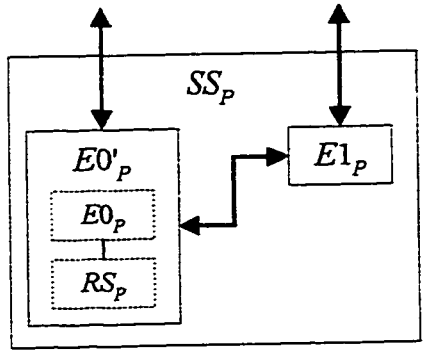

Figure 28: Protocol P'

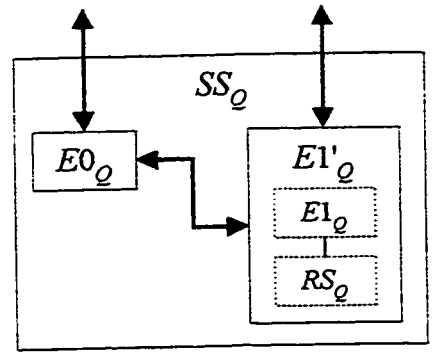

Figure 29: Protocol Q'

The protocol $P' = \{E0'_P, E1_P\}$ (Figure 28) is still a protocol that provides the supplied service (SS_P) of protocol P. The protocol $Q' = \{E0_Q, E1'_Q\}$ (Figure 29) is still a protocol that provides the supplied service (SS_Q) of protocol Q.

Therefore, we will assume that the protocols P and Q are two entity protocols communicating directly. Furthermore, the approaches proposed in the literature assume that the specifications of protocol P and protocol Q adhere to the same communication model. Therefore, we have three cases for protocols P and Q:

1. $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ where the communication model for both protocols is synchronous symmetric.
2. $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ where the communication model for both protocols is synchronous asymmetric.
3. $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ where the communication model for both protocols is asynchronous.

3.3.2 Specification of the conversion requirements

The goal is to find a protocol converter for the two incompatible protocols. There are two important tasks that a converter must perform: it must provide the correct synchronization

between the actions of each protocol, and it must provide the translation of the convertible messages of one protocol to the converted messages of the other protocol. Furthermore, the conversion system must provide the required service of the upper layer protocol.

The specifications of the original protocols alone are not sufficient to determine the requirements of a protocol converter, i.e., what is meaningful communication between the protocol entities. Therefore, it is also necessary to provide a specification of the conversion requirements in a formal style. Ideally, the formal specification will include the specification of the **synchronization requirements**: the requirements on the synchronization between the actions of each protocol; the specification of the **translation requirements**: the requirements on the translation of convertible messages of one protocol to converted messages of the other protocol; and the specification of the **service requirements**: the required service of the conversion system.

The approaches proposed in the literature define different formalisms for the specification of conversion requirements. They can be grouped into two categories: protocol-level conversion requirements and service-level conversion requirements.

Protocol-level conversion requirements

Protocol-level conversion requirements specify the synchronization and the translation requirements in terms of the peer interactions in which the converter participates. They do not specify the service requirements. Protocol-level conversion requirements are found by analysing the protocol-level functionality of the original protocols. These requirements can then be used to construct a converter for the given protocols.

Given two protocol specifications and the specification of protocol-level conversion requirements there are various approaches [RM91][PL92][Oku86][SD92][YCL90a] [CL90d] to construct the formal specification of a converter. The resulting converter would have to be validated against the service requirements of the conversion system.

Service-level conversion requirements

The service-level conversion requirements specify the service requirements in terms of the service primitives of the conversion system or the original protocols. They do not specify the protocol-level requirements. Service-level conversion requirements can be

further split into two categories. Some formal approaches use the required service specification of the conversion system as the conversion requirements for deriving a converter. Given two protocol specifications and the required service specification of the conversion system, there are various approaches [KH93][CL89a][CL89b][TBD95] [YL91] to construct the formal specification of a converter. Other approaches use a specification of a service interface adapter as the conversion requirements. Given two protocol specifications and a service interface adapter which defines how the communication services of the two interconnected protocols are related to one another, there are various approaches [Boc90][Oku90][YL92][KLNS91][KLNS93] to construct the formal specification of a converter. The required service specification can be used to find the service interface adapter. The converter resulting from an approach based on service-level conversion requirements will have to be reduced to satisfy the protocol-level requirements in such a way that the service-level requirements are not violated.

3.3.3 Satisfying conversion requirements

It is not only necessary to specify the conversion requirements formally, but there must also be a definition of what it means for a conversion system to satisfy the conversion requirements. Each approach proposed in the literature has a set of properties that the conversion system must satisfy in order to say that the conversion system satisfies the conversion requirements. A superset of these properties is given in this section.

Given $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ we want to find a protocol converter C for the conversion system $PQ = \{E0_P, C, E1_Q\}$. $E0_P$ and $E1_Q$ will be collectively referred to as the **participating entities**. $E1_P$ and $E0_Q$, the entities replaced by the converter, will be collectively referred to as the **involved entities**.

The following three properties refer to the behaviour of the conversion system $PQ = \{E0_P, C, E1_Q\}$ at its upper layer interface with respect to the conversion requirements specified by a required service specification for the conversion system CS:

1. **Safety Property:** Safety deals with the possible behaviour of the conversion system. PQ satisfies CS with respect to safety if a possible behaviour of PQ is a possible behaviour of CS.
2. **Conformity Property:** This property is stronger than the safety property. PQ satisfies CS with respect to conformity if a possible behaviour of PQ is a possible behaviour of CS and a possible behaviour of CS is a possible behaviour of PQ.
3. **Progress Property:** Progress deals with the desired behaviour of the conversion system and is only concerned with behaviour that is safe. PQ satisfies CS with respect to progress if PQ satisfies CS with respect to safety, and if CS can not deadlock with a particular environment then PQ can not deadlock with the same environment.

The following property refers to the behaviour of the conversion system $PQ = \{E0_P, C, E1_Q\}$ with respect to the conversion requirements specified by a protocol-level specification for the conversion system:

1. **Semantic equivalence:** The conversion system must satisfy the semantics described by the protocol-level conversion requirements. This property varies with each approach that uses protocol-level conversion requirements, and thus will be defined when each approach is reviewed.

The following five properties refer to the internal behaviour of the conversion system $PQ = \{E0_P, C, E1_Q\}$ with respect to the internal behaviour of the original protocols P and Q.

1. **External equivalence**: This property is important for the transparency of the conversion system. PQ satisfies external equivalence if any message sequence between $E0_P$ and C is accepted as if $E0_P$ was communicating with $E1_P$ and any message sequence between $E1_Q$ and C is accepted as if $E1_Q$ was communicating with $E0_Q$.

2. **Freedom from unspecified receptions**: Protocol PQ must be free from unspecified receptions, under the assumption that protocols P and Q are both free from unspecified receptions.

3. **Freedom from deadlock**: Protocol PQ must be free from deadlock, under the assumption that protocols P and Q are both free from deadlock.

4. **Freedom from channel overflow:** Protocol PQ must be free from channel overflow, under the assumption that protocols P and Q are both free from channel overflow. (applies to asynchronous model of communication only)

5. **Freedom from livelock:** Protocol PQ must be free from livelock, under the assumption that protocols P and Q are both free from livelock. (applies to models that specify a final state set in their CFSM definition).

It is possible that there exists more than one protocol converter C such that $PQ = \{E0_P, C, E1_Q\}$ satisfies the conversion requirements. It is desirable that a solution to the protocol conversion problem finds the maximal converter for the two protocols with respect to the conversion requirements. It is also possible that there does not exist a protocol converter C such that $PQ = \{E0_P, C, E1_Q\}$ satisfies the conversion requirements.

A general solution to the protocol conversion problem should be able to produce a protocol converter that is maximal, or provide an indication that no such converter exists. It is important to note that the non-existence of a protocol converter for protocols P and Q is relative to the given conversion requirements and the definition of what it means for a conversion system to satisfy the conversion requirements. The notions of hard mismatch and soft mismatch, introduced in Chapter 2, can be interpreted in this context. The satisfaction of the conversion requirements should define the boundary between a hard and soft mismatch.

Chapter Four

Formal Methods for Protocol Conversion

Since Green [Gre86] suggested that the formal methods used in protocol engineering [BS80][GTN93] might form the basis for a "deeper and more systematic calculus of conversion", several approaches to the synthesis of protocol converters based on formal methods have been proposed in the literature. In this chapter we will review the subset of the approaches that satisfy the following criteria:

- The formal model is based on the CFSM model [BZ83] for protocol specification;
- The approach is algorithmic;
- The approach proposes to solve the problem specified in Chapter 3 where the architecture includes a single converter implemented in a gateway;

[Boc90][CL89a][CL89b][CL90d][KH93][KLNS91][KLNS93][Oku86][Oku90][PL92] [RM91][SD92][TBD95][YCL90a][YL91][YL92].

Other approaches not reviewed in this chapter include non-algorithmic or non-formal approaches found in [Aue89][Aue90][CL88][Lam86][Lam88a][Lam88b][SD93][SJR95]; approaches to synthesize a protocol converter for an end node found in [Cal92][CL90a]; approaches to synthesize two process protocol converters for a half-gateway architecture found in [HCLY94][JL95][SL89][SL90][SL91][YCL90b]; and approaches based on other formal models found in [AM91][TG92]. A few formal approaches are proposed for other interconnection problems including the protocol complementation problem [CL90c][DD95][DGDS92]. Published surveys on protocol conversion can be found in [CL90b] and [PL93].

4.1 Protocol Converter Synthesis from Protocol-level Conversion Requirements

The approaches discussed in this section [CL90d] [Oku86] [PL92] [RM91] [SD92] [YCL90a] are based on protocol-level conversion requirements. Given protocols $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ each approach constructs the protocol converter from the involved entities, $E1_P$ and $E0_Q$, and the protocol-level conversion requirements. As stated in Chapter 3, protocol-level conversion requirements specify the synchronization requirements of the converter and the translation requirements of the converter in terms of the peer protocol messages of the involved entities.

These approaches benefit from the design flexibility the conversion requirements provide with respect to the choice of end-to-end significance versus the local significance of peer protocol messages. Unfortunately, these approaches also suffer from the following problems. The specification of the conversion requirements in terms of peer protocol messages can be difficult to obtain, and difficult to validate. This is because the requirements are based on an intuitive understanding of the protocols. Since the approaches are very sensitive to the way the functionality is represented in the conversion requirements, it is difficult to determine if a failure to find a converter was due to a hard mismatch or due to the specification of the conversion requirements. Finally, if the required service specification of the conversion system is known, there is no way to input the service requirements into these approaches. Therefore, the resulting conversion system constructed by one of these approaches would have to be verified against the service specification after the converter is constructed.

In fact, the service provided by the original protocols and the final conversion system are not considered at any stage of these approaches. In the specification models used by the approaches reviewed in this section, a specification of a protocol entity does not include transitions labeled by service primitives.

4.1.1 Converter construction from executable protocol traces

In [RM91] an approach to construct a protocol converter from executable protocol traces is proposed. The communication model is asynchronous over unbounded channels. The

specifications of the protocol entities are deterministic; i.e., they contain no internal action transitions.

The protocol-level conversion requirements used in [RM91] specify the translation requirements separately from the synchronization requirements. The translation requirements are specified by a message relationship function that maps the peer reception actions of $E1_P$ to the peer transmission actions of $E0_Q$, and vice versa. It is assumed that semantically and syntactically different actions in a protocol are named uniquely, and that the mapping is for actions under the same protocol phases of the two protocols.

The *Message Relationship Function* Ψ is defined as follows: given two peer interactions $+m_1$ defined in $E1_P$ ($E0_Q$), and $-m_2$ defined in $E0_Q$ ($E1_P$), function Ψ maps the message $+m_1$ to the message $-m_2$, if m_1 is convertible or partially convertible to m_2. In [RM91] a protocol analysis technique is briefly described that may simplify the otherwise difficult task of determining the message relationship function.

A *converter specification* M_K for a protocol converter C in protocol $\{E0_P, C, E1_Q\}$ consists of action sets R, S, and N. R consists of peer reception actions that are mapped by Ψ to the set S of peer transmission actions, i.e., $\psi: R \rightarrow S$. R is called the *convertible set of receptions* and S is called the *converted set of transmissions*. N, called the *nonconvertibles*, consists of other peer reception and transmission actions. These sets satisfy the following: $R \cap S = \varnothing$; $N \cap S = \varnothing$; $R \cap N = \varnothing$.

The synchronization requirements of the conversion requirements are specified by a *semantic specification* S_K that specifies a sequence of protocol functions in the overall protocol execution. The semantic specification determines whether functions of the original protocols are partially interconnected or not interconnected. [RM91] does not provide a formal mechanism for specifying the semantic specification. It is stated in a natural language as a list of functions to be performed in the specified order by the converter. Deriving the semantic specification for the conversion system is based on the designers understanding of the protocols and the functions that must be performed by the converter.

Before we discuss what it means for the conversion system to satisfy the conversion requirements, some concepts must be introduced.

The set of **executable traces** of a protocol entity in a protocol is defined to be the set of all suffixes of all executable sequences of the protocol entity in the protocol.

A **legal trace** t is defined as an executable trace of length ≥ 1 of H in protocol $\{E0_P, H, E1_Q\}$, where $H = E1_P \times E0_Q$, provided it satisfies the following conditions:

1. The actions of t belong to sets R, S, or N, and are constrained by M_K and S_K.
2. For every trace action $t(i) \in R$ there exists one and only one action $t(j) = \psi(t(i))$, where $j > i$. This means that action $t(i)$ cannot be repeated in the same trace.
3. For trace actions $t(i_1)$, $t(i_2) \in R$ where $i_2 > i_1$, there exists actions $t(j_1) = \psi(\ t(i_1))$ and $t(j_1) = \psi(\ t(i_2))$, where $j_2 > j_1$. This means that the conversion of messages is done obeying the FIFO nature of the queues.
4. No prefix of a legal trace is a legal trace.
5. The first legal trace begins from the initial state. All legal traces may begin only from the first state of a legal trace or from the last state of a legal trace.

A converter C satisfies **semantic equivalence** if the set of legal traces T_C of C in protocol $\{E0_P, C, E1_Q\}$ also satisfies $T_C \subseteq T_H$, where T_H is the set of legal traces of H in protocol $\{E0_P, H, E1_Q\}$, where $H = E1_P \times E0_Q$.

A conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the protocol-level conversion requirements M_K and S_K if it satisfies external equivalence, semantic equivalence, freedom from deadlock and freedom from unspecified receptions (weak).

Given protocols P and Q that are deadlock free and unspecified reception free, the conversion algorithm proposed in [RM91] searches the state space of $E1_P \times E0_Q$ to obtain the set of legal traces for M_K and S_K. All states or transitions not included in a legal trace are removed from $E1_P \times E0_Q$ to obtain the converter C. Note that C may contain

executable traces that are not legal. For example, if a transition defined at the initial state is never the first transition in a legal trace then any trace starting with that transition is not a legal trace. However, if a legal trace includes the transition elsewhere then the transition will not be removed from $E1_P \times E0_Q$.

[RM91] provides the conditions under which the resulting conversion system is deadlock free and unspecified reception free. Since the input protocols are free from these errors, and since C was obtained from the involved entities, the detection of these errors is straightforward. If the resulting conversion system does not satisfy the protocol-level requirements then a valid converter is not possible for the given conversion requirements. Otherwise, the converter constructed provides the maximum possible conversion that satisfies the conversion requirements.

The computational complexity of finding the set of legal traces in $E1_P \times E0_Q$ is $O((|R|+|N|)!)$. The computational complexity of finding $E1_P \times E0_Q$, reducing it based on the set of legal traces, and verifying whether it is free from errors is linear in the size of $E1_P \times E0_Q$.

The application of this approach is limited by the specification of the conversion requirements. The message mapping function maps a single convertible message to a single converted message; and the definition of a legal trace imposes that a convertible message of one protocol be translated immediately to the converted message of the other protocol, without the possibility of reordering the converted messages. For many practical protocols it is possible that the messages produced in one protocol need to be mapped into a sequence of messages, or re-ordered after they are converted. The immediate mapping of one message from one protocol to another message from the other protocol indicates full interconnection of function. Full interconnection requires that the semantics be the same for both protocols.

4.1.2 Converter construction from valid closed paths

In [PL92] an approach to construct a protocol converter from valid closed paths is proposed. The communication model is asynchronous over unbounded channels. The specifications of the protocol entities are deterministic. Furthermore, this approach

assumes that the protocols are designed to continue forever, executing cycles of transitions through the initial state.

The protocol-level conversion requirements used in [PL92] specify the translation requirements and the synchronization requirements together as a set of ordered mapping sets. An *ordered mapping set* is similar to the message relationship function in [RM91], except that it allows multi-message translations; i.e., a message reception action can be mapped to a sequence of message transmission actions.

Given a protocol $P = \{E0_P, E1_P\}$ and a protocol $Q = \{E0_Q, E1_Q\}$ a *set of ordered mapping sets* $F = \{F_1, F_2, \ldots, F_N\}$ formally specifies the protocol-level conversion requirements where:

1. An ordered mapping set F_i contains only convertible/converted messages of $E1_P$ and $E0_Q$.
2. An ordered mapping set F_i contains two or more convertible/converted messages of $E1_P$ and $E0_Q$.
3. If an action $+m$ is convertible to an action $-m'$ then both $+m$ and $-m'$ are contained in the same ordered mapping set F_i with $+m$ preceding $-m'$ in F_i.
4. If an action $+m$ is convertible to a sequence of actions $-m_1-m_2\ldots-m_n$ then $+m$, $-m_1$, $-m_2,\ldots$, and $-m_n$ are contained in the same ordered mapping set F_i with $+m$ preceding $-m_1$, $-m_2,\ldots$, and $-m_n$ in F_i.
5. The order imposed on actions by $E1_P$ and $E0_Q$ is obeyed by the actions in each ordered mapping set F_i.
6. Convertible messages with different functionality are assigned to different ordered mapping sets.
7. An ordered mapping set is independent from another ordered mapping set.

Before we discuss what it means for the conversion system to satisfy the conversion requirements, some concepts must be introduced.

A **closed path** p in a CFSM E is a sequence of transitions from the initial state of E to the initial state of E such that:

1. the initial state does not appear elsewhere in the closed path, but any other state may appear zero or more times,
2. a transition appears in the closed path at most once.

A closed path p is a **valid closed path** with respect to $F = \{F_1, F_2, \ldots, F_N\}$ if and only if one of the following two conditions is satisfied:

1. p does not contain a transition labeled by an action in an ordered mapping set.
2. Every action in F_i $(i = 1, \ldots, N)$ labels a transition in p and appears in p in the same order as in F_i.

   Note that p can satisfy more than one F_i and can satisfy the same F_i more than once.

The converter C satisfies **semantic equivalence** if every closed path of C is a valid closed path with respect to F, and every ordered mapping set in F is satisfied by at least one closed path in C.

The conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the protocol-level conversion requirements F if it satisfies external equivalence, semantic equivalence, freedom from deadlock and freedom from unspecified receptions (strong).

Given protocols P and Q that are deadlock free and unspecified reception free, the algorithm proposed in [PL92] searches the state space $E1_P \times E0_Q$ for the set of valid closed paths satisfying F. The valid closed paths are then combined in such a way to form a converter C that contains all the valid closed paths of $E1_P \times E0_Q$ and no invalid closed paths of $E1_P \times E0_Q$. This is accomplished by merging the initial state (and final state) of each closed path to form a very large, non-deterministic, finite state machine. The resulting finite state machine is minimized by merging equivalent states [PL92] to produce a CFSM C. Note that this step has the potential to suffer from state space explosion if the number of valid closed paths is large.

C is reduced as necessary to ensure the resulting conversion system is deadlock free and unspecified reception free. Finally, C is verified against the conversion requirements. If C satisfies the conversion requirements then C is a protocol converter; otherwise, no converter exists for the given conversion requirements.

This approach is algorithmic, although the algorithms are not provided in [PL92]. The computational complexity depends on the number of valid closed paths. No further discussion of the complexity is provided in [PL92].

The application of this approach is limited by the specification of the conversion requirements. The nonconvertible messages are not included in the conversion requirements. It is possible that nonconvertible messages are important to the specification of the synchronization requirements.

4.1.3 Converter construction from a conversion seed: synchronous

In [Oku86] an approach to construct a protocol converter from a conversion seed is proposed. The communication model is synchronous asymmetric. The specifications of the protocol entities are deterministic. Furthermore, the approach assumes that the protocols provided are designed to continue forever, executing cycles of transitions through the initial state.

The protocol-level conversion requirements used in [Oku86] specify the translation requirements and the synchronization requirements together as a conversion seed. A *conversion seed* is a CFSM that specifies how the convertible messages of one protocol are to be mapped to the converted messages of the other protocol, and also the order in which significant messages are to be sent and received by the converter. The significant messages include the convertible messages, converted messages and any nonconvertible message whose relative order may impact the desired behaviour of the conversion system with respect to the interconnection of functions. Any convertible message, or sequence of messages, which is received by the converter should appear before the converted message, or sequence of messages, which is sent by the converter. The conversion seed must contain all allowed orderings of the significant messages since the approach does not allow any other orderings in the final converter.

Before we discuss what it means for the conversion system to satisfy the conversion requirements, some concepts must be introduced.

A sequence of state transitions defined in a CFSM E is a **loop** if the sequence of transitions starts at the initial state of E and ends at the initial state of E.

A loop defined in a CFSM E is a **valid loop** with respect to conversion seed X if and only if the subsequence of its associated actions which contains only actions defined in X is accepted by X.

The converter C satisfies **semantic equivalence** with respect to conversion seed X if and only if all loops of C are valid loops.

A protocol entity CFSM E is **effective** in a protocol P if every sequence of actions accepted by E is an executable sequence of E in P.

For two CFSMs E_1 and E_2, E_1 is a **reduced-CFSM** of E_2 if

1. for every path in E_1 there is a corresponding path in E_2 that has the same sequence of actions, and
2. if a path in E_2 that corresponds to some path in E_1 can be extended by a message reception action, then the corresponding path E_2 can also be extended by the same message reception action.

For two CFSMs E_1 and E_2, E_1 is a **sub-CFSM** of E_2 if the state set of E_1 is a subset of the state set of E_2, the initial state of E_1 is the initial state of E_2, the action set of E_1 is a subset of the action set of E_2 and a state transition (s_1, σ, s_2) in E_2 is defined in E_1 whenever s_1, s_2 and σ are defined in E_1.

The conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the protocol-level conversion requirements X if it satisfies external equivalence, semantic equivalence, freedom from deadlock and freedom from unspecified receptions (strong).

[Oku86] shows that if $E1_P$ is unspecified reception free and effective in protocol P and if $E0_Q$ is unspecified free and effective in protocol Q then a CFSM C is a converter for $\{E0_P, C, E1_Q\}$ with respect to conversion seed X if and only if

1. C is a sub-CFSM of $E1_P \times E0_Q \times X$.

   This property is a necessary and sufficient condition for external equivalence and semantic equivalence.

2. C is a reduced-CFSM of $E1_P \times E0_Q$.

   This property is a necessary and sufficient condition for C to be unspecified reception free in $\{E0_P, C, E1_Q\}$.

3. C has no states with no outgoing transitions defined.

   This is necessary for $\{E1_P, C, E1_Q\}$ to be deadlock free.

Because there are a finite number of sub-CFSMs of $E1_P \times E0_Q \times X$ and because conditions 2 and 3 are decidable, an algorithm exists that will find a converter and return nil if one does not exist. The algorithm given in [Oku86] recursively removes from $E1_P \times E0_Q \times X$ any state and its incoming and outgoing transitions that either has no outgoing transitions defined or does not have sufficient reception transitions defined to satisfy condition 2. If the state space is empty at the completion of the algorithm then a converter for protocols P and Q and conversion seed X does not exist.

The computational complexity of the algorithm is linear in the size of $E1_P \times E0_Q \times X$. [Oku86] proposes an efficient approach for the synthesis of the converter for protocols P and Q, provided a suitable conversion seed can be found. The conversion seed does not suffer from the limitations of the previous approaches' specification of conversion requirements: in a conversion seed the specification of the synchronization requirements and the translation requirements is very flexible. Therefore, translation can be specified between single messages or sequences of messages, with or without reordering. Furthermore, the synchronization requirements can be in terms of nonconvertible

messages or convertible/converted messages. Unfortunately, this flexibility makes a correct conversion seed very difficult to define.

4.1.4 Extensions to Okumura's conversion seed approach

In [SD92] an extension to the conversion seed approach [Oku86] is proposed. The extension attempts to reduce the possible state space explosion inherent when finding the product of CFSMs, by reducing the involved entities of the input protocols before applying the conversion seed approach [Oku86] to find a converter.

The idea of the reduction is to merge states that are connected only by insignificant actions (i.e., actions not defined in the conversion seed X) into composite states and remove the insignificant transitions between them. A reduced converter is synthesized from the reduced involved entities and the conversion seed X using Okumura's conversion seed algorithm [Oku86]. Finally, the reduced converter is expanded using three decomposition rules [SD92] to obtain the complete converter C. The converter C obtained by this approach is functionally equivalent to the converter obtained by Okumura's approach with the same input. This approach lowers the computational complexity and contains the state space explosion to some extent in comparison to [Oku86].

Many communication protocols go through different phases, performing distinct functions in each phase. Based on that observation, [YCL90a] proposes a four-step approach for finding a protocol converter, which applies Okumura's conversion seed approach [Oku86]. If the protocols P and Q can be decomposed into distinct functions then the difficulty of determining one conversion seed for protocols P and Q is reduced to finding a less complex conversion seed for each pair of compatible functions of protocols P and Q.

The first step is the decomposition of protocols P and Q into distinct functions. A function F of a protocol can be represented by a set of entities $\{F1, F2\}$, where each entity is specified by a CFSM. A function of a protocol performs a particular task for that protocol. The final states of each entity are the exit points where other functions can be entered. A dependency graph is used to define the order in which the distinct functions are executed by the protocol. The decomposition of the protocol into distinct functions is not

discussed in [YCL90a], however, the conditions for a protocol to be *decomposable* are provided. Once the protocols are decomposed into their distinct functions, the compatible functions of the two protocols can then be recognized.

The second step is the construction of a converter for each pair of compatible functions of the two protocols using the conversion seed approach [Oku86]. Therefore, a set of conversion seeds is required as part of the protocol-level conversion requirements for this step. One conversion seed is required for each pair of compatible functions.

The third step involves merging local functions (incompatible functions) of the two protocols with the functional converters to form a single converter. Therefore, a dependency graph that defines the order in which the functions are executed by the conversion system protocol is needed as part of the protocol-level conversion requirements for this step. A merge algorithm is described in [YCL90a] that merges the set of functional converters, and any local functions of the original protocols, according to the dependency graph.

Finally, all states and transitions that cause deadlock or unspecified receptions are removed from the converter. If a local function or functional converter is removed from the converter with this step, then a converter that satisfies the conversion requirements does not exist.

4.1.5 Converter construction from a conversion seed: asynchronous

In [CL90d] an approach to construct a protocol converter from a conversion seed is presented. The communication model is asynchronous over bounded channels. The protocol-level conversion requirements are specified as a single CFSM conversion seed X as in [Oku86].

The conversion system $PQ = \{E0_P, C, E1_Q\}$ satisfies **semantic equivalence** with respect to conversion seed X if and only if for every α, such that α is an executable path of C in PQ, the subsequence of α that contains only actions defined in X is accepted by X.

The conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the protocol-level conversion requirements X if it satisfies external equivalence, semantic equivalence, freedom from

deadlock, freedom from unspecified receptions (strong), and freedom from channel overflow.

This approach transforms the problem of finding a protocol converter into a protocol validation problem [CL90d]. The first step is to modify the involved entities $E1_P$ and $E0_Q$, and the conversion seed X, to obtain three new entities $E1'_P$, $E0'_Q$, and X' such that the three new entities are considered to be communicating within a protocol $\{E1'_P, X', E0'_Q\}$. The communication model for this protocol is also asynchronous over bounded channels.

[CL90d] shows that if both P and Q are deadlock free, unspecified reception free and channel overflow free, and that if $E1_P$ and $E0_Q$ have no states with both peer message transmission and reception transitions defined, then there exists a converter with respect to conversion seed X if and only if there is a subgraph SG of the global state graph of protocol $\{E1'_P, X', E0'_Q\}$, GSG, that satisfies the following conditions:

1. SG is not empty.
2. SG contains the initial state of GSG.
3. SG has no deadlock states, unspecified reception states or channel overflow states.
4. if a path in GSG that corresponds to some path in SG can be extended by a message reception action, then the corresponding path SG can also be extended by the same message reception action.

The global state graph of protocol $\{E1'_P, X', E0'_Q\}$, GSG, is ensured to be finite because the channels are bounded. GSG can be found using many existing protocol validation techniques [GY84][II83][KWN86][RW82][Sab88][Wes78][Wes86]. Because there are a finite number of subgraphs of GSG, and because conditions 1 to 4 are decidable, a terminating algorithm exists that will find SG if one exists. Once SG is found it must be modified to reverse the modifications made to the original protocol entities. A sketch of the algorithm is presented in [CL90d] where the complexity is not discussed.

4.2 Protocol Converter Synthesis from the Required Service Specification

The approaches discussed in this section are based on service-level conversion requirements in the form of the required service specification of the conversion system *CS*. Given protocols $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ each approach constructs the protocol converter from the participating entities, $E0_P$ and $E1_Q$, and *CS*. As stated in Chapter 3, service-level conversion requirements in the form of the required service specification specify the service requirements of the resulting conversion system in terms of the service primitives of the participating entities.

The synchronization requirements and translation requirements of the converter are not included in the specification of the conversion requirements. Therefore, the approaches reviewed in this section will find a converter that satisfies the service requirements of the conversion system with respect to safety, and possibly progress, if one exists. The converter will have to be reduced to satisfy the translation requirements and synchronization requirements. The safety property will hold for a reduced converter obtained by removing states and transitions from a converter satisfying the safety property; however, the progress property will not necessarily hold for a reduced converter obtained by removing states and transitions from a converter satisfying the progress property. Therefore, if the converter is reduced by removing states and transitions, the progress property will have to be reevaluated. Another limitation of the approaches reviewed in this section is that the communication model is synchronous.

4.2.1 Synthesis of protocol converters using submodule construction

In [KH93] an approach to construct a protocol converter using extensions to the submodule construction approach [MB83] is proposed. The submodule construction problem is extended to the synchronous asymmetric communication model. The specifications of the protocol entities are deterministic. In this approach, *CS* is also deterministic. The progress criterion proposed in [KH93] is based on the notion of task completion. The approach requires an additional specification as part of the conversion requirements called the task-completing sublanguage of *CS*. The task completing

sublanguage of *CS*, *TC*, is a subset of L(*CS*) such that $\varepsilon \notin TC$. *TC* specifies the set of tasks in *CS*.

Before we discuss what it means for the conversion system to satisfy the conversion requirements, some concepts must be introduced.

A **task completing trace** of $PQ = \{E0_P, C, E1_Q\}$ is a non-null executable path in *PQ* that ends with an action defined in *CS* such that the subsequence of the executable path when restricted to actions defined in *CS* is an element of *TC*.

Progress is any behaviour of the conversion system that does not preclude it from completing at least one task completing trace.

PQ is making **infinite progress** with respect to *CS* and *TC* if and only if every executable path in *PQ* is a proper prefix of a task completing trace.

A conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the service-level conversion requirements *CS* and *TC* if it satisfies the safety property, freedom from unspecified receptions (strong) and infinite progress.

The first step of the approach proposed in [KH93] is to find a CFSM *C* such that $PQ = \{E0_P, C, E1_Q\}$ satisfies the safety property. This is accomplished with the specific formulation of the submodule construction problem for the synchronous asymmetric communication model [KH93]. The resulting *C* is maximal with respect to the safety property.

In [KH93] an algorithm is provided that takes as input a CFSM *C* such that $\{E0_P, C, E1_Q\}$ satisfies the safety property, and removes the minimal number of paths from *C* to obtain a solution *C′* such that $\{E0_P, C', E1_Q\}$ satisfies the safety property and freedom from unspecified receptions. The main idea of the algorithm is to obtain the global state machine of $\{E0_P, C, E1_Q\}$, identify the unspecified reception states in $\{E0_P, C, E1_Q\}$, and remove the minimal number of paths from *C*, i.e., the ones that are responsible for unspecified receptions, to obtain *C′*.

Another algorithm provided in [KH93] takes as input a CFSM C such that $\{E0_P, C, E1_Q\}$ satisfies the safety property, and removes the minimal number of paths from C to obtain a solution C' such that $\{E0_P, C', E1_Q\}$ satisfies the safety property and infinite progress. The main idea of the algorithm is to find all the task completing traces of $\{E0_P, C, E1_Q\}$, identify the set of executable paths in $\{E0_P, C, E1_Q\}$ that are not proper prefixes of a task completing trace, and remove the corresponding paths from C to obtain C'. This step must be repeated interatively until C does not change.

The algorithm that removes unspecified receptions may result in a converter that violates the progress condition. The algorithm that removes the infinite progress violations may result in a converter with unspecified receptions. Therefore, to find a protocol converter that satisfies the conversion requirements, the solution is to iteratively apply the algorithms to the CFSM that satisfies the safety property until the outcome does not change.

The computational complexity of the algorithm to find the safe solution is exponential in the size of $E0_P \times CS \times E1_Q$. The computational complexity of the algorithm to remove unspecified receptions is exponential in the size of the safe solution. There is no proof of termination for the algorithm to remove progress violations.

4.2.2 The quotient approach

In [CL89a][CL89b] a solution to the quotient problem is proposed and then used to construct a protocol converter. The communication model is synchronous symmetric. The specification of the protocol entities may be non-deterministic, i.e., transitions labeled by internal actions may be defined. The non-determinism in the specification of the protocol entities is assumed to be fair: any transition that is repeatedly enabled will eventually occur. CS may also be non-deterministic as long as it satisfies the normal form property [CL89a][CL89b]. The non-determinism in the service specification is not assumed to be fair.

A conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the service-level conversion requirements CS if it satisfies the safety property and the progress property.

Let A be a service specification and let B specify one component (entity) of an implementation. B has two interfaces: one is external and is the same as the interface of A, and one is internal, comprising of a set of possible interactions with another component C. The goal is to specify C, which interacts with B via its internal interface, so that the behaviour observed at B's external interface *satisfies* the service defined by A. The **quotient problem** is the problem of finding C such that $\{B, C\}$ satisfies A. $\{B, C\}$ satisfies A if the safety property is satisfied and the progress property is satisfied. [CL90a][CL90b] presents an algorithm that produces a specification of C only if one exists. [CL90a][CL90b] uses the algorithm to solve the protocol conversion problem where $A = CS$, and $B = E0_P \otimes E1_Q$.

[CL90a][CL90b] proposes an algorithm that finds C such that $\{B, C\}$ satisfies A with respect to safety. The algorithm is essentially the same as the submodule construction algorithm [MB83][HU96]. The CFSM specification of C is built inductively starting with the initial state. The result is a specification of C with the largest trace set consistent with the safety of $\{B, C\}$.

Another algorithm is proposed in [CL90a][CL90b] that removes from the safe solution, C, all states that are responsible for possible progress violations by $\{B, C\}$. The states that are removed do not have sufficient outgoing transitions to prevent a possible progress violation. If there are any reachable states left at the end of the process then C (a quotient) is found such that $\{B, C\}$ satisfies A with respect to safety and progress. Furthermore, it is the maximal quotient in the sense that for any other quotient D, any executable path of D in $\{B, D\}$ is an executable path of C in $\{B, C\}$. If the initial state is removed by the algorithm then no quotient exists.

The computational complexity of the safety algorithm is exponential in the size of $A \times B$; the computational complexity of the progress algorithm is polynomial in the size of the safe solution output from the safety algorithm.

The constructed quotient has the largest possible trace set of any quotient, so it is likely that it will contain extra states and extra transitions that do not contribute to system progress. Furthermore, the constructed quotient will likely contain states and transitions

that violate the synchronization requirements and translation requirements of the converter.

4.2.3 Optimized converter

In [TBD95] an approach to find an optimized protocol converter from the participating entities, the involved entities, and *CS* is presented. A protocol converter is optimized if it does not contain any extra states and extra transitions that do not contribute to system progress. The communication model is synchronous symmetric. The protocol entities may be nondeterministic. *CS* may also be non-deterministic.

A conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the service-level conversion requirements *CS* if it satisfies the safety property, the progress property and external equivalence.

[TBD95] claims that there is a deterministic converter *C* such that $PQ = \{E0_P, C, E1_Q\}$ satisfies *CS* if and only if there exists a CFSM *H* with $L(H) \subseteq L(E0_P \times E1_Q)$ such that

1. $\{H'\}$ satisfies *CS* with respect to safety and progress, where H' is *H* except all transitions labeled by actions not defined in *CS* are replaced by internal transitions.
2. For any $\sigma \in L(H)$, if $\sigma.m \in L(E0_P \times E1_Q)$, where *m* is a peer interaction or internal action, then $\sigma.m \in L(H)$.
3. For any $\sigma, \sigma' \in L(H)$ such that the subsequence of σ consisting of only peer interactions is the same as the subsequence of σ' consisting of only peer interactions, if $\sigma.m$ and $\sigma'.m \in L(E0_P \times E1_Q)$ then either both $\sigma.m$ and $\sigma'.m \in L(H)$ or both $\sigma.m$ and $\sigma'.m \notin L(H)$.

In [TBD95] an algorithm is presented to find such a CFSM *H*. The algorithm reduces a product machine constructed from $E0_P$, $E1_Q$, and *CS* by eliminating all paths that violate conditions 2 and 3 above. In addition, states and transitions that violate the safety property or progress property are identified and removed while retaining conditions 2 and 3. If the initial state is removed during this process then no converter exists. Otherwise the result is the CFSM *H*.

The projection of H over its set of peer interactions results in a CFSM C' such that $\{E0_P, C', E1_Q\}$ satisfies CS with respect to safety and progress. To find a C such that $\{E0_P, C, E1_Q\}$ satisfies CS, the product of C' with involved entities $E1_P$ and $E0_Q$, (with their service primitives and internal actions removed by projection), is constructed. It is shown in [TBD95] that the result is a protocol converter such that the conversion system satisfies the conversion requirements. The complexity of the algorithm is exponential in the size of $E0_P \times CS \times E1_Q$.

This approach differs from the quotient approach [CL90a][CL90b] in that it allows the service specification to be nondeterministic and not restricted to the normal form property [CL90a][CL90b]. Furthermore, this approach provides one algorithm to find a converter that satisfies safety and progress as opposed to providing an algorithm with two separate phases. Finally, this approach goes further to find a converter such that the conversion system satisfies external equivalence. If the involved entities of the original protocols do not contain extra states and transitions then the resulting converter will also not contain extra states and transitions. Therefore, the converter constructed by this approach is optimal. Recall, however, that the converter constructed by this approach will likely contain states and transitions that violate the synchronization requirements and translation requirements of the converter.

4.2.4 Constructing a protocol converter with guaranteed service

In [YL91] an approach to construct a protocol converter that considers the conformity property is proposed. The communication model is synchronous symmetric. The protocol entities are deterministic and contain the specification of a designated subset of the state set called final states. In this approach, CS is deterministic.

A conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the service-level conversion requirements CS if it satisfies the conformity property, freedom from deadlock and freedom from livelock.

The approach outlined in [YL91] first removes transitions and states from $G = E0_P \times CS \times E1_Q$ until G is properly synchronized; and contains no livelock states or

deadlock states. If $G = E0_P \times CS \times E1_Q$ satisfies $G = E0_P \times \text{Proj}_\Delta(G) \times E1_Q$, where Δ is the set of peer interactions defined in G, then G is **properly synchronized**.

The next step is to verify that $CS = \text{Proj}_\Delta(G)$, where Δ is the set of actions defined in CS. If the verification passes then let $C = \text{Proj}_\Delta(G)$, where Δ is the set of peer interactions defined in G. The complexity of the algorithm is exponential in the size of $E0_P \times CS \times E1_Q$.

There is no discussion [YL91] as to what it means if the reduction of G results in a nil CFSM or what is means if the verification step fails. [YL91] does not discuss progress in terms of the service provided, however, the liveness condition is a requirement for the conversion system protocol PQ. This approach enforces that the conversion system to satisfy the conformity property in order to satisfy the conversion requirements. Since CS is deterministic this is sufficient property for the satisfaction the progress property.

4.3 Protocol Converter Synthesis from a Service Interface Adapter

Service-level interconnection is one approach to the interconnection of networks at a gateway. This is accomplished with a service interface adapter within the gateway. The approaches in this section consider the problem of finding a protocol converter, however, in each case the significant first step is the construction of a service interface adapter which is then used in the construction of a protocol converter. Recall from Chapter 3, a service interface adapter provides the synchronization and translation of the service primitives of the involved entities such that the conversion system provides the required service. Therefore, the conversion requirements for the approaches reviewed in this section are specified as the required service of the conversion system. As with the approaches reviewed in the previous section; they do not consider the synchronization requirements and translation requirements of the converter.

4.3.1 Bochmann's observation

It was observed in [Boc90] that once the interconnection problem is solved at the service level, a protocol converter for interconnection at the protocol level can be derived automatically from the given protocol specifications and the service interface adapter.

More specifically, the protocol converter is constructed by the composition of the entities in the dotted box of Figure 30.

A comparison of Figures 30 and 31 shows that the subsystem composed of the layer-*N* entity, the layer-*M* entity, and the service interface adapter (*SIA*) within the gateway is a protocol converter; although the protocol converter may have additional properties. This observation leads to the automatic derivation of a specification for a protocol converter from the given protocol specifications to be interconnected and the definition of a service interface adapter (*SIA*). *SIA* defines how the communication services of the two interconnected systems are related to one another.

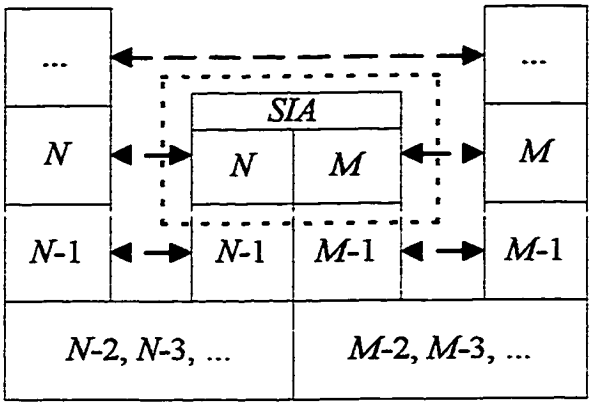

Figure 30: Conversion at the service level

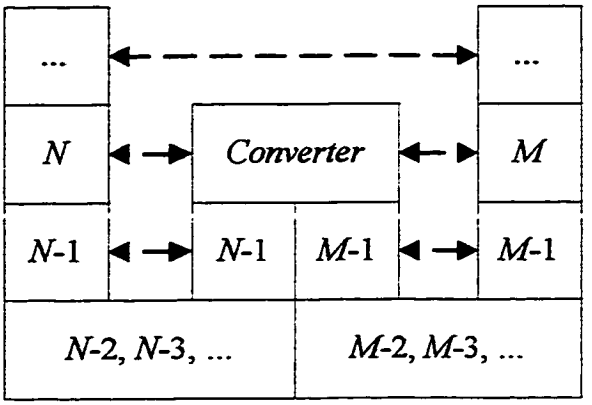

Figure 31: Conversion at the protocol level

4.3.2 Okumura's approach

Okumura [Oku90] uses the observation made in [Boc90] and presents a formal approach to construct a service interface adapter and then a protocol converter. The communication model is asynchronous or synchronous asymmetric. The specification of the entities may be nondeterministic. The specification of *CS* may also be nondeterministic.

A conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the service-level conversion requirements *CS* if it satisfies the safety property.

The first step in the approach proposed in [Oku90] is to generate a service interface adapter for the conversion system. That is, find a service interface adapter *SIA* so that $\{SS_P, SIA, SS_Q\}$ satisfies *CS*. This is an instance of the submodule construction problem

[MB83][HU96]. Let Comp(B, A) represent the largest complement submodule found by the submodule construction approach for module A and submodule B.

Assume that the required service specification of the conversion system is given, CS, and the service specifications of protocols P and Q are given, SS_P and SS_Q, respectively. [Oku90] shows that if service-level interconnection by a service interface adapter is possible then there exists a non-null CFSM $SIA = \text{Comp}(SS_P \times SS_Q, CS)$ and any execution of the service interface adapter is also a path of SIA. If $\text{Comp}(SS_P \times SS_Q, CS) = \textit{null}$ then service-level interconnection by a service interface adapter is not possible.

Once SIA is known an equivalent converter can be automatically derived, as observed by Bochmann [Boc90]: $C = E1_P \otimes SIA \otimes E0_Q$. However, it is usually the case that a more compact converter is desired which will make the conversion system more efficient than using a service interface adapter. [Oku90] also shows that $C' \subseteq C$ is also a converter if $C' \neq \textit{null}$.

A conversion system using a converter constructed as described above will satisfy freedom from channel overflow, however, there is no guarantee that the conversion system will satisfy freedom from deadlocks or freedom from unspecified receptions. Okumura [Oku90] provides the conditions that the states of the converter must satisfy for the inheritance of these properties by the conversion system. In her discussion of the inherited properties Okumura assumes asynchronous communication between the protocol entities and furthermore that at most one message reception action is defined at each state in the protocol entities.

If the converter obtained by $C = E1_P \otimes SIA \otimes E0_Q$ does not satisfy the conditions for freedom from deadlock and unspecified receptions then any state s that does not satisfy the conditions can be removed and all preceding states that come to fail the conditions after the removal of s can be recursively eliminated. If $C \neq \varnothing$ at the completion of this step then a C is a converter that satisfies the desirable properties.

Okumura's approach is completely algorithmic, however, the algorithms are not provided in [Oku90]. The computational complexity of the approach is $O(2^N)$, where $N = |SS_P| \times |SS_Q| \times |CS|$.

If a service interface adapter cannot be found, i.e., $SIA = \varnothing$, then a service interface adapter does not exist. If, after removing states and transitions that violate the conditions for freedom from deadlock and unspecified receptions, $C = \varnothing$ then a converter cannot be found using this approach that satisfies the desirable properties. This does not imply that a converter that satisfies the desirable properties does not exist.

4.3.3 Yao and Liu's approach: guaranteed service

In [YL92] a similar approach is presented to construct a protocol converter. The communication model is synchronous symmetric. The specification of a protocol entity is deterministic and contains the specification of a designated subset of the state set called final states. In this approach, CS is deterministic.

A conversion system $PQ = \{E0_P, C, E1_Q\}$ **satisfies** the service-level conversion requirements CS if it satisfies the conformity property and freedom from deadlock and livelock.

In the approach outlined in [YL92], the first step is to generate a service interface adapter and *new* service specifications for the original protocols for the conversion system. That is, find a service interface adapter SIA, a new service specification for protocol P, new-SS_P, and a new service specification for protocol Q, new-SS_Q, so that {new-SS_P, SIA, new-SS_Q} satisfies CS.

The CFSM $G = SS_P \times CS \times SS_Q$ is modified to make G well-formed by removing transitions and/or inserting new auxiliary transitions into G. If $G = SS_P \times CS \times SS_Q$ satisfies $G = \text{Proj}_{\Delta_1}(G) \times \text{Proj}_{\Delta_2}(G) \times \text{Proj}_{\Delta_3}(G)$, where Δ_1 is the set of service primitive actions defined in $E0_P$ and $E1_P$, Δ_2 is the set of service primitive actions defined in $E1_P$ and $E0_Q$ and Δ_3 the set of service primitive actions defined in $E0_Q$ and $E1_Q$, then G is **well formed**.

The next step is a verification step to ensure the well-formed G can satisfy the conversion requirements, i.e., verify that $CS = \text{Proj}_{\Delta}(G)$, where Δ is the set of service primitive actions defined in $E0_P$ and $E1_Q$. If the verification step passes then construct new-$SS_P = \text{Proj}_{\Delta}(G)$, where Δ is the set of service primitive actions of protocol P, construct new-$SS_Q = \text{Proj}_{\Delta}(G)$, where Δ is the set of service primitive actions of protocol Q and construct $SIA = \text{Proj}_{\Delta}(G)$, where Δ is the set of service primitive actions of $E1_P$ and $E0_Q$. Note that these action sets include the new service primitive actions labeling the transitions added to G to make it well-formed.

Since G is well-formed graph we know that new-$SS_P \otimes SIA \otimes$ new-$SS_Q = G$. We also know that {new-SS_P, SIA, new-SS_Q} satisfies CS. Therefore we have a conversion system that satisfies the required service specification, however, the original protocols' service specifications have been modified by the removal of some transitions and by the insertion of new transitions with new service primitive actions as labels. Recall that the service primitive actions that were added belong to the involved entities that will be replaced by the converter. Therefore, the next step is to generate the new specifications for the involved entities. i.e., find new-$E1_P$ such that {$E0_P$, new-$E1_P$} provides new-SS_P and new-$E0_Q$ such that {new-$E0_Q$, $E1_Q$} provides new-SS_Q. Under certain conditions outlined in [YL92] this can be accomplished. If new involved entities can be constructed then, using Bochmann's observation, a protocol converter can be constructed automatically by $C =$ new-$E1_P \times SIA \times$ new-$E0_Q$. The last step is to optimize C to some extent. Steps are provided to remove deadlock and livelock states from C and transitions corresponding to the service primitive actions of the composed entities.

There are verification steps in the approach in [YL92] without the accompanying discussion about what it means if one of the verification steps fails. The approach is restricted to the synchronous model of communication only. One of the benefits of converter construction via a service interface adapter, as mentioned for Okumura's approach, is the fact that the approach can be independent of the type of communication between the entities. Unfortunately, because of the number of verification steps in Yao and

Liu's approach which require the construction of a reachability graph, an extension of this approach to the asynchronous model is not straightforward.

By modifying the involved entities of the original protocols Yao and Liu provide an approach that may find a service interface adapter for protocols that Okumura's approach cannot. Also, by modifying the original involved entities they may be able to find a conversion system them satisfies the conformity property for protocols that Okumura's approach cannot. Yao and Liu used auxiliary transitions to provide extra synchronization ability such that the service interface adapter, if in existence, together with the new service specifications is guaranteed to support the required service specification.

If their goal was to provide interconnection at the service level then this is not ideal since the original protocol implementations of the involved entities would have to be changed which removes one of the benefits of service-level interconnection. However, the goal is to find a protocol converter that will replace the involved entities so this is not a major concern.

4.3.4 Extension to converter construction via SIA : protocol pruning

In [KLNS93] the observation is made that if a converter is constructed according to Bochmann's observation [Boc90], i.e., the converter is the composition of $E1_P$, *SIA*, and $E0_Q$, then it may contain extra states and transitions that are not needed by the largest common service subset of the two protocols. [KLNS93] presents an efficient algorithm to determine the largest common service subset and then remove the spurious states and transitions from the converter.

The communication model is synchronous asymmetric. The protocol entities may be nondeterministic. The approach assumes that a service interface adapter, *SIA*, that satisfies the conversion requirements is known.

Each service primitive action defined in $E1_P$ will typically have a corresponding service primitive action defined in $E0_P$; and each service primitive action defined in $E0_Q$ will typically have a corresponding service primitive action defined in $E1_Q$. Let every service primitive action defined in $E1_P$ or $E0_Q$ that is not defined in *SIA* be identified as an unmatched service primitive action. Furthermore, for each unmatched service primitive

action, its corresponding service primitive action in $E0_P$ or $E1_Q$ is also identified as an unmatched service primitive action.

