Operator Based Protocol Synthesis for LOTOS Specification

6 P − 6

Bhed Bahadur BISTA, Zi-xue CHENG and Norio SHIRATORI
Faculty of Engineering, TOHOKU University

1 INTRODUCTION

Designing communication protocols is complex and time consuming. It is desirable to have support methods which help protocol designers to design protocols systematically such that their correctness can be ensured. Some protocol synthesis methods have already been discussed in [1]and[2]. In [1] a single entity is designed using Petri Net and its peer entity is generated in Petri Net. In [2] specification primitives(components) are used to synthesize protocols which are designed using Finite State Machine(FSM). In this paper we propose a method which generates a peer entity from a single given entity which is designed using LOTOS[3], a Formal Discreption Technique developed within ISO.

2 PROTOCOL SYNTHESIS

2.1 Overview of the Synthesis Procedure

Our protocol synthesis problem can be defined as follows.
[Definition 1] (Protocol Synthesis Problem):
Given a single entity we want to construct its peer entity such that their interactions are complete, deadlock-free and properly terminated.□
Completeness ensures that each send message in an entity appears as a receive message in its peer entity. Deadlock-freeness guarantees that no communicating entities are waiting for each other forever. Proper termination ensures that whenever one entity terminates successfully then its peer entity also terminates successfully. We briefly summarize our approach to solve this problem which is shown in figure 1. (1) Design a single entity(process) using basic LOTOS. (2) Decompose the process into sub-processes. (3) Construct sub-processes of the peer entity(process) using the synthesis rules. (4) Combine sub-processes in step 3 to construct the complete peer entity which also is in basic LOTOS.