We can now compute the largest common subset of services offered by the interconnection of P and Q with SIA. Modify SS_P and SS_Q to obtain SS'_P and SS'_Q respectively by removing all transitions labeled by unmatched service primitives. The largest common subset of services is $W = SS'_P \otimes SIA \otimes SS'_Q$.

Two algorithms are presented in [KLNS93] to find a pruned converter. The first algorithm involves the composition of many CFSMs, which suffers from state space explosion. To remedy this a second algorithm is presented which avoids the space expensive composition operations and lowers the time complexity as well.

Consider the CFSMs W, $E0_P$, $E1_P$, SIA, $E0_Q$, $E1_Q$. The second algorithm proceeds by removing all transitions with unmatched service primitive actions from each of the CFSMs. For each CFSM, compute the strongly connected component that starts at its initial state and discard the rest of the CFSM. As a result of this step there may now be some unmatched message transmissions or unmatched message receptions in the CFSMs. Remove all transitions with unmatched message receptions or unmatched message transmissions from each of the CFSMs. For each CFSM that changes compute the strongly connected component that starts at its initial state and discard the rest of the CFSM. Repeat this process until there are no more transitions with unmatched actions to obtain the set of CFSMs containing W, $E0'_P$, $E1'_P$, SIA, $E0'_Q$ and $E1'_Q$. The pruned converter is $C = E1'_P \otimes SIA \otimes E0'_Q$.

The algorithm presented is polynomial in the size of the total number of transitions defined in all the CFSMs.

4.4 Observations

Of the various forms of protocol-level conversion requirements reviewed in this chapter, the conversion seed [Oku86][CL90d] is the most expressive in specifying the translation requirements and synchronization requirements of the converter. The benefit of the conversion seed approaches [Oku86][CL90d] over the executable protocol trace approach

[RM91] is that we are not restricted to implementing a converter without memory. It is not practical to convert all messages one-to-one from one protocol to another because of various protocols mismatches. The valid closed path approach of [PL92] is also not limited to a memoryless converter, however, it only specifies the mapping of convertible messages and assumes that the different functions should proceed independent of one another. The conversion seed approach allows nonconvertible messages to be significant to the converter and thus restrictions on their ordering relative to other convertible or unconvertible messages can be specified as desired.

Of the approaches based on protocol-level conversion requirements, the conversion seed approaches [Oku86][CL90d] are the most formal and precise approaches to the protocol conversion problem. If a suitable conversion seed can be found they provide efficient, algorithmic solutions to determine if a protocol converter exists and generate one if it does.

Given the service-level requirements of the protocol conversion system in the form of the required service specification, *CS*, there are two families of approaches for constructing a converter. One family [CL89a][CL89b][TBD95][YL91] constructs the converter from the participating entities and *CS*. The other family [Boc90][Oku90][YL92] finds a service interface adapter *SIA* that when used in service-level conversion satisfies *CS* and then constructs a converter from the involved entities and *SIA*.

The algorithms found in [CL89a][CL89b][TBD95][YL91] provide alternatives to the submodule construction problem [MB83][HU96] for safety and in the case of [CL89a] [CL89b][TBD95] for safety and progress. Of these approaches [TBD95] provides the most general model, allowing both the protocol entities and required service specification to be nondeterministic. [TBD95] also provides the conditions to determine if a converter exists, and presents an algorithm to construct the converter. Furthermore, the approach in [TBD95] considers external equivalence. The side effect of this property is that the resulting converter of this approach will be optimal if the original protocols are optimal.

If a service interface adapter can be found such that when used in service-level conversion the conversion system satisfies the required service specification then there

exists a set of correct protocol converters that can be derived [Oku90][Boc90]. The approach presented in [Oku90] depends on the existence of a service interface adapter, and since a service interface adapter may not exist, the use of this approach is limited. The non-existence of a service interface adapter does not imply the non-existence of a converter. [YL92] proposes an approach that may find a service interface adapter that [Oku90] cannot. Unfortunately, [YL92] is much less general in terms of the model and the expectations of the input when compared to [Oku90]; and [YL92] is not as formal and precise as [Oku90].

Table 1 characterizes selected approaches from the approaches based on the protocol-level requirements, the approaches based on the required service specification, and the approaches based on the existence of a service interface adapter.

	[Oku86] [CL90d]	[TBD95]	[Oku90]
protocols *P* and *Q*	involved entities (deterministic, no service primitives)	involved entities, participating entities (nondeterministic)	involved entities, supplied service
conversion requirements	conversion seed	*CS* (nondeterministic)	*CS* (nondeterministic)
communication model	asynchronous bounded, or synchronous asymmetric	synchronous symmetric	asynchronous bounded, asynchronous unbounded or synchronous asymmetric
assumptions on protocols *P* and *Q*	optimal, deadlock free, unspecified reception free, channel overflow free	optimal	optimal, channel overflow free
properties inherited by the conversion system	optimal, deadlock free, unspecified reception free, channel overflow free	optimal	optimal, channel overflow free
correctness	external equivalence, semantic equivalence, deadlock free, unspecified reception free, channel overflow free	safety, progress, external equivalence	safety, external equivalence
decidable (based on correctness definition)	Yes	Yes	No

Table 1: Characterization of Selected Approaches

Chapter Five

Synthesis of Protocol Conversion Requirements

In Chapter 3, Section 3.1, we presented a formulation of the protocol conversion problem between network X and network Y which is based on the following assumptions:

1. Layer M protocol (called protocol P) of network X and layer N protocol (called protocol Q) of network Y are two different protocols.
2. There is a common peer protocol U which is layer M+1 protocol of network X and layer N+1 protocol of network Y. Thus, protocol U is common to both networks X and Y.
3. The required service (RS_U) for the common peer protocol U is defined.

If there is a gap between SS_P and RS_U, two interface adapters ($A0_P$ and $A1_P$) are used to supply the required service (RS_U) for protocol U. Otherwise, SS_P provides RS_U. In turn, if there is a gap between SS_Q and RS_U, two interface adapters ($A0_Q$ and $A1_Q$) can be used to supply the required service (RS_U) for protocol U. Otherwise, SS_Q provides RS_U.

We have observed that there is a gap between our formulation of the problem described above and the starting point of each of the approaches to the solution of this problem proposed in the literature. In terms of service requirements, most approaches assume that the required service specification for the conversion system, CS, is known. It would be nice to formally generate the required service of the conversion system, CS, and then apply one of the approaches based on the required service specification to generate a converter that satisfies the service requirements with respect to safety, and depending on the selected approach, with respect to progress. Given that it is likely that protocol-level conversion requirements exist in addition to the service requirements, it would be nice to specify the

service-level conversion requirements in terms of the peer protocol messages of the involved entities, for both the synchronous and the asynchronous model of communication. Then we could apply approaches based on protocol-level requirements to generate a converter that satisfies the service-level requirements and the protocol-level conversion requirements, for either the asynchronous or the synchronous model of communication.

As we saw in Chapter 4, Section 4.1, [Oku86] and [CL90d] each propose a solution for an instance of the protocol conversion problem. Both solutions determine the existence of a converter that satisfies the given conversion requirements for a particular class of protocols. The two approaches are similar in that they both rely on protocol-level conversion requirements specified as a conversion seed [Chapter 4]. In both approaches, the conversion seed declares the semantic relation between two dissimilar protocols in terms of their peer protocol messages. Neither approach considers a required service specification for the conversion system. In fact, both approaches expect that the service primitives are not included in the specification of the protocol entities. If a required service specification for the conversion system is known, in both cases, there is currently no way of inputting these global properties directly. Instead a conversion seed would have to be manually created and then the resulting protocol would have to be checked against the required service specification to ensure that it is satisfied.

In this chapter, an approach to synthesize the required service specification for the conversion system is proposed. Furthermore, an approach to construct a conversion seed that captures the semantic relation between the two protocols as it is specified in the required service specification is proposed. Section 5.1 describes the problem statement. Section 5.2 introduces the necessary formalisms for the material in the sequel. In Section 5.3, we propose an approach to synthesize the conversion system specification. In Section 5.4, we propose an approach to synthesize a conversion seed from the required service specification of the conversion system. Extensions to the approach in Section 5.4 are proposed in Sections 5.5 and 5.6.

5.1 Statement of the Problem

Assume that the layers at which the conversion will take place between network X and network Y are layer M and layer N, respectively. The specification of the layer M protocol of network X is given by protocol $P = \{E0_P, E1_P\}$, which provides the service specified by SS_P (Figure 13). The specification of the layer N protocol of network Y is given by protocol $Q = \{E0_Q, E1_Q\}$, which provides the service specified by SS_Q (Figure 17). We also assume that the required service for the common upper layer protocol, RS_U, is given, and that any interface adapters needed so that protocol P and protocol Q can provide RS_U are given (Figures 14, 15, 18, 19). With this information, we can find the required service specification for the conversion system, CS, and the conversion seed necessary to use existing methods to find a converter. Depending on whether or not adapters are needed for protocol P and protocol Q to provide RS_U, we have four general cases for the statement of the problem.

Case 1: If SS_P provides RS_U, as in Figure 15, and SS_Q provides RS_U, as in Figure 19, find CS and the conversion seed X so that existing methods can be employed to construct C so that $E0_P$, $E1_Q$ and C provide CS, as in Figure 21, such that CS can provide RS_U, as in Figure 27.

Case 2: If SS_P provides RS_U, as in Figure 15, and SS_Q together with $A0_Q$ and $A1_Q$ provides RS_U, as in Figure 18, find CS and the conversion seed X so that existing methods can be employed to construct C so that $E0_P$, $E1_Q$ and C provide CS, as in Figure 21, such that CS together with $A1_Q$ can provide RS_U, as in Figure 26.

Case 3: If SS_P together with $A0_P$ and $A1_P$ provides RS_U, as in Figure 14, and SS_Q provides RS_U, as in Figure 19, find CS and the conversion seed X so that existing methods can be employed to construct C so that $E0_P$, $E1_Q$ and C provide CS, as in Figure 21, such that CS together with $A0_P$ can provide RS_U, as in Figure 25.

Case 4: If SS_P together with $A0_P$ and $A1_P$ provides RS_U, as in Figure 14, and SS_Q together with $A0_Q$ and $A1_Q$ provides RS_U, as in Figure 18, find CS and the conversion seed X so that existing methods can be employed to construct C so that $E0_P$, $E1_Q$ and C

provide CS, as in Figure 21, such that CS together with $A0_P$ and $A1_Q$ can provide RS_U, as in Figure 24.

5.2 Preliminaries

Communicating finite state machines (CFSMs) will be used to specify protocol entities, service specifications, interface adapters and conversion seeds.

A **CFSM** E is a quadruple, $E = (S, \Sigma, \delta, s0)$, where:

S is the finite set of states,

Σ is the finite set of actions,

δ: $S \times \Sigma \rightarrow S$ is the partial transition function, and

$s0 \in S$ is the initial state.

The set of actions for which δ is defined at a state $s \in S$ is

$$\Sigma(s) = \{ \sigma \in \Sigma \mid \delta(s, \sigma) \text{ is defined } \}.$$

A state at which there are no outgoing transitions defined is called a dead state,

i.e., a state $s \in S$ is a **dead state** if and only if $\Sigma(s) = \varnothing$.

Consider states s, $s' \in S$ such that there is a transition t defined from s to s', i.e., $\delta(s, \sigma) = s'$, for an action $\sigma \in \Sigma$. The state s is called the **tail state** of t and state s' is called the **head state** of t.

Consider state $s \in S$. Let the number of outgoing transitions from s be the **outdegree**(s), the number of incoming transitions to s be the **indegree**(s) and the **degree**(s) = outdegree(s) + indegree(s).

To describe the behaviour of E on a sequence of actions, δ is extended to a sequence of actions, δ^*: $S \times \Sigma^* \rightarrow S$, where

$\delta^*(s, \varepsilon) = s$,

$\delta^*(s, \sigma.\alpha) = \delta^*(\delta(s, \sigma), \alpha)$, for all $\alpha \in \Sigma^*$ and $\sigma \in \Sigma$.

A sequence of actions α is said to be **accepted** by a CFSM E if and only if $\delta^*(s0, \alpha)$ is defined.

The **language** of a CFSM E is defined as $L(E) = \{ \alpha \in \Sigma^* \mid \alpha \text{ is accepted by } E. \}$

The **largest prefix closed subset** of a **language** of a CFSM E is denoted $\text{Pre}(L(E))$.

It is sometimes useful to construct the complement of a CFSM. There must be one non-accepting state in the specification of a CFSM E in order to find its complement. Therefore a state designated as *na* can be added to a CFSM with the necessary transitions to complete the transition function.

A CFSM $E = (S, \Sigma, \delta, s0)$ can be modified to form a **CFSM with a designated non-accepting state**, $E' = (S', \Sigma', \delta', s0')$, as follows:

$S' = S \cup \{na\}$, where *na* is the only non-accepting state in the modified CFSM,

$\Sigma' = \Sigma$,

$s0' = s0$,

$\delta' : S' \times \Sigma' \rightarrow S'$, where

$\delta'(s, \sigma) = \delta(s, \sigma)$, if $s \in S$ and $\sigma \in \Sigma(s)$.

$= na$ otherwise.

The **complement** of a CFSM with a designated non-accepting state $E = (S, \Sigma, \delta, s0)$, denoted $\neg E = (S', \Sigma', \delta', s0')$, is defined as:

$S' = S \setminus \{na\} \cup \{a\}$, where *a* is the only accepting state in the complement CFSM,

$\Sigma' = \Sigma$,

$s0' = s0$,

$\delta' : S' \times \Sigma' \rightarrow S'$, where

$\delta'(s, \sigma) = \delta(s, \sigma)$, if $\delta(s, \sigma) \neq na$.

$= a$ if $\delta(s, \sigma) = na$.

A sequence of actions α is said to be **accepted** by a CFSM with a designated non-accepting state E if and only if $\delta^*(s0, \alpha)$ is defined and $\delta^*(s0, \alpha) \neq na$.

A sequence of actions α is said to be **accepted** by a complement CFSM E if and only if $\delta^*(s0, \alpha)$ is defined and $\delta^*(s0, \alpha) = a$.

Given a sequence of actions α, the **projection** of α over V, $|_V : U^* \rightarrow V^*$ with U, V two sets of actions and $V \subseteq U$, for $\alpha \in U^*$, $\sigma \in U$, is defined inductively as:

$$
\begin{aligned}
(\varepsilon)|_V &= \varepsilon, \\
(\sigma.\alpha)|_V &= \sigma.(\alpha)|_V, && \text{if } \sigma \in V, \\
&= (\alpha)|_V, && \text{if } \sigma \notin V.
\end{aligned}
$$

Given a set of sequences of actions L, $L \subseteq U^*$, the **projection** of L over V, with U, V two sets of actions and $V \subseteq U$, for $\alpha \in U^*$, $\beta \in V^*$, is defined as:

$$\text{Proj}_V(L) = \{ \beta \mid \beta = \alpha|_V \text{ and } \alpha \in L \}.$$

There are four types of actions for which a transition in a CFSM E can be defined. They are distinguishable from each other by their labels in Σ.

1. A *peer message transmission* is an action denoted by a message identifier, say m, prefixed with a '–', i.e., $-m$. A transition labeled by a peer message transmission semantically corresponds to the entity specified as CFSM E sending message m to another entity in a set of CFSMs. If the entity to receive the message m is specified as CFSM E' then the action set of E' must contain the action $+m$.
2. A *peer message reception* is an action denoted by a message identifier, say m, prefixed with a '+', i.e., $+m$. A transition labeled by a peer message reception semantically corresponds to the entity specified as CFSM E receiving message m which was sent by another entity in a set of CFSMs. If the entity that sent the message m is specified as CFSM E' then the action set of E' must contain the action $-m$.

3. A *service primitive* is an action denoted by a service primitive identifier, say m, and is also prefixed by a '+' or '–' , i.e., $+m$ or $-m$, however, an action $+m$ or $-m$ will not appear in the action set of any other entity in a set of CFSMs. A transition labeled with a service primitive action semantically corresponds to a message exchange (transmission or reception) between the entity specified as CFSM E and the environment.

4. An *internal action* is an action with no sign, i.e., the action identifier is not prefixed by a '+' or '–'. A transition labeled by an internal action semantically corresponds to an action that the entity specified as CFSM E can perform without any interaction with the environment or other CFSMs.

As discussed in Chapter 3, Section 3.1, direct communication between a pair of communicating entities is modeled by a pair of FIFO queues. Each FIFO queue will be called a channel.

A **channel** C_{ij} is a FIFO queue from CFSM E_i to CFSM E_j. The content of C_{ij} is a sequence of zero or more messages represented by c_{ij}. An empty channel is denoted by ε. There can be a limit on the contents of a channel called channel bound which, if specified, is an integer represented by b_{ij}. If b_{ij} is not specified then there is no limit on the contents of the channel. If the bound is zero then the communication is synchronous and there is no need to consider the channels between the entities.

Let $I = \{1, 2, \ldots, n\}$ be a finite index set, with $n \geq 2$. Consider a set of CFSMs, $\{ E_i \mid i \in I \}$, such that any two CFSMs E_i and E_j, $i \neq j$, $i, j \in I$, communicate synchronously, communicate indirectly through another CFSM, or do not communicate. Let the Boolean function $sync(E_i, E_j)$ identify the synchronous communication between all pairs of CFSMs, E_i, E_j, $i \neq j$, $i, j \in I$.

i.e., $sync(E_i, E_j)$ = true, if E_i and E_j communicate synchronously.
= false, otherwise.

Let $I = \{1, 2, \ldots, n\}$ be a finite index set, with $n \geq 2$. Consider a set of CFSMs, $\{ E_i \mid i \in I \}$, such that any two CFSMs E_i and E_j, $i \neq j$, $i, j \in I$, communicate asynchronously over FIFO channels, C_{ij} and C_{ji}, or do not communicate. Let the Boolean function $async(E_i, E_j)$ identify the asynchronous communication between all pairs of CFSMs, E_i, E_j, $i \neq j$, $i, j \in I$.

i.e., $async(E_i, E_j)$ = true, if E_i and E_j communicate asynchronously
= false, otherwise.

For a pair of CFSMs E_i, E_j from a set of CFSMs $\{ E_i \mid i \in I \}$ with communication relationship $(a)sync$, it is useful to define the message sets relating to their communication with each other. Let the communication between E_i and E_j be direct and let M_{ij} represent the set of messages sent by CFSM E_i to be received by CFSM E_j: i.e., $M_{ij} = \{ m \mid -m \in \Sigma_i$ and $+m \in \Sigma_j$ and $(a)sync(E_i, E_j) = \text{true} \}$. Therefore, $M_{ij} = \varnothing$ if $(a)sync(E_i, E_j) = \text{false}$.

For a CFSM E_i from a set of CFSMs $\{ E_i \mid i \in I \}$ it is useful to define the subsets of the action set relating to the four types of actions.

1. Let Σ_i^- represent the set of peer transmission actions of E_i:

$$\Sigma_i^- = \{ -m \in \Sigma_i \mid +m \in \bigcup_{j \in I, j \neq i} \Sigma_j \}.$$

2. Let Σ_i^+ represent the set of peer reception actions of E_i:

$$\Sigma_i^+ = \{ +m \in \Sigma_i \mid -m \in \bigcup_{j \in I, j \neq i} \Sigma_j \}.$$

3. Let Σ_i^U represent the set of service primitive actions of E_i:

$$\Sigma_i^U = \{ -m \in \Sigma_i \mid +m \notin \bigcup_{j \in I, j \neq i} \Sigma_j \} \cup \{ +m \in \Sigma_i \mid -m \notin \bigcup_{j \in I, j \neq i} \Sigma_j \}.$$

4. Let Σ_i^I represent the set of internal actions of E_i:

$$\Sigma_i^I = \{ \sigma \in \Sigma_i \mid \sigma \neq -m \in \Sigma_i^- \cup \Sigma_i^U \text{ and } \sigma \neq +m \in \Sigma_i^+ \cup \Sigma_i^U \}.$$

Given an action $+m \in \Sigma^+ \cup \Sigma^U$, $-m \in \Sigma^- \cup \Sigma^U$, or $n \in \Sigma^I$, the **inverse** of the action is defined as follows:

$(+m)^{-1} = -m,$

$(-m)^{-1} = +m,$

$(n)^{-1} = n.$

Given a sequence actions the **inverse** of the sequence is defined inductively as follows:

$(\varepsilon)^{-1} = \varepsilon,$

$(\sigma.\alpha)^{-1} = (\sigma)^{-1}.(\alpha)^{-1}, \sigma \in \Sigma, \alpha \in \Sigma^*.$

Given a CFSM $E = (S, \Sigma, \delta, s0)$, the **inverse** of E is $E^{-1} = (S^{-1}, \Sigma^{-1}, \delta^{-1}, s0^{-1})$ where

$S^{-1} = S,$

$s0^{-1} = s0,$

$\Sigma^{-1} = \{ (\sigma)^{-1} \mid \sigma \in \Sigma \},$

$\delta^{-1}: S^{-1} \times \Sigma^{-1} \rightarrow S^{-1},$

$\delta^{-1}(s, \sigma) = \delta(s, (\sigma)^{-1}),$ if $(\sigma)^{-1} \in \Sigma(s)$,

$= \text{undefined}$ otherwise.

Given a CFSM $E = (S, \Sigma, \delta, s0)$, the **reverse** of E is $E^R = (S^R, \Sigma^R, \delta^R, s0^R)$ where

$S^R = S,$

$s0^R = s0,$

$\Sigma^R = \Sigma,$

$\delta^R: S^R \times \Sigma^R \rightarrow 2^{S^R},$

$s' \in \delta^R(s, \sigma)$ if $\delta(s', \sigma) = s.$

Given a CFSM $E = (S, \Sigma, \delta, s0)$ and a set $\Delta \subseteq \Sigma$, for a state $s \in S$, the **Δ-closure** of s is $\text{Cls}_\Delta(s) = \{ s' \in S \mid \exists \alpha \in \Delta^*, \delta^*(s, \alpha) = s' \}$. Note that $s \in \text{Cls}_\Delta(s)$ since $s = \delta^*(s, \varepsilon)$ and $\varepsilon \in \Delta^*$.

Given a CFSM $E = (S, \Sigma, \delta, s0)$ and $\Sigma_1 \subseteq \Sigma$, the **projection** of E over Σ_1 is defined by $\text{Proj}_{\Sigma_1}(E) = (S_1, \Sigma_1, \delta_1, p0)$, where

$S_1 = \{ p \in (2^S \setminus \{\varnothing\}) \mid \exists \alpha \in \Sigma_1^*, \delta_1^*(p0, \alpha) = p \},$

$p0 = \mathrm{Cls}_\Delta(s0)$, where $\Delta = \Sigma \setminus \Sigma_1$,

$\delta_1 : S_1 \times \Sigma_1 \rightarrow S_1,$

$$\delta_1(p, \sigma) = \bigcup_{s \in p} \mathrm{Cls}_\Delta(\delta(s, \sigma)).$$

Note that $\mathrm{Cls}_\Delta(\delta(s, \sigma)) = \varnothing$ whenever $\delta(s, \sigma)$ is undefined, and $\delta_1(p, \sigma) =$ undefined whenever $\bigcup_{s \in p} \mathrm{Cls}_\Delta(\delta(s, \sigma)) = \varnothing$.

Given $E_1 = (S_1, \Sigma_1, \delta_1, s0_1)$ and $E_2 = (S_2, \Sigma_2, \delta_2, s0_2)$, E_2 is a **sub-CFSM** of E_1 if and only if $S_2 \subseteq S_1$, $\Sigma_2 \subseteq \Sigma_1$, $s0_2 = s0_1$, and $\delta_2(s, \sigma) = \delta_1(s, \sigma)$ whenever $s \in S_2$, $\sigma \in \Sigma_2$, and $\delta_1(s, \sigma) \in S_2$.

The symmetric product is the product of two CFSMs such that the transitions labeled by actions in $(\Sigma_1 \cap \Sigma_2)$ are synchronized and the transitions labeled by actions in $((\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1))$ are interleaved. This operation is used on two CFSMs to construct a product CFSM when the two CFSMs are not considered to be in communication with each other (or their communication is not being considered). The definition of the symmetric product is intended to be general in that it does not consider the semantics of each transition (i.e., service primitive, peer message transmission or reception, internal). The set of actions whose transitions must be synchronized is simply determined by $(\Sigma_1 \cap \Sigma_2)$ and the set of actions whose transitions are interleaved is determined by $((\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1))$. Therefore, if $+m \in \Sigma_1$ and $+m \in \Sigma_2$ then transitions labeled with $+m$ must be synchronized in the product. However, if $+m \in \Sigma_1$, $-m \notin \Sigma_1$ and $+m \notin \Sigma_2$, $-m \in \Sigma_2$ then the transitions labeled by $+m$ and $-m$ are interleaved since they are considered different actions by this definition.

Given $E_1 = (S_1, \Sigma_1, \delta_1, s0_1)$ and $E_2 = (S_2, \Sigma_2, \delta_2, s0_2)$, the **symmetric product** of E_1 and E_2 is a CFSM defined by $E = E_1 \times E_2 = (S, \Sigma, \delta, s0)$, where

$S = \{ (s_1, s_2) \in S_1 \times S_2 \mid \exists \alpha \in \Sigma^*, \delta^*((s0_1, s0_2), \alpha) = (s_1, s_2) \},$

$s0 = (s0_1, s0_2),$

$\Sigma = \Sigma_1 \cup \Sigma_2,$

$\delta((s_1, s_2), \sigma) = (\delta_1(s_1, \sigma), \delta_2(s_2, \sigma)),$ if $\sigma \in \Sigma_1 \cap \Sigma_2$

$$
\begin{aligned}
& && \text{and } \sigma \in \Sigma_1(s_1) \text{ and } \sigma \in \Sigma_2(s_2), \\
&= (\delta_1(s_1, \sigma), s_2), && \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and } \sigma \in \Sigma_1(s_1), \\
&= (s_1, \delta_2(s_2, \sigma)), && \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and } \sigma \in \Sigma_2(s_2), \\
&= \text{undefined} && \text{otherwise.}
\end{aligned}
$$

A **protocol** P is a set of CFSMs, $P = \{ E_i \mid i \in I \}$, with communication relationship (a)sync$_P$, such that

1. A CFSM E_i, $i \in I$, cannot send the same message to more than one CFSM:

   i.e., $M_{ij} \cap M_{ik} = \varnothing, j, k \in I, j \neq k, j \neq i, k \neq i.$

2. A CFSM E_i, $i \in I$, cannot receive the same message from more than one CFSM:

   i.e., $M_{ji} \cap M_{ki} = \varnothing, j, k \in I, j \neq k, j \neq i, k \neq i.$

3. The internal actions must be distinct for every CFSM:

   i.e., $\Sigma_i^I \cap \Sigma_j^I = \varnothing, i, j \in I, i \neq j.$

4. The service primitive actions must be distinct for every CFSM:

   i.e., $\Sigma_i^U \cap \Sigma_j^U = \varnothing, i, j \in I, i \neq j.$

5. For any two indirectly communicating or non-communicating CFSMs, their action sets must be distinct:

   i.e., $\Sigma_i \cap \Sigma_j = \varnothing, i, j \in I, i \neq j$ and (a)sync$_P(E_i, E_j)$ = false.

The following two sections provide parallel definitions for properties of protocols. The first section defines the properties relating to the synchronous model of communication. Recall, in this case the bound on the channels is zero thus any two CFSMs in the protocol communicate directly and synchronously, communicate indirectly through another CFSM or do not communicate. The second section defines the same properties for the asynchronous model of communication. In this case any two entities communicate directly and asynchronously through channels or do not communicate.

5.2.1 Properties of protocols with synchronous communication

Synchronously communicating CFSMs synchronize on directly coupled actions ($-m$, $+m$), where m represents the message that is being sent by one CFSM and received by the other CFSM. Within a protocol each pair of CFSMs communicates synchronously, communicate indirectly through another CFSM, or do not communicate. The Boolean function *sync* identifies the synchronous communication between all pairs of CFSMs in a protocol. The synchronous asymmetric product is the product of two CFSMs from a protocol with a synchronous communication relationship. If the two CFSMs communicate directly (synchronously) then the transitions labeled by the directly coupled actions are synchronized and the other transitions are interleaved. When the transitions labeled by directly coupled actions ($-m$, $+m$) are synchronized the transition in the product CFSM is labeled by the internal action identified by m, i.e., without the prefix '+' or '–'. If the two CFSMs do not communicate directly then all transitions are interleaved in the product. The definition of the synchronous asymmetric product is used on two CFSMs from the same protocol to construct a single CFSM that represents the joint behaviour of the two CFSMs within the protocol.

Given two entities, $E_i = (S_i, \Sigma_i, \delta_i, s0_i)$ and $E_j = (S_j, \Sigma_j, \delta_j, s0_j)$, from protocol $P = \{ E_k \mid k \in I \}$ with communication relationship sync_P, the **synchronous asymmetric product** of E_i and E_j is a CFSM defined by $E = E_i \otimes E_j = (S, \Sigma, \delta, s0)$, where:

$$S = \{ (s_i, s_j) \in S_i \times S_j \mid \exists \alpha \in \Sigma^*, \delta^*((s0_i, s0_j), \alpha) = (s_i, s_j) \},$$

$$s0 = (s0_i, s0_j),$$

$$\Sigma = \Sigma_i \setminus \{ \{ +m \mid m \in M_{ji} \} \cup \{ -m \mid m \in M_{ij} \} \}$$
$$\cup \Sigma_j \setminus \{ \{ +m \mid m \in M_{ij} \} \cup \{ -m \mid m \in M_{ji} \} \}$$
$$\cup \{ m \mid m \in M_{ij} \cup M_{ji} \},$$

$$\delta : S \times \Sigma \rightarrow S,$$

$$\delta((s_i, s_j), \sigma) = (\delta_i(s_i, -m), \delta_j(s_j, +m)), \quad \sigma = m, \text{ if } m \in M_{ij}, -m \in \Sigma_i(s_i) \text{ and } +m \in \Sigma_j(s_j),$$

$$= (\delta_i(s_i, +m), \delta_j(s_j, -m)), \quad \sigma = m, \text{ if } m \in M_{ji}, +m \in \Sigma_i(s_i)$$

$$
\begin{aligned}
& && \text{and } -m \in \Sigma_j(s_j), \\
&= (\delta_i(s_i, \sigma), s_j), && \text{if } \sigma \in \Sigma_i \setminus \{\{ +m \mid m \in M_{ji} \} \\
& && \cup \{ -m \mid m \in M_{ij} \}\} \text{ and } \sigma \in \Sigma_i(s_i), \\
&= (s_i, \delta_j(s_j, \sigma)), && \text{if } \sigma \in \Sigma_j \setminus \{\{ +m \mid m \in M_{ij} \} \\
& && \cup \{ -m \mid m \in M_{ji} \}\} \text{ and } \sigma \in \Sigma_j(s_j), \\
&= \text{undefined} && \text{otherwise.}
\end{aligned}
$$

A **global state** G of a protocol $P = \{ E_i \mid i \in I \}$ is a list of states, $G = < s_i >_{i \in I}$, where each s_i is the local state of entity E_i in G, for all $i \in I$. In particular, the **initial global state** of P is $G0 = < s0_i >_{i \in I}$, such that each $s0_i$ is the initial state of entity E_i, for all $i \in I$.

A defined transition $\delta_i(s_i, \sigma) = s_i'$ of E_i is **executable** at global state $G = < s_i >_{i \in I}$ if and only if the current state of E_i in G is s_i, and

1. $\sigma \in \Sigma_i^I$; i.e., the transition is an internal action transition, or
2. $\sigma \in \Sigma_i^U$; i.e., the transition is a service primitive transition, or
3. $\sigma \in \Sigma_i^-$ such that $\sigma = -m$, $m \in M_{ij}$, the current state of E_j in G is s_j, and $\delta_j(s_j, +m)$ is defined; or
4. $\sigma \in \Sigma_i^+$ such that $\sigma = +m$, $m \in M_{ji}$, the current state of E_j in G is s_j, and $\delta_j(s_j, -m)$ is defined.

Let $G = < s_i >_{i \in I}$, and $H = < s_i' >_{i \in I}$, be global states and define $\boldsymbol{G \rightarrow H}$ if and only if

1. $\exists i \in I$, such that H can be derived from G by executing a single transition $\delta_i(s_i, \sigma) = s_i'$ for some $\sigma \in \Sigma_i^I \cup \Sigma_i^U$, or
2. $\exists i, j \in I, i \neq j$, such that H can be derived from G by executing the following transitions synchronously: $\delta_i(s_i, -m) = s_i'$ and $\delta_j(s_j, +m) = s_j'$, for some $m \in M_{ij}$.

All elements of G that are not affected by the transition(s) remain the same in H.

Let $\rightarrow^*$ denote the reflexive, transitive closure of $\rightarrow$. H is **reachable from** G if and only if $G \rightarrow^* H$. H is a **reachable global state** if and only if $G0 \rightarrow^* H$.

The **language** of a protocol $P = \{ E_i \mid i \in I \}$ is defined as $L(P) = \{ \alpha \in ((\bigcup_{i,j \in I, j \neq i} M_{ij}) \cup (\bigcup_{k \in I} (\Sigma_k^I \cup \Sigma_k^U)))^* \mid G0 \rightarrow^* H$ through a sequence of executable transitions whose sequence of labels is α }.

A protocol P **satisfies** a required service specification RS_P with respect to **safety** if $\text{Proj}_{\Sigma_{RSP}}(L(P)) \subseteq L(RS_P)$.

A reachable global state $G = < s_i >_{i \in I}$ of a protocol $P = \{ E_i \mid i \in I \}$ is a **deadlock state** if and only if there are no executable transitions defined at G.

A protocol is **deadlock free** if and only if no reachable global state is a deadlock state.

A reachable global state $G = < s_i >_{i \in I}$ of a protocol $P = \{ E_i \mid i \in I \}$ is an **unspecified reception state** for entity E_i if and only if there exists an entity E_j such that $\delta_j(s_j, -m)$ is defined, $m \in M_{ji}$, and $\delta_i(s_i, +m)$ is not defined.

An entity is **unspecified reception free** in a protocol if and only if no reachable global state is an unspecified reception state for the entity.

A protocol is **unspecified reception free** if and only if no reachable global state is an unspecified reception state for any entity in the protocol.

5.2.2 Properties of protocols with asynchronous communication

Asynchronously communicating CFSMs communicate over channels where channel bound is greater than zero. Within a protocol each pair of CFSMs communicates asynchronously or does not communicate. The Boolean function *async* identifies the asynchronous communication between all pairs of CFSMs in a protocol.

A **global state** G of a protocol $P = \{ E_i \mid i \in I \}$ is represented as a pair (S, C) where S is a list of states, $S = < s_i >_{i \in I}$, where each s_i is the state of entity E_i in G, for all $i \in I$, and C is a list of channel contents, $C = < c_{ij} >_{i,j \in I, i \neq j}$, where each c_{ij} is the content of channel

C_{ij} in G, for all $i, j \in I$. In particular, the **initial global state** of P is $G0 = (< s0_i >_{i \in I}, < c_{ij} >_{i,j \in I, i \neq j})$ such that each $s0_i$ is the initial state of entity E_i, for all $i \in I$, and $c_{ij} = \varepsilon$, for all $i, j \in I$, $i \neq j$.

A defined transition $\delta_i(s_i, \sigma) = s_i'$ of E_i is **executable** at global state $G = (< s_i >_{i \in I}, < c_{ij} >_{i,j \in I, i \neq j})$ if and only if the current state of E_i in G is s_i, and

1. $\sigma \in \Sigma_i^I$; i.e., the transition is an internal action transition, or
2. $\sigma \in \Sigma_i^U$; i.e., the transition is a service primitive transition, or
3. $\sigma \in \Sigma_i^-$; i.e., the transition is a peer message transmission transition, or
4. $\sigma \in \Sigma_i^+$ such that $\sigma = +m$, $m \in M_{ji}$, and front$(c_{ji}) = m$; i.e., the transition is a peer message reception transition and the message to be received is at the front of the appropriate channel.

Let $G = (< s_i >_{i \in I}, < c_{ij} >_{i,j \in I, i \neq j})$ and $H = (< s_i' >_{i \in I}, < c_{ij}' >_{i,j \in I, i \neq j})$ be global states and define $G \rightarrow H$ if and only if $\exists i \in I$, such that H can be derived from G by executing one of the following transitions:

1. $\delta_i(s_i, \sigma) = s_i'$ for some $\sigma \in \Sigma_i^I \cup \Sigma_i^U$, or
2. $\delta_i(s_i, -m) = s_i'$ and $c_{ij}' = c_{ij}.m$, for some $m \in M_{ij}$, or
3. $\delta_i(s_i, +m) = s_i'$ and $c_{ji} = m.c_{ji}'$, for some $m \in M_{ji}$.

All elements of G that are not affected by the transition remain the same in H.

Let $\rightarrow^*$ denote the reflexive, transitive closure of $\rightarrow$. H is **reachable from** G if and only if $G \rightarrow^* H$. H is a **reachable global state** if and only if $G0 \rightarrow^* H$.

The **language** of a protocol $P = \{ E_i \mid i \in I \}$ is defined as $L(P) = \{ \alpha \in (\bigcup_{k \in I} \Sigma_k)^* \mid G0 \rightarrow^* H$ through a sequence of executable transitions whose sequence of labels is $\alpha \}$.

A protocol P **satisfies** a required service specification RS_P with respect to **safety** if $\text{Proj}_{\Sigma_{RSP}}(L(P)) \subseteq L(RS_P)$.

A reachable global state $G = (< s_i >_{i \in I}, < c_{ij} >_{i,j \in I, i \neq j})$ of a protocol $P = \{ E_i \mid i \in I \}$ is a **deadlock state** if and only if there are no executable transitions defined at G and all $c_{ij} = \varepsilon$, $i, j \in I$, $i \neq j$.

A protocol is **deadlock free** if and only if no reachable global state is a deadlock state.

A reachable global state $G = (< s_i >_{i \in I}, < c_{ij} >_{i,j \in I, i \neq j})$ of a protocol $P = \{ E_i \mid i \in I \}$ is an **unspecified reception state** for entity E_i if and only if there exists a channel C_{ji} such that $\text{front}(c_{ji}) = m$, and $\delta_i(s_i, +m)$ is not defined.

An entity is **unspecified reception free** in a protocol if and only if no reachable global state is an unspecified reception state for the entity.

A protocol is **unspecified reception free** if and only if no reachable global state is an unspecified reception state for any entity in the protocol.

A reachable global state $G = (< s_i >_{i \in I}, < c_{ij} >_{i,j \in I, i \neq j})$ of a protocol $P = \{ E_i \mid i \in I \}$ is a **channel overflow state** if and only if there exists a channel C_{ij} such that $\text{length}(c_{ij}) > b_{ij}$.

A protocol is **channel overflow free** if and only if no reachable global state is a channel overflow state.

5.2.3 Generalizing the model: direct communication only

To simplify the discussion that follows we will consider the case of direct communication only. That is, we will consider protocols in the form $P = \{E0_P, E1_P\}$ where $(a)\text{sync}_P(E0_P, E1_P) = \text{true}$ (Figure 13), and $Q = \{E0_Q, E1_Q\}$ where $(a)\text{sync}_Q(E0_Q, E1_Q) = \text{true}$ (Figure 17). If a protocol P is given as in Figure 12, or a protocol Q is given as in Figure 16, where the protocol entities communicate indirectly through a communication

medium that provides the required service for the protocol, the protocol can be transformed into a protocol where the protocol entities communicate directly.

For example, if $P = \{E0_P, RS_P, E1_P\}$, where $sync_P(E0_P, RS_P) = true$, $sync_P(RS_P, E1_P) = true$, and $sync_P(E0_P, E1_P) = false$, then $P' = \{E0'_P, E1_P\}$, where $sync_{P'}(E0'_P, E1_P) = true$ and $E0'_P = E0_P \otimes RS_P$. If $Q = \{E0_Q, RS_Q, E1_Q\}$, where $sync_Q(E0_Q, RS_Q) = true$, $sync_Q(RS_Q, E1_Q) = true$, and $sync_Q(E0_Q, E1_Q) = false$, then $Q' = \{E0_Q, E1'_Q\}$, where $sync_{Q'}(E0_Q, E1'_Q) = true$ and $E1'_Q = E1_Q \otimes RS_Q$.

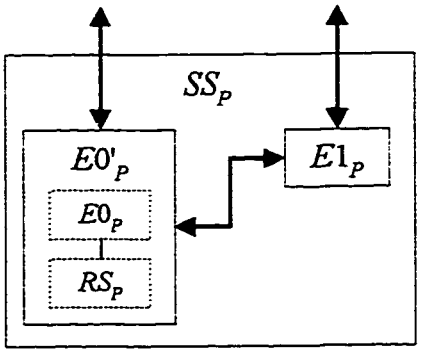

Figure 32: Protocol P'

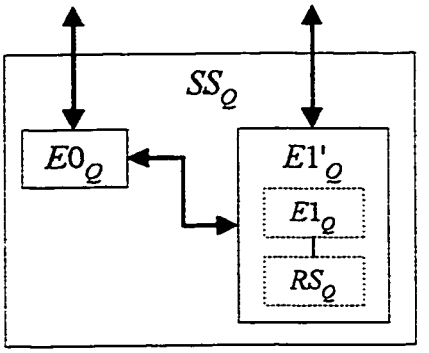

Figure 33: Protocol Q'

The protocol $P' = \{E0'_P, E1_P\}$(Figure 32) is still a protocol that provides the supplied service (SS_P) of protocol P. The protocol $Q' = \{E0_Q, E1'_Q\}$ (Figure 33) is still a protocol that provides the supplied service (SS_Q) of protocol Q.

5.3 Finding the Required Service Specification

In the case where no protocol adapters are necessary for protocol P to provide RS_U and protocol Q to provide RS_U, as in case 1 of the problem statement, the conversion system specification is the required service of protocol U, i.e., $CS = RS_U$.

If adapters are specified to provide the required service for protocol U, as in cases 2, 3 and 4 of the problem statement, then we must construct the required service specification of the conversion system. This can be done using the submodule construction approach.

5.3.1 Applying the submodule construction approach

The submodule construction problem (SCP) as stated in [MB83] is that given the specification of a system (module) and that of its $n - 1$ submodules, determine the

specification of the *n*th submodule that together with the given n - 1 submodules will satisfy the given system specification. [HU96] recasts the *SCP* in a formal setting and presents an algorithm for the solution of *SCP* where submodules are prefix-closed finite state machines. The problem as stated in [MB83] [HU96] assumes that the communication between entities is synchronous and the directly coupled actions are identified by common action labels. The problem also assumes that the specification of the module includes the definition of a final state set. Our model uses an asymmetric notation for communication between entities and assumes that entities are prefixed closed, i.e., all states are final states, and thus the reformulation of the problem here is simply notational.

The **submodule construction problem** [HU96] can be formulated as follows: Given the specification of a module $E_0 = (S_0, \Sigma_0, \delta_0, s0_0)$ and the specification of a submodule $E_1 = (S_1, \Sigma_1, \delta_1, s0_1)$, find a submodule $E_2 = (S_2, \Sigma_2, \delta_2, s0_2)$, where $\Sigma_2 = \{ \sigma \mid \sigma \in \Sigma_0 \setminus \Sigma_1 \} \cup \{ \sigma \mid \sigma^{-1} \in \Sigma_1 \setminus \Sigma_0 \}$, such that $\text{Proj}_{\Sigma_0}(\text{L}(E_1) \otimes \text{L}(E_2)) \subseteq \text{L}(E_0)$.

Theorem 5.1 [HU96]: Given the specification of a module (CFSM with designated non-accepting state) $E_0 = (S_0, \Sigma_0, \delta_0, s0_0)$, the specification of a submodule (CFSM) $E_1 = (S_1, \Sigma_1, \delta_1, s0_1)$, and $\Sigma_2 = \{ \sigma \mid \sigma \in \Sigma_0 \setminus \Sigma_1 \} \cup \{ \sigma \mid \sigma^{-1} \in \Sigma_1 \setminus \Sigma_0 \}$, then E_2 that satisfies

$$\text{L}(E_2) = \text{Pre}(\text{Proj}_{\Sigma_2}(\text{L}((E_0^{-1} \otimes E_1)^{-1})) \setminus \text{Proj}_{\Sigma_2}(\text{L}((\neg E_0^{-1} \otimes E_1)^{-1}))),$$

also satisfies

$$\text{Proj}_{\Sigma_0}(\text{L}(E_1 \otimes E_2)) \subseteq \text{L}(E_0),$$

and $\text{L}(E_2)$ is maximal with this property.

We are given the required service for protocol *U*: $RS_U = (S_{RS_U}, \Sigma_{RS_U}, \delta_{RS_U}, s0_{RS_U})$. We also are given one or two interface adapters that will provide the translation between protocol *U* and the conversion system: $A0_P = (S_{A0_P}, \Sigma_{A0_P}, \delta_{A0_P}, s0_{A0_P})$ and/or $A1_Q = (S_{A1_Q}, \Sigma_{A1_Q}, \delta_{A1_Q}, s0_{A1_Q})$. We need to construct the required service specification of the conversion system, *CS*, so that the *CS* and the interface adapter(s) provide RS_U.

Our model presents a special case of the submodule construction problem in that all states are accepting states in a CFSM. Therefore a final state set is not included in the

definition of a CFSM. The submodule construction approach requires one non-accepting state in the specification of the CFSM E_0 in order to find the complement. Therefore a state is added to RS_U with the necessary transitions to complete the transition function to form E_0. This new state will be the only non-accepting state, i.e., *na*, of E_0.

1. Define $E_0 = (S_0, \Sigma_0, \delta_0, s0_0)$ where [cases 2, 3 or 4 of the problem statement]

   $S_0 = S_{RS_U} \cup \{na\}$,

   $\Sigma_0 = \Sigma_{RS_U}$,

   $s0_0 = s0_{RS_U}$,

   $\delta_0 : S_0 \times \Sigma_0 \rightarrow S_0$, where

   $\delta_0(s, \sigma) = \delta_{RS_U}(s, \sigma)$, if $s \in S_{RS_U}$ and $\sigma \in \Sigma_{RS_U}(s)$.

   $= na$ otherwise.

   Or go to step 5. [case 1 of the problem statement]

2. Define $E_1 = A1_Q$, [case 2 of problem statement]

   or define $E_1 = A0_P$, [case 3 of problem statement]

   or define $E_1 = A0_P \otimes A1_Q$. [case 4 of problem statement]

3. Define $\Sigma_2 = \{ \sigma \mid \sigma \in \Sigma_0 \setminus \Sigma_1 \} \cup \{ \sigma \mid \sigma^{-1} \in \Sigma_1 \setminus \Sigma_0 \}$.

4. Find E_2 such that E_2 satisfies

   $L(E_2) = \text{Pre}(\text{Proj}_{\Sigma_2}(L((E_0^{-1} \otimes E_1)^{-1})) \setminus \text{Proj}_{\Sigma_2}(L((\neg E_0^{-1} \otimes E_1)^{-1})))$.

5. $CS = RS_U$ [case 1 of the problem statement]

   or $CS = E_2$ [cases 2, 3 or 4 of the problem statement].

Corollary 5.1: The specification *CS* found according to the above steps satisfies

$\text{Proj}_{\Sigma_{RSU}}(L(CS)) \subseteq L(RS_U)$ [case 1],

$\text{Proj}_{\Sigma_{RSU}}(L(CS \otimes A1_Q)) \subseteq L(RS_U)$ [case 2],

$\text{Proj}_{\Sigma_{RSU}}(L(A0_P \otimes CS)) \subseteq L(RS_U)$ [case 3],

$\text{Proj}_{\Sigma_{RSU}}(\text{L}(A0_P \otimes CS \otimes A1_Q)) \subseteq \text{L}(RS_U)$ [case 4],

and furthermore, L(CS) is maximal with the property.

5.3.2 Algorithm for submodule construction

An algorithm to solve the submodule construction problem is given in [HU96]. The algorithm has been modified in two ways for our model. First, the algorithm has been modified to take into account the asymmetric notation of our model. This only impacts the prefixes of the labels in the resulting CFSM of the algorithm. Second, the resulting CFSM does not contain a final state set. The modifications from the original algorithm are shown in bold. As pointed out in [HU96], when the input CFSM E_0 contains only the one designated non-accepting state and E_1 contains only accepting states, it can be easily shown for this case that $(\neg E_0^{-1} \otimes E_1)^{-1} = \neg((E_0^{-1} \otimes E_1)^{-1})$. Hence it is sufficient to only form the CFSM $E = (E_0^{-1} \otimes E_1)^{-1}$.

A state of E is a pair (s_1, s_2), where s_1 is a state of E_0^{-1} and s_2 is a state of E_1. Let $NA = \{ (s_1, s_2) \in S_E \mid s_1 = na \}$. A state in E_2 is a set of states of E. A state p of E_2 is **invalid** if $p \cap NA \neq \varnothing$.

procedure SCP(E : CFSM; Σ_0, Σ_1 : ACTIONS; var E_2 : CFSM);

input: $E = (\boldsymbol{E_0^{-1}} \otimes \boldsymbol{E_1})^{-1} = (S, \Sigma, \delta, s0)$, Σ_0, Σ_1

output: $E_2 = (S_2, \Sigma_2, \delta_2, p0_2)$ if exists, and "no solution" otherwise.

```
begin
  Σ2 = { σ | σ ∈ Σ0 \ Σ1 } ∪ { σ | σ^-1 ∈ Σ1 \ Σ0 };
  Δ := Σ^I;
  p02 := Cls_Δ(s0);
  if p02 is invalid
     then
        begin
           report "no solution"
        end {then}
     else
        begin
           new := ∅;
           old := {p02};
```

```
        S2 := {p02};
        while old ≠ ∅ do
          begin
            for p ∈ old do
              begin
                for σ ∈ Σ2 do
                  begin
                    p' := ∪ Cls_Δ(δ(s, σ));
                          s∈p
                    if p' ≠ ∅ and valid
                      then
                        begin
                          δ2(p, σ) := p';
                          if p' ∉ S2
                            then
                              begin
                                S2 := S2 ∪ {p'};
                                new := new ∪ {p'};
                              end; {then}
                          {endif}
                        end {then}
                      else
                        begin
                          δ2(p, σ) is undefined;
                        end {else}
                    {endif}
                  end; {for}
              end; {for}
            old := new;
            new := ∅;
          end; {while}
      end; {else}
  {endif}
end. {procedure SCP}
```

Potentially, all subsets of the state set of the input CFSM could be generated, thus making the algorithm exponential in time and space complexity. More specifically, in the worst case, the time and space complexity is $O(2^N)$, where $N = |(E_0^{-1} \otimes E_1)^{-1}|$.

Theorem 5.2: Given CFSMs E_0 and E_1, the CFSM E_2 constructed by the proposed algorithm satisfies: $\text{Proj}_{\Sigma_0}(\text{L}(E_1 \otimes E_2)) \subseteq \text{L}(E_0)$. Moreover, $\text{L}(E_2)$ is maximal with this property.

Proof: by proof in [HU96] and the following observations:

1. Our model does not include a final state set because the implicit assumption is that all states are final states. The original algorithm always returns a final state set equal to the state set of the resulting machine therefore this modification does not affect the results in [HU96].
2. The input has changed from $(E_0 \times E_1)$, where E_0 and E_1 are specified in symmetric notation, to $(E_0^{-1} \otimes \text{E}_1)^{-1}$, where E_0 and E_1 are specified in asymmetric notation. Note that $(E_0 \times E_1)$ is equivalent to $(E_0^{-1} \otimes E_1)^{-1}$ with the exception of the 'signs' on the transition labels.
3. The action set of E_2 changed from $(\Sigma_0 \setminus \Sigma_1) \cup (\dot{\Sigma}_1 \setminus \Sigma_0)$, where E_0 and E_1 are specified in symmetric notation, to $\{ \sigma \mid \sigma \in \Sigma_0 \setminus \Sigma_1 \} \cup \{ \sigma \mid \sigma^{-1} \in \Sigma_1 \setminus \Sigma_0 \}$, where E_0 and E_1 are specified in asymmetric notation. Note that $(\Sigma_0 \setminus \Sigma_1) \cup (\Sigma_1 \setminus \Sigma_0)$ is equivalent to $\{ \sigma \mid \sigma \in \Sigma_0 \setminus \Sigma_1 \} \cup \{ \sigma \mid \sigma^{-1} \in \Sigma_1 \setminus \Sigma_0 \}$ with the exception of the 'signs' on the transition labels.
4. The delta set of actions changed from $(\Sigma_0 \cup \Sigma_1) \setminus \Sigma_2$, where E_0 and E_1 are specified in symmetric notation, to Σ^{I}, where E is specified in asymmetric notation. Note that $(\Sigma_0 \cup \Sigma_1) \setminus \Sigma_2$ is equivalent to Σ^{I}. □

5.4 Finding a Conversion Seed: Approach_1

The following is based on the assumption that the service primitives that are referred to in *CS* are those of $E0_P$ and $E1_Q$; i.e., $\Sigma_{CS} = \Sigma^U_{E0_P} \cup \Sigma^U_{E1_Q}$.

The conversion seed problem: Given the specification of the conversion system requirements, *CS*, and the specification of the protocols $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$, find a conversion seed *X*, such that a conversion system protocol $\{E0_P, C, E1_Q\}$ that satisfies external equivalence and semantic equivalence with respect to *X* also satisfies the safety property with respect to *CS*.

We propose a solution to the conversion seed problem for protocols that can be specified by the CFSM model introduced in Section 5.2 for synchronous communication and for a subset of the protocols that can be specified by the CFSM model introduced in Section 5.2 for asynchronous communication. The subset of protocol specifications for the asynchronous model are those in which the specification of the protocol entities satisfy the following conditions:

C1 There is no state *s* defined such that the set of transitions defined at states in $Cls_\Delta(s)$ contain both peer protocol message transmission transitions and peer protocol message reception transitions, where $\Delta = \Sigma_{CS}$.

C2 There is no state *s* defined such that there are both peer protocol message transitions and service primitive transitions defined at *s*.

C3 There are no two sequences of actions $\alpha.\sigma$ and $\beta.\rho$ accepted by an entity, where σ and ρ are peer protocol message transitions, such that $\alpha.\sigma|_{\Sigma_{CS}} \neq \beta.\rho|_{\Sigma_{CS}}$ and $\alpha.\sigma|_{\Sigma\Sigma_{CS}} = \beta.\rho|_{\Sigma\Sigma_{CS}}$.

5.4.1 Significant transitions and the significant action set

Consider the transitions of $E0_P$ and $E1_Q$ where the communication between these entities and the converter is asynchronous. In the conversion system, Figure 21, the peer protocol message transmissions of $E0_P$ and $E1_Q$ are observable by the converter and the peer protocol message receptions of $E0_P$ and $E1_Q$ are controllable by the converter. The service primitive message transmissions and receptions of $E0_P$ and $E1_Q$ are neither observable nor controllable by the converter.

Now consider the transitions of $E0_P$ and $E1_Q$ where the communication between these entities and the converter is synchronous. In the conversion system, Figure 21, the peer

protocol message transmissions of $E0_P$ and $E1_Q$ are observable and controllable by the converter and the peer protocol message receptions of $E0_P$ and $E1_Q$ are observable and controllable by the converter. The service primitive message transmissions and receptions of $E0_P$ and $E1_Q$ are neither observable nor controllable by the converter.

The converter maps the messages of one protocol to the messages of the other protocol. It also provides the synchronization between the two protocol entities. It can only accomplish this task through the observable and controllable transitions. Our specification of the conversion system is given in terms of the service primitives of $E0_P$ and $E1_Q$. For a service primitive transition, t, in $E0_P$ or $E1_Q$, we want to determine the set of transitions that the converter can control such that once the converter participates in the execution of one of those transitions, there is a sequence of zero or more transitions uncontrollable by the converter which leads to the enabling of t. We also want to find the set of transitions that the converter can observe such that when the converter participates in the execution of one of the transitions, t has been executed. These sets of transitions contain the significant transitions of service primitive transition t. The set of actions labeling the significant transitions of all the service primitive transitions is the significant action set.

Before the substeps are given to find the significant transitions and significant action sets, some additional definitions are needed.

For any transition $t = (s, \sigma, s')$ in CFSM $E = (S, \Sigma, \delta, s0)$, the *set of predecessor transitions*, pre(t), is defined by pre(t) = $\{ (r, \rho, s) \mid r \in S, \rho \in \Sigma, \delta(r, \rho) = s \}$.

For any transition $t = (s, \sigma, s')$ in CFSM $E = (S, \Sigma, \delta, s0)$, the *set of controllable predecessor transitions*, c_pre(t), can be constructed as follows:

- c_pre(t) = $\varnothing$, if $s \in \mathrm{Cls}_\Delta(s')$, where $\Delta = \Sigma_{CS} \setminus \{\sigma\}$.
- $t_1 \in$ c_pre(t), if $t_1 \in$ pre(t) and t_1 is controllable by the converter,
- $t_2 \in$ c_pre(t), if $t_2 \in$ c_pre(t_1) and $t_1 \in$ pre(t) and t_1 is not controllable by the converter.

For any transition $t = (s, \sigma, s')$ in CFSM $E = (S, \Sigma, \delta, s0)$, the *set of significant predecessor transitions*, sig_pre(t), is defined by:

sig_pre(t) = { $(r, \rho, r') \mid (r, \rho, r') \in$ c_pre(t) and $\nexists (q, \rho', r) \in$ c_pre(t), $q, r, r' \in S$, $\rho, \rho' \in \Sigma$ }.

For any transition $t = (s, \sigma, s')$ in CFSM $E = (S, \Sigma, \delta, s0)$, the *set of successor transitions*, suc(t), is defined by: suc(t) = { $(s', \rho, q) \mid q \in S, \rho \in \Sigma, \delta(s', \rho) = q$ }.

For any transition $t = (s, \sigma, s')$ in CFSM $E = (S, \Sigma, \delta, s0)$, the *set of observable successor transitions*, o_suc(t), can be constructed as follows:

- o_suc(t) = $\varnothing$, if $s \in \text{Cls}_{\Delta}(s')$, where $\Delta = \Sigma_{CS} \setminus \{\sigma\}$.
- $t_1 \in$ o_suc(t), if $t_1 \in$ suc(t) and t_1 is observable by the converter,
- $t_2 \in$ o_suc(t), if $t_2 \in$ o_suc(t_1) and $t_1 \in$ suc(t) and t_1 is not observable by the converter.

For any transition $t = (s, \sigma, s')$ in CFSM $E = (S, \Sigma, \delta, s0)$, the *set of significant successor transitions*, sig_suc(t), is defined as follows:

sig_suc(t) = { $(r, \rho, r') \mid (r, \rho, r') \in$ o_suc(t) and $\nexists (r', \rho', q) \in$ o_suc(t), $q, r, r' \in S$, $\rho, \rho' \in \Sigma$ }.

Then, the following is performed to find the significant transitions and the significant action set.

1. For every transition (s, σ, s') in $E0_P$ such that $\sigma \in \Sigma_{CS}$, $s, s' \in \Sigma_{E0_P}$, find sig_pre((s, σ, s')) and sig_suc((s, σ, s')).
2. For every transition (s, σ, s') in $E1_Q$ such that $\sigma \in \Sigma_{CS}$, $s, s' \in \Sigma_{E1_Q}$, find sig_pre((s, σ, s')) and sig_suc((s, σ, s')).
3. Let the significant action set $SA = \{ \sigma \mid (s, \sigma, s') \in \text{sig\_suc}(t) \cup \text{sig\_pre}(t), \forall t = (r, \rho, r')$ in $E0_P$ and in $E1_Q$, $\rho \in \Sigma_{CS}\}$.

The conversion seed found using the approach presented in the sequel is dependent on the sets of significant transitions. If any of the sets of significant predecessor transitions is

empty then a conversion seed cannot be found using this method. To find the most general conversion seed, and in some cases to find a conversion seed at all, it is necessary that for a service primitive transition t, sig_suc(t) = o_suc(t). It is possible to modify $E0_P$ or $E1_Q$ by adding states and transitions to make sig_suc(t) = o_suc(t). In the worst case the size of the CFSM doubles.

For every state r' such that there exists a non-empty set of incoming transitions, $\{(r, \rho, r') \mid (r, \rho, r') \in$ o_suc(t) \ sig_suc(t), where t is any service primitive transition in $E0_P$ ($E1_Q$)$\}$ perform the following steps in order to ensure that sig_suc(t) = o_suc(t):

1. Add a new state $r'a$.
2. For every $\sigma \in \Sigma(r')$ do the following steps:
   a) Define $\delta(r'a, \sigma) = \delta(r', \sigma)$, if $\delta(r', \sigma) \neq r'$.
   b) Define $\delta(r'a, \sigma) = r'a$, if $\delta(r', \sigma) = r'$.
   c) If $\sigma \in \Sigma_{CS}$ then define sig_pre($(r'a, \sigma, \delta(r', \sigma))$) = $\varnothing$ and define sig_suc($(r'a, \sigma, \delta(r', \sigma))$) = sig_suc($(r', \sigma, \delta(r', \sigma))$).
   d) If transition $(r', \sigma, \delta(r', \sigma)) \in$ sig_pre(t) where t is a service primitive transition then add transition $(r'a, \sigma, \delta(r', \sigma))$ to sig_pre(t).
3. For each transition (r, ρ, r') in the set $\{(r, \rho, r') \mid (r, \rho, r') \in$ o_suc(t) \ sig_suc(t), where t is any service primitive transition in $E0_P$ ($E1_Q$)$\}$ do the following steps:
   a) Remove transition (r, ρ, r').
   b) Add transition $(r, \rho, r'a)$.
   c) If transition $(r, \rho, r') \in$ sig_suc((s, σ, s')) where (s, σ, s') is a service primitive transition then replace transition (r, ρ, r') with transition $(r, \rho, r'a)$ in sig_suc((s, σ, s')).
   d) If transition $(r, \rho, r') \in$ sig_pre((s, σ, s')) where (s, σ, s') is a service primitive transition and $r' \neq s$ then replace transition (r, ρ, r') with transition $(r, \rho, r'a)$ in sig_pre((s, σ, s')).

e) If transition $(r, \rho, r') \in \text{sig\_pre}((s, \sigma, s'))$ where (s, σ, s') is a service primitive transition and $r' = s$ then remove transition (r, ρ, r') from $\text{sig\_pre}((s, \sigma, s'))$.

f) If transition $(r, \rho, r') \in \text{o\_suc}((s, \sigma, s')) \setminus \text{sig\_suc}((s, \sigma, s'))$, where (s, σ, s') is a service primitive transition then add transition $(r, \rho, r'a)$ to $\text{sig\_suc}((s, \sigma, s'))$.

g) For every $(r'a, \sigma, s')$ defined, $\sigma \in \Sigma_{CS}$, add transition $(r, \rho, r'a)$ to $\text{sig\_pre}((r'a, \sigma, s'))$.

h) Add action ρ to SA.

Theorem 5.3: States and transitions added by the steps above ensure that $\text{sig\_suc}(t) = \text{o\_suc}(t)$, for all service primitive transitions t, and that the language accepted by the CFSM does not change.

Proof: Let E' be the CFSM obtained by applying steps 1 to 3 above to CFSM E for a state r' in E such that there exists a non-empty set of incoming transitions, $\{(r, \rho, r') \mid (r, \rho, r') \in \text{o\_suc}(t) \setminus \text{sig\_suc}(t)$, where t is any service primitive transition in $E\}$.

(1) Observe that the new state $r'a$ with its outgoing transitions, added by steps 1 and 2 of the procedure, is *equivalent* to the state r' already defined in the CFSM. Two states s_1 and s_2 in a CFSM are **equivalent** if and only if for every $\alpha \in \Sigma^*$, $\delta(s_1, \alpha) = \delta(s_2, \alpha)$. Furthermore, in step 3 every incoming transition to r' removed is replaced by an incoming transition to $r'a$ with the same tail state. Since r' and $r'a$ are equivalent the behaviour is the same. Therefore, $L(E') = L(E)$.

(2) Observe that the new transitions outgoing from the new state $r'a$, added by steps 1 and 2 of the procedure, cannot be observable successor transitions (and therefore cannot be significant successor transitions) of the original service primitive transitions. This is because the only transitions defined to the new state $r'a$ in E' are observable transitions [step 3].

(3) For every $(r, \rho, r') \in o\_suc(t) \setminus sig\_suc(t)$ in E, where t is any service primitive transition in E, if $(r, \rho, r') \in o\_suc(t)$ in E then $(r, \rho, r'a) \in o\_suc(t)$ in E'. Furthermore, in E', $(r, \rho, r'a) \notin sig\_suc(t) \Rightarrow \exists (r'a, \theta, s) \in o\_suc(t) \Rightarrow$ there exists a path from the head state of t to $r'a$ that contains only unobservable transitions $\Rightarrow$ there exists an unobservable transition with head state $r'a$, which leads to a contradiction. Therefore, the result of step 3 of the procedure is the replacement of (r, ρ, r') with $(r, \rho, r'a)$ in the sets of observable successors, since the transition (r, ρ, r') is replaced with $(r, \rho, r'a)$ in E', and the addition of $(r, \rho, r'a)$ to the corresponding sets of significant successor transitions.

(4) Any new service primitive transition added with tail state $r'a$ will have the same set of observable successor transitions and the same set of significant successor transitions as the corresponding original service primitive transition will tail state r', since $r'a$ and r' are equivalent.

From (2), (3) and (4), we can say that in E', the set $\{(r, \rho, r') \mid (r, \rho, r') \in o\_suc(t) \setminus sig\_suc(t)$, where t is any service primitive transition in $E0_P$ ($E1_Q$), r is any state$\}$ is empty and the set $\{(r, \rho, r'a) \mid (r, \rho, r') \in o\_suc(t) \setminus sig\_suc(t)$, where t is any service primitive transition in $E0_P$ ($E1_Q$), r is any state$\}$ is empty. This implies that if we apply the steps of the procedure to every state in the original E then in the resulting CFSM $o\_suc(t) = sig\_suc(t)$, for every service primitive transition t. □

5.4.2 Constructing the conversion seed

Given the specification of the conversion system requirements, *CS*, and the specification of the protocols $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$, along with the significant action set *SA*, and the sets of significant transitions, the following steps are taken to find a conversion seed X, such that a conversion system protocol $\{E0_P, C, E1_Q\}$ that satisfies external equivalence and semantic equivalence with respect to X also satisfies the safety property with respect to *CS*.

1. Construct $(E0_P \times E1_Q \times CS)$~red as follows:

   i) Extend *CS* by adding a dead state, *d*, and the following transitions:

$\forall s \in S_{CS}$, $s \neq d$, $\forall \sigma \in \Sigma_{CS}$, if $\delta(s, \sigma)$ is undefined then define $\delta(s, \sigma) = d$.

$\forall s \in S_{CS}$, $s \neq d$, $\forall \sigma \in ((\Sigma_{E0_P} \cup \Sigma_{E1_Q}) \setminus \Sigma_{CS})$, define $\delta(s, \sigma) = s$.

$\forall \sigma \in (\Sigma_{CS} \cup \Sigma_{E0_P} \cup \Sigma_{E1_Q})$, $\delta(d, \sigma)$ is undefined.

II) Construct $Y = (E0_P \times E1_Q) \times CS = (S_Y, \Sigma_Y, \delta_Y, s0_Y)$.

III) Reduce Y in the following manner to obtain $(E0_P \times E1_Q \times CS)$~red as the result of the reduction:

A) Repeat for every dead state (s_1, s_2, s_3) in Y:

i) Repeat for every incoming transition $((r_1, r_2, r_3), \sigma, (s_1, s_2, s_3))$:

a) $\delta_Y((r_1, r_2, r_3), \sigma)$ = undefined

b) if $\sigma \in \Sigma_{CS}$ then

- find $SP = \{((q_1, q_2, q_3), \rho, (p_1, p_2, p_3)) \mid ((q_1, q_2, q_3), \rho, (p_1, p_2, p_3))$ is the last transition preceding (r_1, r_2, r_3) in a path ending at state (r_1, r_2, r_3) satisfying ($\sigma \in \Sigma_{E0_p}$ and $(q_1, \rho, p_1) \in$ sig_pre$((r_1, \sigma, s_1))$) or ($\sigma \in \Sigma_{E1_Q}$ and $(q_2, \rho, p_2) \in$ sig_pre$((r_2, \sigma, s_2))$)$\}$.
- $\forall((q_1, q_2, q_3), \rho, (p_1, p_2, p_3)) \in SP$, $\delta_Y((q_1, q_2, q_3), \rho) = na$.

ii) $S_Y := S_Y \setminus \{(s1, s2, s3)\}$

B) Repeat for every service primitive transition $((r_1, r_2, r_3), \sigma, (s_1, s_2, s_3))$, $\sigma \in \Sigma_{CS}$:

i) find $SS = \{((q_1, q_2, q_3), \rho, (p_1, p_2, p_3)) \mid ((q_1, q_2, q_3), \rho, (p_1, p_2, p_3))$ is the first transition following (s_1, s_2, s_3) in a path starting at state (s_1, s_2, s_3) satisfying ($\sigma \in \Sigma_{E0_p}$ and $(q_1, \rho, p_1) \in$ sig_suc$((r_1, \sigma, s_1))$) or ($\sigma \in \Sigma_{E1_Q}$ and $(q_2, \rho, p_2) \in$ sig_suc$((r_2, \sigma, s_2))$.

ii) $\forall((q_1, q_2, q_3), \rho, (p_1, p_2, p_3)) \in SS$,

a) find $SP = \{ ((v_1, v_2, v_3), \theta, (w_1, w_2, w_3)) \mid ((v_1, v_2, v_3), \theta, (w_1, w_2, w_3))$ is the first transition in a path from (s_1, s_2, s_3) to (q_1, q_2, q_3) such that $((\sigma \in \Sigma_{E0_P}$ and $(v_2, \theta, w_2) \in$ sig_pre(t), for some $t = (r, \rho', r')$ in $E1_Q$, $\rho' \in \Sigma_{CS})$ or $(\sigma \in \Sigma_{E1_Q}$ and $(v_1, \theta, w_1) \in$ sig_pre(t), for some $t = (r, \rho', r')$ in $E0_P$, $\rho' \in \Sigma_{CS})$.

b) $\forall((v_1, v_2, v_3), \theta, (w_1, w_2, w_3)) \in SP$, $\delta_Y((v_1, v_2, v_3), \theta) = na$.

2. Find W such that

$$L(W) = \text{Pre}(L(\text{Proj}_{SA}(E0_P \times E1_Q \times CS)\sim\text{red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red})))$$

3. $X = W^{-1}$ is the conversion seed.

The possible behaviour of a conversion system with participating entities $E0_P$ and $E1_Q$ is the interleaving of the language of $E0_P$ with the language of $E1_Q$. We constructed the intermediate CFSMs $(E0_P \times E1_Q \times CS)\sim$red [step 1] and W [step 2] in such a way that given a possible interleaving of a sequence of actions accepted by $E0_P$ with a sequence of actions accepted by $E1_Q$, if the sequence of actions is not accepted by $(E0_P \times E1_Q \times CS)\sim$red then the projection of the sequence over the significant action set is not accepted by W. Furthermore, if a possible interleaving projected over the significant action set is accepted by W then the possible interleaving is accepted by $((E0_P \times E1_Q \times CS)\sim$red) and the possible interleaving projected over service primitive actions is accepted by CS. In other words, the behaviour can be part of a safe solution. This is formally stated below as Propositions 5.1 and 5.2.

Proposition 5.1: $\forall\alpha \in L(E0_P \times E1_Q), \alpha \notin L((E0_P \times E1_Q \times CS)\sim\text{red}) \Rightarrow$

$$\alpha|_{SA} \notin L(\text{Proj}_{SA}((E0_P \times E1_Q \times CS)\sim\text{red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red})).$$

Proof: by the construction of $((E0_P \times E1_Q \times CS)\sim$red) and W. □

Proposition 5.2: $\forall\alpha \in L(E0_P \times E1_Q)$,

$$\alpha|_{SA} \in L(\text{Proj}_{SA}(E0_P \times E1_Q \times CS)\sim\text{red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red}))) \Rightarrow$$
$$\alpha \in L((E0_P \times E1_Q \times CS)\sim\text{red}) \Rightarrow \alpha|_{\Sigma_{CS}} \in L(CS)$$

Proof: $\alpha \in L(E0_P \times E1_Q)$ and $\alpha|_{\Sigma_{CS}} \notin L(CS)$

$\Rightarrow$ $\alpha \notin L(E0_P \times E1_Q \times CS)$

$\Rightarrow$ $\alpha \notin L((E0_P \times E1_Q \times CS)\sim\text{red})$ [by construction of $((E0_P \times E1_Q \times CS)\sim\text{red})$]

$\Rightarrow$ $\alpha|_{SA} \notin L(\text{Proj}_{SA}((E0_P \times E1_Q \times CS)\sim\text{red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red}))$

[by Proposition 5.1]. □

The possible behaviour of a conversion system with participating entities $E0_P$ and $E1_Q$ and involved entities $E1_P$ and $E0_Q$, that satisfies external equivalence, is the language accepted by the protocol $\{E0_P, E1_P \times E0_Q, E1_Q\}$, where $E0_P$ and $E1_P \times E0_Q$ communicate directly, $E1_P \times E0_Q$ and $E1_Q$ communicate directly, and $E0_P$ and $E1_Q$ do not communicate. It is useful to characterize the part the involved entities, $E1_P \times E0_Q$, play in this behaviour, or the part the participating entities, $E0_P$ and $E1_Q$, play in this behaviour. Given a sequence of actions, α, accepted by the protocol $\{E0_P, E1_P \times E0_Q, E1_Q\}$, the **participating part of** α, α_o, is the sequence of actions that represents the part the participating entities play in the behaviour α, and the **involved part of** α, α_I, is the sequence of actions that represents the part the involved entity plays in the behaviour α.

Let $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$, where $\text{sync}(E0_P, E1_P \times E0_Q) = \text{true}$, $\text{sync}(E1_P \times E0_Q, E1_Q) = \text{true}$ and $\text{sync}(E0_P, E1_Q) = \text{false}$.

Define the **participating part of** α, α_o, inductively as follows:

$$\begin{aligned}
\alpha_o &= \varepsilon, && \text{if } \alpha = \varepsilon,\\
(\sigma.\alpha)_o &= +\sigma.\alpha_o && \text{if } \sigma \in M_{E1_P,E0_P} \text{ or } \sigma \in M_{E0_Q,E1_Q}\\
&= -\sigma.\alpha_o && \text{if } \sigma \in M_{E0_P,E1_P} \text{ or } \sigma \in M_{E1_Q,E0_Q}\\
&= \sigma.\alpha_o, && \text{if } \sigma \in \Sigma_{CS}.
\end{aligned}$$

Define the **involved part of** α, α_I, inductively as follows:

$$\begin{aligned}
\alpha_I &= \varepsilon, && \text{if } \alpha = \varepsilon,\\
(\sigma.\alpha)_I &= -\sigma.\alpha_I && \text{if } \sigma \in M_{E1_P,E0_P} \text{ or } \sigma \in M_{E0_Q,E1_Q}\\
&= +\sigma.\alpha_I && \text{if } \sigma \in M_{E0_P,E1_P} \text{ or } \sigma \in M_{E1_Q,E0_Q}
\end{aligned}$$

$= \alpha_I$, if $\sigma \in \Sigma_{CS}$.

Let $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$, where async$(E0_P, E1_P \times E0_Q)$ = true, async$(E1_P \times E0_Q, E1_Q)$ = true and async$(E0_P, E1_Q)$ = false.

Define the **participating part of** α, $\alpha_o = \alpha|_{\Sigma_{E0P} \cup \Sigma_{E1Q}}$.

Define the **involved part of** α, $\alpha_I = \alpha|_{\Sigma_{E1P} \cup \Sigma_{E0Q}}$.

In the case where the protocols P and Q communicate using the synchronous model of communication, if we are given a sequence of actions, α, that is a possible behaviour of the conversion system that satisfies external equivalence, we can show that the involved part of α projected over the alphabet of the conversion seed is accepted by the conversion seed if and only if the inverse of the participating part of α projected over the significant action set is accepted by the conversion seed. This is formalized as follows:

Proposition 5.3: $\forall \alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$, where sync$(E0_P, E1_P \times E0_Q)$ = true, sync$(E1_P \times E0_Q, E1_Q)$ = true and sync$(E0_P, E1_Q)$ = false,

$\alpha_I|_{\Sigma_X} \in L(X) \Leftrightarrow (\alpha_o|_{SA})^{-1} \in L(X)$.

Proof: $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$ with communication relationship sync

$\Rightarrow$ $\alpha_I = (\alpha_o|_\Delta)^{-1}$, where $\Delta = (\Sigma_{E0_P} \cup \Sigma_{E1_Q}) \setminus \Sigma_{CS}$.

$\Rightarrow$ $\alpha_I|_{\Sigma_X} = (\alpha_o|_\Delta)^{-1}|_{\Sigma_X}$

$\Rightarrow$ $\alpha_I|_{\Sigma_X} = ((\alpha_o|_\Delta)|_{SA})^{-1}$

$\Rightarrow$ $\alpha_I|_{\Sigma_X} = (\alpha_o|_{SA})^{-1}$

$\Rightarrow$ $\alpha_I|_{\Sigma_X} \in L(X) \Leftrightarrow (\alpha_o|_{SA})^{-1} \in L(X)$ □

Proposition 5.3 demonstrates the relationship between the conversion seed and the participating part of a sequence of actions that is a possible behaviour of a conversion system that satisfies external equivalence in the case of synchronous communication. We must also consider the case of asynchronous communication. In asynchronous communication, the fact that the transmission of a message and the corresponding

reception of that message are two separate events that can be separated by the occurrence of certain other events involving the same or different entities, complicates the relationship between the conversion seed and the participating part of the behaviour.

Consider the protocol $\{E0_P, E1_P \times E0_Q, E1_Q\}$, where async$(E0_P, E1_P \times E0_Q)$ = true, async$(E1_P \times E0_Q, E1_Q)$ = true and async$(E0_P, E1_Q)$ = false. Given the involved part for an unknown behaviour $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$, α_I, a **possible participating part consistent with** α_I, is any sequence $\chi \in E0_P \times E1_Q$ that satisfies all of the following conditions:

α_o_1 $(\chi|_\Delta).\alpha' = (\alpha_I)^{-1}|_\Delta$, where $\Delta = \Sigma^+_{E0_P}$ and $\alpha' \in (\Sigma^+_{E0_P})^*$

α_o_2 $\chi|_\Delta = (\alpha_I)^{-1}|_\Delta.\alpha'$, where $\Delta = \Sigma^-_{E0_P}$ and $\alpha' \in (\Sigma^-_{E0_P})^*$

α_o_3 $(\chi|_\Delta).\alpha' = (\alpha_I)^{-1}|_\Delta$, where $\Delta = \Sigma^+_{E1_Q}$ and $\alpha' \in (\Sigma^+_{E1_Q})^*$

α_o_4 $\chi|_\Delta = (\alpha_I)^{-1}|_\Delta.\alpha'$, where $\Delta = \Sigma^-_{E1_Q}$ and $\alpha' \in (\Sigma^-_{E1_Q})^*$

α_o_5 if the action $+m$ precedes the action $-n$ in α_I then the corresponding action $-m$ precedes the corresponding action $+n$ in χ.

The conditions α_o_1, α_o_2, α_o_3 and α_o_4 are due to the channels with bounds greater than zero between the entities. Condition α_o_5 is due to the observation that if the message m is received by $E1_P \times E0_Q$ before it sends message n then the message m must be sent by $E0_P$ or $E1_Q$ before $E0_P$ or $E1_Q$ receives the message n.

Given α_I for an unknown $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$, we can construct the set of possible participating parts consistent with α_I, $A_O(\alpha_I)$, as follows:

1. Find β, such that β is the longest sequence in $L(E0_P \times E1_Q)$ that satisfies $\alpha_I = (\beta|_\Delta)^{-1}$, where $\Delta = (\Sigma_{E0_P} \cup \Sigma_{E1_Q}) \setminus \Sigma_{CS}$, and add β to $A_O(\alpha_I)$. β is the **root sequence** of $A_O(\alpha_I)$.
2. For $\beta' \in A_O(\alpha_I)$ modify β' by one of the following steps and add the modified sequence to $A_O(\alpha_I)$, until no more unique sequences are added.

I) swap adjacent reception actions, "$+m+n$" $\rightarrow$ "$+n+m$",
if ($+m \in \Sigma_{E0_P}$, $+n \in \Sigma_{E1_Q}$) or ($+n \in \Sigma_{E0_P}$, $+m \in \Sigma_{E1_Q}$).

II) swap adjacent transmission actions, "$-m-n$" $\rightarrow$ "$-n-m$",
if ($-m \in \Sigma_{E0_P}$, $-n \in \Sigma_{E1_Q}$) or ($-n \in \Sigma_{E0_P}$, $-m \in \Sigma_{E1_Q}$).

III) swap a reception action with the following transmission action,
"$+m-n$" $\rightarrow$ "$-n+m$", if ($+m \in \Sigma_{E0_P}$, $-n \in \Sigma_{E1_Q}$) or ($-n \in \Sigma_{E0_P}$, $+m \in \Sigma_{E1_Q}$).

IV) swap a service primitive action with an adjacent action, "$\sigma_1\sigma_2$" $\rightarrow$ "$\sigma_2\sigma_1$",
if ($\sigma_1 \in \Sigma_{CS}$ or $\sigma_2 \in \Sigma_{CS}$) and (($\sigma_1 \in \Sigma_{E0_P}$, $\sigma_2 \in \Sigma_{E1_Q}$) or ($\sigma_2 \in \Sigma_{E0_P}$, $\sigma_1 \in \Sigma_{E1_Q}$)).

V) remove the last action of β', σ, if $\sigma = +m$, $+m \in \Sigma_{E0_P} \cup \Sigma_{E1_Q}$ or $\sigma \in \Sigma_{CS}$.

VI) add an action to the end of β', σ, if $\sigma = -m$, $-m \in \Sigma_{E0_P} \cup \Sigma_{E1_Q}$ or $\sigma \in \Sigma_{CS}$, and $\beta'.\sigma \in L(E0_P \times E1_Q)$.

Recall that this approach is applicable to a subset of the protocols that can be specified by the CFSM model for asynchronous communication introduced in Section 5.2. Given protocols P and Q in that subset, we can show that a sequence of actions that is an interleaving of the behaviour of the participating entities $E0_P$ and $E1_Q$ is a possible participating part consistent with a given involved part if and only if the sequence of actions is in $A_O(\alpha_I)$. This is formalized as follows:

Proposition 5.4: Consider the protocol $\{E0_P, E1_P \times E0_Q, E1_Q\}$, where async($E0_P$, $E1_P \times E0_Q$) = true, async($E1_P \times E0_Q$, $E1_Q$) = true and async($E0_P$, $E1_Q$) = false. If given $E0_P$, $E1_P$, $E0_Q$, and $E1_Q$ that satisfy conditions C1, C2 and C3 given in Section 5.4; and given α_I for an unknown $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$, then
$\beta \in L(E0_P \times E1_Q)$ and $\beta \in A_O(\alpha_I) \Leftrightarrow \beta$ is a possible α_o consistent with α_I.

Proof:

($\Rightarrow$ part) Observe the following:

(1) The root sequence of $A_O(\alpha_I)$, β, satisfies conditions α_o_1, α_o_2, α_o_3, α_o_4 and α_o_5 given earlier. Therefore $\beta \in L(E0_P \times E1_Q) \Rightarrow \beta$ is a possible α_o consistent with α_I.

(2) The sequence obtained by performing any one of the six modifications in the definition of $A_O(\alpha_I)$ to a possible α_o consistent with α_I will also satisfy conditions α_o_1, α_o_2, α_o_3, α_o_4 and α_o_5 given earlier. Therefore, the sequence obtained by performing any of the six modifications in the definition of $A_O(\alpha_I)$ is also a possible α_o consistent with α_I.

Thus, $\beta \in L(E0_P \times E1_Q)$ and $\beta \in A_O(\alpha_I) \Rightarrow \beta$ is a possible α_o consistent with α_I. [by (1) and (2)]

($\Leftarrow$ part) β is a possible α_o consistent with $\alpha_I \Rightarrow \beta \in L(E0_P \times E1_Q)$ [by definition of a possible α_o consistent with α_I.]

Now we must show that β is a possible α_o consistent with $\alpha_I \Rightarrow \beta \in A_O(\alpha_I)$.

Assume there is a sequence $\beta \in L(E0_P \times E1_Q)$ which is a possible α_o consistent with α_I and $\beta \notin A_O(\alpha_I)$. Let $\chi \in A_O(\alpha_I)$, $\chi \in L(E0_P \times E1_Q)$, be the shortest sequence that has the longest prefix in common with a prefix of β and let β_1 be the longest common prefix of β and χ; i.e., $\chi = \beta_1.\chi_2$, $\beta = \beta_1.\beta_2$.

(1) $\beta_2 = \varepsilon$; $\chi_2 = \varepsilon$;

$\Rightarrow \quad \beta = \chi$

$\Rightarrow \quad \beta \in A_O(\alpha_I)$, which leads to a contradiction.

(2) $\beta_2 = \varepsilon$; $\chi_2 = \sigma.\chi'_2$; $\sigma = +m$ or $\sigma \in \Sigma_{CS}$

χ is the shortest sequence that has the longest prefix in common with a prefix of β. Therefore χ'_2 must contain some peer protocol message transmission actions that were not added by step VI and therefore were in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow$ β does not contain some peer protocol message transmission actions that were in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow \quad \beta|_\Delta \neq (\alpha_I)^{-1}|_\Delta.\alpha'$,

$(\Delta = \Sigma_{\overline{E}0_P}$ and $\alpha' \in (\Sigma_{\overline{E}0_P})^*)$ or $(\Delta = \Sigma_{\overline{E}1_Q}$ and $\alpha' \in (\Sigma_{\overline{E}1_Q})^*)$

$\Rightarrow$ β is not a possible α_o consistent with α_I, which leads to a contradiction.

(3) $\beta_2 = \varepsilon$; $\chi_2 = -m.\chi'_2$

χ is the shortest sequence that has the longest prefix in common with a prefix of β. Therefore the peer protocol message transmission action $-m$ was not added by step VI and therefore was in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow$ β does not contain a peer protocol message transmission action that was in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow$ $\beta|_\Delta \neq (\alpha_I)^{-1}|_\Delta.\alpha'$,

$(\Delta = \Sigma_{\overline{E}0_P}$ and $\alpha' \in (\Sigma_{\overline{E}0_P})^*)$ or $(\Delta = \Sigma_{\overline{E}1_Q}$ and $\alpha' \in (\Sigma_{\overline{E}1_Q})^*)$

$\Rightarrow$ β is not a possible α_o consistent with α_I, which leads to a contradiction.

(4) $\beta_2 = +m.\beta'_2$; $\chi_2 = \varepsilon$; $+m \in \Sigma^+_{E0_P}$

β is a possible α_o consistent with α_I and therefore $+m$ is in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow$ During the transformation steps from the root sequence to χ, the action $+m$ was removed by step V.

$\Rightarrow$ $\exists \chi' \in A_O(\alpha_I)$, $\chi' \in L(E0_P \times E1_Q)$, such that χ' satisfies $\chi' = \beta_1.\delta.+m$, where $\delta \in (\Sigma^+_{\overline{E}1_Q} \cup \Sigma^U_{\overline{E}1_Q})^*$.

Note that δ cannot contain peer protocol message transmission actions since $\beta_1 = \chi \in A_O(\alpha_I)$ and there is no step in the definition of $A_O(\alpha_I)$ that removes peer protocol message transmission actions. Furthermore, δ cannot contain peer protocol message receptions from entity $E0_P$ since β is a possible α_o consistent with α_I. Finally, δ cannot contain service primitives from entity $E0_P$ since $E0_P$ satisfies condition C2 given in Section 5.4.

$\Rightarrow$ $\exists \chi'' \in A_O(\alpha_I)$ such that χ'' satisfies $\chi'' = \beta_1.+m.\delta$ [by steps I, IV]

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that χ''' satisfies $\chi''' = \beta_1.+m$ [by step V]

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that the common prefix of χ''' and β is $\chi.+m = \beta_1.+m$.

$\Rightarrow$ $\chi \in A_O(\alpha_I)$ is not the shortest sequence that has the *longest prefix* in common with a prefix of β, which leads to a contradiction.

The same argument holds for $\beta_2 = +m.\beta'_2$; $\chi_2 = \varepsilon$; $+m \in \Sigma^{+}_{E1_Q}$

(5) $\beta_2 = \sigma.\beta'_2$; $\chi_2 = \varepsilon$; $\sigma = -m$ or $\sigma \in \Sigma_{CS}$

$\Rightarrow$ $\chi.\sigma \in A_O(\alpha_I)$ [by step VI]

$\Rightarrow$ $\exists\chi' \in A_O(\alpha_I)$ such that the common prefix of χ' and β is $\chi.\sigma = \beta_1.\sigma$.

$\Rightarrow$ $\chi \in A_O(\alpha_I)$ is not the shortest sequence that has the *longest prefix* in common with a prefix of β, which leads to a contradiction.

(6) $\beta_2 = \sigma_1.\beta'_2$; $\chi_2 = \sigma_2.\chi'_2$; $((\sigma_1 \in \Sigma^{-}_{E0_P}, \sigma_2 \in \Sigma_{E0_P})$ or $(\sigma_1 \in \Sigma_{E1_Q}, \sigma_2 \in \Sigma_{E1_Q}))$.

χ is the shortest sequence that has the longest prefix in common with a prefix of β. Therefore σ_2 has not been added by step VI and therefore exists in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow$ $\beta|_\Delta \neq (\alpha_I)^{-1}|_\Delta.\alpha'$,

$(\Delta = \Sigma^{-}_{E0_P}$ and $\alpha' \in (\Sigma^{-}_{E0_P})^*)$ or $(\Delta = \Sigma^{-}_{E1_Q}$ and $\alpha' \in (\Sigma^{-}_{E1_Q})^*)$,

if both σ_1 and σ_2 are peer protocol message receptions;

or $(\beta|_\Delta).\alpha' \neq (\alpha_I)^{-1}|_\Delta$,

$(\Delta = \Sigma^{+}_{E0_P}$ and $\alpha' \in (\Sigma^{+}_{E0_P})^*)$ or $(\Delta = \Sigma^{+}_{E1_Q}$ and $\alpha' \in (\Sigma^{+}_{E1_Q})^*)$,

if both σ_1 and σ_2 are peer protocol message transmissions.

$\Rightarrow$ β is not a possible α_o consistent with α_I, which leads to a contradiction.

Note that the case where σ_1 is a peer protocol message transmission and σ_2 is a peer protocol message reception (or vice versa) cannot occur because the original protocols satisfy condition C1 given in Section 5.4. Also, the case where σ_1 is a peer protocol message and σ_2 is a service primitive (or vice versa) cannot occur because the original protocols satisfy C2 given in Section 5.4. Finally, the case where σ_1 is a service primitive and σ_2 is a service primitive cannot occur because the original protocols satisfy condition C3 given in Section 5.4.

(7) $\beta_2 = +m.\beta'_2$; $\chi_2 = -n.\chi'_2$; $+m \in \Sigma_{E0_P}$, $-n \in \Sigma_{E1_Q}$

β is a possible α_o consistent with the given α_I and therefore $+m$ is in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow$ $\exists\chi' \in A_O(\alpha_I)$, $\chi' \in L(E0_P \times E1_Q)$, such that χ' satisfies $\chi' = \beta_1.-n.\delta.+m.\delta'$, where $\delta \in \Sigma^*_{E1_Q}$.

Note that δ cannot contain peer protocol message receptions from entity $E0_P$ since β is a possible α_o consistent with α_I. Furthermore, δ cannot contain service primitives or peer protocol message transmissions from entity $E0_P$ since $E0_P$ satisfies conditions C1 and C2 given in Section 5.4.

$\Rightarrow$ $\exists\chi'' \in A_O(\alpha_I)$ such that χ'' satisfies $\chi'' = \beta_1.-n.+m.\ \delta.\delta'$ [by steps I, III, IV]

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that χ''' satisfies $\chi''' = \beta_1.+m.-n.\delta.\delta'$, if $+m$ preceded $-n$ in the root sequence[by step III]; or β is not a possible α_o consistent with α_I, if $+m$ did not precede $-n$ in the root sequence.

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that the common prefix of χ'' and β is $\beta_1.+m$

or β is not a possible α_o consistent with α_I.

$\Rightarrow$ $\chi \in A_O(\alpha_I)$ is not the shortest sequence that has the *longest prefix* in common with a prefix of β, which leads to a contradiction, or β is not a possible α_o consistent with α_I , which also leads to a contradiction.

The same argument holds for $\beta_2 = +m.\beta'_2$; $\chi_2 = -n.\chi'_2$; $+m \in \Sigma_{E1_Q}$, $-n \in \Sigma_{E0_P}$

(8) $\beta_2 = +m.\beta'_2$; $\chi_2 = \sigma.\chi'_2$; $+m \in \Sigma_{E0_P}$, $\sigma \in \Sigma_{E1_Q}$, $\sigma = +n$ or $\sigma \in \Sigma_{CS}$

β is a possible α_o consistent with the given α_I and therefore $+m$ is in the root sequence of $A_O(\alpha_I)$.

$\Rightarrow$ $\exists\chi' \in A_O(\alpha_I)$, $\chi' \in L(E0_P \times E1_Q)$, such that χ' satisfies $\chi' = \beta_1.\sigma.\delta.+m.\delta'$, where $\delta \in \Sigma^*_{E1_Q}$.

Note that δ cannot contain peer protocol message receptions from entity $E0_P$ since β is a possible α_o consistent with α_I. Furthermore, δ cannot contain service primitives or peer protocol message transmissions from entity $E0_P$ since $E0_P$ satisfies conditions C1 and C2 given in Section 5.4.

$\Rightarrow$ $\exists\chi'' \in A_O(\alpha_I)$ such that χ'' satisfies $\chi'' = \beta_1.\sigma.+m.\ \delta.\delta'$ [by steps I, III, IV]

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that χ''' satisfies $\chi''' = \beta_1.+m.\sigma.\delta.\delta'$ [by rules I, IV]

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that the common prefix of χ'' and β is $\beta_1.+m$.

$\Rightarrow$ $\chi \in A_O(\alpha_I)$ is not the shortest sequence that has the *longest prefix* in common with a prefix of β, which leads to a contradiction.

The same argument holds for $\beta_2 = +m.\beta'_2$; $\chi_2 = \sigma.\chi'_2$; $+m \in \Sigma_{E1_Q}$, $\sigma \in \Sigma_{E0_P}$; $\sigma = +n$ or $\sigma \in \Sigma_{CS}$

(9) $\beta_2 = \sigma_1.\beta'_2$, $\chi_2 = \sigma_2.\chi'_2$; $\sigma_1 \in \Sigma_{E0_P}$, $\sigma_2 \in \Sigma_{E1_Q}$, $\sigma_1 = -m$ or $\sigma_1 \in \Sigma_{CS}$

a) σ_1 is in the root sequence of $A_O(\alpha_I)$

$\Rightarrow$ $\exists\chi' \in A_O(\alpha_I)$, $\chi' \in L(E0_P \times E1_Q)$, such that χ' satisfies $\chi' = \beta_1.\sigma_2.\delta.\sigma_1.\delta'$, where $\delta \in \Sigma^*_{E1_Q}$.

Note than δ cannot contain actions from entity $E0_P$ since β is a possible α_o consistent with α_I and since entity $E0_P$ satisfies conditions C1 and C2 given in Section 5.4.

$\Rightarrow$ $\exists\chi'' \in A_O(\alpha_I)$ such that χ'' satisfies $\chi'' = \beta_1.\sigma_2.\sigma_1.\ \delta.\delta'$ [by steps II, III, IV]

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that χ''' satisfies $\chi''' = \beta_1.\sigma_1.\sigma_2.\delta.\delta'$ [by steps II, III, IV]

$\Rightarrow$ $\exists\chi''' \in A_O(\alpha_I)$ such that the common prefix of χ'' and β is $\beta_1.\sigma_1$.

$\Rightarrow$ $\chi \in A_O(\alpha_I)$ is not the shortest sequence that has the *longest prefix* in common with a prefix of β, which leads to a contradiction.

b) σ_1 is not in the root sequence of $A_O(\alpha_I)$

$\Rightarrow$ $\exists \chi' \in A_O(\alpha_I)$, $\chi' \in L(E0_P \times E1_Q)$, such that χ' satisfies $\chi' = \beta_1.\sigma_2.\delta$, where $\delta \in \Sigma^*_{E1_Q}$.

Note that δ cannot contain actions from entity $E0_P$ since β is a possible α_o consistent with α_I and since entity $E0_P$ satisfies condition C1 and C2 given in Section 5.4.

$\Rightarrow$ $\exists \chi'' \in A_O(\alpha_I)$ such that χ'' satisfies $\chi'' = \beta_1.\sigma_2.\delta.\sigma_1$ [by step VI]

$\Rightarrow$ $\exists \chi''' \in A_O(\alpha_I)$ such that χ''' satisfies $\chi''' = \beta_1.\sigma_2.\sigma_1.\delta$ [by steps II, III, IV]

$\Rightarrow$ $\exists \chi'''' \in A_O(\alpha_I)$ such that χ'''' satisfies $\chi'''' = \beta_1.\sigma_1.\sigma_2.\delta$ [by steps II, III, IV]

$\Rightarrow$ $\exists \chi'''' \in A_O(\alpha_I)$ such that the common prefix of χ'''' and β is $\beta_1.\sigma_1$.

$\Rightarrow$ $\chi \in A_O(\alpha_I)$ is not the shortest sequence that has the *longest prefix* in common with a prefix of β, which leads to a contradiction. □

Given a sequence of actions that is accepted by the intermediate CFSM $(E0_P \times E1_Q \times CS)$~red, if we modify the sequence of actions by applying one of the six construction steps of the definition of $A_O(\alpha_I)$, we show in the following three lemmas that the resulting sequence of actions is also accepted by the intermediate CFSM $(E0_P \times E1_Q \times CS)$~red. It then follows that if the root sequence for the set of possible participating parts consistent with α_I is accepted by $(E0_P \times E1_Q \times CS)$~red then all sequences of actions in the set of possible participating parts consistent with α_I are accepted by $(E0_P \times E1_Q \times CS)$~red. This is formalized by the following three lemmas.

Lemma 5.1: If $\beta \in L((E0_P \times E1_Q \times CS)\text{~red})$ and β' is the result of applying step I, II, III or IV of the definition of $A_O(\alpha_I)$ to β then $\beta' \in L((E0_P \times E1_Q \times CS)\text{~red})$.

Proof: Let $\beta = \beta1.\sigma1.\sigma2.\beta2$, where $\sigma1$ and $\sigma2$ are the adjacent actions swapped to form $\beta' = \beta1.\sigma2.\sigma1.\beta2$. Observe the following:

(1) $\beta \in L((E0_P \times E1_Q \times CS)\text{~red})$

$\Rightarrow \quad \beta \in L(E0_P \times E1_Q \times CS)$

$\Rightarrow \quad \beta \in L(E0_P \times E1_Q)$

$\Rightarrow \quad \beta|_{\Sigma_{CS}} \in L(CS)$

(2) $((\sigma_1 \in \Sigma_{E0_P}, \sigma_2 \in \Sigma_{E1_Q})$ or $(\sigma_2 \in \Sigma_{E0_P}, \sigma_1 \in \Sigma_{E1_Q}))$ [by def'n of steps I, II, III, IV]

(3) $\beta' \in L(E0_P \times E1_Q)$ [by (1) and (2)]

(4) if none, or one, of $\sigma 1$ and $\sigma 2$ is in Σ_{CS} then $\beta'|_{\Sigma_{CS}} = \beta|_{\Sigma_{CS}}$ and $\beta' \in L(E0_P \times E1_Q \times CS)$ [by (3)]

(5) if both of $\sigma 1$ and $\sigma 2$ are in Σ_{CS} then

$\beta 1.\sigma 2.\sigma 1.\beta 2 \notin L(E0_P \times E1_Q \times CS)$

$\Rightarrow \beta 1.\sigma 2.\sigma 1 \notin L(E0_P \times E1_Q \times CS)$

$\Rightarrow \beta 1.\sigma 2 \notin L((E0_P \times E1_Q \times CS)\sim\text{red})$

$\Rightarrow \beta 1 \notin L((E0_P \times E1_Q \times CS)\sim\text{red})$

$\Rightarrow \beta \notin L((E0_P \times E1_Q \times CS)\sim\text{red})$, which leads to a contradiction.

$\Rightarrow \beta' \in L(E0_P \times E1_Q \times CS)$

(6) The following states and transitions are defined in $E0_P \times E1_Q \times CS$: [by (4) and (5)]

if $(\sigma_2 \in \Sigma_{E0_P}, \sigma_1 \in \Sigma_{E1_Q})$:

the initial state $(p0, q0)$, $(p0, q0) \text{—}\beta 1\text{→} (p, q)$, $(p, q) \text{—}\sigma 1\text{→} (p, q')$,

$(p, q') \text{—}\sigma 2\text{→} (p', q')$, $(p, q) \text{—}\sigma 2\text{→} (p', q)$, $(p', q) \text{—}\sigma 1\text{→} (p', q')$.

if $(\sigma_1 \in \Sigma_{E0_P}, \sigma_2 \in \Sigma_{E1_Q})$:

the initial state $(p0, q0)$, $(p0, q0) \text{—}\beta 1\text{→} (p, q)$, $(p, q) \text{—}\sigma 1\text{→} (p', q)$,

$(p', q) \text{—}\sigma 2\text{→} (p', q')$, $(p, q) \text{—}\sigma 2\text{→} (p, q')$, $(p, q') \text{—}\sigma 1\text{→} (p', q')$.

(7) if $(\sigma_2 \in \Sigma_{E0_P}, \sigma_1 \in \Sigma_{E1_Q})$:

if $(q, \sigma 1, q')$ is a significant transition in $E1_Q$ and the state, transition sequence

$(p0, q0)$—$\beta 1$→(p, q)—$\sigma 1$→(p, q')—$\sigma 2$→(p', q') is defined in

$(E0_P \times E1_Q \times CS)$~red then the transition (p, q') —$\sigma 1$→(p', q') is also defined in

$(E0_P \times E1_Q \times CS)$~red.

(8) if $(\sigma_2 \in \Sigma_{E0_P}, \sigma_1 \in \Sigma_{E1_Q})$:

if $(p, \sigma 2, p')$ is a significant transition in $E0_P$ and the state, transition sequence

$(p0, q0)$—$\beta 1$→(p, q)—$\sigma 1$→(p, q')—$\sigma 2$→(p', q') is defined in

$(E0_P \times E1_Q \times CS)$~red then the transition (p, q) —$\sigma 2$→(p', q) is also defined in

$(E0_P \times E1_Q \times CS)$~red.

(9) if $(\sigma_2 \in \Sigma_{E0_P}, \sigma_1 \in \Sigma_{E1_Q})$:

if $(q, \sigma 1, q')$ is not a significant transition in $E1_Q$ and $(p, \sigma 2, p')$ is not a

significant transition in $E0_P$ then (p, q') —$\sigma 1$→(p', q') and (p, q) —$\sigma 2$→(p', q)

are defined in $(E0_P \times E1_Q \times CS)$~red.

(10) if $(\sigma_1 \in \Sigma_{E0_P}, \sigma_2 \in \Sigma_{E1_Q})$:

if $(p, \sigma 1, p')$ is a significant transition in $E0_P$ and the state transitions sequence

$(p0, q0)$—$\beta 1$→(p, q)—$\sigma 1$→(p', q)—$\sigma 2$→(p', q') is defined in

$(E0_P \times E1_Q \times CS)$~red then the transition (p, q') —$\sigma 1$→(p', q') is also defined in

$(E0_P \times E1_Q \times CS)$~red.

(11) if $(\sigma_1 \in \Sigma_{E0_P}, \sigma_2 \in \Sigma_{E1_Q})$:

if $(q, \sigma 2, q')$ is a significant transition in $E1_Q$ and the state transitions sequence

$(p0, q0)$—$\beta 1$→(p, q)—$\sigma 1$→(p', q)—$\sigma 2$→(p', q') is defined in

$(E0_P \times E1_Q \times CS)$~red then the transition (p, q)—$\sigma 2$→(p, q') is also defined in

$(E0_P \times E1_Q \times CS)$~red.

(12) if $(\sigma_1 \in \Sigma_{E0_P}, \sigma_2 \in \Sigma_{E1_Q})$:

if $(p, \sigma 1, p')$ is not a significant transition in $E0_P$ and $(q, \sigma 2, q')$ is not a significant transition in $E1_Q$ then $(p, q') \text{—}\sigma 1\rightarrow (p', q')$ and $(p, q) \text{—}\sigma 2\rightarrow (p', q)$ are defined in $(E0_P \times E1_Q \times CS)$~red.

Thus, $\beta' \in L((E0_P \times E1_Q \times CS)\text{~red})$ [from (1) to (12)] □

Lemma 5.2: If $\beta \in L((E0_P \times E1_Q \times CS)\text{~red})$ and β' is the result of applying step V to β then $\beta' \in L((E0_P \times E1_Q \times CS)\text{~red})$.

Proof: Let $\beta = \beta'.\sigma$, where σ is the action removed to form β'.

$\beta \in L((E0_P \times E1_Q \times CS)\text{~red})$

$\Rightarrow \quad \beta'.\sigma \in L((E0_P \times E1_Q \times CS)\text{~red})$

$\Rightarrow \quad \beta' \in L((E0_P \times E1_Q \times CS)\text{~red})$. □

Lemma 5.3: If $\beta \in L((E0_P \times E1_Q \times CS)\text{~red})$ and β' is the result of applying step VI to β then $\beta' \in L((E0_P \times E1_Q \times CS)\text{~red})$.

Proof: Let $\beta' = \beta.\sigma$, where σ is the action added to form β'. Observe the following:

(1) $\beta \in L((E0_P \times E1_Q \times CS)\text{~red})$

$\Rightarrow \quad \beta \in L(E0_P \times E1_Q \times CS)$

$\Rightarrow \quad \beta \in L(E0_P \times E1_Q)$

$\Rightarrow \quad \beta|_{\Sigma_{CS}} \in L(CS)$

(2) $\sigma = -m$, $-m \in \Sigma_{E0_P} \cup \Sigma_{E1_Q}$ or $\sigma \in \Sigma_{CS}$, and $\beta' \in L(E0_P \times E1_Q)$ [by step VI]

(3) if $\sigma \notin \Sigma_{CS}$ then $\beta'|_{\Sigma_{CS}} = \beta|_{\Sigma_{CS}}$ and $\beta' \in L(E0_P \times E1_Q \times CS)$ [from (2)]

(4) if $\sigma \in \Sigma_{CS}$ then

$\beta.\sigma \notin L(E0_P \times E1_Q \times CS)$

$\Rightarrow \beta \notin L((E0_P \times E1_Q \times CS)\text{~red})$, which leads to a contradiction.

$\Rightarrow \beta' \in L(E0_P \times E1_Q \times CS)$

(5) The following states and transitions are defined in $E0_P \times E1_Q \times CS$: [by (3) and (4)]

if $\sigma \in \Sigma_{E0_P}$:

the initial state $(p0, q0)$, $(p0, q0) \text{—}\beta\text{→} (p, q)$, $(p, q) \text{—}\sigma\text{→} (p', q)$.

if $(\sigma \in \Sigma_{E1_Q})$:

the initial state $(p0, q0)$, $(p0, q0) \text{—}\beta\text{→} (p, q)$, $(p, q) \text{—}\sigma\text{→} (p, q')$.

(6) if $\sigma \in \Sigma_{E0_P}$:

if σ is a peer protocol message transmission or service primitive from $E0_P$ and the state transition sequence $(p0, q0) \text{—}\beta\text{→} (p, q)$ is defined in $(E0_P \times E1_Q \times CS)\sim$red then $(p, q) \text{—}\sigma\text{→} (p', q)$ is also defined in $(E0_P \times E1_Q \times CS)\sim$red.

(7) if $\sigma \in \Sigma_{E1_Q}$:

if σ is a peer protocol message transmission or service primitive from $E1_Q$ and the state transition sequence $(p0, q0) \text{—}\beta\text{→} (p, q)$ is defined in $(E0_P \times E1_Q \times CS)\sim$red then $(p, q) \text{—}\sigma\text{→} (p, q')$ is also defined in $(E0_P \times E1_Q \times CS)\sim$red.

Thus, $\beta' \in L((E0_P \times E1_Q \times CS)\sim\text{red})$ [from (1) to (7)]. □

Corollary 5.2: If the root sequence of $A_O(\alpha_I)$ is β and $\beta \in L((E0_P \times E1_Q \times CS)\sim\text{red})$ then $\beta' \in A_O(\alpha_I) \Rightarrow \beta' \in L((E0_P \times E1_Q \times CS)\sim\text{red})$.

Proof: follows from Lemmas 5.1, 5.2, and 5.3. □

For a sequence of actions that is a possible behaviour of a conversion system that satisfies external equivalence, the sequence projected over the service primitives is accepted by *CS* if and only if the participating part of the sequence projected over the service primitives is accepted by *CS*. This is formalized as follows:

Proposition 5.5: $\forall \alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\}),\ \alpha_o|_{\Sigma_{CS}} \in L(CS) \Leftrightarrow \alpha|_{\Sigma_{CS}} \in L(CS)$

Proof: The proof follows by noting that service primitives only defined in $E0_P$ and $E1_Q$. □

For a sequence of actions that is a possible behaviour of a conversion system that satisfies external equivalence, if the involved part of the sequence projected over the alphabet of the conversion seed is accepted by the conversion seed then the sequence projected over the service primitives is accepted by *CS*. i.e., the behaviour can be part of a safe solution. This is formalized as follows:

Proposition 5.6: $\forall \alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\}),\ \alpha_I|_{\Sigma_X} \in L(X) \Rightarrow \alpha|_{\Sigma_{CS}} \in L(CS)$

Proof (synchronous case): $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$ and $\alpha_I|_{\Sigma_X} \in L(X)$

$\Rightarrow$ $(\alpha_o|_{SA})^{-1} \in L(X)$ [by Proposition 5.3]

$\Rightarrow$ $\alpha_o|_{SA} \in L(\text{Proj}_{SA}(E0_P \times E1_Q \times CS)\sim\text{red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red})))$ [by construction of *X*]

$\Rightarrow$ $\alpha_o|_{\Sigma_{CS}} \in L(CS)$ [by Proposition 5.2]

$\Rightarrow$ $\alpha|_{\Sigma_{CS}} \in L(CS)$ [by Proposition 5.5] □

Proof (asynchronous case): $\alpha \in L(\{E0_P, E1_P \times E0_Q, E1_Q\})$ and $\alpha_I|_{\Sigma_X} \in L(X)$

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and $\alpha_I|_{\Sigma_X} \in L(X)$

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and the root sequence of $A_O(\alpha_I)$ is β, where β is the longest sequence in $L(E0_P \times E1_Q)$ that satisfies $\alpha_I = (\beta|_\Delta)^{-1}$, where $\Delta = (\Sigma_{E0_P} \cup \Sigma_{E1_Q}) \setminus \Sigma_{CS}$, and $\alpha_I|_{\Sigma_X} \in L(X)$.

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and $\alpha_I|_{\Sigma_X} = (\beta|_\Delta)^{-1}|_{\Sigma_X}$ and $\alpha_I|_{\Sigma_X} \in L(X)$.

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and $\alpha_I|_{\Sigma_X} = ((\beta|_\Delta)|_{SA})^{-1}$ and $\alpha_I|_{\Sigma_X} \in L(X)$.

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and $\alpha_I|_{\Sigma_X} = (\beta|_{SA})^{-1}$ and $\alpha_I|_{\Sigma_X} \in L(X)$.

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and $(\beta|_{SA})^{-1} \in L(X)$.

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and

$\beta|_{SA} \in$ L(Proj$_{SA}$($E0_P \times E1_Q \times CS$)~red)) \ L(Proj$_{SA}$(¬($E0_P \times E1_Q \times CS$)~red)))

$\Rightarrow$ $\alpha_o \in A_O(\alpha_I)$ and $\beta \in$ L(($E0_P \times E1_Q \times CS$)~red) [by Proposition 5.2]

$\Rightarrow$ $\alpha_o \in$ L(($E0_P \times E1_Q \times CS$)~red) [by Corollary 5.2]

$\Rightarrow$ $\alpha_o|_{\Sigma_{CS}} \in$ L(CS) [by Proposition 5.2]

$\Rightarrow$ $\alpha|_{\Sigma_{CS}} \in$ L(CS) [by Proposition 5.5] □

Theorem 5.4: Given CS, P and Q, if $\{E0_P, C, E1_Q\}$ satisfies external equivalence and semantic equivalence with respect to a conversion seed X generated by Approach_1 then $\{E0_P, C, E1_Q\}$ satisfies the safety property with respect to CS.

Proof: $\{E0_P, C, E1_Q\}$ satisfies external equivalence and semantic equivalence

$\Rightarrow$ $\forall\alpha \in$ L($\{E0_P, C, E1_Q\}$), $\alpha_C \in$ L($E1_P \times E0_Q$) and $\alpha_C|_{\Sigma_X} \in$ L(X) [by definitions of external equivalence and semantic equivalence]

$\Rightarrow$ $\forall\alpha \in$ L($\{E0_P, C, E1_Q\}$), $\alpha|_{\Sigma_{CS}} \in$ L(CS) [by Proposition 5.6]

$\Rightarrow$ Proj$_{\Sigma_{CS}}$(L($\{E0_P, C, E1_Q\}$)) $\subseteq$ L(CS)

$\Rightarrow$ $\{E0_P, C, E1_Q\}$ satisfies the safety property with respect to CS. □

5.4.3 Algorithm to find the significant transitions and the significant action set

procedure *Find_Significant_Transitions_and_Significant_Actions*(E : CFSM;
Σ_{CS} : ACTIONS; var *TST* : TABLE; var *SA* : ACTIONS);

input: $E = (S, \Sigma, \delta, s0)$, Σ_{CS}

output: *TST*, a table containing one entry for each service primitive transition t in E; each entry in *TST* consists of (t, sig_pre(t), sig_suc(t), o_suc(t)); *SA*, the set of significant actions; or "failed" if this approach cannot be used.

function *Find_OC*(E : CFSM; s : STATE; Δ : ACTIONS) : TRANSITIONS;

input: $E = (S, \Sigma, \delta, s0)$, $s \in S$, $\Delta \subseteq \Sigma$

output: *Find_OC*, a set of transitions.

```
begin
  for ∀p ∈ S do
    begin
```

```
        mark[p] := false;
      end; {for}
   Tail := { s' ∈ S | δ(s', σ) = s, σ ∈ Σ_CS }
   Find_OC := ∅;
   Cls_Δ := {s};
   mark[s] := true;
   for q ∈ Cls_Δ do
      begin
         for ∀q' ∈ S, ∀σ' ∈ Σ, such that δ(q, σ') = q' do
            begin
               if q' ∈ Tail
                  then
                     begin
                        Find_OC := ∅;
                        exit;
                     end; {then}
               {endif}
               if σ' ∈ Δ and not mark[q']
                  then
                     begin
                        Cls_Δ := Cls_Δ ∪ {q'};
                        mark[q'] := true;
                     end {then}
                  else
                     begin
                        Find_OC := Find_OC ∪ {(q, σ', q')};
                     end; {else}
               {endif}
            end; {for}
      end;{for}
end; { function Find_OC }

begin { main }
   INITIALIZE(TST);
   if sync
      then
```

```
        begin
            Δ_P := Σ^U;
            Δ_S := Σ^U;
        end {then}
    else { async }
        begin
            Δ_P := Σ^U ∪ Σ^-;
            Δ_S := Σ^U ∪ Σ^+;
        end {else};
{endif}
SA := ∅;
for ∀s ∈ S do
    begin
        if ∃δ(s, σ) ∈ S, σ ∈ Σ_CS
            then
                begin
                    Rev_Ctrl_Pre := Find_OC(E^R, s, Δ_P);
                    if Rev_Ctrl_Pre = ∅
                        then
                            begin
                                report 'failed';
                                exit;
                            end; {then}
                    {endif}
                    Ctrl_Pre := { (p', σ, p) | (p, σ, p') ∈ Rev_Ctrl_Pre };
                    Sig_Pre := { (r, ρ, r') | (r, ρ, r') ∈ Ctrl_Pre and
                                        ∄(q, ρ', r) ∈ Ctrl_Pre, q, r, r' ∈ S, ρ, ρ' ∈ Σ };
                    SA := SA ∪ { σ | (s, σ, s') ∈ Sig_Pre };
                    for ∀δ(s, σ) = s' ∈ S, σ ∈ Σ_CS do
                        begin
                            INSERT(TST, (s, σ, s'), Sig_Pre, _, _);
                        end; {for}
                end; {then}
        {endif}
        if ∃δ(s', σ) = s, s' ∈ S, σ ∈ Σ_CS
            then
```

```
            begin
               Obs_Suc := Find_OC (E, s, ΔS);
               if Obs_Suc = ∅
                  then
                     begin
                        report 'failed';
                        exit;
                     end; {then}
               {endif}
               Sig_Suc := { (r, ρ, r') | (r, ρ, r') ∈ Obs_Suc and
                                 ∄(r', ρ', q) ∈ Obs_Suc, q, r, r' ∈ S, ρ, ρ' ∈ Σ };
               SA := SA ∪ { σ | (s, σ, s') ∈ Sig_Suc };
               for ∀δ(s', σ) = s, s' ∈ S, σ ∈ ΣCS do
                  begin
                     INSERT(TST, (s', σ, s), _, Sig_Suc, Obs_Suc);
                  end; {for}
            end; {then}
         {endif}
      end; {for}
end. { procedure Find_Significant_Transitions_and_Significant_Actions }
```

The complexity of function *Find_OC* is $O(|E|)$, where $|E| = |S| + |T|$. The main procedure *Find_Significant_Transitions_and_Significant_Actions* calls *Find_OC* at most twice for a state in E; therefore the upper bound on this procedure is $O(|S|*|E|)$, which is polynomial in the size of the input CFSM E.

5.4.4 Algorithm to maximize the significant successor transitions

procedure *Maximize_Sig_Suc*(var E : CFSM; var *TST*: TABLE; var *SA* : ACTIONS);

input: $E = (S, \Sigma, \delta, s0)$, *TST* , *SA*

output: $E = (S, \Sigma, \delta, s0)$, *TST*, a table containing one entry for each service primitive transition t in E; each entry in *TST* consists of (t, sig_pre(t), sig_suc(t), o_suc(t)); *SA*, the set of significant actions. E, *TST* and *SA* are modified such that for all entries in the table *TST* sig_suc(t) = o_suc(t).

```
begin
   for ∀(t, sig_pre(t), sig_suc(t), o_suc(t)) ∈ TST do
      begin
```

```
for ∀(r, σ, s) ∈ o_suc(t) \ sig_suc(t) do
   begin
      if sa ∉ S
         then
            begin
               S := S ∪ sa;
               for ∀ρ ∈ Σ(s) do
                  begin
                     if δ(s, ρ) ≠ s
                        then
                           begin
                              δ(sa, ρ) := δ(s, ρ);
                           end {then}
                        else
                           begin
                              δ(sa, ρ) := sa;
                           end {else}
                     {endif}
                     if ρ ∈ Σ_CS
                        then
                           begin
                              INSERT(TST, (sa, ρ, δ(sa, ρ)), ∅,
                                    sig_suc((s, ρ, δ(s, ρ)), o_suc((s, ρ, δ(s, ρ)));
                           end {then}
                     {endif}
                     for ∀(t', sig_pre(t'), sig_suc(t'), o_suc(t')) ∈ TST, do
                        begin
                           if (s, ρ, δ(s, ρ)) ∈ sig_pre(t')
                              then
                                 begin
                                    sig_pre(t') :=
                                          sig_pre(t') ∪ {(sa, ρ, δ(sa, ρ))};
                                 end {then}
                           {endif}
                        end {for}
                  end {for}
```

```
                end {then}
            {endif}
            δ(r, σ) := sa;
            for ∀(t', sig_pre(t'), sig_suc(t'), o_suc(t')) ∈ TST, do
              begin
                if (r, σ, s) ∈ sig_suc(t')
                  then
                    begin
                      sig_suc(t') := sig_suc(t') \ {(r, σ, s)}∪ {(r, σ, sa)};
                    end {then}
                {endif}
                if (r, σ, s) ∈ sig_pre(t')
                  then
                    begin
                      sig_pre(t') := sig_pre(t') \ {(r, σ, s)};
                      if the head of t' is not s
                        then
                          begin
                            sig_pre(t') := sig_pre(t') ∪ {(r, σ, sa)};
                          end {then}
                      {endif}
                    end {then}
                {endif}
                if the head of t' is s, i.e., t = (s, ρ, s'), s' ∈ S, σ ∈ Σ
                  then
                    begin
                      sig_pre((sa, ρ, s')) := sig_pre((sa, ρ, s')) ∪ {(r, σ, sa)};
                    end; {then}
                {endif}
              end {for}
            SA := SA ∪ {σ};
          end {for}
        sig_suc(t) := o_suc(t);
      end {for}
end. {procedure Maximize_Sig_Suc}
```

The complexity of procedure *Maximize_Sig_Suc* is $O(|TST|*|E|)$, where $|E| = |S| + |T|$ and $|TST|$ is the number of entries in *TST*, which is the number of service primitive transitions in *E*. Therefore the upper bound on this procedure is polynomial in the size of the input CFSM *E*.

5.4.5 Algorithm to construct $(E0_P \times E1_Q \times CS)\sim$red

procedure *Construct*1(var *Y* : CFSM; TST_{E0_P} : TABLE; TST_{E1_Q} : TABLE);

input: $Y = (E0_P \times E1_Q) \times CS = (S, \Sigma, \delta, s0)$, TST_{E0_P}, TST_{E1_Q}

output: $Y = (E0_P \times E1_Q \times CS)\sim\text{red} = (S, \Sigma, \delta, s0)$.

procedure *Rmv_Sig_Pre_Frm_Paths_To_State*((*r*1, *r*2, *r*3), σ, (*s*1, *s*2, *s*3));

comment: The service primitive transition passed as input corresponds with a service primitive transition in $E0_P$ or $E1_Q$. This procedure removes transitions from the CFSM *Y* that correspond with the significant predecessor transitions of the service primitive transition in $E0_P$ or $E1_Q$. The removal is done by explicitly redefining the head state to be the non-accepting state *na*.

```
begin
  for ∀(p1, p2, p3) ∈ S do
    begin
      mark[(p1, p2, p3)] := false;
    end; {for}
  Pre := {(r1, r2, r3)};
  mark(((r1, r2, r3)) := true;
  for (q1, q2, q3) ∈ Pre do
    begin
      for ∀(p1, p2, p3) ∈ S, ∀ρ ∈ Σ, such that δ((p1, p2, p3), ρ) = (q1, q2, q3)
        and not mark((p1, p2, p3)) do
          begin
            if (σ ∈ Σ_E0p and (p1, ρ, q1) ∈ sig_pre((r1, σ, s1)))
              or (σ ∈ Σ_E1Q and (p2, ρ, q2) ∈ sig_pre((r2, σ, s2)))
                then
                  begin
                    δ((q1, q2, q3), ρ) = na
```

```
                    end {then}
                else
                    begin
                        Pre = Pre ∪ {(p1, p2, p3)};
                        mark(((p1, p2, p3)) := true;
                    end {else}
            {endif}
        end {for}
    end {for}
end;
```

procedure *Rmv_Sig_Pre_from_Paths_To_Sig_Suc*($(r1, r2, r3)$, σ, $(s1, s2, s3)$):

comment: The service primitive transition passed as input corresponds with a service primitive transition in $E0_P$ or $E1_Q$. This procedure removes transitions from the CFSM Y that correspond with the significant predecessor transitions of other service primitive transitions in $E0_P$ or $E1_Q$ if they occur in a path from $(s1, s2, s3)$ before the transition corresponding to the significant successor. The removal is done by explicitly redefining the head state to be the non-accepting state *na*.

```
begin
  for ∀(p1, p2, p3) ∈ S do
    begin
      mark[(p1, p2, p3)] := false;
    end; {for}
  Suc := {(s1, s2, s3)};
  mark((s1, s2, s3)) := true;
  for (q1, q2, q3) ∈ Suc do
    begin
      for ∀(p1, p2, p3) ∈ S, ∀ρ ∈ Σ, such that δ((q1, q2, q3), ρ) = (p1, p2, p3)
        and not mark((p1, p2, p3)) do
          begin
            if (σ ∈ Σ_E0_p and (q1, ρ, p1) ∉ Sig_Suc((r1, σ, s1)))
              or (σ ∈ Σ_E1_Q and (q2, ρ, p2) ∉ Sig_Suc((r2, σ, s2))
                then
                  begin
```

```
                        if (σ ∈ Σ_{E0_p} and (q2, ρ, p2) ∈ sig_pre(t),
                            for some t = (r, ρ', r') in E1_Q, ρ' ∈ Σ_CS) or
                            (σ ∈ Σ_{E1_Q} and (q1, ρ, p1) ∈ sig_pre(t),
                            for some t = (r, ρ', r') in E0_P, ρ' ∈ Σ_CS)
                                then
                                    begin
                                        δ((q1, q2, q3), ρ) = na
                                    end {then}
                                else
                                    begin
                                        Suc = Suc ∪ {(p1, p2, p3)};
                                        mark(((p1, p2, p3)) := true;
                                    end; {else}
                            {endif}
                        end {then}
                    {endif}
                end {for}
        end {for}
end.

begin {main }
    for ∀(s1, s2, s3) ∈ S, such that (s1, s2, s3) is a dead state, do
        begin
            for ∀δ((r1, r2, r3), σ) = (s1, s2, s3), (r1, r2, r3) ∈ S, σ ∈ Σ do
                begin
                    if σ ∈ Σ_CS
                        then
                            begin
                                Rmv_Sig_Pre_Frm_Paths_To_State((r1, r2, r3), σ, (s1, s2, s3));
                            end; {then}
                    {endif}
                    δ((r1, r2, r3), σ) = undefined
                end {for}
            S := S \ {(s1, s2, s3)}
        end {for}
    for δ((r1, r2, r3), σ) = (s1, s2, s3) ∈ S, (r1, r2, r3) ∈ S, σ ∈ Σ_CS do
```

```
    begin
      Rmv_Sig_Pre_Frm_Paths_To_Sig_Suc((r1, r2, r3), σ, (s1, s2, s3));
    end; {for}
end. {procedure Construct1}
```

The complexity of both procedures to remove the significant transitions is $O(|E|)$. The main procedure *Construct*1 calls procedure *Rmv_Sig_Pre_Frm_Paths_To_State* once for every dead state in E and calls procedure *Rmv_Sig_Pre_from_Paths_To_Sig_Suc* once for every service primitive transition in E; therefore the complexity is $O(|E|^2)$, which is polynomial in the size of the input CFSM E.

5.4.6 Algorithm to construct the conversion seed from $(E0_P \times E1_Q \times CS)$~red

```
procedure Construct2(E : CFSM; SA : ACTIONS; var X : CFSM);
input: E = (E0_P × E1_Q × CS)~red = (S, Σ, δ, s0), SA
output: X = (S_X, Σ_X, δ_X, p0_X) if exists, and "no solution" otherwise.
begin
  Δ := Σ \ SA;
  Σ_X := (SA)^-1;
  p0_X := Cls_Δ(s0);
  if na ∈ p0_X
    then
      begin
        report "no solution"
      end {then}
    else
      begin
        new := ∅;
        old := {p0_X};
        S_X := {p0_X};
        while old ≠ ∅ do
          begin
            for p ∈ old do
              begin
                for σ ∈ SA do
                  begin
                    p' := ∪ Cls_Δ(δ(s, σ));
                          s∈p
```

```
                        if p' ≠ ∅ and na ∉ p'
                            then
                                begin
                                    δX(p, σ^-1) := p';
                                    if p' ∉ SX
                                        then
                                            begin
                                                SX := SX ∪ {p'};
                                                new := new ∪ {p'};
                                            end; {then}
                                    {endif}
                                end {then}
                            else
                                begin
                                    δX(p, σ^-1) is undefined;
                                end {else}
                        {endif}
                    end; {for}
                end; {for}
            old := new;
            new := ∅;
        end; {while}
    end; {else}
{endif}
end. {procedure Construct2}
```

Potentially, all subsets of the state set of the input CFSM could be generated, thus making the algorithm exponential in time and space complexity. More specifically, in the worst case, the time and space complexity is $O(2^N)$, where $N = |(E0_P \times E1_Q \times CS)\text{~red}|$.

5.4.7 The algorithmic solution

To summarize our approach we give below the order of applying the algorithms presented earlier.

Find_Significant_Transitions_and_Significant_Actions($E0_P$, Σ_{CS}, TST_{E0_P}, SA_{E0_P})

{ returns TST_{E0_P} and SA_{E0_P} or 'failed'; cost: $O(|S_{E0_P}|*|E0_P|)$ }

Maximize_Sig_Suc($E0_P$, TST_{E0_P}, SA_{E0_P})

{ returns $E0_P$, TST_{E0_P} and SA_{E0_P}; cost: $O(|TST_{E0_P}|*|E0_P|)$ }

Find_Significant_Transitions_and_Significant_Actions($E1_Q$, Σ_{CS}, TST_{E1_Q},SA_{E1_Q})

{ returns TST_{E1_Q} and SA_{E1_Q} or 'failed'; cost: $O(|S_{E1_Q}|*|E1_Q|)$ }

Maximize_Sig_Suc($E1_Q$, TST_{E1_Q}, SA_{E1_Q})

{ returns $E1_Q$, TST_{E1_Q} and SA_{E1_Q}; cost: $O(|TST_{E1_Q}|*|E1_Q|)$ }

$Y := (E0_P \times E1_Q) \times CS$

*Construct*1(Y, TST_{E0_P}, TST_{E1_Q})

{ returns $Y = (E0_P \times E1_Q \times CS)$~red; cost: $O(|(E0_P \times E1_Q) \times CS|^2)$ }

Construct2(Y, $SA_{E0_P} \cup SA_{E1_Q}$, X)

{ returns X or no solution; cost: $O(2^N)$, where $N = |\ (E0_P \times E1_Q \times CS)$~red $|$ }

5.5 Finding a Conversion Seed with Reduced Input CFSMs

Once the significant action set is known, it can be used to reduce the size of the input CFSMs, $E0_P$ and $E1_Q$, thus reducing the potential state space explosion that can occur when finding the product of CFSMs. The idea behind this reduction is based on the observation that certain state pairs can be merged while retaining the language of the CFSM projected over the significant action set and the service primitive actions. At the same time, we do not want to merge state pairs that have significant transitions between them to maintain the significant transitions.

5.5.1 Reducing the size of CFSMs

Given a CFSM $E_1 = (S_1, \Sigma, \delta_1, s0_1)$, $\Delta \subseteq \Sigma$, and a set of significant transitions ST, a reduced CFSM, $E_2 = (S_2, \Sigma, \delta_2, s0_2)$ can be constructed such that $\text{Proj}_\Delta(\text{L}(E_1)) = \text{Proj}_\Delta(\text{L}(E_2))$, $|S_2| \leq |S_1|$ and $|T_2| \leq |T_1|$, where $|S_i|$ is the number of states in the state set of the CFSM E_i, and $|T_i|$ is the number of transitions defined in the CFSM E_i, $1 \leq i \leq 2$, by the following steps:

1. Let $E_2 = E_1$.

2. Find a pair of adjacent states, $s_1, s_2 \in S_2$, such that there exists a transition from s_1 to s_2 labeled by an action $\sigma \notin \Delta$, i.e., $\delta_2(s_1, \sigma) = s_2$, and there does not exist a transition from s_1 to s_2, t, such that $t \in ST$ and there does not exist a transition from s_2 to s_1, t', such that $t' \in ST$.

3. If one of the following merge conditions is true

M1 $\nexists\, s \in S_2,\ s \neq s_1,\ \sigma' \in \Sigma$ such that $\delta_2(s, \sigma') = s_2$ and $\nexists\, \sigma' \in \Delta$ such that $\delta_2(s_1, \sigma') = s_2$ and $\forall \sigma' \in \Sigma_2(s_1) \cap \Sigma_2(s_2)$, $\delta_2(s_1, \sigma') = \delta_2(s_2, \sigma')$ and s_2 is not the initial state of E_2.

M2 $\nexists\, s \in S_2,\ s \neq s_2,\ \sigma' \in \Sigma$ such that $\delta_2(s_1, \sigma') = s$ and $\nexists\, \sigma' \in \Delta$ such that $\delta_2(s_1, \sigma') = s_2$.

M3 $\exists\ \sigma' \notin \Delta$ such that $\delta_2(s_2, \sigma') = s_1$ and $\forall \sigma' \in \Sigma_2(s_1) \cap \Sigma_2(s_2)$, $\delta_2(s_1, \sigma') = \delta_2(s_2, \sigma')$.

then the states can be merged as follows

a) $S_2 = S_2 \cup \{\{s_1, s_2\}\}$.

b) $\forall \sigma \in \Sigma_2(s_1)$, define $\delta_2(\{s_1, s_2\}, \sigma) = \delta_2(s_1, \sigma)$ and $\delta_2(s_1, \sigma) =$ undefined.

c) $\forall \sigma \in \Sigma_2(s_2)$, define $\delta_2(\{s_1, s_2\}, \sigma) = \delta_2(s_2, \sigma)$ and $\delta_2(s_2, \sigma) =$ undefined.

d) $\forall s \in S_2$, $\forall \sigma \in \Sigma$ such that $\delta_2(s, \sigma) = s_1$ or $\delta_2(s, \sigma) = s_2$, define $\delta_2(s, \sigma) = \{s_1, s_2\}$.

e) $S_2 = S_2 \setminus \{s_1, s_2\}$.

f) $\forall \sigma \in \Sigma \setminus \Delta$ such that $\delta_2(\{s_1, s_2\}, \sigma) = \{s_1, s_2\}$, $\delta_2(\{s_1, s_2\}, \sigma) =$ undefined.

4. Repeat steps 2 and 3 until no more states can be merged.

The cost of finding and merging all $|S|$ states of a CFSM is between $O(|S| + |T|)$ and $O(|S| + \log(|S|)*|T|)$, where $|S|$ is the number of states in the CFSM and $|T|$ is the number of transitions in the CFSM, depending on the strategy chosen to find the next state pair to merge.

Theorem 5.5: Given a CFSM $E_1 = (S_1, \Sigma, \delta_1, s0_1)$, $\Delta \subseteq \Sigma$ and a set of significant transitions ST, the above steps constructs a CFSM $E_2 = (S_2, \Sigma, \delta_2, s0_2)$ such that $\text{Proj}_\Delta(\text{L}(E_1)) = \text{Proj}_\Delta(\text{L}(E_2))$, $|S_2| \leq |S_1|$ and $|T_2| \leq |T_1|$, where $|S_i|$ is the number of states in the state set of the CFSM E_i, and $|T_i|$ is the number of transitions defined in the CFSM E_i, $1 \leq i \leq 2$.

Proof:

(1) If no two states of E_1 can be merged then clearly $E_1 = E_2$ which implies that $\text{Proj}_\Delta(\text{L}(E_1)) = \text{Proj}_\Delta(\text{L}(E_2))$, $|S_2| = |S_1|$ and $|T_2| = |T_1|$.

(2) Let $E_2 = (S_2, \Sigma, \delta_2, s0_2)$ be the reduced CFSM of $E_1 = (S_1, \Sigma, \delta_1, s0_1)$ after merging two states $s_1, s_2 \in S_1$ into state $\{s_1, s_2\}$, by steps a) to f) above, where $\Delta \subseteq \Sigma$. By the merge procedure we have:

$S_2 = (S_1 \cup \{\{s_1, s_2\}\}) \setminus \{s_1, s_2\}$.

$\delta_2: S_2 \times \Sigma \rightarrow S_2$,

$\delta_2(s, \sigma) = \delta_1(s, \sigma)$, if $s \neq \{s_1, s_2\}$ and $\delta_1(s, \sigma) \notin \{s_1, s_2\}$.

$= \delta_1(s_1, \sigma)$, if $s = \{s_1, s_2\}$, $\delta_1(s_1, \sigma) \notin \{s_1, s_2\}$ and $\sigma \in \Sigma_1(s_1)$.

$= \delta_1(s_2, \sigma)$, if $s = \{s_1, s_2\}$, $\delta_1(s_2, \sigma) \notin \{s_1, s_2\}$ and $\sigma \in \Sigma_1(s_2)$.

$= \{s_1, s_2\}$, if $s \neq \{s_1, s_2\}$, $\delta_1(s, \sigma) \in \{s_1, s_2\}$.

$= \{s_1, s_2\}$, if $s = \{s_1, s_2\}$ and $\sigma \in \Delta$ and

$(\delta_1(s_1, \sigma) \in \{s_1, s_2\}$ or $\delta_2(s_1, \sigma) \in \{s_1, s_2\})$.

undefined otherwise.

$\Rightarrow |S_2| < |S_1|$, and

a transition defined in E_1 whose head or tail is not s_1 or s_2 is defined in E_2, and

a transition defined in E_1 whose tail is s_1 and whose head is not s_1 or s_2 is defined in E_2 with tail $\{s_1, s_2\}$, and

a transition defined in E_1 whose tail is s_2 and whose head is not s_1 or s_2 is defined in E_2 with tail $\{s_1, s_2\}$, and

a transition defined in E_1 whose head is s_1 or s_2 and whose tail is not s_1 or s_2 is defined in E_2 with head $\{s_1, s_2\}$, and

a transition labeled by an action in Δ defined in E_1 whose head is s_1 or s_2 and whose tail is s_1 or s_2 is defined in E_2 with head $\{s_1, s_2\}$ and tail $\{s_1, s_2\}$, and

a transition labeled by an action not in Δ defined in E_1 whose head is s_1 or s_2 and whose tail is s_1 or s_2 is undefined in E_2, (there must be at least one such transition between s_1 and s_2 according to step 2 of the reduction procedure) and

an undefined transition in E_1 is undefined in E_2.

$\Rightarrow |S_2| < |S_1|$, and $|T_2| < |T_1|$.

We have shown that $|S_2| < |S_1|$ and $|T_2| < |T_1|$, now we must show that $\alpha \in \text{Proj}_\Delta(\text{L}(E_1)) \Leftrightarrow \alpha \in \text{Proj}_\Delta(\text{L}(E_2))$.

($\Rightarrow$ part) $\alpha \in \text{Proj}_\Delta(L(E_1))$

$\Rightarrow$ $\exists\beta \in \text{L}(E_1)$ such that $\alpha = \beta|_\Delta$.

$\Rightarrow$ There is a sequence of state transitions starting from the initial state of E_1 such that the corresponding sequence of actions is β and $\alpha = \beta|_\Delta$.

case 1: the states s_1 and s_2 do not appear in the sequence of state transitions:

$\Rightarrow$ There is a sequence of state transitions starting from the initial state of E_2 such that the corresponding sequence of actions is β', where $\beta' = \beta$, and $\alpha = \beta|_\Delta$. [By construction: the sequence of state transitions in E_2 is the same as the sequence of state transitions in E_1.]

case 2: the states s_1 or s_2 appear in the sequence of state transitions:

$\Rightarrow$ There is a sequence of state transitions from the initial state of E_2 such that the corresponding sequence of actions is β', where $\beta' = \beta$ minus zero or more actions *not* in Δ, and $\alpha = \beta|_\Delta$. [By construction: the sequence of state transitions in E_2 differs from the sequence of state transitions in E_1 in only the following ways:

- each occurrence of s_1 or s_2 is replaced by $\{s_1, s_2\}$.
- each state transition $(\{s_1, s_2\}, \sigma, \{s_1, s_2\})$ where $\sigma \notin \Delta$, is replaced by $\{s_1, s_2\}$.]

$\Rightarrow$ $\exists\beta' \in L(E_2)$ such that $\alpha = \beta'|_\Delta$.

$\Rightarrow$ $\alpha \in \text{Proj}_\Delta(L(E_2))$.

($\Leftarrow$ part) $\alpha \notin \text{Proj}_\Delta(L(E_1))$, $\alpha \in \Delta^*$

$\Rightarrow$ for all $\beta \in \Sigma^*$ such that $\alpha = \beta|_\Delta$, $\beta \notin L(E_1)$.

Assume for all $\beta \in \Sigma^*$ such that $\alpha = \beta|_\Delta$, $\beta \notin L(E_1)$ and there is a $\chi \in \Sigma^*$ such that $\alpha = \chi|_\Delta$, $\chi \in L(E_2)$, and furthermore, $\chi = \chi 1.\sigma$, where $\chi 1 \in L(E_1)$.

case 1: The sequence of state transitions from the initial state of E_1 with the corresponding action sequence $\chi 1$, ends at state $s \in S_1$, $s \notin \{s_1, s_2\}$.

$\Rightarrow$ The sequence of state transitions from the initial state of E_2 with the corresponding action sequence $\chi 1$, ends at state $s \in S_2$, $\sigma \notin \Sigma_1(s)$, and $\sigma \in \Sigma_2(s)$, which leads to a contradiction.

This is impossible since the merge procedure does not add, remove or change transitions defined at states that are not s_1 or s_2.

case 2: The sequence of state transitions from the initial state of E_1 with the corresponding action sequence $\chi 1$, ends at state $s_1 \in S_1$.

$\Rightarrow$ The sequence of state transitions from the initial state of E_2 with the corresponding action sequence $\chi 1$, ends at state $\{s_1, s_2\} \in S_2$, $\sigma \notin \Sigma_1(s_1)$, and $\sigma \in \Sigma_2(\{s_1, s_2\})$.

$\Rightarrow \sigma \notin \Sigma_1(s_1)$, and $\sigma \in \Sigma_1(s_2)$.

Recall that to merge s_1 and s_2 there exists a transition from s_1 to s_2 labeled by an action $\rho \notin \Delta$, i.e., $\delta_2(s_1, \rho) = s_2$.

$\Rightarrow$ There is a sequence $\chi 1.\rho.\sigma \in L(E_1)$ such that $(\chi 1.\rho.\sigma)|_\Delta = \alpha$, which leads to a contradiction.

This is a contradiction of the assumption that for all $\beta \in \Sigma^*$ such that $\alpha = \beta|_\Delta$, $\beta \notin L(E_1)$

case 3: The sequence of state transitions from the initial state of E_1 with the corresponding action sequence $\chi 1$, ends at state $s_2 \in S_1$.

$\Rightarrow$ The sequence of state transitions from the initial state of E_2 with the corresponding action sequence $\chi 1$, ends at state $\{s_1, s_2\} \in S_2$, $\sigma \notin \Sigma_1(s_2)$, and $\sigma \in \Sigma_2(\{s_1, s_2\})$.

$\Rightarrow \sigma \notin \Sigma_1(s_2)$, and $\sigma \in \Sigma_1(s_1)$.

Recall that to merge s_1 and s_2 that satisfy condition M1 given earlier there exists one or more transitions from s_1 to s_2 labeled by an action $\rho \notin \Delta$, i.e., $\delta_1(s_1, \rho) = s_2$; and these are the only incoming transitions to s_2, and s_2 is not the initial state.

Recall that to merge s_1 and s_2 that satisfy condition M2 given earlier there exists one or more transitions from s_1 to s_2 labeled by an action $\rho \notin \Delta$, i.e., $\delta_1(s_1, \rho) = s_2$; and these are the only outgoing transitions from s_1.

Recall that to merge s_1 and s_2 that satisfy condition M3 given earlier there exists one or more transitions from s_1 to s_2 labeled by an action $\rho \notin \Delta$, i.e., $\delta_2(s_1, \rho) = s_2$; and there exists one or more transitions from s_2 to s_1 labeled by an action $\rho \notin \Delta$, i.e., $\delta_1(s_2, \rho) = s_1$.

$\Rightarrow$ If s_1 and s_2 satisfy condition M1 given earlier, then there exists $\chi 1'.\sigma \in L(E_1)$, where $\chi 1 = \chi 1'.\rho$, such that $(\chi 1'.\sigma)|_\Delta = \alpha$, which leads to a contradiction. If s_1 and s_2 satisfy condition M2 then $\sigma \notin \Delta$ which implies that $\chi 1 \in L(E_1)$, and $(\chi 1)|_\Delta = \alpha$, which leads to a contradiction. If s_1 and s_2 satisfy condition M3 given earlier, then there exists $\chi 1.\rho.\sigma \in L(E_1)$, such that $(\chi 1.\rho.\sigma)|_\Delta = \alpha$, which leads to a contradiction.

This is a contradiction of the assumption that for all $\beta \in \Sigma^*$ such that $\alpha = \beta|_\Delta$, $\beta \notin L(E_1)$

$\Rightarrow$ for all $\beta \in \Sigma^*$ such that $\alpha = \beta|_\Delta$, $\beta \notin L(E_2)$.

$\Rightarrow$ $\alpha \notin \text{Proj}_\Delta(L(E_2))$.

From (1) and (2), we have shown that $\text{Proj}_\Delta(L(E_1)) = \text{Proj}_\Delta(L(E_2))$, $|S_2| \leq |S_1|$ and $|T_2| \leq |T_1|$. □

5.5.2 Finding a conversion seed: Approach_2

Approach_2 is based on the same assumptions as Approach_1 [Section 5.4], solves the same problem as Approach_1 [Section 5.4] for the same sets of protocols as Approach_1 [Section 5.4]. Approach_2 extends Approach_1 by making use of the reduction technique introduced in Section 5.5.1. Once the significant transitions and the significant action set are known, they can be used to reduce the size of the input CFSMs, $E0_P$ and $E1_Q$, thus reducing the potential state space explosion that can occur when finding the product of CFSMs.

1. Find the significant transitions and the significant action set and maximize the significant successor transitions. (Section 5.4.1).
2. Let $\Delta = SA \cup \Sigma_{CS}$.
3. Let $ST = \{ t \mid t \in (\text{sig\_suc}(t') \cup \text{sig\_pre}(t')), \forall t' = (r, \rho, r')$ in $E0_P$ and in $E1_Q$, $\rho \in \Sigma_{CS}\}$.
4. Find $R\text{-}E0_P$ = reduction of $E0_P$ based on Δ and ST using the approach in Section 5.5.1.

5. Find $R\text{-}E1_Q$ = reduction of $E1_Q$ based on Δ and ST using the approach in Section 5.5.1.
6. Update the significant transition information to account for the merged states. No service primitives or significant transitions were removed, however, the head states or tail states may have been changed.
7. Find the conversion seed (Section 5.4.2) replacing $E0_P$ with $R\text{-}E0_P$ and replacing $E1_Q$ with $R\text{-}E1_Q$.

Let β be a possible interleaving of a sequence of actions accepted by $E0_P$ with a sequence of actions accepted by $E1_Q$ and let α be the projection of β over the subset of actions including the significant action set and the service primitives of CS. The following proposition shows that α is accepted by the product CFSM ($E0_P \times E1_Q \times CS$) projected over the same subset of actions if and only if α is accepted by the CFSM which is the product of the product CFSM ($E0_P \times E1_Q$), projected over the same subset of actions, with CS.

Proposition 5.7: $\forall \beta \in L(E0_P \times E1_Q)$, $\alpha = \beta|_{SA \cup \Sigma_{CS}}$,

$$\alpha \in L(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q \times CS)) \Leftrightarrow \alpha \in L(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q) \times CS)$$

Proof: $\alpha \in L(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q \times CS))$

$\Leftrightarrow$ $\exists \alpha' \in L(E0_P \times E1_Q \times CS)$ such that $\alpha'|_{SA \cup \Sigma_{CS}} = \alpha$

$\Leftrightarrow$ $\exists \alpha' \in L(E0_P \times E1_Q)$ such that $\alpha'|_{\Sigma_{CS}} \in L(CS)$ and $\alpha'|_{SA \cup \Sigma_{CS}} = \alpha$

[observe that $\alpha'|_{\Sigma_{CS}} = (\alpha'|_{SA \cup \Sigma_{CS}})|_{\Sigma_{CS}} = \alpha|_{\Sigma_{CS}}$]

$\Leftrightarrow$ $\exists \alpha' \in L(E0_P \times E1_Q)$ such that $\alpha|_{\Sigma_{CS}} \in L(CS)$ and $\alpha'|_{SA \cup \Sigma_{CS}} = \alpha$

$\Leftrightarrow$ $\alpha \in L(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q))$ and $\alpha|_{\Sigma_{CS}} \in L(CS)$

$\Leftrightarrow$ $\alpha \in L(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q) \times CS)$ □

Let β be a possible interleaving of a sequence of actions accepted by $E0_P$ with a sequence of actions accepted by $E1_Q$ and let α be the projection of β over the subset of actions including the significant action set and the service primitives of CS. The following proposition shows that α is accepted by the product CFSM ($E0_P \times E1_Q$) projected over

the same subset of actions if and only if α is accepted by the CFSM which is the product of the CFSM $E0_P$, projected over the same subset of actions, with $E1_Q$, projected over the same subset of actions.

Proposition 5.8: $\forall\beta \in L(E0_P \times E1_Q)$, $\alpha = \beta|_{SA \cup \Sigma_{CS}}$,

$$\alpha \in L(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q)) \Leftrightarrow \alpha \in L(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q))$$

Proof: $\Sigma_{E0_P} \cap \Sigma_{E1_Q} = \varnothing$ □

Let β be a possible interleaving of a sequence of actions accepted by $E0_P$ with a sequence of actions accepted by $E1_Q$ and let α be the projection of β over the subset of actions including the significant action set.

In Approach_1 [Section 5.4] the intermediate CFSM found in step one of finding the conversion seed [Section 5.4.2] is $(E0_P \times E1_Q \times CS)$~red. If we replace $E0_P$ with the projection of $E0_P$ over the subset of actions including the significant action set and the service primitive actions, and we replace $E1_Q$ with the projection of $E1_Q$ over the same subset of actions, then the intermediate CFSM found in step one is $((\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)$~red). The following proposition shows that α is accepted by $((\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)$~red) projected over the significant action set and NOT accepted by the complement of $((\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)$~red) projected over the significant action set if and only if α is accepted by $(E0_P \times E1_Q \times CS)$~red projected over the significant action set and NOT accepted by the complement of $(E0_P \times E1_Q \times CS)$~red projected over the significant action set.

Proposition 5.9: $\forall\beta \in L(E0_P \times E1_Q)$, $\alpha = \beta|_{SA}$,

$$\alpha \in L(\text{Proj}_{SA}((\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\text{~red})))$$
$$\setminus L(\text{Proj}_{SA}(\neg(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\text{~red})))$$
$$\Leftrightarrow \alpha \in L(\text{Proj}_{SA}((E0_P \times E1_Q \times CS)\text{~red}))) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\text{~red})))$$

Proof: We prove by showing that

(1) $\alpha \in L(Proj_{SA}((Proj_{SA \cup \Sigma_{CS}}(E0_P) \times Proj_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\sim red)))$

$\Leftrightarrow \alpha \in L(Proj_{SA}((E0_P \times E1_Q \times CS)\sim red)))$, and

(2) $\alpha \in L(Proj_{SA}(\neg(Proj_{SA \cup \Sigma_{CS}}(E0_P) \times Proj_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\sim red)))$

$\Leftrightarrow \alpha \in L(Proj_{SA}(\neg(E0_P \times E1_Q \times CS)\sim red)))$

(1) $\alpha \in L(Proj_{SA}((Proj_{SA \cup \Sigma_{CS}}(E0_P) \times Proj_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\sim red)))$

$\Leftrightarrow$ $\alpha \in L(Proj_{SA}(((Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q)) \times CS)\sim red)))$ [by Proposition 5.8]

$\Leftrightarrow$ $\exists \alpha' \in L((Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q) \times CS)\sim red)$ such that $\alpha'|_{SA} = \alpha$

$\Leftrightarrow$ $\exists \alpha' \in L(Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q) \times CS)$ such that $\alpha'|_{SA} \doteq \alpha$ and α' does not violate any reduction rules.

$\Leftrightarrow$ $\exists \alpha' \in L(Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q \times CS))$ such that $\alpha'|_{SA} = \alpha$ and α' does not violate any reduction rules. [by Proposition 5.7]

$\Leftrightarrow$ $\exists \beta' \in L(E0_P \times E1_Q \times CS)$ such that $\beta'|_{SA \cup \Sigma_{CS}} = \alpha'$, $\alpha'|_{SA} = \alpha$ and α' does not violate any reduction rules.

$\Leftrightarrow$ $\exists \beta' \in L(E0_P \times E1_Q \times CS)$ such that $\beta'|_{SA} = \alpha$ and β' does not violate any reduction rules.

$\Leftrightarrow$ $\exists \beta' \in L((E0_P \times E1_Q \times CS)\sim red)$ such that $\beta'|_{SA} = \alpha$.

$\Leftrightarrow$ $\alpha \in L(Proj_{SA}((E0_P \times E1_Q \times CS)\sim red)))$

(2) $\alpha \in L(Proj_{SA}(\neg(Proj_{SA \cup \Sigma_{CS}}(E0_P) \times Proj_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\sim red)))$

$\Leftrightarrow$ $\alpha \in L(Proj_{SA}(\neg((Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q)) \times CS)\sim red)))$ [by Proposition 5.8]

$\Leftrightarrow$ $\exists \alpha' \in L(\neg(Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q) \times CS)\sim red)$ such that $\alpha'|_{SA} = \alpha$

$\Leftrightarrow$ $\exists \alpha' \in L(Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q) \times CS)$ such that $\alpha'|_{SA} = \alpha$ and α' violates a reduction rule.

$\Leftrightarrow$ $\exists \alpha' \in L(Proj_{SA \cup \Sigma_{CS}}(E0_P \times E1_Q \times CS))$ such that $\alpha'|_{SA} = \alpha$ and α' violates a reduction rule. [by Proposition 5.7]

$\Leftrightarrow$ $\exists \beta' \in L(E0_P \times E1_Q \times CS)$ such that $\beta'|_{SA \cup \Sigma_{CS}} = \alpha'$, $\alpha'|_{SA} = \alpha$ and α' violates a reduction rule.

$\Leftrightarrow$ $\exists \beta' \in L(E0_P \times E1_Q \times CS)$ such that $\beta'|_{SA} = \alpha$ and β' violates a reduction rule.

$\Leftrightarrow$ $\exists \beta' \in L(\neg(E0_P \times E1_Q \times CS)\sim\text{red})$ such that $\beta'|_{SA} = \alpha$.

$\Leftrightarrow$ $\alpha \in L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red})))$. □

Theorem 5.6: The conversion seed generated by Approach_2 is equivalent to the conversion seed generated by Approach_1.

Proof:

$L(\text{Proj}_{SA}((R\text{-}E0_P \times R\text{-}E1_Q \times CS)\sim\text{red})) \setminus L(\text{Proj}_{SA}(\neg((R\text{-}E0_P \times R\text{-}E1_Q \times CS)\sim\text{red}))$

= $L(\text{Proj}_{SA}((\text{Proj}_{SA \cup \Sigma_{CS}}(R\text{-}E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(R\text{-}E1_Q) \times CS)\sim\text{red})))$
$\setminus L(\text{Proj}_{SA}(\neg(\text{Proj}_{SA \cup \Sigma_{CS}}(R\text{-}E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(R\text{-}E1_Q) \times CS)\sim\text{red})))$

[by Proposition 5.9]

= $L(\text{Proj}_{SA}((\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\sim\text{red})))$
$\setminus L(\text{Proj}_{SA}(\neg(\text{Proj}_{SA \cup \Sigma_{CS}}(E0_P) \times \text{Proj}_{SA \cup \Sigma_{CS}}(E1_Q) \times CS)\sim\text{red})))$

[by Theorem 5.5]

= $L(\text{Proj}_{SA}((E0_P \times E1_Q \times CS)\sim\text{red}))) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red})))$

[by Proposition 5.9] □

Corollary 5.3: Given CS, P and Q, if $\{E0_P, C, E1_Q\}$ satisfies external equivalence and semantic equivalence with respect to a conversion seed X generated by the above steps then $\{E0_P, C, E1_Q\}$ satisfies the safety property with respect to CS.

Proof: by Theorem 5.4 and Theorem 5.6. □

5.6 Finding a Conversion Seed with Pruned Input CFSMs

Both approaches in Section 5.4 and Section 5.5 are based on the assumption that $\Sigma_{CS} = \Sigma^U_{E0_P} \cup \Sigma^U_{E1_Q}$. If this is not the case then protocol pruning is an efficient approach to reduce the size of the input CFSMs and to ensure that the above assumption is met.

5.6.1 Pruning the input protocols

Protocol pruning [LNS95] is a method that computes those parts of the protocol machine that are necessary to provide the required service of the protocol where the supplied service is a superset of the required service. An algorithm that prunes protocols in polynomial time and space is given in [LNS95].

Given a protocol $\{E_0, E_1\}$, with (a)sync(E_0, E_1) = true, and a set of unmatched service primitives $USP \subseteq \Sigma^U_0 \cup \Sigma^U_1$, prune($E_0$, E_1, USP) finds the pruned E_0 and the pruned E_1 by the following steps.

1. Remove all transitions with unmatched service primitives as labels from E_0 and E_1.
2. For each CFSM, compute the strongly connected component that starts at the initial state, discarding the rest of the machine.
3. Remove any resulting transitions with unmatched service primitive or peer protocol messages as labels from the CFSMs.
4. For any CFSM that changes recompute the strongly connected component that starts at the initial state, discarding the rest of the machine.
5. Continue iteratively until all transitions are matched and each machine is a strongly connected component. The resulting CFSMs are the pruned E_0 and the pruned E_1.

This method has complexity $O(d_{E_0} \times |T_{E_0}| + d_{E_1} \times |T_{E_1}|)$, where d_E is the number of transitions deleted from CFSM E and $|T_E|$ is the number of transitions in the CFSM E.

5.6.2 Finding a conversion seed: Approach_3

The following approach is based on the assumption that $\Sigma_{CS} \subseteq \Sigma^U_{E0_P} \cup \Sigma^U_{E1_Q}$.

If entity $E0_P$ contains transitions labeled by service primitive actions that are not in Σ_{CS} then we want to construct the pruned $E0_P$. Each service primitive in $E0_P$ typically has a corresponding service primitive in $E1_P$. The unmatched service primitives of protocol P are those service primitive actions labeling transitions in $E0_P$ that are not in Σ_{CS}, and their corresponding service primitives in $E1_P$.

If entity $E1_Q$ contains transitions labeled by service primitive actions that are not in Σ_{CS} then we want to construct the pruned $E1_Q$. Each service primitive in $E1_Q$ typically has a corresponding service primitive in $E0_Q$. The unmatched service primitives of protocol Q are those service primitive actions labeling transitions in $E1_Q$ that are not in Σ_{CS}, and their corresponding service primitives in $E0_Q$.

1. USP = all service primitive actions in $\Sigma_0^U \setminus \Sigma_{CS}$ and their corresponding service primitive actions in $\Sigma_{E1_P}^U$.
2. Let $P\text{-}E0_P$ be the result of prune($E0_P$, $E1_P$, USP).
3. USP = all service primitive actions in $\Sigma_{E1_Q}^U \setminus \Sigma_{CS}$ and their corresponding service primitive actions in $\Sigma_{E0_Q}^U$.
4. Let $P\text{-}E1_Q$ be the result of prune($E0_Q$, $E1_Q$, USP).
5. Find the significant transitions and the significant action set and maximize the significant successor transitions (Section 5.4.1) replacing $E0_P$ with $P\text{-}E0_P$ and replacing $E1_Q$ with $P\text{-}E1_Q$.
6. Let $\Delta = SA \cup \Sigma_{CS}$.
7. Let $ST = \{ t \mid t \in (\text{sig\_suc}(t') \cup \text{sig\_pre}(t')), \forall t' = (r, \rho, r')$ in $E0_P$ and in $E1_Q$, $\rho \in \Sigma_{CS}\}$.
8. Find $RP\text{-}E0_P$ = reduction of $P\text{-}E0_P$ based on Δ and ST (Section 5.5.1).
9. Find $RP\text{-}E1_Q$ = reduction of $P\text{-}E1_Q$ based on Δ and ST (Section 5.5.1).

10. Update the significant transition information to account for the merged states. No service primitives or significant transitions were removed, however, the head states or tail states may have been changed.

11. Find the conversion seed (Section 5.4.2) replacing $E0_P$ with $RP\text{-}E0_P$ and replacing $E1_Q$ with $RP\text{-}E1_Q$.

Corollary 5.4: Given CS, P and Q, if $\{E0_P, C, E1_Q\}$ satisfies external equivalence and semantic equivalence with respect to a conversion seed X generated by the above steps then $\{E0_P, C, E1_Q\}$ satisfies the safety property with respect to CS.

Proof: The proof follows from the correctness of the pruning algorithm [LNS95], Theorem 5.4 and Corollary 5.3. □

Chapter Six

Application to Conversion Seed Approach: Synchronous

6.1 More on Okumura's Conversion Seed Approach

In [Oku86] an approach to construct a protocol converter from a conversion seed is proposed for a particular class of protocols. The class of protocols are those that can be specified using the CFSM model for synchronous communication described in Chapter 5, Section 5.2. As we saw in Chapter 4, Section 4.1.3, given two distinct protocols $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ where the communication model for both protocols is synchronous, Okumura proposes a solution to find a protocol converter C for the conversion system $PQ = \{E0_P, C, E1_Q\}$ such that the conversion system *satisfies* [Section 4.1.3] a given conversion seed X. If the original protocols P and Q are effective and unspecified reception free then the failure of the approach to find a protocol converter indicates that a protocol converter that satisfies the given conversion seed X does not exist for the given protocols P and Q. The failure to generate a converter does not imply that a converter does not exist for a different conversion seed.

The results of Okumura's paper are the following theorems and an algorithm for the construction of a converter.

Theorem 6.1: Let X be a conversion seed. If there exists a converter C for two protocols P and Q with respect to X, then there exists a converter D which is a sub-CFSM of $E1_P \times E0_Q \times X$. [Oku86]

Theorem 6.2: Assume that $E1_P$ is unspecified reception free and effective in protocol $P = \{E0_P, E1_P\}$ and $E0_Q$ is unspecified reception free and effective in protocol $Q = \{E0_Q, E1_Q\}$. Then there exists an algorithm which decides whether a converter for given protocols P and Q with conversion seed X exists. [Oku86]

In [Oku86] Okumura shows that if $E1_P$ is unspecified reception free and effective in protocol P and if $E0_Q$ is unspecified reception free and effective in protocol Q then a CFSM C is a converter for $\{E0_P, C, E1_Q\}$ that satisfies conversion seed X if and only if

1. C is a sub-CFSM of $E1_P \times E0_Q \times X$.

   This property is necessary and sufficient condition for external equivalence and semantic equivalence. [Oku86]

2. C is a reduced-CFSM of $E1_P \times E0_Q$.

   This property is necessary and sufficient condition for C to be unspecified reception free in $\{E0_P, C, E1_Q\}$.[Oku86]

3. C has no states with no outgoing transitions defined.

   This is necessary for $\{E0_P, C, E1_Q\}$ to be deadlock free. [Oku86]

Because there are a finite number of sub-CFSMs of $E1_P \times E0_Q \times X$ and because conditions 2 and 3 are decidable, an algorithm exists that will find a converter and return nil if one does not exist. The algorithm given in [Oku86] recursively removes from $Z = E1_P \times E0_Q \times X$ any state and its incoming and outgoing transitions that either has no outgoing transitions defined or does not have sufficient reception transitions defined to satisfy condition 2. If the state space is empty at the completion of the algorithm then there does not exist a converter for the protocols and conversion seed X.

Assumptions for Okumura's algorithm for the construction of the converter are listed below:

1. $E1_P$ is given and furthermore:

   - $E1_P$ is unspecified reception free in protocol P.

- The internal action transitions and service primitive transitions have been removed from $E1_P$. This can be achieved by projection of $E1_P$ over the subset of Σ_{E1_P} that includes peer protocol messages only. i.e., let $\Delta = \Sigma^{-}_{E1_P} \cup \Sigma^{+}_{E1_P}$ and replace $E1_P$ by $\text{Proj}_{\Delta}(E1_P)$.

2. $E0_Q$ is given and furthermore:
   - $E0_Q$ is unspecified reception free in protocol Q.
   - The internal action transitions and service primitive transitions have been removed from $E0_Q$. This can be achieved by projection of $E0_Q$ over the subset of Σ_{E0_Q} that includes peer protocol messages only. i.e., let $\Delta = \Sigma^{-}_{E0_Q} \cup \Sigma^{+}_{E0_Q}$ and replace $E0_Q$ by $\text{Proj}_{\Delta}(E0_Q)$.
3. A conversion seed X is given and furthermore:
   - X is extended by adding a dead state, d, and the following transitions:

     a) $\forall s \in S_X, s \neq d, \forall \sigma \in \Sigma_X$, if $\delta(s, \sigma)$ is undefined then define $\delta(s, \sigma) = d$.

     b) $\forall s \in S_X, s \neq d, \forall \sigma \in ((\Sigma_{E1_P} \cup \Sigma_{E0_Q}) \setminus \Sigma_X)$, define $\delta(s, \sigma) = s$.

     c) $\forall \sigma \in (\Sigma_X \cup \Sigma_{E1_P} \cup \Sigma_{E0_Q})$, $\delta(d, \sigma)$ is undefined.
4. The action sets of $E0_P$ and $E1_Q$ are distinct: $\Sigma_{E0_P} \cap \Sigma_{E1_Q} = \varnothing$.

algorithm for finding a converter

input: $Z = E1_P \times E0_Q \times X = (S_Z, \Sigma_Z, \delta_Z, s0_Z)$: CFSM

output: $C = (S_C, \Sigma_C, \delta_C, s0_C)$: CFSM or *nil* if no converter was found.

method: find a sub-CFSM of the input CFSM Z such that all states have outgoing transitions and it is a reduced-CFSM of $E1_P \times E0_Q$, if such a CFSM exists.

algorithm for removing a state and its transitions

input: s : state; $A = (S_A, \Sigma_A, \delta_A, s0_A)$: CFSM

output: A : CFSM (reduced)

method: remove all outgoing and incoming transitions of s; if another state loses an outgoing reception transition or becomes a dead state then remove that state as well with the same procedure.

```
procedure CONVERTER ( Z : CFSM; var C : CFSM );

   procedure REMOVE( s : state; var A : CFSM );
      begin
         (* remove outgoing transitions *)
         for all σ ∈ Σ_A such that δ_A(s, σ) = t ∈ S_A do
            δ_A(s, σ) := undefined;

         (* remove incoming transitions *)
         for all r ∈ S_A, σ ∈ Σ_A such that δ_A(r, σ) = s do
            begin
               δ_A(r, σ) := undefined;

               (* Also remove the tail state of the incoming transition if   *)
               (* 1) the transition represents a message reception, or       *)
               (* 2) the tail state is a dead state.                         *)
               if ( σ = +m ) or ( ∀σ' ∈ Σ_A, δ_A(s, σ') = undefined )
                  then
                     REMOVE(r, A);
            end; (* for *)

         (* remove the state from the state set *)
         S_A := S_A \ {s};
      end; (* procedure REMOVE *)
   BEGIN
      C := Z;

      (* remove all dead states *)
      for s ∈ S_C do
         begin
            if ∀σ ∈ Σ_Z, δ_Z(s, σ) = undefined
               then
                  REMOVE(s, C);
         end; (* for *)
```

```
    if C has a non-empty loop from initial state to initial state
       then
          MINIMIZE(C)
       else
          C := nil;
END. (* procedure CONVERTER *)
```

The space complexity of the algorithm is $O(|Z|)$, i.e., the size of the input CFSM Z. C is a sub-CFSM of Z. In the worst case no states (and thus no transitions) are removed from Z.

The worst case scenario when considering computational complexity occurs when every state and transition is removed. The procedure CONVERTER visits each state at most once to determine if it is a dead state and whether it should be removed. Assume state s is identified for removal. In procedure REMOVE, each outgoing transition from s is removed; and each incoming transition to s is removed and its label is tested for a reception action label. Furthermore, the tail state of each incoming transition is tested to determine if it has become a dead state. Finally, the state is removed. Therefore the procedure REMOVE has time complexity $O(\text{degree}(s))$, where s is the state to be removed. With the observation that each transition is only checked and removed once and each state can only be removed once, the time to remove all states and transitions is $O(|\ Z\ |)$, where $|\ Z\ | = |\ S_Z\ |$ (number of states) + $|\ T_Z\ |$ (number of transitions).

The loop in procedure CONVERTER finds all the dead states of Z that need to be removed in $O(|\ S_Z\ |)$ time. Any other states that need to be removed as a result of the removal of an adjacent state are identified within the procedure REMOVE and therefore identifying them takes no additional time.

Thus the time to find all states that need to be removed and remove them along with their incoming and outgoing transitions, in the worst case, is $O(|\ Z\ |)$.

6.2 Example 1

In this section, we provide an example for constructing the required service for the conversion system, *CS*, using the approach in Section 5.3, for constructing a conversion seed according to Approach_1 and then for applying Okumura's approach to construct a protocol converter.

This example demonstrates the following procedures for the synchronous model of communication:

- generalizing the protocol model to direct communication (cf. Section 5.2.3);
- construction of *CS* using submodule construction;
- modification of the input CFSMs to ensure that sig_suc(t) = o_suc(t), for every service primitive transition t;
- constructing the conversion seed; and
- application of Okumura's approach using the constructed conversion seed.

6.2.1 The given protocols and adapters

Assume that the layers at which the protocol conversion will take place are known: layer M protocol of network X (called protocol P) and layer N protocol of network Y (called protocol Q). The required service for the common upper layer peer protocol U is RS_U. Protocol U is layer M+1 protocol of network X and layer N+1 protocol of network Y. The specification of RS_U is given as a CFSM in Figure 34.

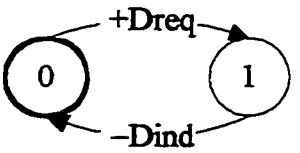

Figure 34: RS_U

The layer M protocol of network X (Figure 35) is protocol P. The peer protocol entities communicate indirectly through a communication medium providing the required service

for the protocol where the communication type between the entities and the communication medium is synchronous. The specification of the protocol entities of protocol $P = \{E0_P, RS_P, E1_P\}$ are given in Figures 37, 38 and 39, where $sync_P(E0_P, RS_P) = true$, $sync_P(RS_P, E1_P) = true$, and $sync_P(E0_P, E1_P) = false$. Protocol P provides the service specified by SS_P (Figure 40). Protocol P is an Alternating Bit Protocol which performs error recovery. The two protocol entities, $E0_P$ and $E1_P$, are based on an unreliable medium, RS_P. In entity $E0_P$ we use an internal transition labeled by tm to represent a timeout waiting for an acknowledgment. In RS_P, we use internal transitions labeled by d-ls and a-ls to represent the loss of a data message and acknowledgment message, respectively. Protocol P is deadlock free and unspecified reception free. In network X there is a gap between SS_P and RS_U, therefore two interface adapters ($A0_P$ and $A1_P$) can be used to supply the required service (RS_U) for protocol U (Figure 36). The specifications of the interface adapters for network X are given in Figures 41 and 42.

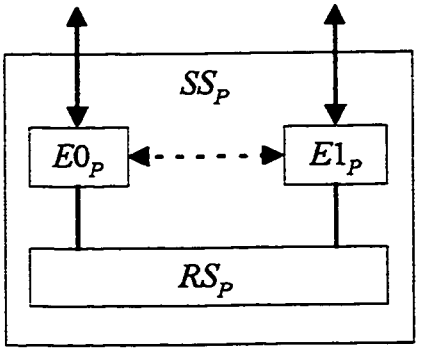

Figure 35: Layer M of network X

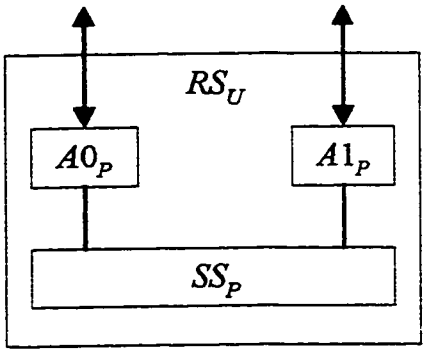

Figure 36: $SS_P \neq RS_U$ in network X

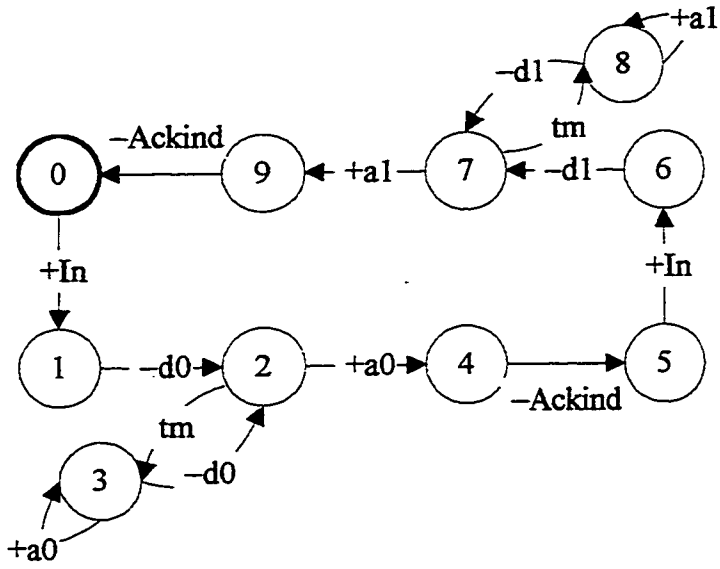

Figure 37: $E0_P$

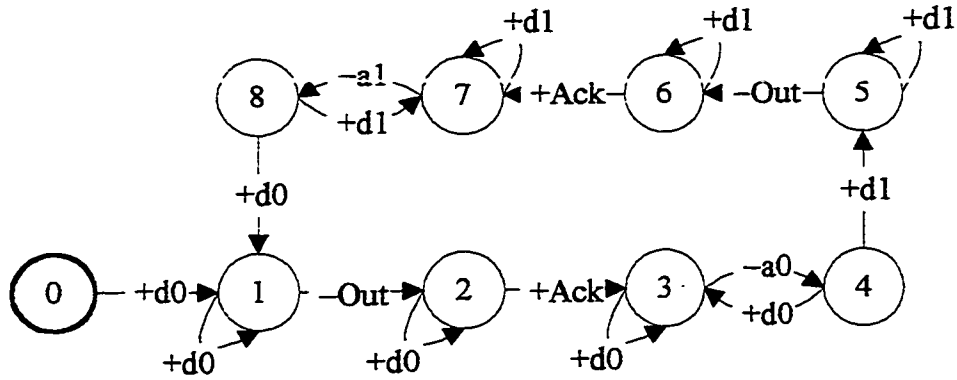

Figure 38: $E1_P$

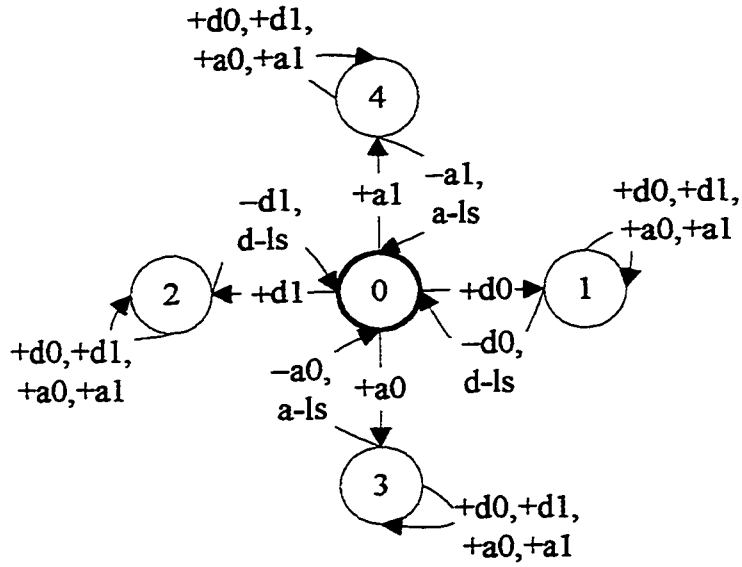

Figure 39: RS_P

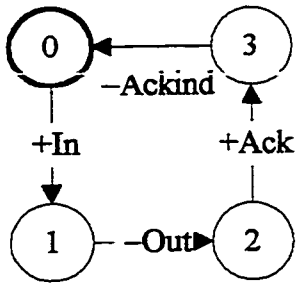

Figure 40: SS_P

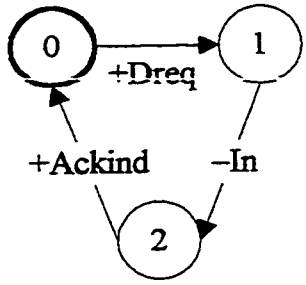

Figure 41: $A0_P$

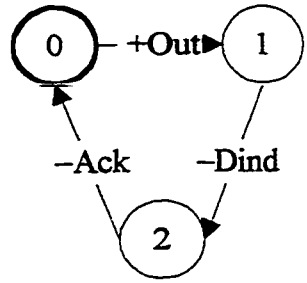

Figure 42: $A1_P$

The layer N protocol of network Y (Figure 43) is protocol Q. The peer protocol entities communicate directly providing the required service for the protocol where the communication type between the entities is synchronous. The specification of the protocol entities of protocol $Q = \{E0_Q, E1_Q\}$ are given in Figures 45 and 46, where $sync_Q(E0_Q, E1_Q) = true$. Protocol Q provides the service specified by SS_Q (Figure 47). Protocol Q is a simple data transfer protocol with an empty medium. No error recovery is performed in this protocol. Protocol Q is deadlock free and unspecified reception free. In network Y there is a gap between SS_Q and RS_U, therefore two interface adapters ($A0_Q$ and $A1_Q$) can be used to supply the required service (RS_U) for protocol U (Figure 44). The specifications of the interface adapters of network Y are given in Figures 48 and 49.

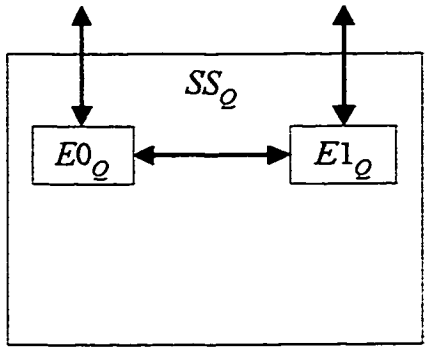

Figure 43: Layer N of network Y

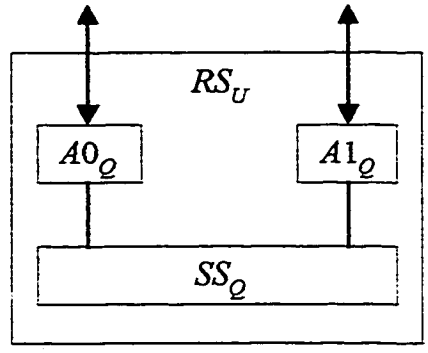

Figure 44: $SS_Q \neq RS_U$ in network Y

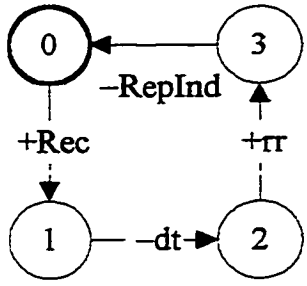

Figure 45: $E0_Q$

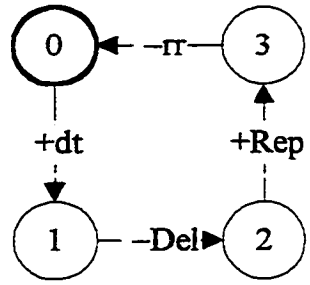

Figure 46: $E1_Q$

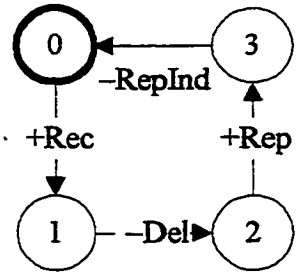

Figure 47: SS_Q

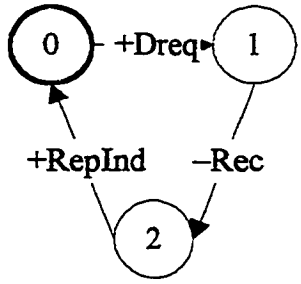

Figure 48: $A0_Q$

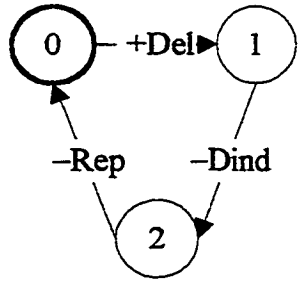

Figure 49: $A1_Q$

We will find CS, and then C, such that the three entities $E0_P$, C, and $E1_Q$, together with the communication medium provide a new protocol that is externally equivalent to Protocol P and Protocol Q, deadlock free and unspecified reception free, and whose service specification satisfies CS (Figure 50). Furthermore, the new protocol whose service

specification satisfies CS together with the existing adapters will satisfy the required service (RS_U) for the common peer protocol U (Figure 51).

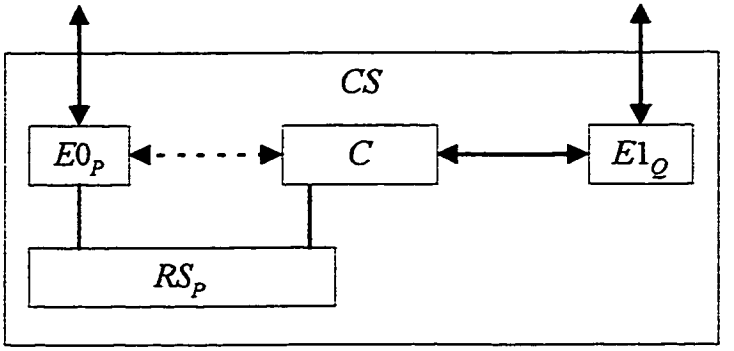

Figure 50: Network Z

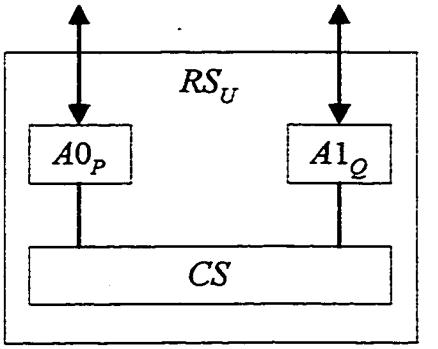

Figure 51: $SS_P \neq RS_U$ in network X, $SS_Q \neq RS_U$ in network Y

6.2.2 Direct communication only

Approach_1 assumes the input protocols communicate directly [Chapter 5, Section 5.2.3]. Therefore, we must transform protocol P into protocol $P' = \{E0'_P, E1_P\}$ where $sync_{P'}(E0'_P, E1_P) = true$ and $E0'_P = E0_P \otimes RS_P$. In Figure 52 the specification of $Proj_\Delta(E0'_P)$ is shown, where $\Delta = \{+In, -AckInd, -d0, -d1, +a0, +a1\}$. To simplify the CFSM for this presentation we replaced $E0'_P$ with $Proj_\Delta(E0'_P)$, i.e., the internal transitions of $E0'_P = E0_P \otimes RS_P$, {d0, a0, d1, a1, tm, d-ls, a-ls}, were removed by projecting over Δ.

Note: In the remainder of this example, $E0_P$ will refer to $Proj_\Delta(E0'_P)$ unless otherwise stated.

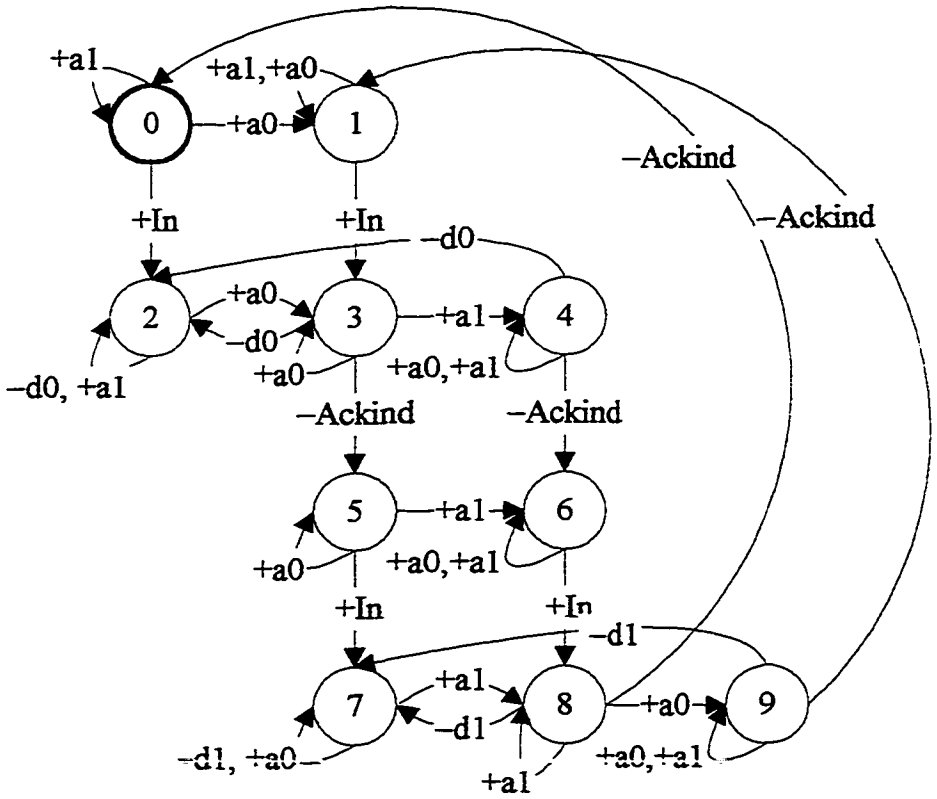

Figure 52: $\text{Proj}_{\Delta}(E0_P \otimes RS_P)$, Δ = {+In, –AckInd, –d0, –d1, +a0, +a1}

6.2.3 Finding the required service specification for the conversion system

Since for both protocols P and Q, adapters are specified to provide the required service for protocol U, this example satisfies case 4 of the problem statement given in the beginning of Section 5.1. We must construct the required service specification of the conversion system using the submodule construction approach [Chapter 5, Section 5.3.1].

Step 1: Define E_0 (= RS_U extended)

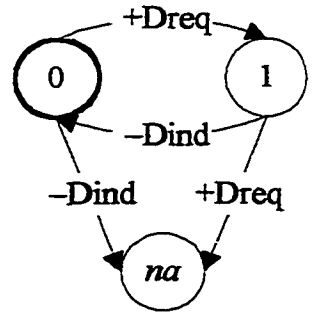

Figure 53: E_0 (= RS_U extended)

Step 2: Define $E_1 = A0_P \otimes A1_Q$.

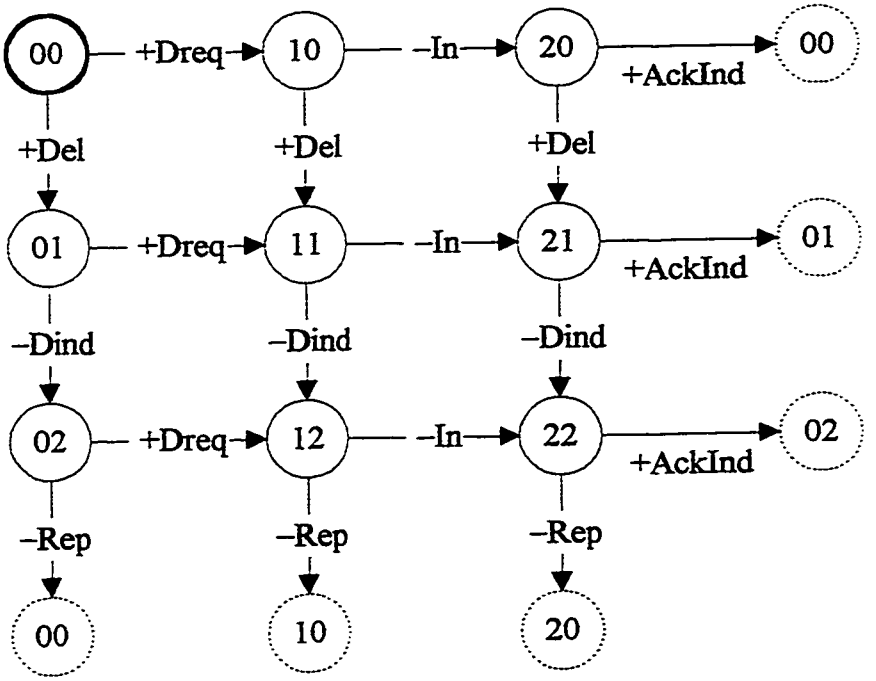

Figure 54: $E_1 = A0_P \otimes A1_Q$

Step 3: Define $\Sigma_2 = \{ \sigma \mid \sigma \in \Sigma_0 \setminus \Sigma_1 \} \cup \{ \sigma \mid \sigma^{-1} \in \Sigma_1 \setminus \Sigma_0 \}$

$\Sigma_2 = \{$ +In, -AckInd, -Del, +Rep $\}$.

Step 4: Find E_2 such that E_2 satisfies

$$L(E_2) = \text{Pre}(\text{Proj}_{\Sigma_2}(L((E_0^{-1} \otimes E_1)^{-1})) \setminus \text{Proj}_{\Sigma_2}(L((\neg E_0^{-1} \otimes E_1)^{-1}))).$$

As pointed out in Chapter 5, Section 5.3.2, it is sufficient to only form the CFSM $E = (E_0^{-1} \otimes E_1)^{-1}$ (Figure 55). The projection of E over Σ_2 is shown in Figure 56. The shaded states are invalid and will be removed to form E_2 (Figure 57).

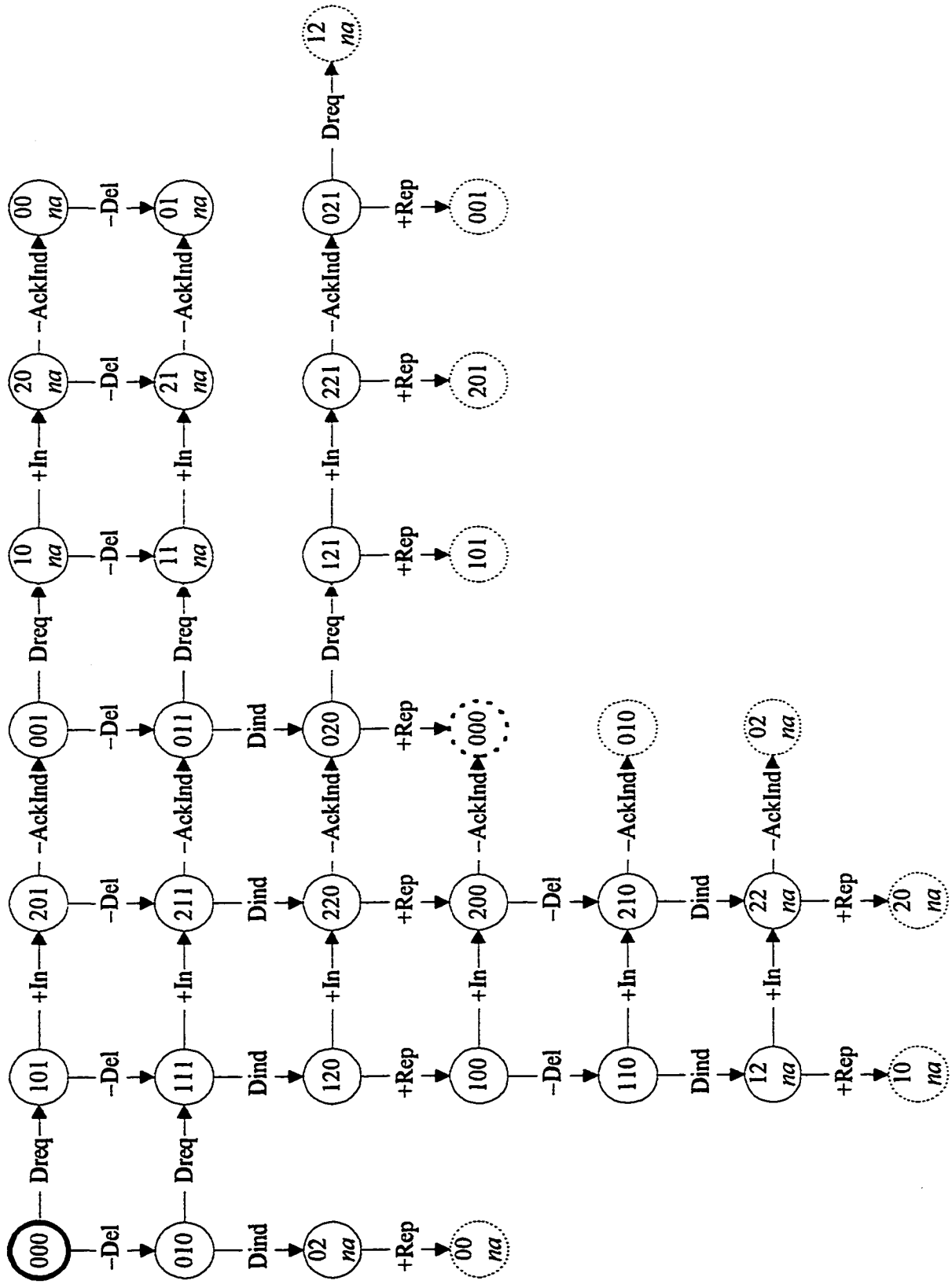

Figure 55: $E = (E_0^{-1} \otimes E_1)^{-1}$

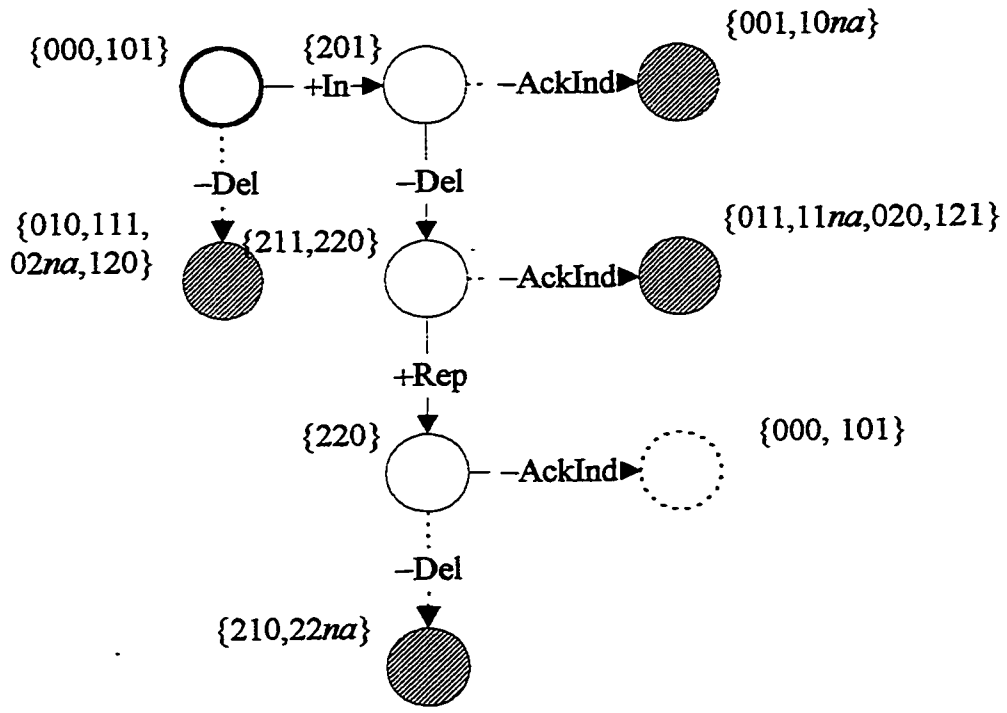

Figure 56: $\text{Proj}_{\Sigma_2}((E_0^{-1} \otimes E_1)^{-1})$ where Σ_2 = { +In, –AckInd, –Del, +Rep }, the shaded states are "invalid".

Step 5: $CS = E_2$

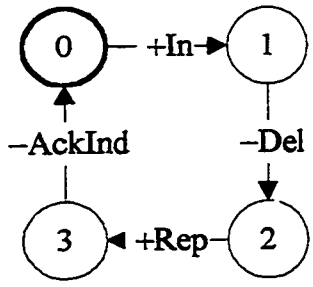

Figure 57: $CS = E_2$ (relabeled)

6.2.4 Pruning the input CFSMs

Protocol pruning is not necessary for this example since $\Sigma_{CS} = \Sigma^U_{E0_P} \cup \Sigma^U_{E1_Q}$.

6.2.5 Significant transitions and the significant action set

At this stage we find the sets of significant transitions for each service primitive transition in $E0_P$ and $E1_Q$, and construct the significant action set according to Chapter 5, Section 5.4.1. The CFSM $E0_P$ contains a number of observable successor transitions that are not

significant for their respective service primitive transition. Therefore, $E0_P$, the sets of significant transitions and the significant action set were modified according to the steps in Chapter 5, Section 5.4.1, to ensure that the every observable successor transition for a service primitive transition is also a significant successor transition for that service primitive transition. The state transition table defining the new $E0_P$ is given in Table 2. The significant action set for the new $E0_P$ is SA = {+a1, +a0, −d0, −d1, +dt, −rr}. The significant transitions of the new $E0_P$ are given in Table 3. Table 4 provides the significant transitions of $E1_Q$.

state	+In	−Ackind	−d0	+a0	−d1	+a1
0	2			1		0*a*
0*a*	2			1		0*a*
1	3			1*a*		1*a*
1*a*	3			1*a*		1*a*
2			2*a*	3		2*a*
2*a*			2*a*	3		2*a*
3		5	2	3*a*		4
3*a*		5	2	3*a*		4
4		6	2	4		4
5	7			5*a*		6
5*a*	7			5*a*		6
6	8			6*a*		6*a*
6*a*	8			6*a*		6*a*
7				7*a*	7*a*	8
7*a*				7*a*	7*a*	8
8		0		9	7	8*a*
8*a*		0		9	7	8*a*
9		1		9	7	9

Table 2: New $E0_P$ (with $sig\_suc(t) = obs\_suc(t)$)

transition *t*	**sig_pre(*t*)**	**sig_suc(*t*)**
(0,+In,2)	(3,+a1,4)(3*a*,+a1,4)(5,+a1,6) (5*a*,+a1,6)(7,+a1,8)(7*a*,+a1,8)	(2,–d0,2*a*) (2,+a0,3) (2,+a1,2*a*)
(0*a*,+In,2)	(0,+a1,0*a*)	same as above
(1,+In,3)	(0,+a0,1) (0*a*,+a0,1) (8,+a0,9) (8*a*,+a0,9)	(7,+a0,7*a*)(7,–d1,7*a*)(7,+a1,8)(5,+a0,5*a*) (5,+a1,6)(3,–d0,2)(3,+a0,3*a*)(3,+a1,4)
(1*a*,+In,3)	(1,+a0,1*a*) (1,+a1,1*a*)	same as above
(3,–AckInd,5)	(0,+a0,1) (0*a*,+a0,1) (8,+a0,9) (8*a*,+a0,9)(2,+a0,3) (2*a*,+a0,3)	(7,+a0,7*a*) (7,–d1,7*a*) (7,+a1,8) (5,+a0,5*a*) (5,+a1,6)
(3*a*,–AckInd,5)	(3,+a0,3*a*)	same as above
(4,–AckInd,6)	(3,+a1,4) (3*a*,+a1,4)	(2,–d0,2*a*) (2,+a0,3) (2,+a1,2*a*) (0,+a0,1) (0,+a1,0*a*) (8,+a0,9) (8,–d1,7) (8,+a1,8*a*) (6,+a0,6*a*) (6,+a1,6*a*)
(5,+In,7)	(0,+a0,1) (0*a*,+a0,1) (8,+a0,9) (8*a*,+a0,9)(2,+a0,3) (2*a*,+a0,3)	(7,+a0,7*a*) (7,–d1,7*a*) (7,+a1,8)
(5*a*,+In,7)	(5,+a0,5*a*)	same as above
(6,+In,8)	(3,+a1,4) (3*a*,+a1,4) (5,+a1,6) (5*a*,+a1,6)	(2,–d0,2*a*) (2,+a0,3) (2,+a1,2*a*) (0,+a0,1) (0,+a1,0*a*) (8,+a0,9) (8,–d1,7) (8,+a1,8*a*)
(6*a*,+In,8)	(6,+a0,6*a*) (6,+a1,6*a*)	same as above
(8,–AckInd,0)	(3,+a1,4) (3*a*,+a1,4) (5,+a1,6) (5*a*,+a1,6)(7,+a1,8) (7*a*,+a1,8)	(2,-d0,2*a*) (2,+a0,3) (2,+a1,2*a*) (0,+a0,1) (0,+a1,0*a*)
(8*a*,–AckInd,0)	(8, +a1, 8*a*)	same as above
(9,–AckInd,1)	(8,+a0,9) (8*a*,+a0,9)	(7,+a0,7*a*)(7,–d1,7*a*) (7,+a1,8) (5,+a0,5*a*) (5,+a1,6) (3,–d0,2) (3,+a0,3*a*) (3,+a1,4) (1,+a0,1*a*) (1,+a1,1*a*)

Table 3: Significant Transitions of New $E0_P$

Note: In the remainder of this example, $E0_P$ will refer to the new $E0_P$ constructed in this step unless otherwise stated.

transition t	sig_pre(t)	sig_suc(t)
(1,–Del,2)	(0,+dt,1)	(3,–rr,0)
(2,+Rep,3)	(0,+dt,1)	(3,–rr,0)

Table 4: Significant Transitions of $E1_Q$

6.2.6 Reducing the input CFSMs

Given the significant transitions and the significant action set, the entities $E0_P$ and $E1_Q$ may be reduced by merging states according to the steps in Chapter 5, Section 5.5.1. In this case, neither $E0_P$ nor $E1_Q$ have states that can be merged.

6.2.7 Constructing the conversion seed

Given CS, $E0_P$, $E1_Q$, the significant transitions and the significant action set SA, construct the conversion seed according to the steps in Chapter 5, Section 5.4.2.

Step 1: Construct $(E0_P \times E1_Q \times CS)$~red

First, extend CS to include a dead state and the appropriate transitions (Figure 58). Next, the CFSM $(E0_P \times E1_Q \times CS)$ is constructed and then reduced to obtain $(E0_P \times E1_Q \times CS)$~red (Figure 59).

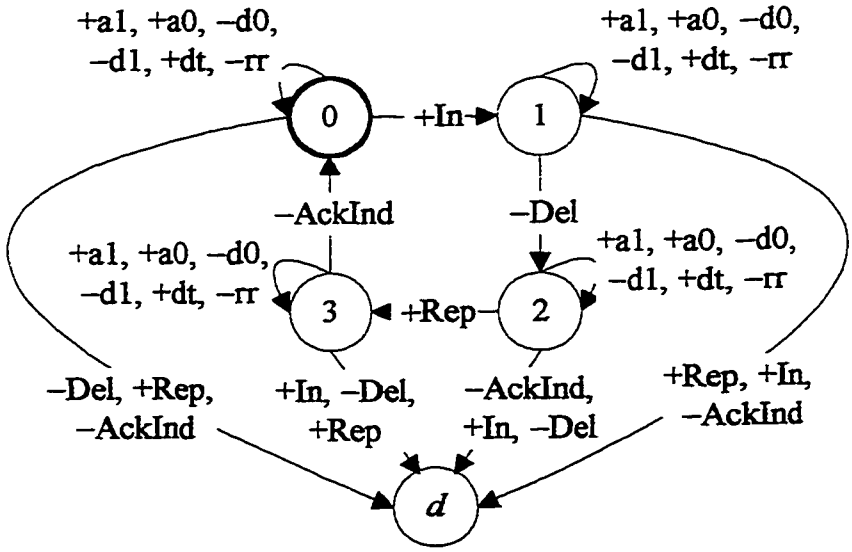

Figure 58: CS (extended)

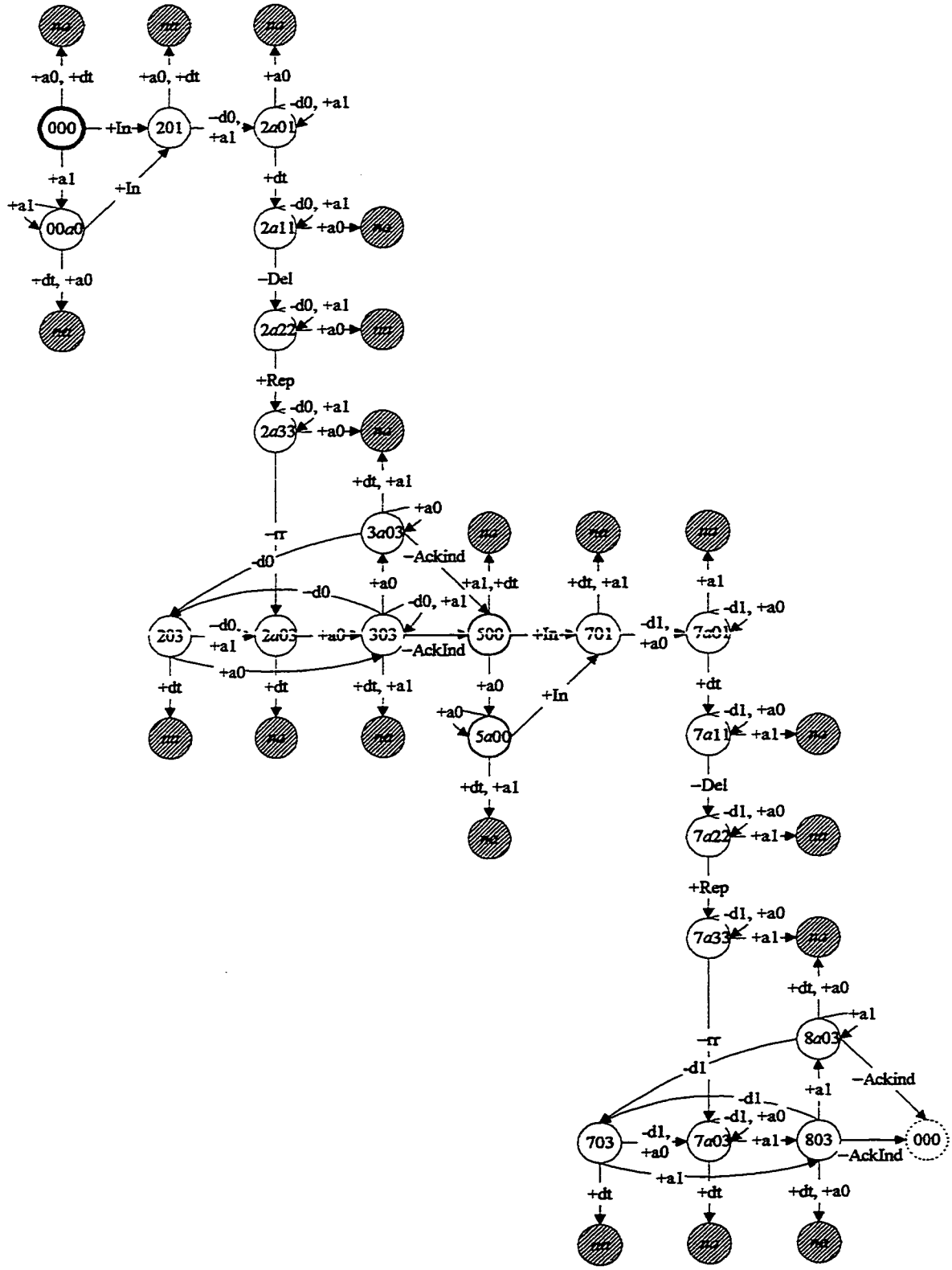

Figure 59: $(E0_P \times E1_Q \times CS)$~red

Step 2 : Find W such that

$$L(W) = \text{Pre}(L(\text{Proj}_{SA}(E0_P \times E1_Q \times CS)\sim\text{red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\sim\text{red})))$$

Step 3 : $X = W^{-1}$ is the conversion seed.

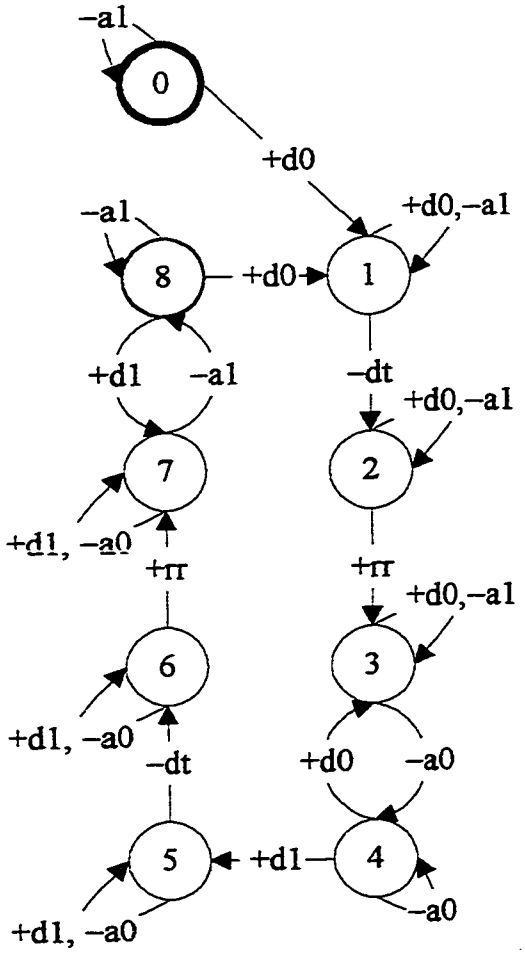

Figure 60: Conversion Seed $X = W^{-1}$ (relabeled and minimized)

6.2.8 Constructing a converter: Okumura's approach

The conversion seed X (Figure 60) can now be used as input to Okumura's approach along with involved entities $E1_P$ (Figure 38) and $E0_Q$ (Figure 45). First, they must be modified to satisfy the assumptions [Section 6.1] for the construction of a converter using Okumura's algorithm. Specifically, the entities $E1_P$ and $E0_Q$ may not contain service primitive transitions and the conversion seed X must be extended. The new $E1_P$, the new $E0_Q$ and the new X are shown in Figures 61, 62 and 63, respectively.

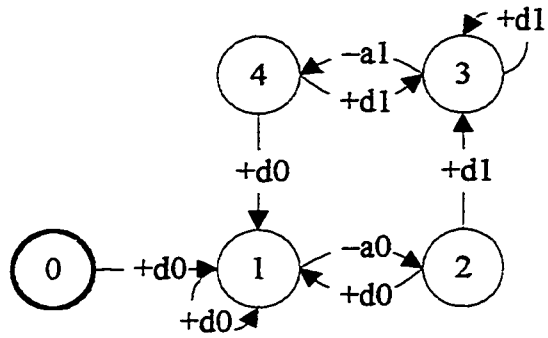

Figure 61: new $E1_P$ (service primitive transitions removed)

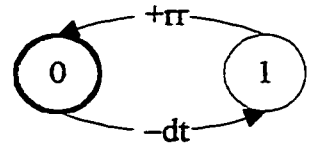

Figure 62: new $E0_Q$ (service primitive transitions removed)

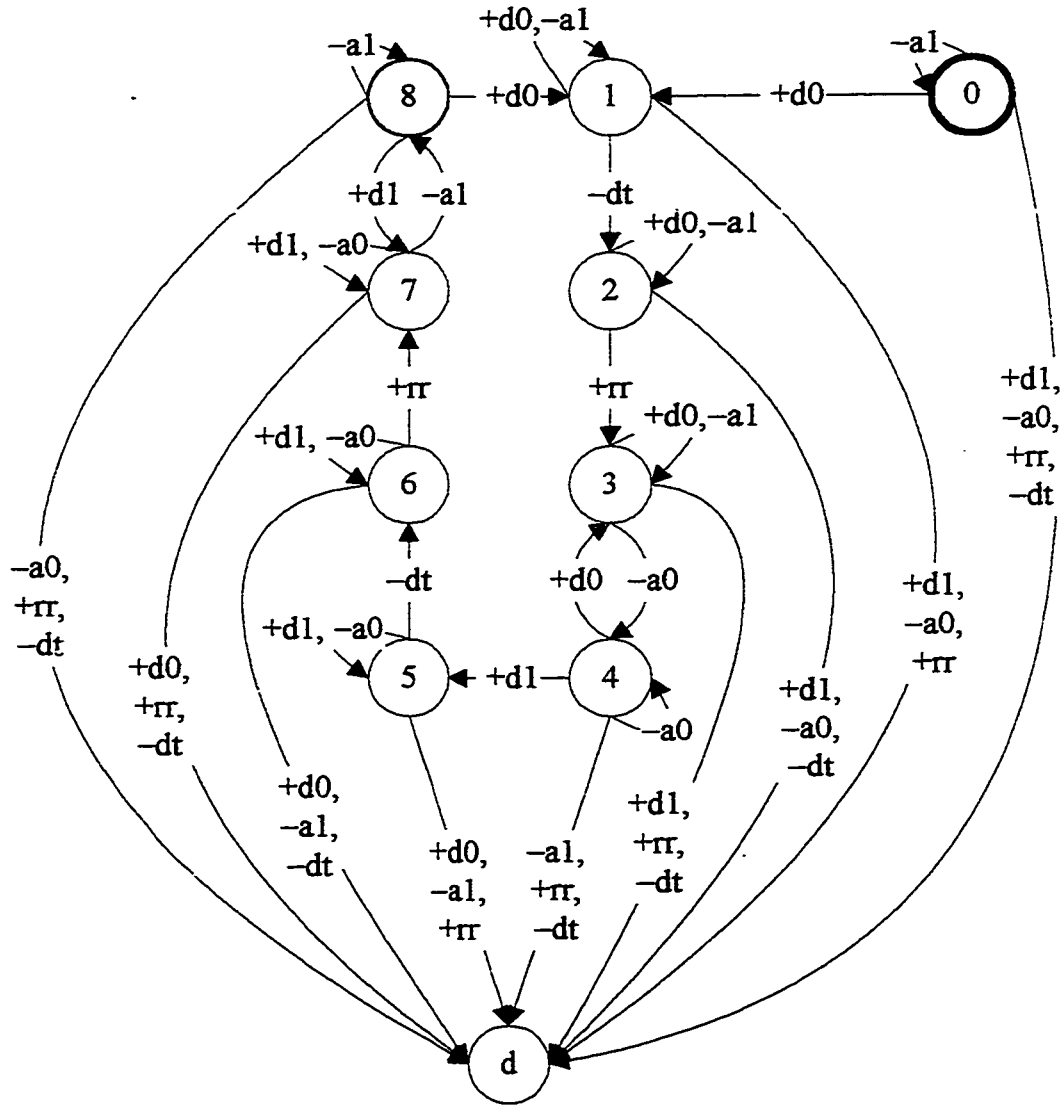

Figure 63: new X (extended)

Note: In the remainder of this example, $E1_P$ will refer to the new $E1_P$ constructed in this step unless otherwise stated, $E0_Q$ will refer to the new $E0_Q$ constructed in this step unless otherwise stated, and X will refer to the new X constructed in this step unless otherwise stated.

The input to Okumura's algorithm is the CFSM $Z = E1_P \times E0_Q \times X$.

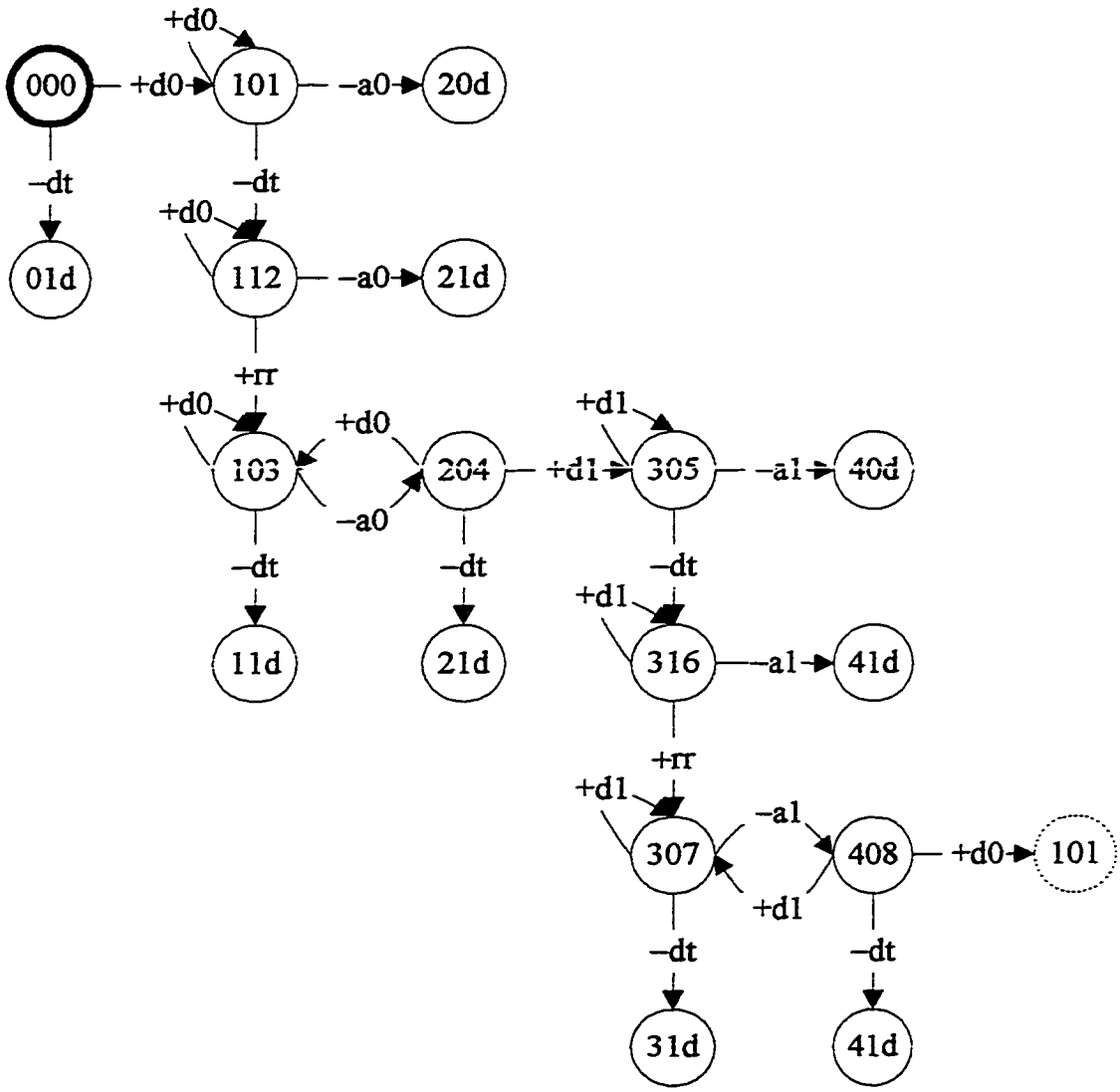

Figure 64: $Z = E1_P \times E0_Q \times X$

Okumura's algorithm finds a sub-CFSM of Z (Figure 64) such that all the states have outgoing transitions and it is a reduced-CFSM of $E1_P \times E0_Q$ (Figure 65), if such a CFSM exists. The resulting sub-CFSM (Figure 66) is the converter. Figure 67 shows the converter with the states relabeled.

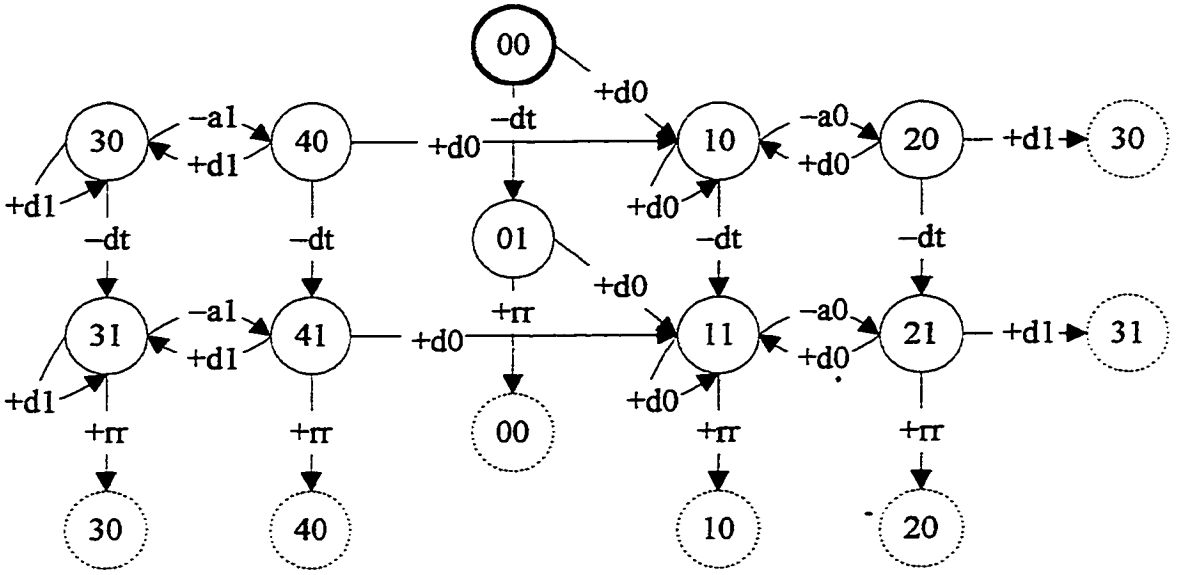

Figure 65: $E1_P \times E0_Q$

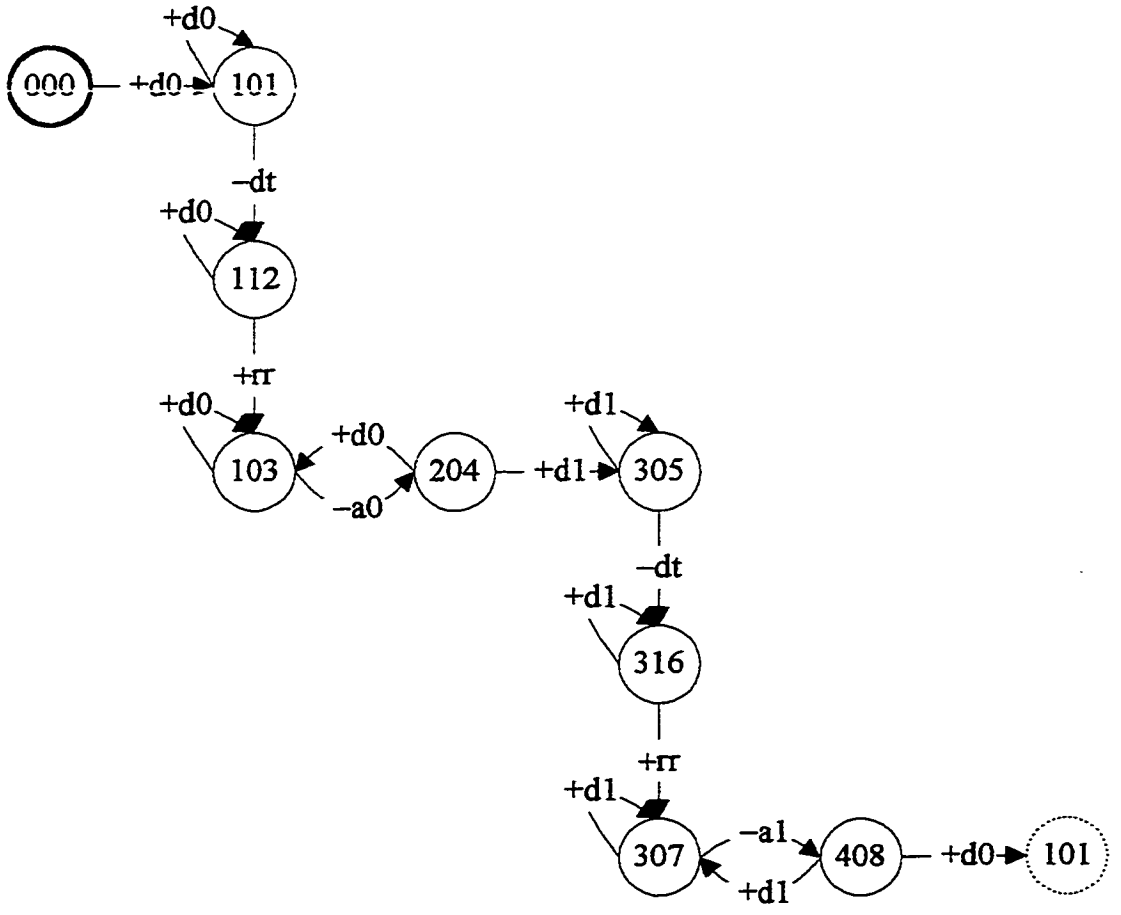

Figure 66: Z (reduced)

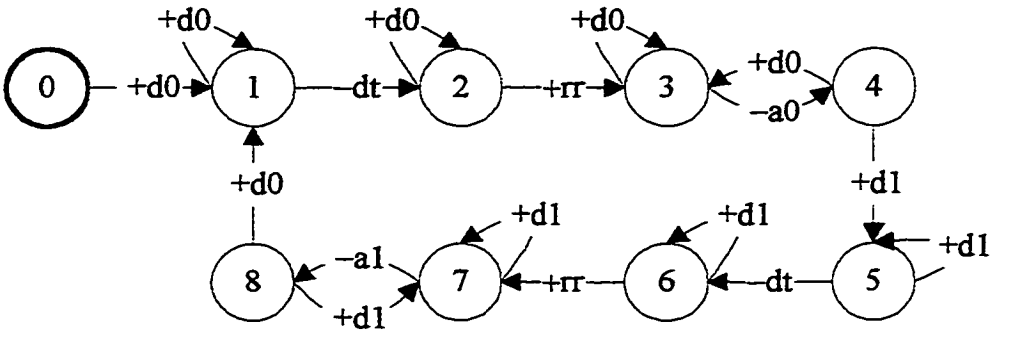

Figure 67: The converter, C (relabeled)

The *original* protocol entities $E0_P$ (Figure 37), RS_P (Figure 39) and $E1_Q$ (Figure 46), and the converter C (Figure 67), together form a new protocol $\{E0_P, RS_P, C, E1_Q\}$ that satisfies external equivalence with respect to the protocols P and Q, semantic equivalence with respect to the conversion seed X, and freedom from deadlock and unspecified receptions. Furthermore, the service provided by the new protocol satisfies CS with respect to safety; which, in turn, satisfies RS_U with respect to safety.

6.3 Example 2

In this section, we provide an example for first constructing a conversion seed according to Approach_3 and then applying Okumura's approach to construct a protocol converter.

This example demonstrates the following procedures for the synchronous model of communication:

- pruning the input protocols;
- reducing the input protocols;
- the reduction steps while constructing the conversion seed; and
- the reduction steps of Okumura's algorithm while applying Okumura's approach using the constructed conversion seed.

6.3.1 The given protocols and adapters

Assume that the layers at which the protocol conversion will take place are known: layer M protocol of network X (called protocol P) and layer N protocol of network Y (called

protocol Q). The required service for the common upper layer peer protocol U is RS_U. Protocol U is layer M+1 protocol of network X and layer N+1 protocol of network Y. The specification of RS_U is given as a CFSM in Figure 68.

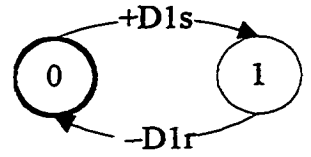

Figure 68: RS_U

The layer M protocol of network X (Figure 69) is protocol P. The peer protocol entities communicate directly where the communication type is synchronous. The specification of the protocol entities of protocol $P = \{E0_P, E1_P\}$ are given in Figures 71 and 72, where $sync_P(E0_P, E1_P) = true$. Protocol P provides the service specified by SS_P (Figure 73). Protocol P operates in the following manner: on receiving the data D1s, the sender of protocol P ($E0_P$) sends it as a data unit d to the receiver ($E1_P$), followed by a message indicating the end of data, e. On receiving the data unit d and the end of data message e the receiver ($E1_P$) sends the data D1r to the user and sends an acknowledgment message, ack, to the sender ($E0_P$). Protocol P is deadlock free and unspecified reception free. In network X there is no gap between SS_P and RS_U (Figure 70), therefore interface adapters are not needed.

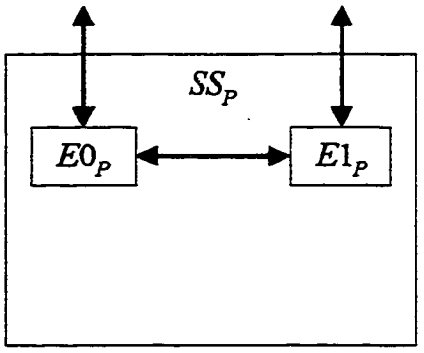

Figure 69: Layer M of network X

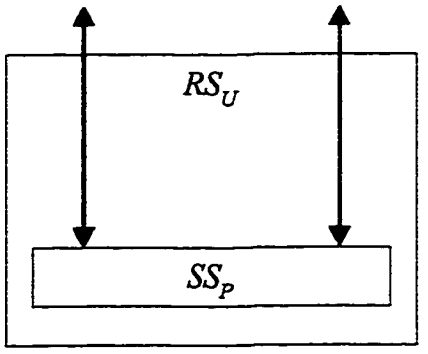

Figure 70: $SS_P = RS_U$ in network X

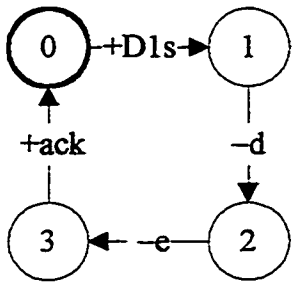

Figure 71: $E0_P$ (sender)

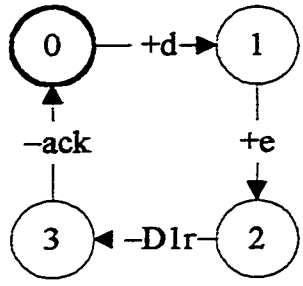

Figure 72: $E1_P$ (receiver)

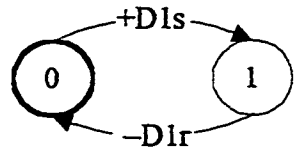

Figure 73: SS_P

The layer N protocol of network Y (Figure 74) is protocol Q. The peer protocol entities communicate directly providing the required service for the protocol where the communication type between the entities is synchronous. The specification of the protocol entities of protocol $Q = \{E0_Q, E1_Q\}$ are given in Figures 76 and 77, where $sync_Q(E0_Q, E1_Q) = true$. Protocol Q provides the service specified by SS_Q (Figure 78). Protocol Q functions as follows: on receiving the data D1s, the sender ($E0_Q$) sends it as a data unit x to the receiver ($E1_Q$). On receiving the data D2s, the sender ($E0_Q$) sends it as a data unit y to the receiver ($E1_Q$). On receiving a data unit, x or y, the receiver ($E1_Q$) sends either an acknowledgment (z) or a negative acknowledgment (nz) to the sender ($E0_Q$). In the case of a negative acknowledgment, the sender resends the data unit and the receiver waits for the retransmission of the data unit. After a positive acknowledgment the receiver transmits the data, D1r or D2r, to the user. Note that the sender always waits for a poll signal from the receiver before sending either data unit x or data unit y to the receiver. Protocol Q is deadlock free and unspecified reception free. In network Y there is no gap between SS_Q and RS_U (Figure 75), therefore interface adapters are not needed.

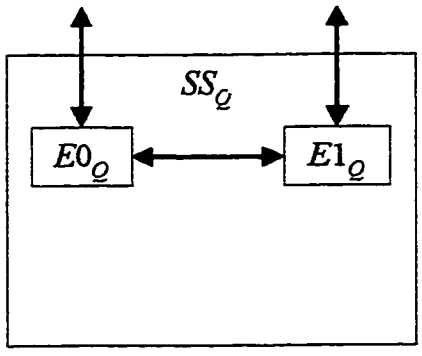

Figure 74: Layer N of network Y

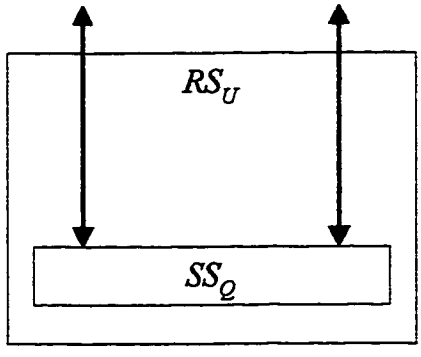

Figure 75: $SS_Q = RS_U$ in network Y

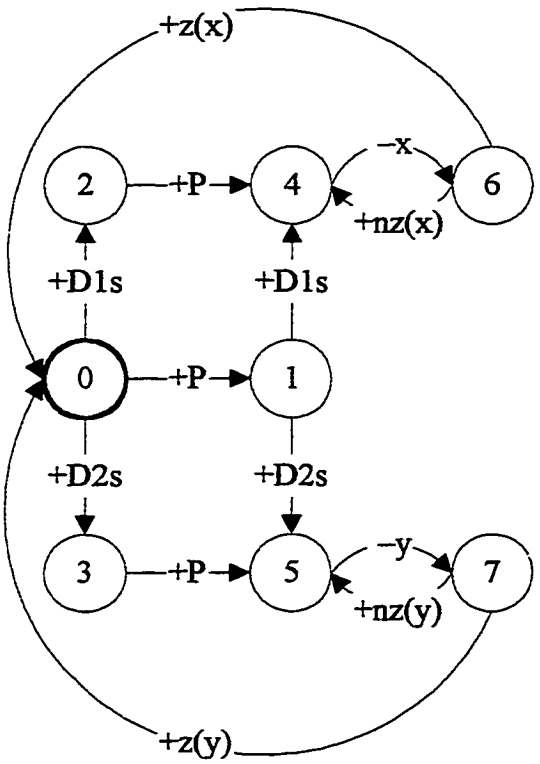

Figure 76: $E0_Q$

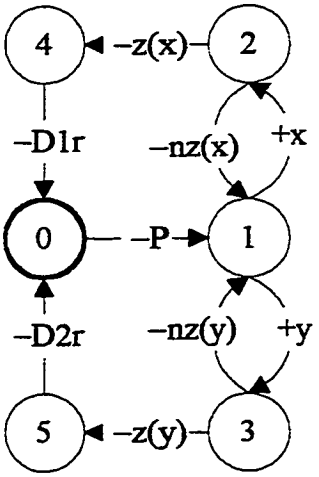

Figure 77: $E1_Q$

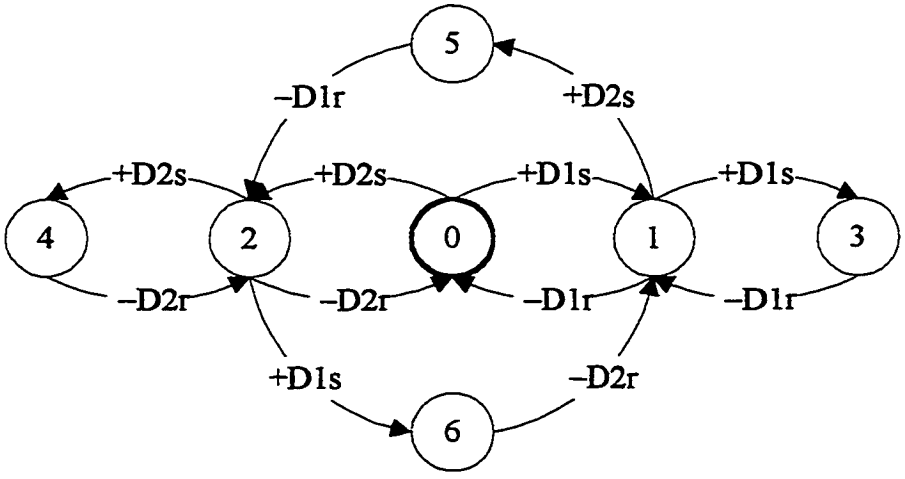

Figure 78: SS_Q

We will find *CS*, and then *C*, such that the three entities $E0_P$, *C*, and $E1_Q$, provide a new protocol that is externally equivalent to Protocol *P* and Protocol *Q*, deadlock free and unspecified reception free, and whose service specification satisfies *CS* (Figure 79).

Furthermore, the new protocol whose service specification satisfies *CS* will satisfy the required service (RS_U) for the common peer protocol *U* (Figure 80).

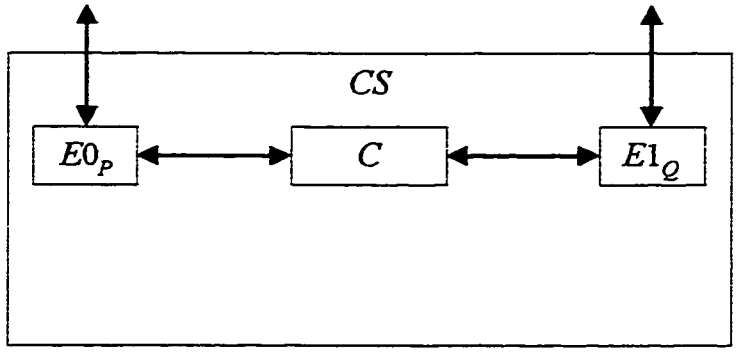

Figure 79: Network *Z*

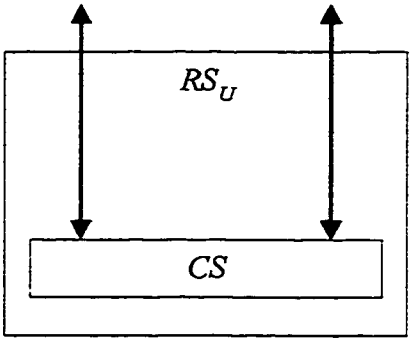

Figure 80: $SS_P = RS_U$ in network *X*, $SS_Q = RS_U$ in network *Y*

6.3.2 Direct communication only

Approach_3 assumes the input protocols communicate directly [Chapter 5, Section 5.2.3]. The given protocols satisfy this assumption; therefore no transformation is necessary.

6.3.3 Finding the required service specification for the conversion system

Since no adapters are specified to provide the required service for protocol *U*, this example satisfies case 1 of the problem statement given in the beginning of Section 5.1. According to Chapter 5, Section 5.3.1, the specification for the conversion system *CS* is RS_U.

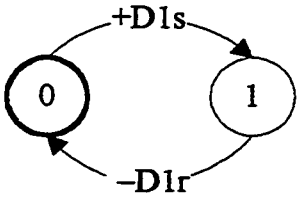

Figure 81: $CS = RS_U$

6.3.4 Pruning the input CFSMs

Protocol pruning is necessary for this example since $\Sigma_{CS} \subset \Sigma^U_{E0_P} \cup \Sigma^U_{E1_Q}$. The protocols are pruned according to the steps in Chapter 5, Section 5.6.1. For Protocol *P*, the set of

unmatched service primitives, USP_P, is the set of all service primitive actions in $\Sigma^U_{E0_P} \setminus \Sigma_{CS}$ and their corresponding service primitive actions in $\Sigma^U_{E1_P}$. Therefore, $USP_P = \varnothing$ which implies that protocol P can not be pruned. For Protocol Q, the set of unmatched service primitives, USP_Q, is the set of all service primitive actions in $\Sigma^U_{E1_Q} \setminus \Sigma_{CS}$ and their corresponding service primitive actions in $\Sigma^U_{E0_Q}$. Therefore, USP_Q = {D2s, D2r} which implies that Protocol Q can be pruned to remove the parts of the protocol entities that only contribute to that service.

Step 1: Remove all transitions with unmatched service primitives as labels from $E0_Q$ and $E1_Q$. Currently, the unmatched service primitives are {D2s, D2r}. The resulting CFSMs are shown in Figures 82 and 83, respectively.

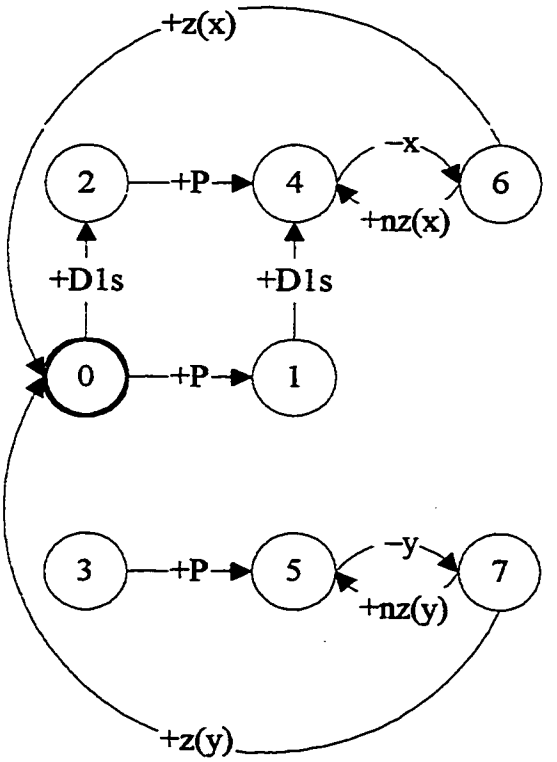

Figure 82: Step 1 $E0_Q$

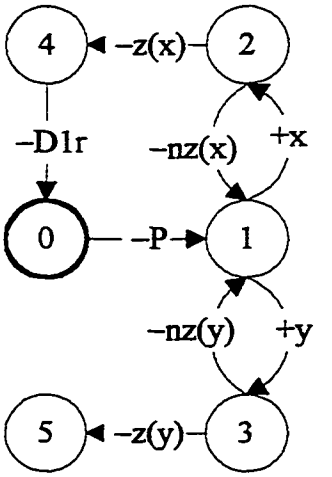

Figure 83: Step 1 $E1_Q$

Step 2: For each CFSM, compute the strongly connected component that starts at the initial state, discarding the rest of the CFSM. The resulting CFSMs are shown in Figures 84 and 85, respectively.

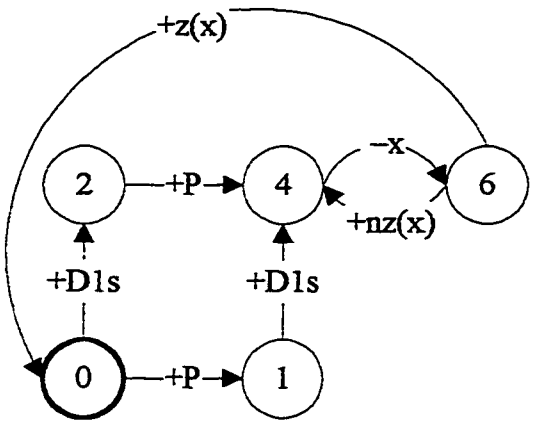

Figure 84: Step 2 $E0_Q$

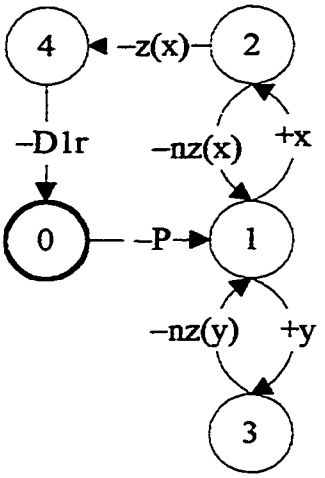

Figure 85: Step 2 $E1_Q$

Step 3: Remove any resulting transitions with unmatched service primitive or peer protocol messages as labels from $E0_Q$ and $E1_Q$. Currently, there are no unmatched service primitives. With the removal of all transitions labeled by "–y" and "+nz(y)" from $E0_Q$ at Step 1, the current set of unmatched peer protocol messages are {+y, –nz(y)}. The transitions related to these messages are removed from $E1_Q$. The resulting CFSMs are shown in Figures 86 and 87, respectively. Note that $E0_Q$ did not change.

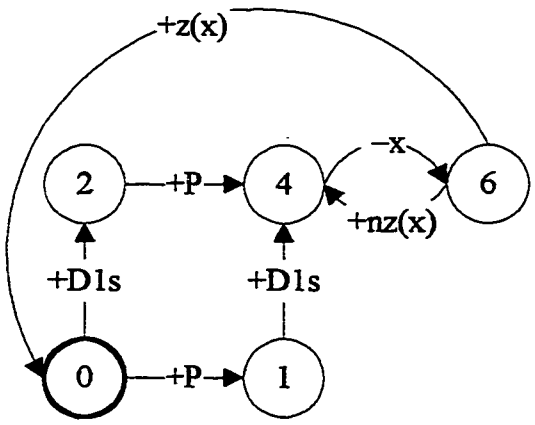

Figure 86: Step 3 $E0_Q$ (no change)

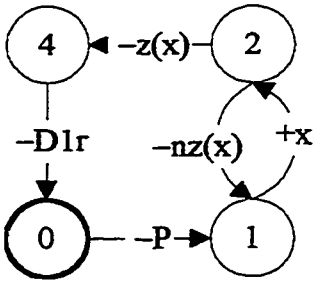

Figure 87: Step 3 $E1_Q$

Step 4: For the CFSM that changed (Figure 87), compute the strongly connected component that starts at the initial state, discarding the rest of the CFSM. The resulting CFSM is the same as Figure 87.

Step 5: All transitions in $E0_Q$ and $E1_Q$ are matched and each CFSM is a strongly connected component. Therefore, the resulting CFSMs are the pruned $E0_Q$ (Figure 88) and the pruned $E1_Q$ (Figure 89).

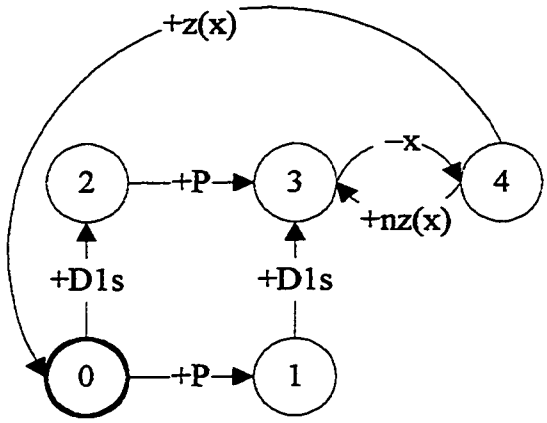

Figure 88: Pruned $E0_Q$ (relabeled)

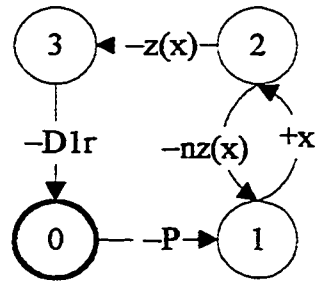

Figure 89: Pruned $E1_Q$ (relabeled)

Note: In the remainder of this example, $E0_Q$ will refer to the pruned $E0_Q$ constructed in this step unless otherwise stated and $E1_Q$ will refer to the pruned $E1_Q$ constructed in this step unless otherwise stated.

6.3.5 Significant transitions and the significant action set

At this stage we find the sets of significant transitions for each service primitive transition in $E0_P$ and $E1_Q$; and construct the significant action set according to Chapter 5, Section 5.4.1. The significant action set is SA = {+ack, −z(x), −d, −P}. The significant transitions of $E0_P$ are given in Table 5. Table 6 provides the significant transitions of $E1_Q$.

transition *t*	**sig_pre(*t*)**	**sig_suc(*t*)**
(0, +D1s, 1)	{(3, +ack, 0)}	{(1, −d, 2)}

Table 5: Significant Transitions of $E0_P$

transition *t*	**sig_pre(*t*)**	**sig_suc(*t*)**
(4, −D1r, 0)	{(2, −z(x), 4)}	{(0, −P, 1)}

Table 6: Significant Transitions of $E1_Q$

6.3.6 Reducing the input CFSMs

Given the significant transitions and the significant action set, the entities $E0_P$ and $E1_Q$ may be reduced by merging states according to the steps in Chapter 5, Section 5.5.1. In this case, both $E0_P$ and $E1_Q$ have states that can be merged.

Δ = {+ack, –z(x), –d, –P, +D1s, –D1r}.

ST = {(3, +ack, 0), (1, –d, 2), (2, –z(x), 4), (0, –P, 1)}

Step 1 ($E0_P$): Reduced $E0_P = E0_P$

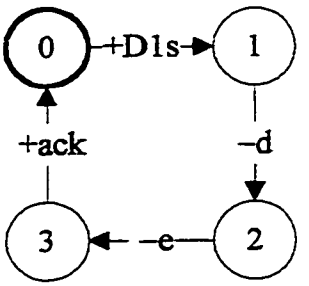

Figure 90: Step 1 $E0_P$

Step 2 ($E0_P$): Find a pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in ST. Such a pair of states is (2, 3).

Step 3 ($E0_P$): M1 and M2 are true for (2, 3) so they can be merged.

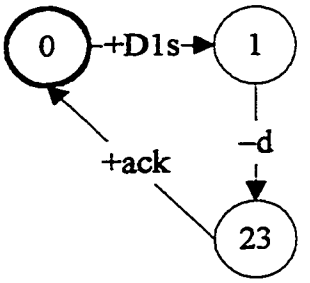

Figure 91: Step 3 $E0_P$ (merge #1)

Step 4 ($E0_P$): There is no pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in ST. The resulting CFSM is the reduced $E0_P$ in Figure 92.

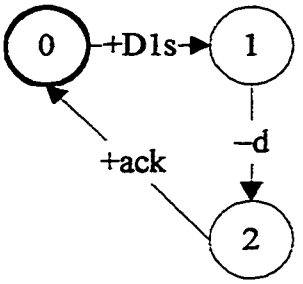

Figure 92: Reduced $E0_P$ (relabeled)

Step 5 ($E0_P$): Update the table of significant transitions to reflect the relabeled states of Reduced $E0_P$.

transition *t*	**sig_pre(*t*)**	**sig_suc(*t*)**
(0, +D1s, 1)	{(2, +ack, 0)}	{(1, –d, 2)}

Table 7: Significant Transitions of Reduced $E0_P$

Step 1 ($E1_Q$): Reduced $E1_Q = E1_Q$

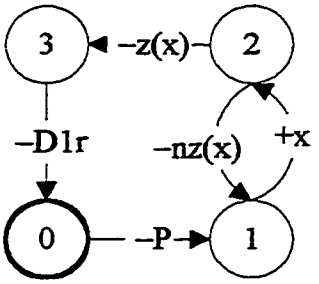

Figure 93: Step 1 $E1_Q$

Step 2 ($E1_Q$): Find a pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in *ST*. Such a pair of states is (1, 2).

Step 3 ($E1_Q$): M3 is true for (1, 2) so they can be merged.

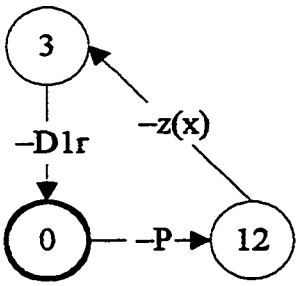

Figure 94: Step 3 $E1_Q$ (merge #1)

Step 4 ($E1_Q$): There is no pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in *ST*. The resulting CFSM is the reduced $E1_Q$ shown in Figure 95.

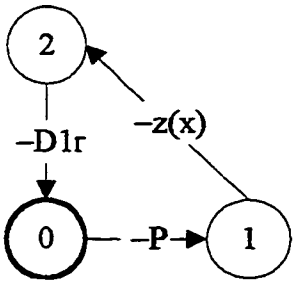

Figure 95: Reduced $E1_Q$ (relabeled)

Step 5 ($E1_Q$): Update the table of significant transitions to reflect the relabeled states of Reduced $E1_Q$.

transition *t*	**sig_pre(*t*)**	**sig_suc(*t*)**
(2, –D1r, 0)	{(1, –z(x), 2)}	{(0, –P, 1)}

Table 8: Significant Transitions of Reduced $E1_Q$

Note: In the remainder of this example, $E0_P$ will refer to the reduced $E0_P$ constructed in this step unless otherwise stated and $E1_Q$ will refer to the reduced $E1_Q$ constructed in this step unless otherwise stated.

6.3.7 Constructing the conversion seed

Given *CS*, $E0_P$, $E1_Q$, the significant transitions and the significant action set *SA*, construct the conversion seed according to the steps in Chapter 5, Section 5.4.2.

Step 1: Construct $(E0_P \times E1_Q \times CS)$~red

First, extend *CS* to include a dead state and the appropriate transitions.

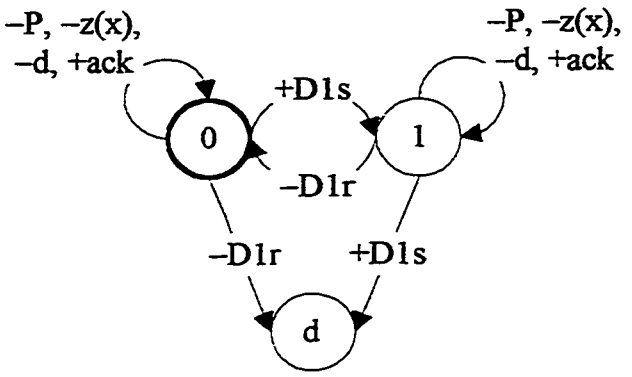

Figure 96: *CS* (extended)

Next, the construct the CFSM $Y = (E0_P \times E1_Q) \times CS$.

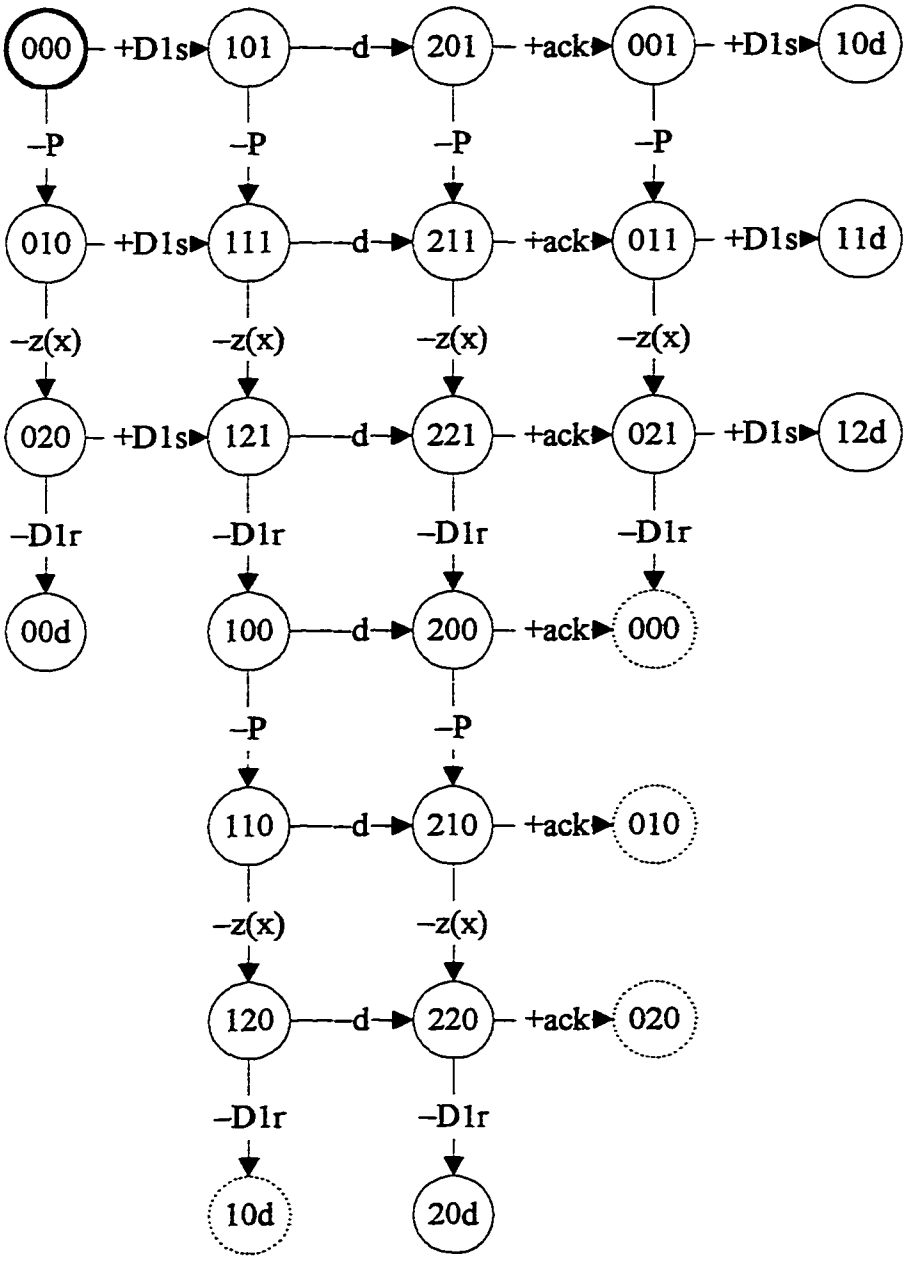

Figure 97: $Y = (E0_P \times E1_Q) \times CS$

The CFSM Y is reduced by explicitly removing transitions that correspond to significant predecessor transitions in $E0_P$ or $E1_Q$. The transitions are "removed" explicitly by redefining the head state to be the non-accepting state *na*. The CFSM in Figure 98 is the result of removing all the significant predecessor transitions of service primitive transitions whose head states are dead states. This CFSM is further reduced by removing all the significant predecessor transitions of service primitive transitions that occur in a path from

a different service primitive transition before the significant successor transition occurs, to obtain $(E0_P \times E1_Q \times CS)$~red (Figure 99). The transitions that have been removed explicitly are shown in **bold** in Figures 98 and 99. States that are unreachable after each step are not shown.

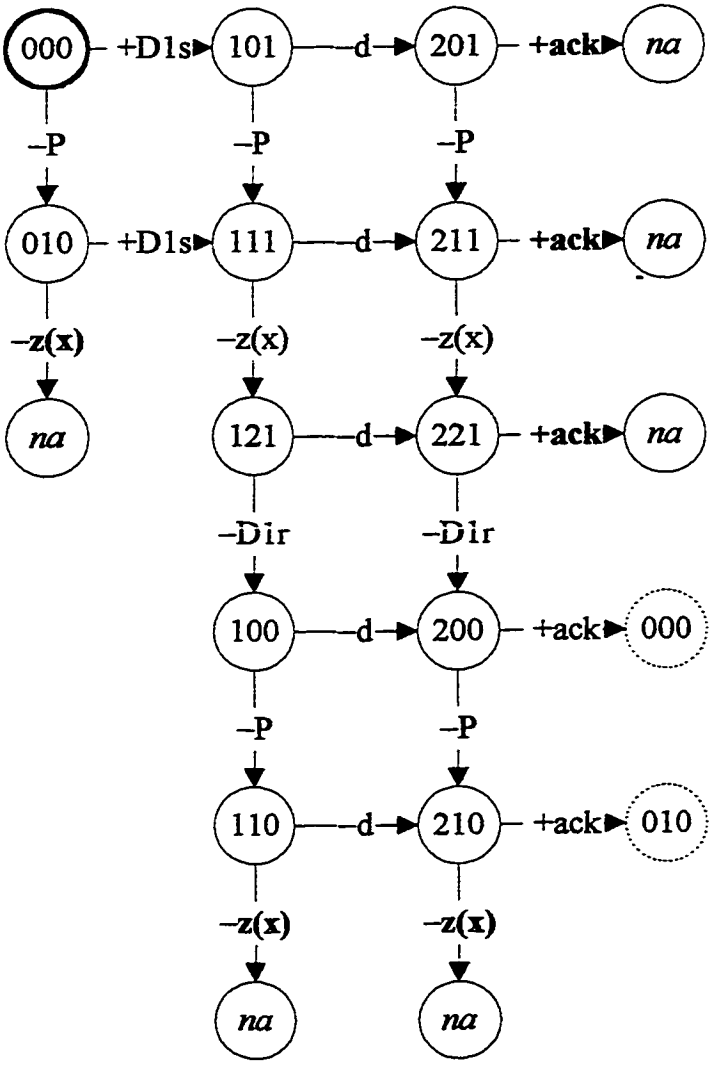

Figure 98: Reduction #1 (reachable states only)

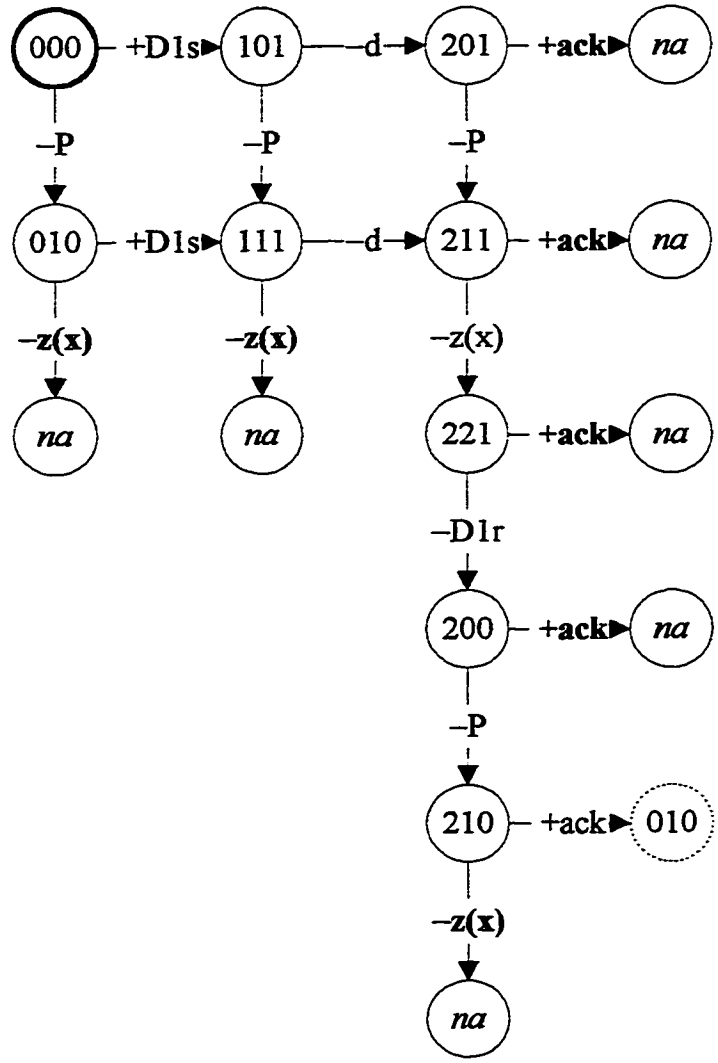

Figure 99: $(E0_P \times E1_Q \times CS)$~red (reachable states only)

Step 2 : Find W such that

$$L(W) = \text{Pre}(L(\text{Proj}_{SA}(E0_P \times E1_Q \times CS)\text{~red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\text{~red})))$$

Figure 100 shows the CFSM $\text{Proj}_{SA}((E0_P \times E1_Q \times CS)$~red). By removing the shaded state and all incoming transitions to the shaded state we obtain W.

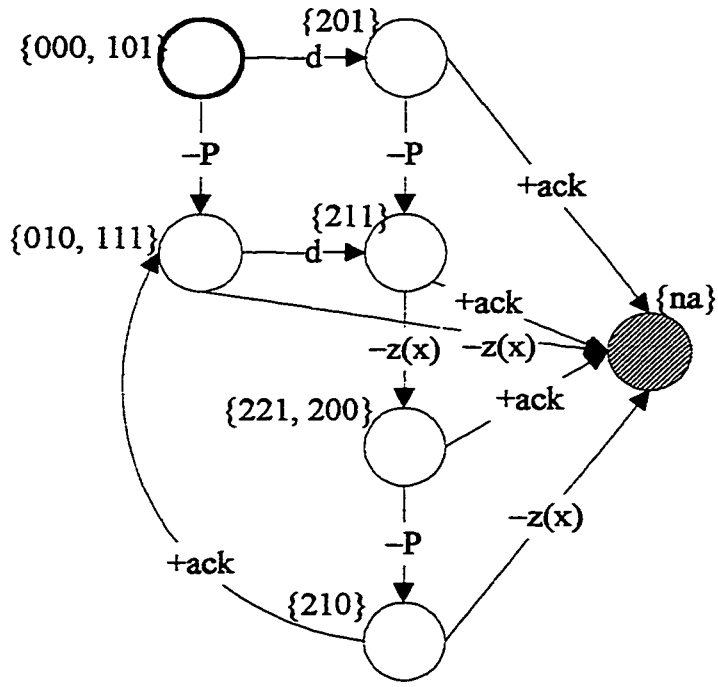

Figure 100: $\text{Proj}_{SA}((E0_P \times E1_Q \times CS)\sim\text{red})$

Step 3 : $X = W^{-1}$ is the conversion seed.

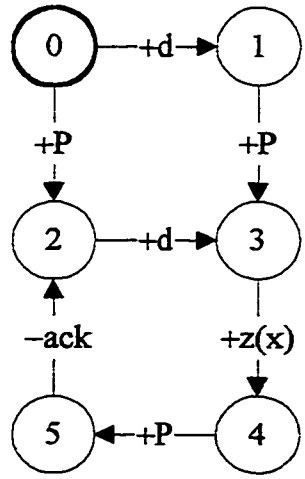

Figure 101: Conversion Seed $X = W^{-1}$ (relabeled and minimized)

6.3.8 Constructing a converter: Okumura's approach

The conversion seed X (Figure 101) can now be used as input to Okumura's approach along with the involved entities $E1_P$ (Figure 72) and $E0_Q$ (Figure 76). First, they must be modified to satisfy the assumptions [Section 6.1] for the construction of a converter using Okumura's algorithm. Specifically, the entities $E1_P$ and $E0_Q$ may not contain service

primitive transitions and the conversion seed X must be extended. The new $E1_P$, the new $E0_Q$ and the new X are shown in Figures 102, 103 and 104, respectively.

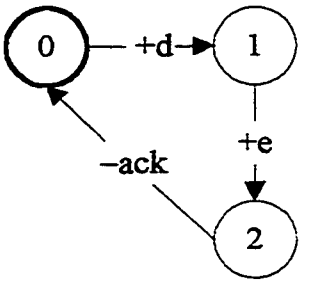

Figure 102: new $E1_P$

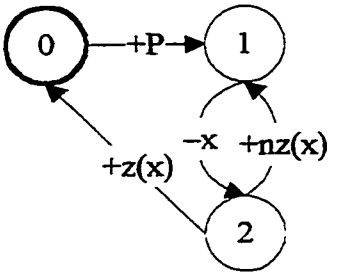

Figure 103: new $E0_Q$

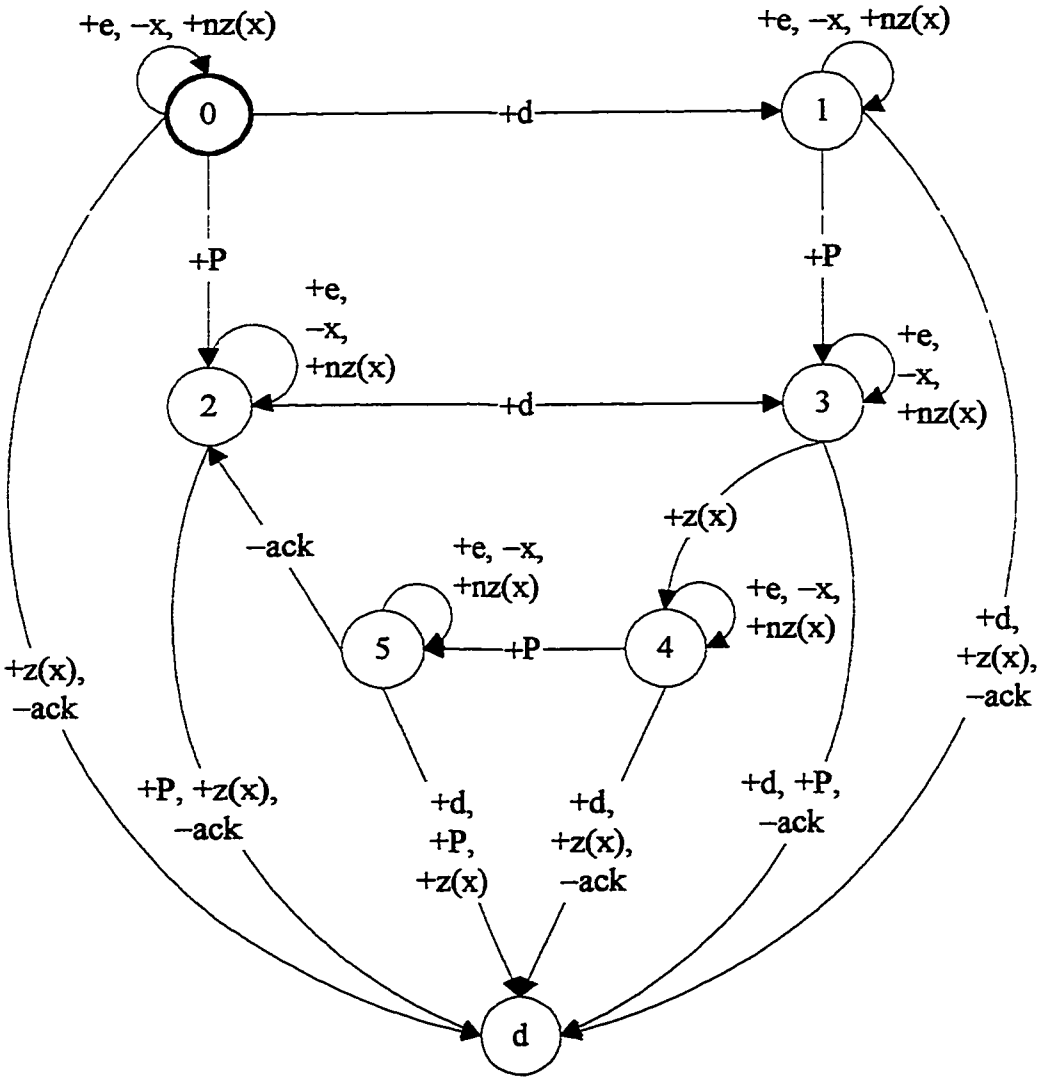

Figure 104: new X (extended)

Note: In the remainder of this example, $E1_P$ will refer to the new $E1_P$ constructed in this step unless otherwise stated, $E0_Q$ will refer to the new $E0_Q$ constructed in this step unless otherwise stated, and X will refer to the new X constructed in this step unless otherwise stated.

The input to Okumura's algorithm is the CFSM $Z = E1_P \times E0_Q \times X$.

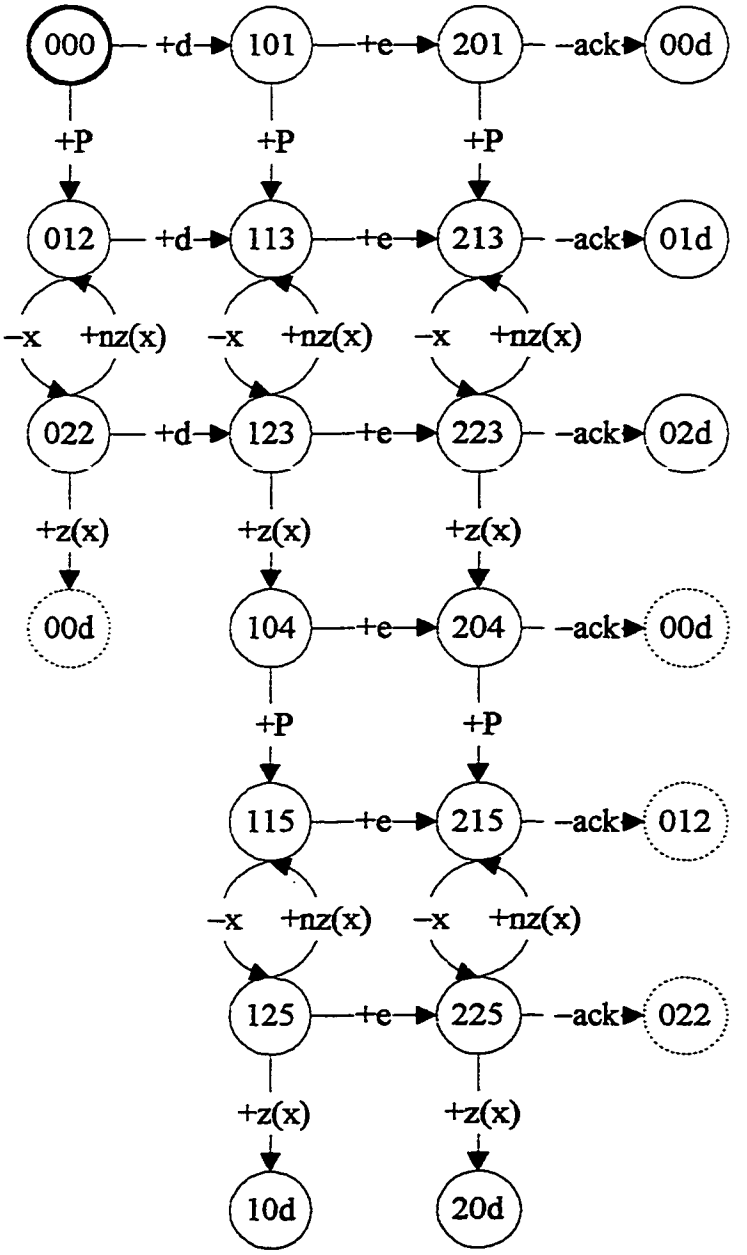

Figure 105: $Z = E1_P \times E0_Q \times X$

Okumura's algorithm finds a sub-CFSM of Z (Figure 105) such that all the states have outgoing transitions and it is a reduced-CFSM of $E1_P \times E0_Q$ (Figure 106), if such a CFSM exists.

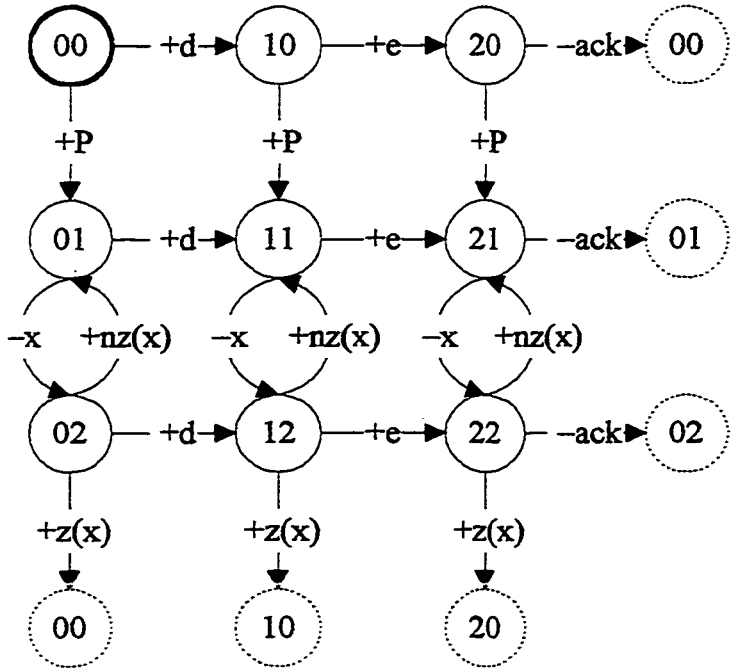

Figure 106: $E1_P \times E0_Q$

Okumura's algorithm works in such a way, for every dead state in $Z = E1_P \times E0_Q \times X$ (Figure 105), all incoming transitions to the state and the state itself are removed. This applies to any state that becomes a dead state as well. Furthermore, if a transition is removed that corresponds to a peer message reception transition of $E1_P$ or $E0_Q$, then its tail state is removed along with all incoming and outgoing transitions. Figure 107 shows $Z = E1_P \times E0_Q \times X$ with the states to be removed by Okumura's algorithm shaded. The resulting sub-CFSM (Figure 108) is the converter. Figure 109 shows the converter with the states relabeled.

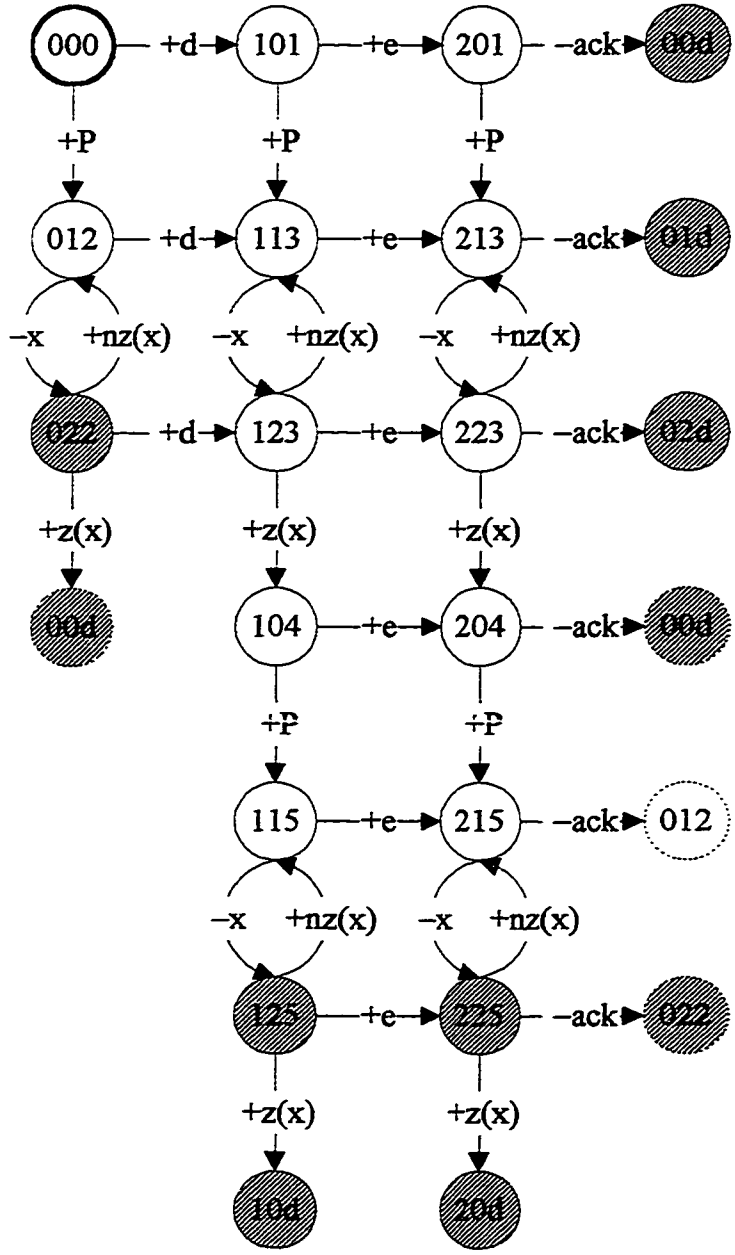

Figure 107: *Z* with states to be removed shaded

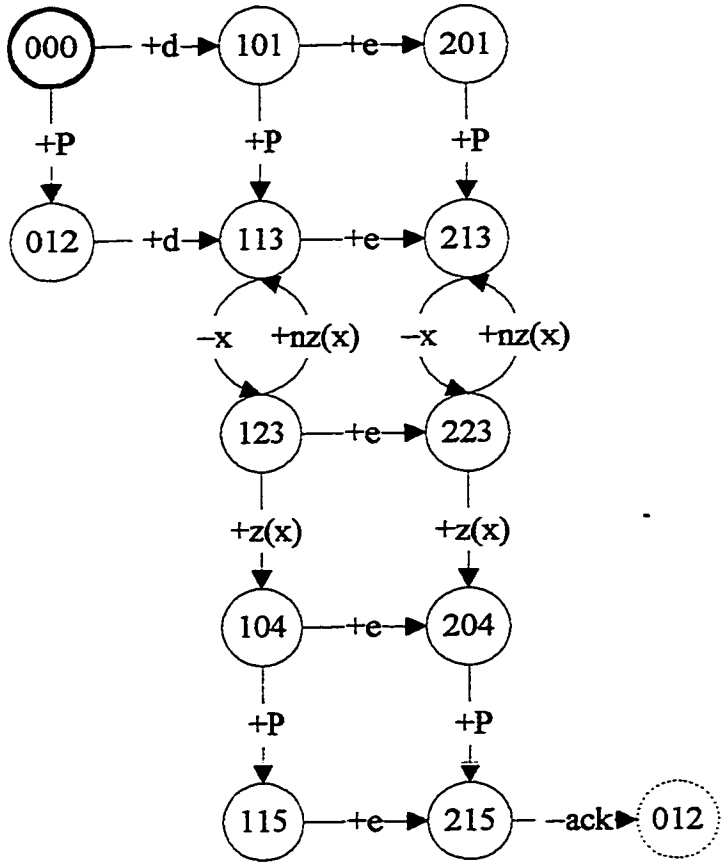

Figure 108: *Z* reduced

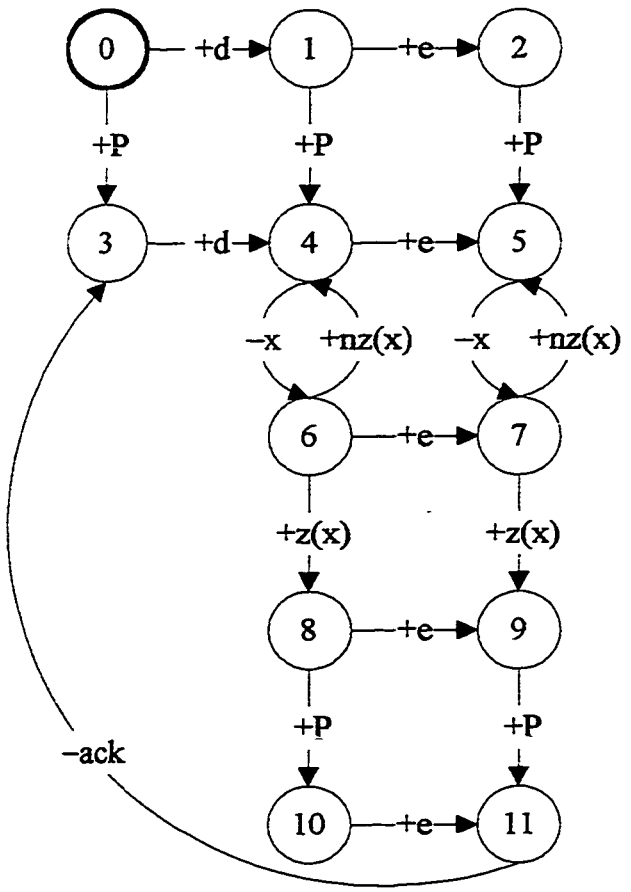

Figure 109: The converter, C (relabeled)

The *original* protocol entities $E0_P$ (Figure 71), and $E1_Q$ (Figure 77), and the converter C (Figure 109), together form a new protocol $\{E0_P, C, E1_Q\}$ that satisfies external equivalence with respect to the protocols P and Q, semantic equivalence with respect to the conversion seed X, and freedom from deadlock and unspecified receptions. Furthermore, the service provided by the new protocol satisfies CS with respect to safety; which, in turn, satisfies RS_U with respect to safety.

Chapter Seven

Application to Conversion Seed Approach: Asynchronous

7.1 More on Chang and Liu's Conversion Seed Approach

In [CL90d] an approach to construct a protocol converter from a conversion seed is presented for a particular class of protocols. The class of protocols are those that can be specified using the CFSM model for asynchronous communication over bounded channels introduced in Chapter 5, Section 5.2. As we saw in Chapter 4, Section 4.1.5, given two distinct protocols $P = \{E0_P, E1_P\}$ and $Q = \{E0_Q, E1_Q\}$ where the communication model for both protocols is asynchronous over bounded channels, Chang and Liu proposes a solution to find a protocol converter C for the conversion system $PQ = \{E0_P, C, E1_Q\}$ such that the conversion system *satisfies* [Section 4.1.5] the conversion seed X. If the protocol entities of protocol P and protocol Q do not contain states with both peer protocol message transmission and peer protocol message reception transitions defined then the failure of the approach to find a protocol converter indicates that a protocol converter that satisfies the conversion seed does not exist for the given protocols. The failure to generate a protocol converter does not imply that a converter does not exist for a different conversion seed.

This approach transforms the problem of finding a protocol converter into a protocol validation problem. The first step is to modify the involved entities $E1_P$ and $E0_Q$, and the conversion seed X, to obtain three new entities $E1_P$-trans, $E0_Q$-trans, and X-trans.

Transformation of involved entities: The entity $E1_P$-trans is the same as $E1_P$ except that any transition labeled by $+m$, where $+m \in \Sigma_X$, is replaced by a sequence of two transitions, the first labeled by $+m$, and the second labeled by $-m'$, connected by a new state; and any

transition labeled by $-m$, where $-m \in \Sigma_X$, is replaced by a sequence of two transitions, the first labeled by $+m'$, and the second labeled by $-m$, connected by a new state. The entity $E0_Q$-trans is constructed from $E0_Q$ in the same manner.

The new states introduced in $E1_P$-trans and $E0_Q$-trans are called **internal states**, and the sequence of transitions labeled by $+m$ and $-m'$, (or $+m'$ and $-m$), are called **coupled transitions**.

Transformation of the Conversion Seed: The entity X-trans is the same as the conversion seed X except that every transition labeled $+m$ is relabeled $+m'$, and every transition labeled $-m$ is relabeled $-m'$.

The three new entities are now viewed as asynchronously communicating entities within a protocol {$E1_P$-trans, X-trans, $E0_Q$-trans}, where async($E1_P$-trans, X-trans) = true, async($E0_Q$-trans, X-trans) = true and async($E1_P$-trans, $E0_Q$-trans) = false. The bounds on the channels are derived from the bounds on the channels in the original protocols: $b_{E1_P\text{-trans},X\text{-trans}} = b_{E0_P,E1_P}$; $b_{X\text{-trans},E1_P\text{-trans}} = b_{E1_P,E0_P}$; $b_{E0_Q\text{-trans},X\text{-trans}} = b_{E1_Q,E0_Q}$, and $b_{X\text{-trans},E0_Q\text{-trans}} = b_{E0_Q,E1_Q}$. The original actions of $E1_P$ and $E0_Q$ are treated as the service primitives of the new protocol.

The next step is to build a CFSM G from the global states and transitions of the protocol {$E1_P$-trans, X-trans, $E0_Q$-trans}:

- The initial state of G is the initial global state of the protocol {$E1_P$-trans, X-trans, $E0_Q$-trans}.
- If G contains the tail state of a global transition labeled by $+m \in \Sigma_X$, then G contains the transition labeled by $+m$, its head state; and one and only one outgoing transition defined at its head state which is labeled by $-m'$ and its head state as well.
- If G contains the tail state of a global transition labeled by $+m'$, $-m \in \Sigma_X$, then G contains the transition labeled by $+m'$, its head state, and one and only one outgoing transition defined at its head state which is labeled by $-m$ and its head state as well.

- If G contains the tail state of a global transition labeled by an insignificant message $\sigma \in ((\Sigma_{E1_P} \cup \Sigma_{E0_Q}) \setminus \Sigma_X)$, and the state does not contain an internal state of $E1_P$-trans or $E0_Q$-trans, then G contains the transition and its head state.
- If G contains the tail state of a global transition labeled by $\sigma \in \Sigma_{X\text{-trans}}$ then G contains the transition and its head state.

The results of [CL90d] are the following theorems and a sketch of an algorithm:

Theorem 7.1: Given that there is a sub-CFSM SG of G that satisfies the following conditions:

1. SG is not empty,
2. SG contains the initial state of G,
3. SG contains no deadlock states, channel overflow states, or unspecified reception states of the protocol $\{E1_P\text{-trans}, X\text{-trans}, E0_Q\text{-trans}\}$.
4. If SG contains the tail state of a transition labeled by a message reception action $+m \in (\Sigma_{E1_P} \cup \Sigma_{E0_Q})$ that is defined in G then SG also contains the transition and its head state.

If original protocols P and Q have no states containing both peer protocol message transmission and peer protocol message reception transitions as its outgoing edges, then there must exist a converter for them with respect to conversion seed X. [CL90d]

Theorem 7.2: Given that there is a protocol converter for protocols P and Q with respect to conversion seed X, there exists a sub-CFSM SG of G satisfying the four conditions of Theorem 7.1. [CL90d]

In [CL90d] Chang and Liu show that the sub-CFSM SG' is a protocol converter for the given protocols P and Q such that $\{E0_P, SG', E1_Q\}$ satisfy the conversion seed X, where the transitions labeled by the new messages added during the transformation of $E1_P$, X, and $E0_Q$ are treated as internal transitions of the entity SG'.

The CFSM SG' can be constructed from SG as follows:

1. For every *transformation* state transition sequence in SG, $(s_1, +m, s_2, -m', s_3, +m', s_4)$, remove all incoming transitions to state s_3 except the transition labeled by $-m'$; and remove all outgoing transitions from state s_3 except the transition labeled by $+m'$.
2. For every *transformation* state transition sequence in SG, $(s_1, -m', s_2, +m', s_3, -m, s_4)$, remove all incoming transitions to state s_2 except the transition labeled by $-m'$; and remove all outgoing transitions from state s_2 except the transition labeled by $+m'$.

The definitions of G and, in turn SG, are sufficient to prove that semantic equivalence and external equivalence are satisfied by $\{E0_P, SG', E1_Q\}$ and that $\{E0_P, SG', E1_Q\}$ will preserve freedom from deadlock and unspecified receptions. The definitions of G, SG and SG' are sufficient to prove that $\{E0_P, SG', E1_Q\}$ will preserve freedom from channel overflow. Finally, the definition of G, the assumption that protocols P and Q have no states containing both peer protocol message transmission and peer protocol message reception transitions as its outgoing edges, and the fact that $\{E0_P, SG', E1_Q\}$ preserves freedom from unspecified receptions and satisfies external equivalence, are sufficient to prove that $\{E0_P, SG', E1_Q\}$ will preserve freedom from deadlock.

SG' is a converter that contains internal transitions. In their sketch of an algorithm, Chang and Liu remove the internal transitions to obtain a smaller converter by replacing each transformation sequence $(s_1, +m, s_2, -m', s_3, +m', s_4)$, or $(s_1, -m', s_2, +m', s_3, -m, s_4)$, by the simple transition, $+m$ or $-m$, respectively.

Assumptions for Chang and Liu's sketch of an algorithm for the construction of the converter are listed below:

1. Protocol $P = \{E0_P, E1_P\}$ is given and furthermore:
   - Protocol P is deadlock free, unspecified reception free and channel overflow free.
   - $E1_P$ contains no states with both peer protocol message transmission and peer protocol message reception transitions defined.

- The internal action transitions and service primitive transitions have been removed from $E1_P$. This can be achieved by projection of $E1_P$ over the subset of Σ_{E1_P} that includes peer protocol messages only. i.e., let $\Delta = \Sigma^{-}_{E1_P} \cup \Sigma^{+}_{E1_P}$ and replace $E1_P$ by $\text{Proj}_{\Delta}(E1_P)$.

2. Protocol $Q = \{E0_Q, E1_Q\}$ is given and furthermore:
   - Protocol Q is deadlock free, unspecified reception free and channel overflow free.
   - $E0_Q$ contains no states with both peer protocol message transmission and peer protocol message reception transitions defined.
   - The internal action transitions and service primitive transitions have been removed from $E0_Q$. This can be achieved by projection of $E0_Q$ over the subset of Σ_{E0_Q} that includes peer protocol messages only. i.e., let $\Delta = \Sigma^{-}_{E0_Q} \cup \Sigma^{+}_{E0_Q}$ and replace $E0_Q$ by $\text{Proj}_{\Delta}(E0_Q)$.
3. A conversion seed X is given.
4. The action sets of $E0_P$ and $E1_Q$ are distinct: $\Sigma_{E0_P} \cap \Sigma_{E1_Q} = \varnothing$.

Sketch of an Algorithm

input: G: CFSM constructed from the global states and transitions of protocol $\{E1_P\text{-trans}, X\text{-trans}, E0_Q\text{-trans}\}$

output: C : CFSM or *nil* if no converter was found

begin

(* construct SG from G *)

Recursively traverse the states of G and remove a state and its incoming and outgoing transitions if it satisfies one of the following conditions, resulting in SG:

1. it is a deadlock state,
2. it is a channel overflow state,
3. it is an unspecified reception state,
4. it had an outgoing transition removed which was labeled by $+m \in (\Sigma_{E1_P} \cup \Sigma_{E0_Q})$,
5. it is unreachable from the initial global state.

(* construct SG' from SG *)
For every state s in SG that satisfies one of the following conditions:
1. s is the exit state of any coupled transitions $+m$ and $-m'$,
2. s is the entry state of any coupled transitions $+m'$ and $-m$,

Do the following to construct SG':
Remove all outgoing transitions from s not labeled by $+m'$.
Remove all incoming transitions to s not labeled by $-m'$.

(* construct C from SG'*)
For every transformation sequence $(s_1, +m, s_2, -m', s_3, +m', s_4)$ in SG':
Replace it with a simple transition $(s_1, +m, s_4)$.
For every transformation sequence $(s_1, -m', s_2, +m', s_3, -m, s_4)$ in SG':
Replace it with a simple transition $(s_1, -m, s_4)$.

If C contains some significant messages then
return C as the converter;
otherwise
return nil;
end.

The complexity of this approach is not discussed in [CL90d]. From the sketch of the algorithm it can be seen that the space and time complexity is $O(|G|)$, i.e., the size of the input global CFSM G. States and transitions are not added to the input so no additional space is required by the algorithm. A recursive algorithm similar to Okumura's algorithm could be implemented to find the sub-CFSM SG of G. As with Okumura's algorithm the time complexity would be in the order of the size of the input CFSM, i.e., $O(|G|)$. To construct SG' from SG, and finally C from SG', would be in the order of the size of the input CFSM, i.e., $O(|SG|)$.

7.2 Example 3

In this section, we provide an example for first constructing a conversion seed according to Approach_2 and then applying Chang and Liu's approach to construct a protocol converter.

This example demonstrates the following procedures for the asynchronous model of communication:

- reducing the input protocols;
- the reduction steps while constructing the conversion seed; and
- the steps of Chang and Liu's approach using the constructed conversion seed.

7.2.1 The given protocols and adapters

Assume that the layers at which the protocol conversion will take place are known: layer M protocol of network X (called protocol P) and layer N protocol of network Y (called protocol Q). The required service for the common upper layer peer protocol U is RS_U. Protocol U is layer M+1 protocol of network X and layer N+1 protocol of network Y. The specification of RS_U is given as a CFSM in Figure 110.

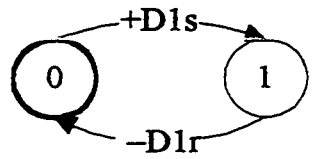

Figure 110: RS_U

The layer M protocol of network X (Figure 111) is protocol P. The peer protocol entities communicate directly where the communication type is asynchronous. The specification of the protocol entities of protocol $P = \{E0_P, E1_P\}$ are given in Figures 113 and 114, where $async_P(E0_P, E1_P) = true$, $b_{E0_P,E1_P} = 2$, and $b_{E1_P,E0_P} = 1$. Protocol P provides the service specified by SS_P (Figure 115). Protocol P operates in the following manner: on receiving the data D1s, the sender of protocol P ($E0_P$) sends it as a data unit d to the receiver ($E1_P$), followed by a message indicating the end of data, e. On receiving the data unit d and the end of data message e the receiver ($E1_P$) sends the data D1r to the user and sends an acknowledgment message, ack, to the sender ($E0_P$). Protocol P is deadlock free, unspecified reception free and channel overflow free. In network X there is no gap between SS_P and RS_U (Figure 112), therefore interface adapters are not needed.

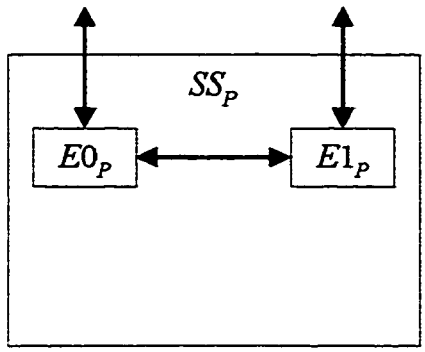

Figure 111: Layer M of network X

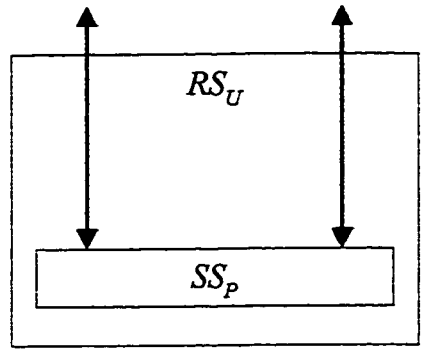

Figure 112: $SS_P = RS_U$ in network X

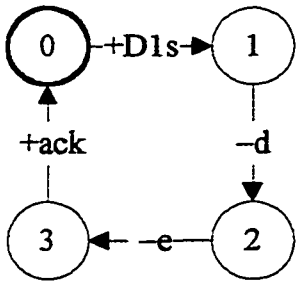

Figure 113: $E0_P$

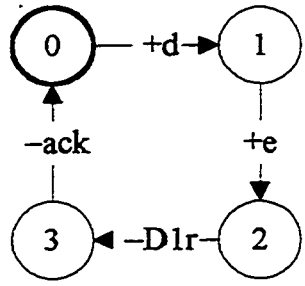

Figure 114: $E1_P$

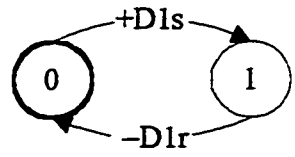

Figure 115: SS_P

The layer N protocol of network Y (Figure 116) is protocol Q. The peer protocol entities communicate directly providing the required service for the protocol where the communication type between the entities is asynchronous. The specification of the protocol entities of protocol $Q = \{E0_Q, E1_Q\}$ are given in Figures 118 and 119, where $\text{async}_Q(E0_Q, E1_Q) = \text{true}$, $b_{E0_Q,E1_Q} = 1$, and $b_{E1_Q,E0_Q} = 2$. Protocol Q provides the service specified by SS_Q (Figure 120). Protocol Q functions as follows: on receiving the data D1s, the sender ($E0_Q$) sends it as a data unit x to the receiver ($E1_Q$). On receiving a data unit, x, the receiver ($E1_Q$) sends either an acknowledgment (z) or a negative acknowledgment (nz) to the sender ($E0_Q$). In the case of a negative acknowledgment, the sender resends the data unit and the receiver waits for the retransmission of the data unit. After a positive acknowledgment the receiver transmits the data, D1r, to the user. Note that the sender always waits for a poll signal from the receiver before sending data unit x to the receiver. Protocol Q is deadlock free, unspecified reception free and channel overflow free. In network Y there is no gap between SS_Q and RS_U (Figure 117), therefore interface adapters are not needed.

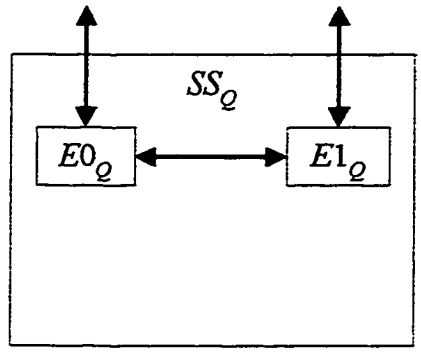

Figure 116: Layer N of network Y

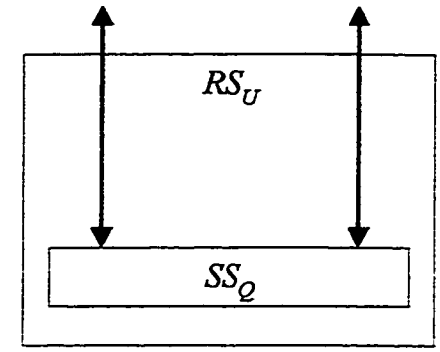

Figure 117: $SS_Q = RS_U$ in network Y

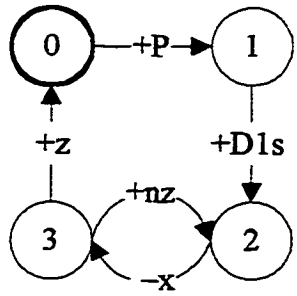

Figure 118: $E0_Q$

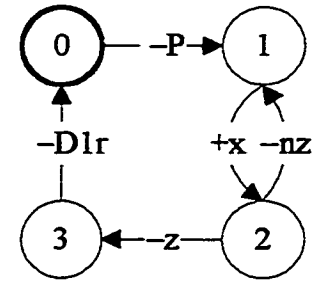

Figure 119: $E1_Q$

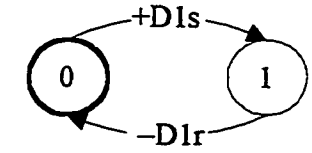

Figure 120: SS_Q

We will find CS, and then C, such that the three entities $E0_P$, C, and $E1_Q$, provide a new protocol that is externally equivalent to Protocol P and Protocol Q, deadlock free, unspecified reception free, and channel overflow free; and whose service specification satisfies CS (Figure 121). Furthermore, the new protocol whose service specification satisfies CS will satisfy the required service (RS_U) for the common peer protocol U (Figure 122).

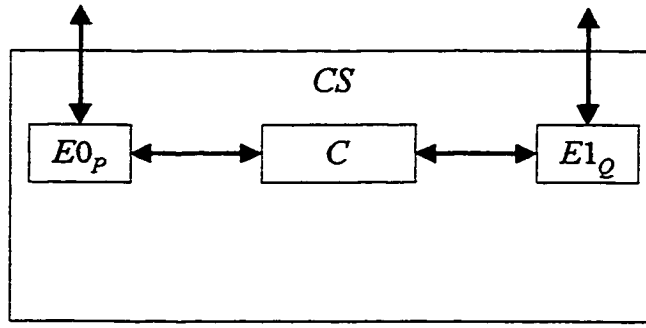

Figure 121: Network Z

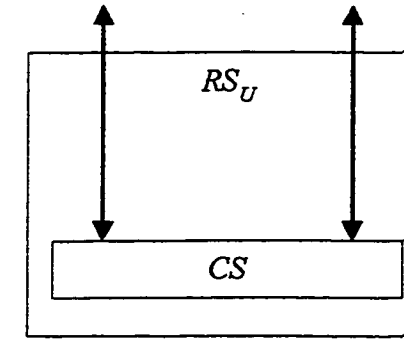

Figure 122: $SS_P = RS_U$ in network X, $SS_Q = RS_U$ in network Y

7.2.2 Direct communication only

Approach_2 assumes the input protocols communicate directly [Chapter 5, Section 5.2.3]. The given protocols satisfy this assumption.

7.2.3 Finding the required service specification for the conversion system

Since no adapters are specified to provide the required service for protocol *U*, this example satisfies case 1 of the problem statement given in the beginning of Section 5.1. According to Chapter 5, Section 5.3.1, the specification for the conversion system *CS* is RS_U.

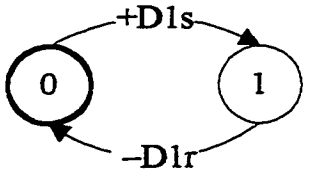

Figure 123: $CS = RS_U$

7.2.4 Pruning the input CFSMs

Protocol pruning is not necessary for this example since $\Sigma_{CS} = \Sigma^U_{E0_P} \cup \Sigma^U_{E1_Q}$.

7.2.5 Significant transitions and the significant action set

At this stage we find the sets of significant transitions for each service primitive transition in $E0_P$ and $E1_Q$, and construct the significant action set according to Chapter 5, Section 5.4.1. The significant action set is *SA* = {+ack, +x, –d, –P}. The significant transitions of $E0_P$ are given in Table 9. Table 10 provides the significant transitions of $E1_Q$.

transition *t*	sig_pre(*t*)	sig_suc(*t*)
(0, +D1s, 1)	{(3, +ack, 0)}	{(1, –d, 2)}

Table 9: Significant Transitions of $E0_P$

transition *t*	sig_pre(*t*)	sig_suc(*t*)
(3, –D1r, 0)	{(1, +x, 2)}	{(0, –P, 1)}

Table 10: Significant Transitions of $E1_Q$

7.2.6 Reducing the input CFSMs

Given the significant transitions and the significant action set, the entities $E0_P$ and $E1_Q$ may be reduced by merging states according to the steps in Chapter 5, Section 5.5.1. In this case, both $E0_P$ and $E1_Q$ have states that can be merged.

Δ = {+ack, +x, –d, –P, +D1s, –D1r}.

ST = {(3, +ack, 0), (1, –d, 2), (1, +x, 2), (0, –P, 1)}

Step 1 ($E0_P$): Reduced $E0_P = E0_P$

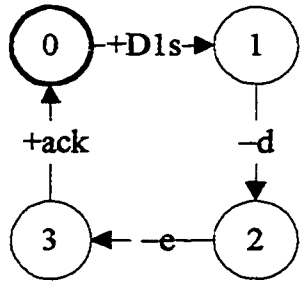

Figure 124: Step 1 $E0_P$

Step 2 ($E0_P$): Find a pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in ST. Such a pair of states is (2, 3).

Step 3 ($E0_P$): M1 and M2 are true for (2, 3) so they can be merged.

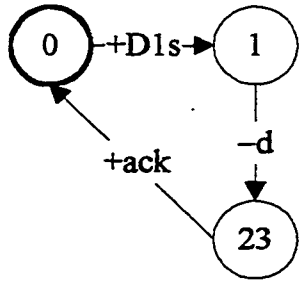

Figure 125: Step 3 $E0_P$ (merge #1)

Step 4 ($E0_P$): There is no pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in ST. The resulting CFSM is the reduced $E0_P$ in Figure 126.

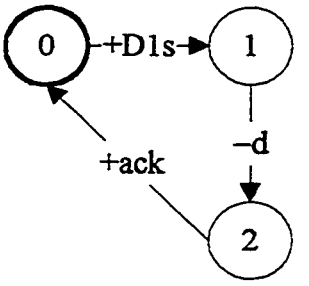

Figure 126: Reduced $E0_P$ (relabeled)

Step 5 ($E0_P$): Update the table of significant transitions to reflect the relabeled states of Reduced $E0_P$.

transition t	sig_pre(t)	sig_suc(t)
(0, +D1s, 1)	{(2, +ack, 0)}	{(1, –d, 2)}

Table 11: Significant Transitions of Reduced $E0_P$

Step 1 ($E1_Q$): Reduced $E1_Q = E1_Q$

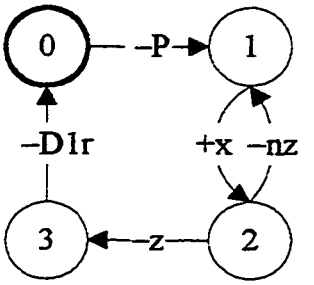

Figure 127: Step 1 $E1_Q$

Step 2 ($E1_Q$): Find a pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in *ST*. Such a pair of states is (2, 3).

Step 3 ($E1_Q$): M1 is true for (2, 3) so they can be merged.

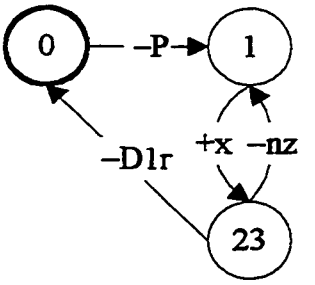

Figure 128: Step 3 $E1_Q$ (merge #1)

Step 4 ($E1_Q$): There is no pair of adjacent states such that there exists a transition between them labeled by an action not in Δ and there does not exist a transition between them in ST. The resulting CFSM is the reduced $E1_Q$ shown in Figure 129.

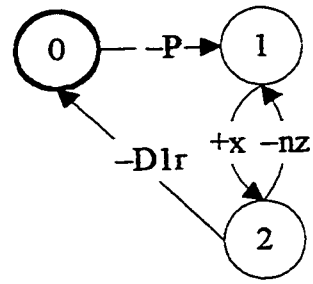

Figure 129: Reduced $E1_Q$ (relabeled)

Step 5 ($E1_Q$): Update the table of significant transitions to reflect the relabeled states of Reduced $E1_Q$.

transition t	sig_pre(t)	sig_suc(t)
(2, -D1r, 0)	{(1, +x, 2)}	{(0, -P, 1)}

Table 12: Significant Transitions of Reduced $E1_Q$

Note: In the remainder of this example, $E0_P$ will refer to the reduced $E0_P$ constructed in this step unless otherwise stated and $E1_Q$ will refer to the reduced $E1_Q$ constructed in this step unless otherwise stated.

7.2.7 Constructing the conversion seed

Given CS, $E0_P$, $E1_Q$, the significant transitions and the significant action set SA, construct the conversion seed according to the steps in Chapter 5, Section 5.4.2.

Step 1: Construct $(E0_P \times E1_Q \times CS)$~red

First, extend CS to include a dead state and the appropriate transitions.

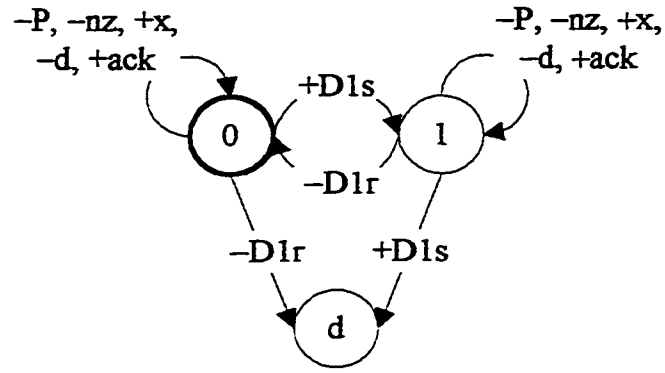

Figure 130: *CS* (extended)

Next, the construct the CFSM $Y = (E0_P \times E1_Q) \times CS$.

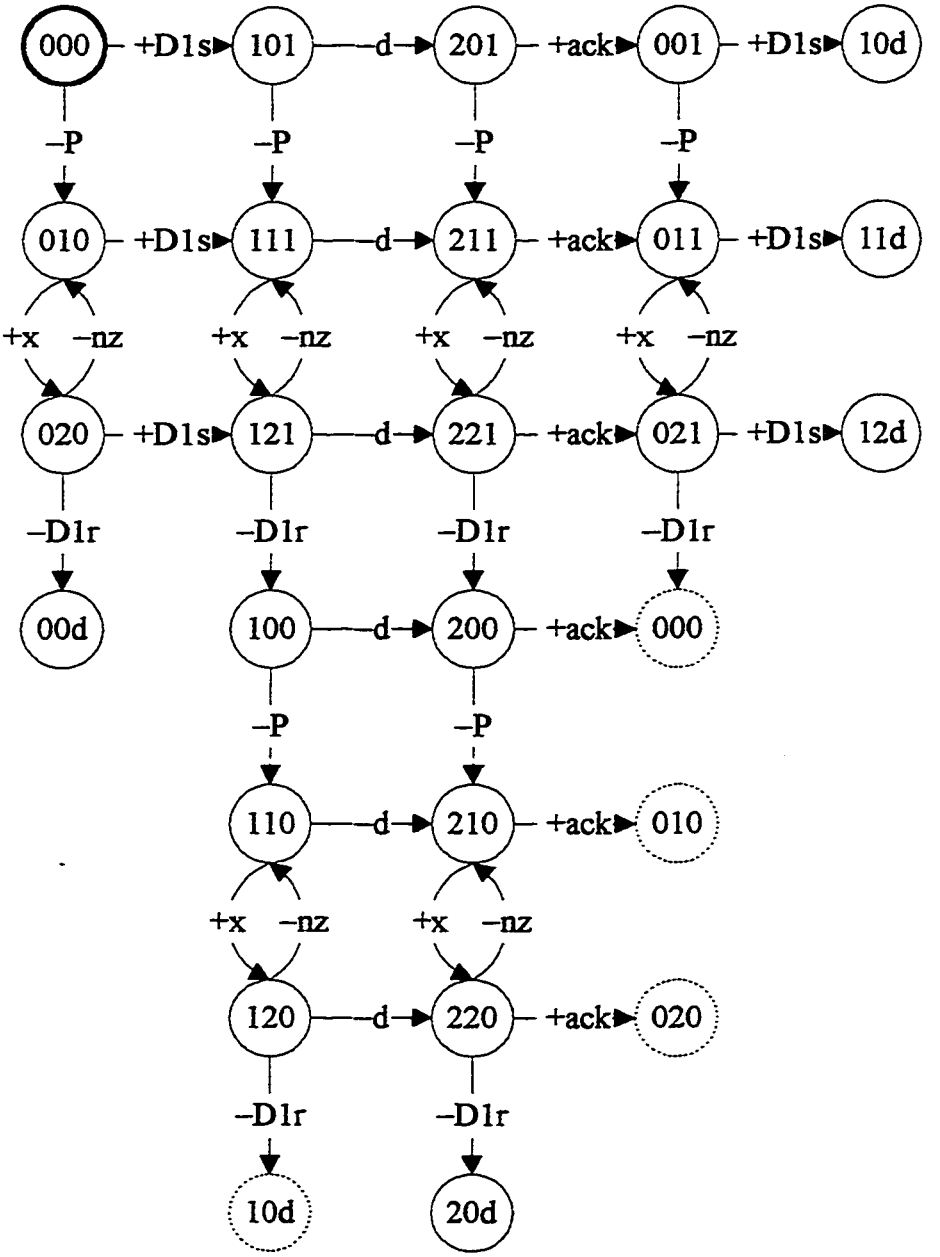

Figure 131: $Y = (E0_P \times E1_Q) \times CS$

The CFSM Y is reduced by explicitly removing transitions that correspond to significant predecessor transitions in $E0_P$ or $E1_Q$. The transitions that are "removed" explicitly by redefining the head state to be the non-accepting state *na*. The CFSM in Figure 132 is the result of removing all the significant predecessor transitions of service primitive transitions whose head states are dead states. This CFSM is further reduced by removing all the significant predecessor transitions of service primitive transitions that occur in a path from

a different service primitive transition before the significant successor transition occurs, to obtain $(E0_P \times E1_Q \times CS)$~red (Figure 133). The transitions that have been removed explicitly are shown in **bold** in Figures 132 and 133. States that are unreachable after each step are not shown.

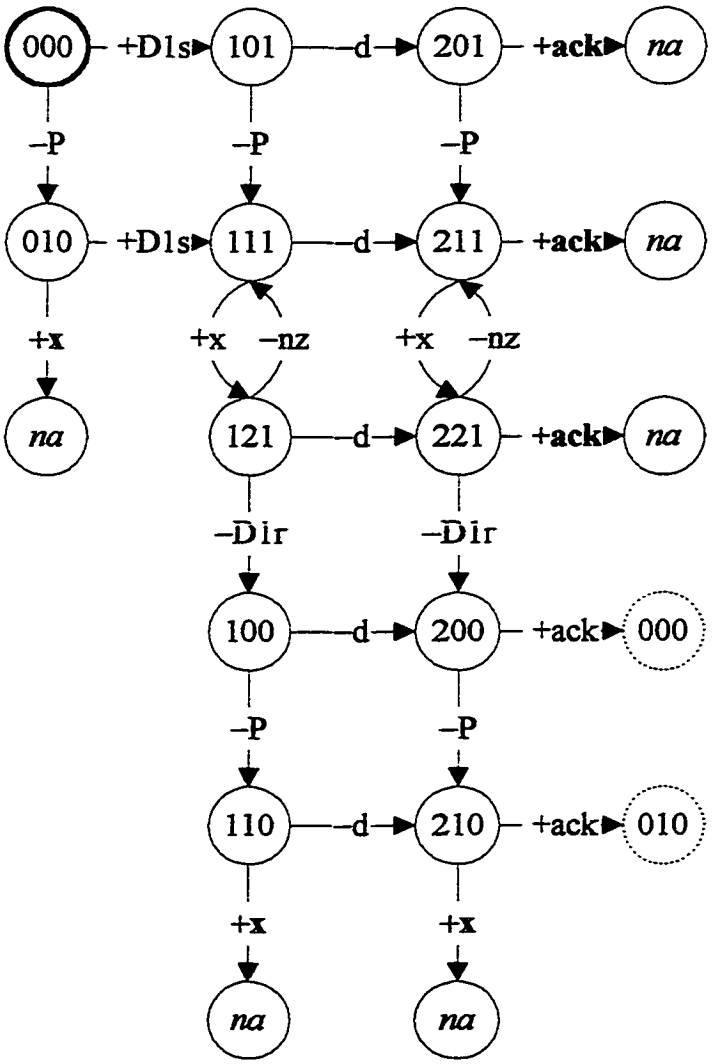

Figure 132: Reduction #1 (reachable states only)

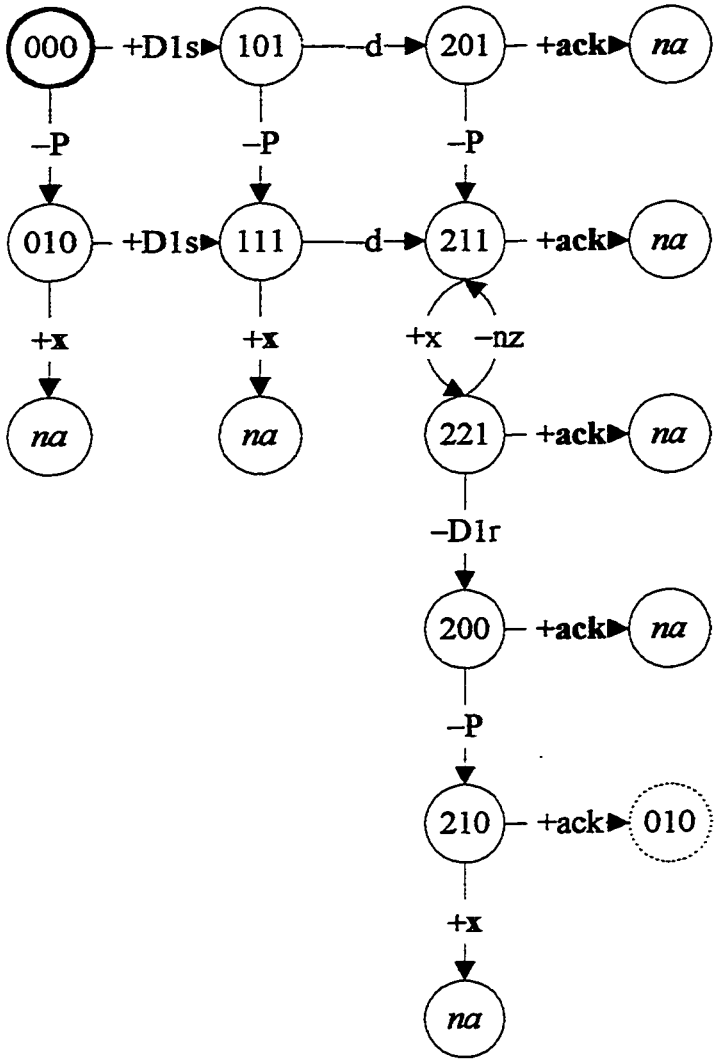

Figure 133: $(E0_P \times E1_Q \times CS)$~red (reachable states only)

Step 2: Find W such that

$$L(W) = \text{Pre}(L(\text{Proj}_{SA}(E0_P \times E1_Q \times CS)\text{~red})) \setminus L(\text{Proj}_{SA}(\neg(E0_P \times E1_Q \times CS)\text{~red})))$$

Figure 134 shows the CFSM $\text{Proj}_{SA}((E0_P \times E1_Q \times CS)$~red). By removing the shaded state and all incoming transitions to the shaded state we obtain W.

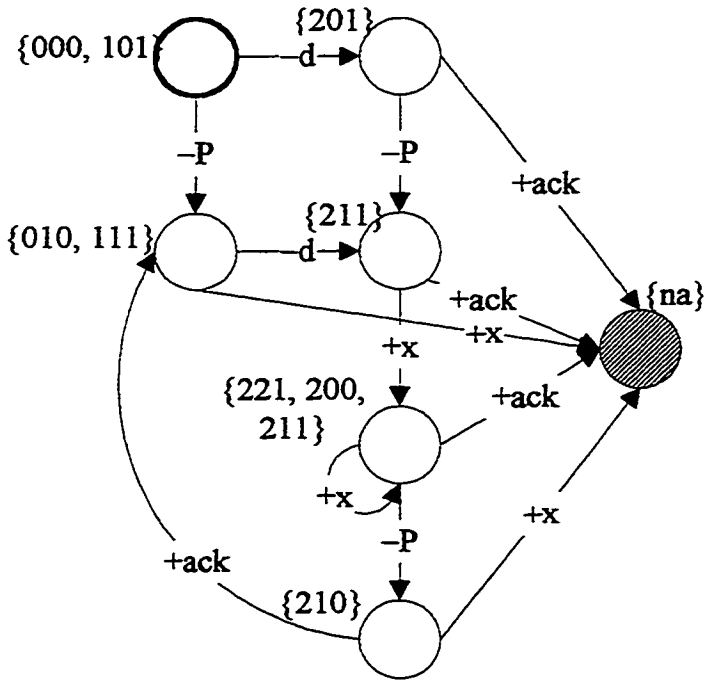

Figure 134: $\text{Proj}_{SA}((E0_P \times E1_Q \times CS)\sim\text{red})$

Step 3: $X = W^{-1}$ is the conversion seed.

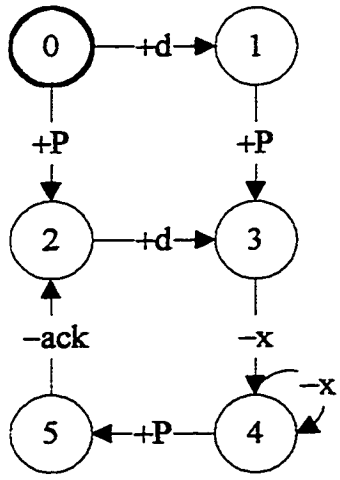

Figure 135: Conversion Seed $X = W^{-1}$ (relabeled and minimized)

7.2.8 Constructing a converter: Chang and Liu's approach

The conversion seed X (Figure 135) can now be used as input to Chang and Liu's approach along with the involved entities $E1_P$ (Figure 114) and $E0_Q$ (Figure 118). First, they must be modified to satisfy the assumptions [Section 7.1] for the construction of a converter using Chang and Liu's approach. Specifically, the entities $E1_P$ and $E0_Q$ may not

contain service primitive transitions. The new $E1_P$ and the new $E0_Q$ are shown in Figures 136 and 137, respectively.

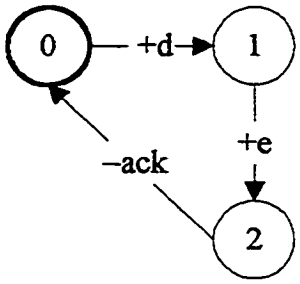

Figure 136: new $E1_P$

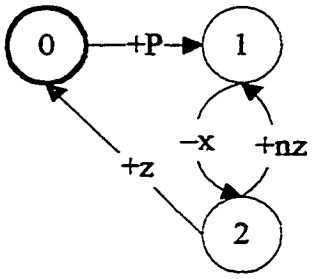

Figure 137: new $E0_Q$

Note: In the remainder of this example, $E1_P$ will refer to the new $E1_P$ constructed in this step unless otherwise stated and $E0_Q$ will refer to the new $E0_Q$ constructed in this step unless otherwise stated.

The first step in Chang and Liu's approach is to modify the involved entities $E1_P$ and $E0_Q$, and the conversion seed X, to obtain three new entities $E1_P$-trans (Figure 138), $E0_Q$-trans (Figure 139) and X-trans (Figure 140). The transformation steps are given in Section 7.1.

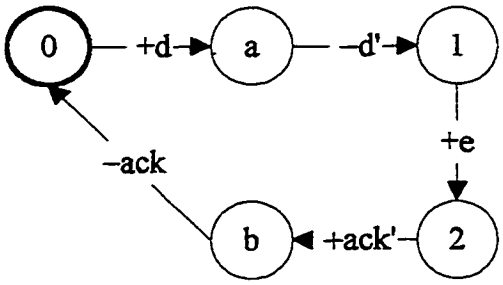

Figure 138: $E1_P$-trans

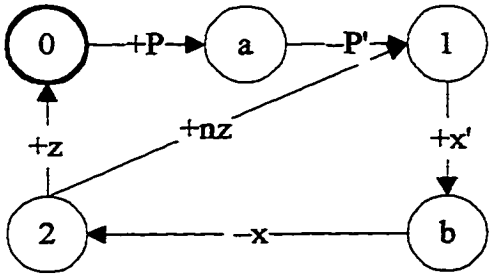

Figure 139: $E0_Q$-trans

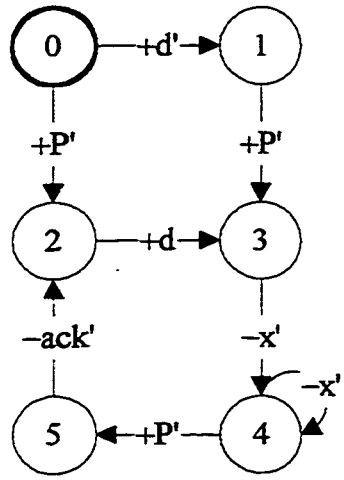

Figure 140: *X*-trans

The three new entities are now viewed as communicating entities within a protocol {$E1_P$-trans, *X*-trans, $E0_Q$-trans}, where async($E1_P$-trans, *X*-trans) = true, async($E0_Q$-trans, *X*-trans) = true and async($E1_P$-trans, $E0_Q$-trans) = false; and $b_{E1_P\text{-trans},X\text{-trans}} = b_{E0_P,E1_P}$; $b_{X\text{-trans},E1_P\text{-trans}} = b_{E1_P,E0_P}$; $b_{E0_Q\text{-trans},X\text{-trans}} = b_{E1_Q,E0_Q}$, and $b_{X\text{-trans},E0_Q\text{-trans}} = b_{E0_Q,E1_Q}$. The original actions of $E1_P$ and $E0_Q$ are treated as the service primitives of the new protocol.

The next step is to build a CFSM *G* from the global states and transitions of the protocol {$E1_P$-trans, *X*-trans, $E0_Q$-trans}. CFSM *G* is input to Chang and Liu's algorithm. In the sketch of their algorithm, the states of *G* are recursively traversed and a state and its incoming and outgoing transitions are removed if the state (1) is a deadlock state of {$E1_P$-trans, *X*-trans, $E0_Q$-trans}, or (2) is a channel overflow state of {$E1_P$-trans, *X*-trans, $E0_Q$-trans}, or (3) is an unspecified reception state of {$E1_P$-trans, *X*-trans, $E0_Q$-trans}, or (4) had an outgoing transition removed by this step which was labeled by $+m \in (\Sigma_{E1_P} \cup \Sigma_{E0_Q})$, or (5) is an unreachable state from the initial state. The result of this reduction is a CFSM *SG* (Figure 141).

The CFSM *SG* is reduced by removing all transitions that potentially break a sequence of transitions that together form a transformation sequence. The reduced *SG* is shown in Figure 142.

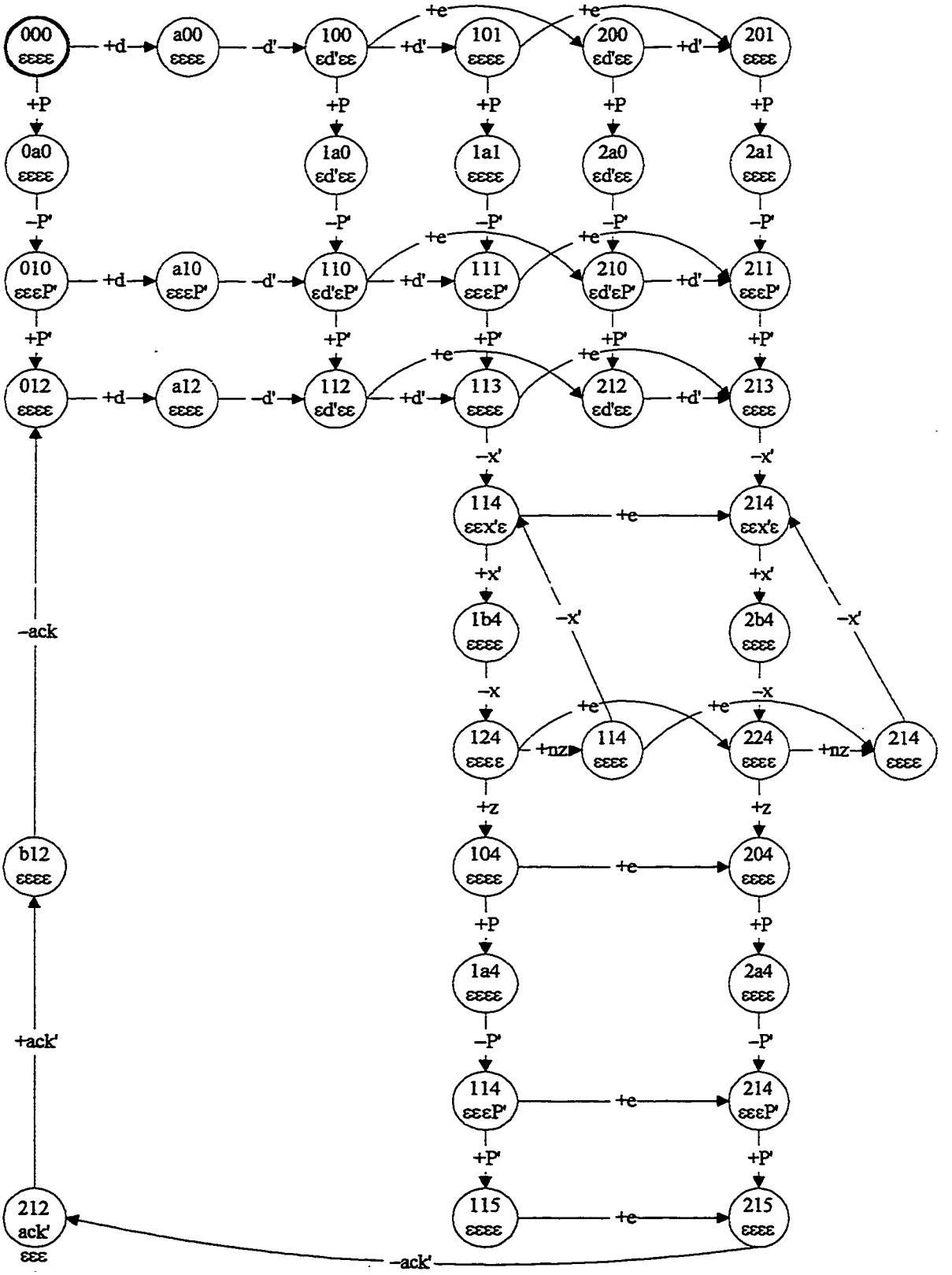

Figure 141: *SG*

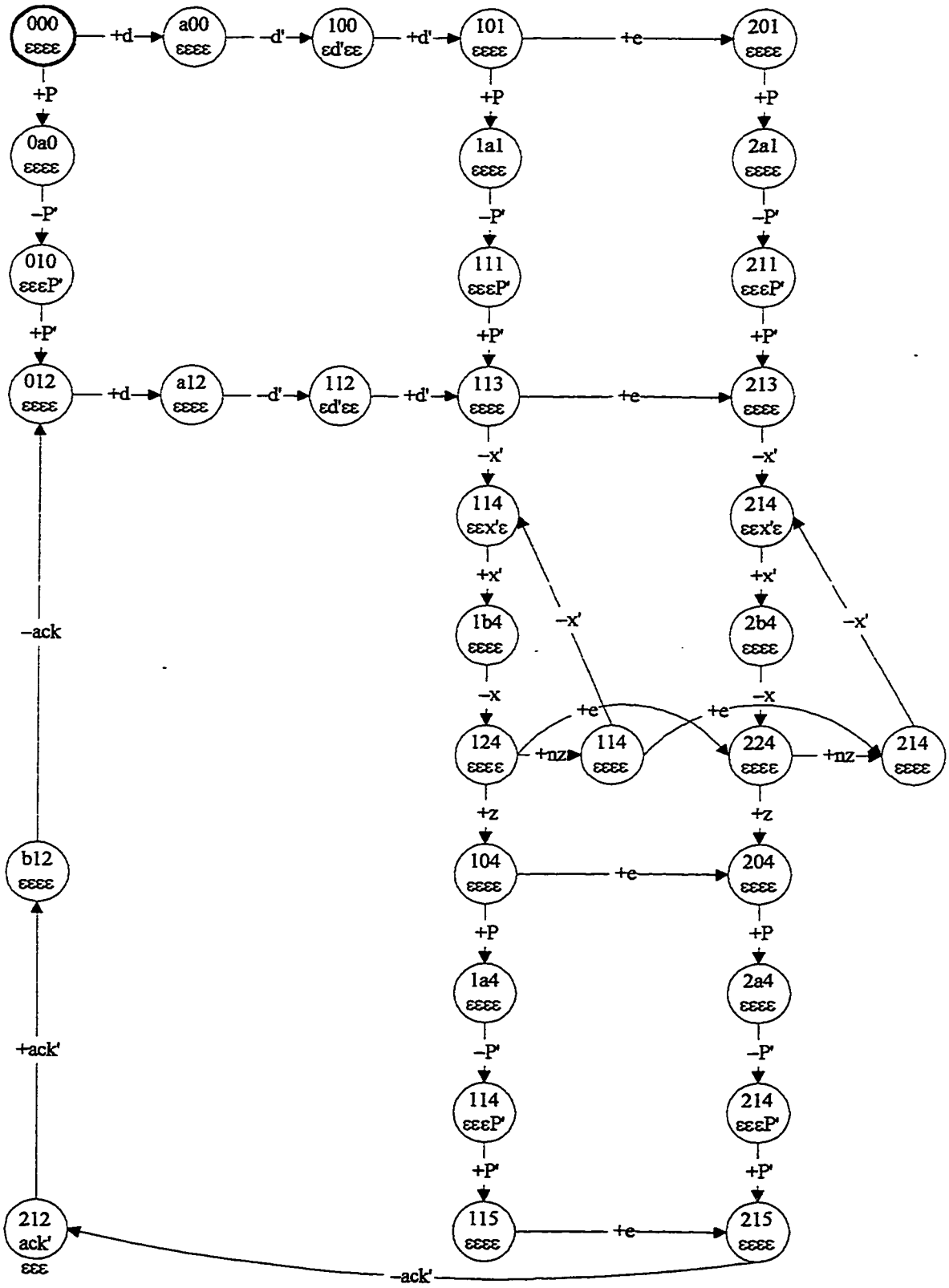

Figure 142: Reduced *SG*

Finally, the converter C is constructed from the reduced SG by replacing the transformation sequences with a simple transition (Figure 143).

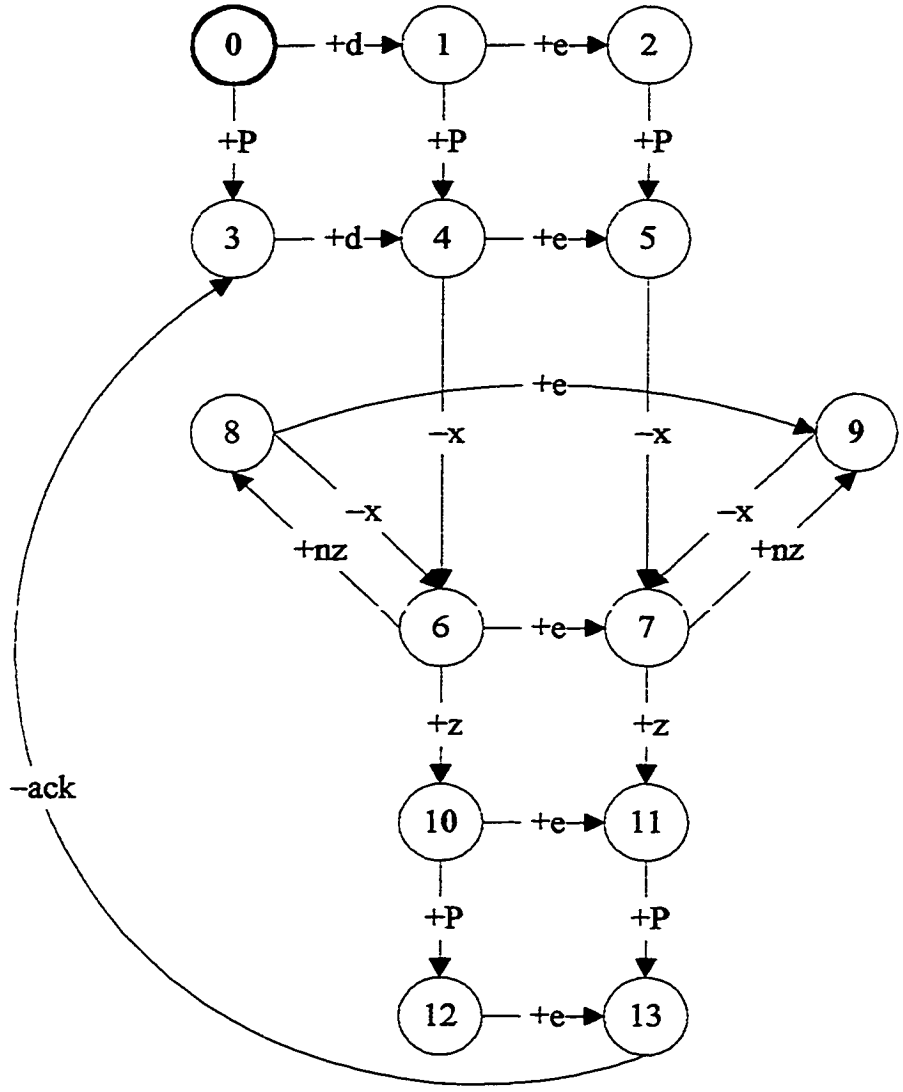

Figure 143: The converter, C (relabeled)

The *original* protocol entities $E0_P$ (Figure 113), and $E1_Q$ (Figure 119), and the converter C (Figure 143), together form a new protocol $\{E0_P, C, E1_Q\}$ that satisfies external equivalence with respect to the protocols P and Q, semantic equivalence with respect to the conversion seed X, and freedom from deadlock, unspecified receptions and channel overflow. Furthermore, the service provided by the new protocol satisfies CS with respect to safety; which, in turn, satisfies RS_U with respect to safety.

Chapter Eight

Conclusions

8.1 Final Observations

Heterogeneous network interconnection at a gateway involves solving the protocol conversion problem. Two different approaches to protocol conversion can be considered: conversion at the service level (via a service interface adapter) or conversion at the protocol level (via a protocol converter).

The easiest approach to heterogeneous network interconnection at a gateway is to interconnect the common service boundary with a service interface adapter (*SIA*) since it is not concerned with the details of the original protocols. The advantage is that existing protocol implementations can be used within the gateway. However, the service level interconnection at a gateway raises valid concerns about the end-to-end significance of the original protocols, the concatenation of reliable and unreliable services, etc., since not all the protocol functions will surface at the service level and this may be relevant to the required service. Each protocol will terminate at the gateway thus any end-to-end synchronization capability of the existing services will be lost.

A more complex but more efficient approach to interconnection at a gateway requires replacing the conversion layers in the gateway with a converter that maps one protocol to the other protocol and vice-versa. The converter function is defined explicitly in terms of the peer protocol messages exchanged within the two interconnected networks according to their respective protocols. Protocol level conversion attempts to provide conversions for all the protocol level common functionality. This can make the converter completely transparent in some cases. Maintenance of the end-to-end semantics with a peer protocol

converter is essential, at least within the common subset of functionality required by the common upper layer protocol. Protocol level interconnection overcomes some of the problems with service level interconnection because it enables end-to-end synchronization at the conversion level.

We have examined the family of formal methods for protocol converter synthesis based on the CFSM model [BZ83] that is widely used to specify communication protocols. From our examination we were able to formulate the protocol conversion problem in a more general framework that considers both direct communication (synchronous or asynchronous over simplex channels) and indirect communication (via a communication medium also modeled by a CFSM).

A formal method for protocol conversion requires that the conversion requirements be defined formally as well. Protocol-level conversion requirements specify the translation requirements and the synchronization requirements of the protocol converter in terms of the peer protocol messages of the two different protocols. The service-level conversion requirements specify the service requirements in terms of the service primitives of the conversion system. The approaches proposed in the literature synthesize protocol converters that satisfy either protocol-level conversion requirements or service-level conversion requirements, but not both.

Service-level conversion requirements are typically specified in the form of a required service specification of the conversion system. Approaches that are based on the required service specification of the conversion system synthesize a protocol converter C such that the resulting conversion system satisfies the required service specification with respect to safety, and in some cases, progress. Based on our formulation of the protocol conversion problem, we can synthesize a required service specification CS using the submodule construction approach. The submodule construction approach will find CS such that CS, along with the existing adapters, satisfies the required service of the common upper layer protocol, RS_U, with respect to safety. Furthermore, L(CS) is maximal with this property. Therefore, CS captures the safety requirements. Progress requirements are specified using non-determinism in a required service specification. CS is always deterministic and does

not capture the progress requirements. If *CS* is input as the conversion requirements for approaches which include safety and progress [CL89a][CL89b][KH93][TBD95] in their correctness definition, then only the safety phase of the approach should be applied. Failure to satisfy the progress phase indicates failure to satisfy the conformity property, which may not be desired. Other service-level approaches which use *CS* to derive a service interface adapter [Boc90][Oku90][YL92] and then go on to find a protocol converter are only concerned with the safety property and thus can use *CS* as the conversion requirements.

Of the various forms of protocol-level conversion requirements proposed in the literature, the conversion seed [Oku86][CL90d] is the most expressive in specifying the translation requirements and synchronization requirements of the converter. Approaches based on protocol-level requirements are ideal because they ensure that the good properties of the original protocols such as freedom from deadlock, freedom from unspecified receptions and freedom from channel overflow are inherited by the conversion system as a side effect of the synthesis approach. Furthermore, if protocol-level requirements exist with respect to the interconnection of functions then there is a means to provide them as input to the synthesis approach. Unfortunately, where service-level requirements are known, it is difficult to specify them in terms of the peer protocol messages of the protocol-level requirements. We have proven that given *CS* and the original protocols, if the conversion system satisfies external equivalence with respect to the original protocols and semantic equivalence with respect to the conversion seed generated using our approach from *CS* and the participating entities, then the conversion system also satisfies *CS* with respect to safety.

As part of this work, we have presented the protocol conversion problem in a more general framework. This provided a more realistic starting point for the protocol conversion problem. Instead of needing specific conversion requirements in addition to the specifications of entities in the given protocol stacks, the approach we took in this thesis only requires the specification of the required service for the common upper layer protocol in addition to the given protocols and any existing adapters. That is, our approach consists

of the synthesis of CS, the synthesis of X, and finally using an existing conversion seed based approach, the synthesis of the protocol converter C. Because our approach is based on the existing conversion seed approaches, it also provides the advantage of being applicable to the synchronous asymmetric model of communication and the asynchronous model of communication. Furthermore, the conversion system found by our approach inherits all the desired protocol properties from the original protocols: freedom from deadlock, freedom from unspecified receptions, freedom from channel overflow and optimality. Moreover, if successful, our approach finds a protocol converter that satisfies safety (RS_U), external equivalence (original protocols), and inherits freedom from deadlock, freedom from unspecified receptions, freedom from channel overflow and optimality. Therefore, the service level requirements are satisfied along with the desired protocol properties.

The success of our approach depends on the existence of a conversion seed. We did not explore the implication of the failure to construct a conversion seed on the existence of a protocol converter. As with the SIA in [Oku90], the failure to find a conversion seed may not imply the non-existence of a protocol converter.

8.2 Summary of Contributions

Below we list the major contributions of this thesis:

- In Chapters 2, 3 and 4 we have provided our formulation of the protocol conversion problem, and a survey and analysis of the current literature on protocol conversion, restricted to the CFSM model for protocol specification.
- In Chapter 5, Section 5.3, a formal approach to synthesize the required service specification for the conversion system is presented which makes use of the submodule construction approach. This approach allows for the maximum safe solution. The required service specification can then be used as the service-level requirements for safety in many of the formal methods for protocol conversion.

- In Chapter 5, Section 5.4, a formal approach to synthesize a conversion seed which captures the service-level conversion requirements for safety as specified in a required service specification for the conversion system is presented. The approach can be applied to both the synchronous model of communication and the asynchronous model of communication. We prove that given *CS* and the original protocols, if the conversion system satisfies external equivalence with respect to the original protocols and semantic equivalence with respect to the conversion seed generated using our approach, then the conversion system also satisfies *CS* with respect to safety.
- In Chapter 5, Section 5.5 we provide an improvement to the formal approach presented in Section 5.4 to synthesize a conversion seed. The improvement is an efficient reduction technique that can reduce the number of states and transitions in order to reduce the state space explosion that occurs when finding the product of CFSMs. We prove that the conversion seed found when this extension is applied is the same as the conversion seed found without using this extension.
- In Chapter 5, Section 5.6, we provide a second improvement to the formal approach presented in Section 5.4 to synthesize a conversion seed. The improvement uses the efficient protocol pruning technique [LNS95][KLNS93] to remove those parts of the input CFSMs that do not contribute to the service to be provided.
- Finally, when we consider the synthesis of *CS*, the synthesis of *X*, and using an existing conversion seed based approach, the synthesis of *C*, we have a formal approach for constructing a protocol converter. To our knowledge this is the first approach not based on the existence of an *SIA*, that addresses the safety property for the asynchronous model of communication.

8.3 Directions for Future Research

It would be interesting to see this work improved and/or extended in the following directions:

- If successful, our approach finds a protocol converter C such that the conversion system satisfies safety (RS_U), external equivalence (original protocols), and inherits freedom from deadlock, freedom from unspecified receptions, freedom from channel overflow and optimality. Is L(C) maximal with this property?
- Our approach may be unsuccessful in finding a conversion seed for the given protocols P and Q, and the derived service specification CS. As well, our approach may be unsuccessful in finding a converter for the derived conversion seed X. Does the failure of our approach to synthesize a correct converter C imply that there is no such converter?
- This work can be used as the starting point for the development of a hybrid approach to synthesizing protocol requirements. By combining the derived protocol-level requirements (generated from the service specification) with known protocol-level requirements (in some form) that take into account the protocol level considerations, one could synthesize a conversion seed that captures both. Then existing methods could be used to synthesize a converter that satisfies both the service specification and the protocol-level requirements. Alternatively, one could determine that both the protocol-level requirements and service-level requirements cannot be satisfied.
- The approach to synthesize a conversion seed can be applied to a subset of the protocols specified using the asynchronous model of communication. The restrictions imposed on asynchronously communicating protocols are stated in terms of the states and transitions of the protocol entities themselves. It would be interesting to determine how these restrictions can be specified in terms of service that is provided by the protocols, or additionally, if the restrictions could be relaxed in any way.

Chapter Nine

References

[AM91] Janaki Akella and Kenneth McMillan, "Synthesizing converters between finite state protocols". In *Proceedings 1991 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 410-413. IEEE Computer Society Press, October 1991.

[Aue89] Joshua Auerbach, "A protocol conversion software toolkit". *Computer Communication Review (ACM SIGCOMM '89)*, 19(4):259-270, September 1989.

[Aue90] Joshua Auerbach, "TACT: A protocol conversion toolkit". *IEEE Journal on Selected Areas in Communications*, 8(1):143-159, January 1990.

[Bie89] Ernst W. Biersack, "A systematic approach for constructing gateways". *Computer Networks and ISDN systems*, 18(2):79-95, 1989.

[BMM90] Gregor v. Bochman and Pierre Mondain-Monval, "Design principles for communication gateways". *IEEE Journal on Selected Areas in Communications*, 8(1):12-21, January 1990.

[Boc90] Gregor v. Bochman, "Deriving protocol converters for communications gateways". *IEEE Transactions on Communications*, 38(9):1298-1300, September 1990.

[BS80] Gregor v. Bochman and Carl A. Sunshine, "Formal methods in communication protocol design". *IEEE Transactions on Communications*, 28(4):624-631, April 1980.

[BZ83] Daniel Brand and Pitro Zafiropulo, "On communicating finite state machines". *Journal of the ACM*, 30(2):323-342, April 1983.

[Cal92] Kenneth L. Calvert, "Module composition and refinement with applications to protocol conversion". In R.J. Linn, Jr. and M. U. Uyar (editors), *Protocol Specification, Testing and Verification, XII*, pp. 383-397. IFIP, Elsevier Science Publishers B.V. (North-Holland), June 1992.

[CL88] Kenneth L. Calvert and Simon S. Lam, "An exercise in deriving a protocol conversion". *Computer Communication Review (ACM SIGCOMM '88)*, 18(4):151-160, August 1988.

[CL89a] Kenneth L. Calvert and Simon S. Lam, "Deriving a protocol converter: A top-down method". *Computer Communication Review (ACM SIGCOMM '89)*, 19(4):247-258, September 1989.

[CL89b] Kenneth L. Calvert and Simon S. Lam, "The protocol conversion problem - finding a quotient of specifications". In Juraj V. Medanic and P. R. Kumar (editors), *Proceedings Twenty-Seventh Annual Allerton Conference on Communication, Control and Computing*, pp. 481-490. University of Illinois at Urbana-Champaign, September 1989.

[CL90a] Kenneth L. Calvert and Simon S. Lam, "Adaptors for protocol conversion". In *Proceedings IEEE INFOCOM '90*, pp. 552-560. IEEE Computer Society Press, June 1990.

[CL90b] Kenneth L. Calvert and Simon S. Lam, "Formal methods for protocol conversion". *IEEE Journal on Selected Areas in Communications*, 8(1):127-142, January 1990.

[CL90c] Jing Chang and Ming T. Liu, "An approach to protocol complementation for interworking". In Peter A. Ng, C.V. Ramamoorthy, Laurence C. Seifert and Raymond T. Yeh (editors), *Proceedings of the First International Conference on Systems Integration*, pp. 205-211. IEEE Computer Society Press, April 1990.

[CL90d] Jing Chang and Ming T. Liu, "Using protocol validation techniques to solve protocol conversion problems". In *Ninth Annual International Phoenix Conference on Computers and Communications: 1990 Conference Proceedings*, pp. 539-546. IEEE Computer Society Press, March 1990.

[DD95] Subir Das and P. Dhar, "Interworking between TP4 and TCP through protocol complementation". In V. L. Narasimhan (editor), *IEEE First International Conference on Algorithms and Architectures for Parallel Processing*, volume 1, pp. 315-323. IEEE Computer Society Press, April 1995.

[DGDS92] P. Dhar, R. C. Ganguly, Subir Das and S. Saha, "Protocol conversion - a protocol complementation approach". In *TENCON '92 Technology Enabling Tomorrow: 1992 IEEE Region 10 International Conference on Computers, Communication and Automation towards the 21st Century*, pp. 116-120. IEEE, November 1992.

[Flu88] Francois Fluckiger, "Gateways and converters in computer networks". *Computer Networks and ISDN systems*, 16(1 & 2):55-59, September 1988.

[Gre86] Paul E. Green, Jr., "Protocol conversion". *IEEE Transactions on Communications*, 34(3):257-268, March 1986.

[Gro86] Inge Groenbaek, "Conversion between the TCP and ISO transport protocols as a method of achieving interoperability between data communication systems". *IEEE Journal on Selected Areas in Communications*, 4(2):288-296, March 1986.

[GTN93] Erry Gunawan, Tan Pek Tong and Shi Nansi, "Survey of formal description techniques (FDTs) for protocol converter design". In Yuan Baozong (editor), *Proceedings TENCON '93: 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, volume 1, pp. 422-425. International Academic Publishers, October 1993.

[GY84] M. G. Gouda and Y.-T. Yu, "Protocol Validation by Maximal Progress State Exploration". *IEEE Transactions on Communications*, 32(1):94-97, January 1984.

[HCLY94] Chung-Ming Huang, Meng-Shu Chiang, Shiun-Wei Lee and Yaw-Wei Yao, "Protocol half-converters derivation using the formal approach". *Proceedings of the National Science Council, Republic of China. Part A, Physical Science and Engineering*, 18(5):440-449, September 1994.

[HU96] Esfandiar Haghverdi and Hasan Ural, "An Algorithm for Submodule Construction". In *Technical Report TR-96-09*, Computer Science Department, University of Ottawa, August 1996.

[II83] M. Itoh and H. Ichikawa, "Protocol Verification Algorithm Using Reduced Reachability Analysis". *Transactions on the IECE of Japan*, 66(2):88-93, February 1983.

[JL95] Hou-Wa J. Jeng and Ming T. Liu, "Protocol conversion in multimedia networks: simulation and algorithm". *Simulation*, 64(1):51-60, January 1995.

[KH93] Samir G. Kelekar and George W. Hart, "Synthesis of protocols and protocol converters using the submodule construction approach". In A. Danthine, G. Leduc and P. Wolper (editors), *Protocol Specification, Testing and Verification XIII*, pp. 307-322. IFIP, Elsevier Science Publishers B.V. (North-Holland), May 1993.

[KLNS91] David M. Kristol, David Lee, Arun N. Netravali and Krishan Sabnani, "Efficient gateway synthesis from formal specifications". *Computer Communication Review (ACM SIGCOMM '91)*, 21(4):89-97, September 1991.

[KLNS93] David M. Kristol, David Lee, Arun N. Netravali and Krishan Sabnani, "A polynomial algorithm for gateway generation from formal specifications". *IEEE/ACM Transactions on Networking*, 1(2):217-229, April 1993.

[KWN86] Y. Kakuda, Y. Wakahara and M. Norigoe, "A New Algorithm for Fast Protocol Validation". In *Proceedings IEEE COMPSAC '86*, pp. 228-236. 1986.

[Lam86] Simon S. Lam, "Protocol conversion - correctness problems". *Computer Communication Review (ACM SIGCOMM '86)*, 16(3):19-28, August 1986.

[Lam88a] Simon S. Lam, "Protocol conversion". *IEEE Transactions on Software Engineering*, 14(3):353-362, March 1988.

[Lam88b] Simon S. Lam, "Protocol conversion via projections". In *Proceedings International Computer Science Conference '88 Artificial Intelligence: Theory and Applications*, pp. 121-128. IEEE Computer Society Press, December 1988.

[LNS95] David Lee, Arun N. Netravali and Krishan K. Sabnani, "Protocol pruning". *Proceedings of the IEEE*, 83(10):1357-1371, October 1995.

[MB83] Philip Merlin and Gregor v. Bochmann, "On the construction of submodule specifications and communication protocols". *ACM Transactions on Programming Languages and Systems*, 5(1):1-25, January 1983.

[Oku86] Kaoru Okumura, "A formal protocol conversion method". *Computer Communication Review (ACM SIGCOMM '86)*, 16(3):30-37, August 1986.

[Oku90] Kaoru Okumura, "Generation of proper adapters and converters from a formal service specification". In *Proceedings IEEE INFOCOM '90*, pp. 564-571. IEEE Computer Society Press, June 1990.

[PL92] Mohammad Peyravian and Chin-Tau Lea, "An algorithmic method for protocol conversion". In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pp. 328-335. IEEE Computer Society Press, June 1992.

[PL93] Mohammad Peyravian and Chin-Tau Lea, "Construction of protocol converters using formal methods". *Computer Communications*, 16(4):215-228, April 1993.

[RM91] Murali Rajagopal and Raymond E. Miller, "Synthesizing a protocol converter from executable protocol traces". *IEEE Transactions on Computers*, 40(4):487-499, April 1991.

[RW82] J. Rubin and C. H. West, "An Improved Protocol Validation Technique". *Computer Networks*, pp. 65-73, 1982.

[Sab88] Krishan Sabnani, "An algorithmic technique for protocol verification". *IEEE Transactions on Communications*, 36(8):924-931, August 1988.

[SD92] D. Saha and P. Dhar, "A fast protocol conversion technique using reduction of state transition graphs". In *Eleventh Annual International Phoenix Conference on Computers and Communications: 1992 Conference Proceedings*, pp. 628-635. IEEE Computer Society Press, April 1992.

[SD93] D. Saha and P. Dhar, "Conversion between computer network protocols using graph matching technique". *Journal of the IETE*, 39(5):291-303, September/October 1993.

[SJR95] Kassem Saleh, Mansour Jaragh and Omar Rafiq, "A methodology for the synthesis of communication gateways for network interoperability". *Computer Standards and Interfaces*, 17:193-207, 1995.

[SL89] J. C. Shu and Ming T. Liu, "A synchronization model for protocol conversion". In *Proceedings IEEE INFOCOM '89*, pp. 276-284. IEEE Computer Society Press, April 1989.

[SL90] J. C. Shu and Ming T. Liu, "Protocol conversion between complex protocols". In *Ninth Annual International Phoenix Conference on Computers and Communications: 1990 Conference Proceedings*, pp. 584-590. IEEE Computer Society Press, March 1990.

[SL91] J. C. Shu and Ming T. Liu, "An approach to indirect protocol conversion". *Computer Networks and ISDN systems*, 21(2):93-108, April 1991.

[Sun90] Carl A. Sunshine, "Network interconnection and gateways". *IEEE Journal on Selected Areas in Communications*, 8(1):4-11, January 1990.

[TBD95] Z. P. Tao, Gregor v. Bochmann and R. Dssouli, "A top-down method for synthesizing optimized protocol converters". In *Conference Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications*, pp. 270-276. IEEE Computer Society Press, March 1995.

[TG92] Z. P. Tao and M. Goossens, "Synthesizing communication protocol converter: a model and method". In Jagan P. Agrawal, Vijay Kumar and Virgil Wallentine (editors), *1992 ACM Computer Science Conference: Communications Proceedings*, pp. 17-24. ACM Press, March 1992.

[YCL90a] Yow-Wei Yao, Wen-Shyen Chen and Ming T. Liu, "A modular approach to constructing protocol converters". In *Proceedings IEEE INFOCOM '90*, pp. 572-579. IEEE Computer Society Press, June 1990.

[YCL90b] Yow-Wei Yao, Wen-Shyen Chen and Ming T. Liu, "A parallel model for constructing protocol converters". In *Globecom '90: IEEE Global Telecommunications Conference and Exhibition*, volume 3, pp. 1902-1907. IEEE, December 1990.

[YL91] Yow-Wei Yao and Ming T. Liu, "Constructing protocol converters with guaranteed service". In *Proceedings IEEE INFOCOM '91*, pp. 960-969. IEEE Computer Society Press, April 1991.

[YL92] Yow-Wei Yao and Ming T. Liu, "Constructing protocol converters from service specifications". In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pp. 344-351. IEEE, June 1992.

[Wes78] Colin H. West, "An Automated Technique of Communications Protocol Validation". *IEEE Transactions on Communications*, 26(8):1271-1275, August 1978.

[Wes86] Colin H. West, "Protocol Validation by Random State Exploration". In *Protocol Specification, Testing and Verification, VI*, pp. 233-242. IFIP, Elsevier Science Publishers B.V. (North-Holland), 1986.

[Zim80] Hubert Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection". *IEEE Transactions on Communications*, 28(4):425-432, April 1980.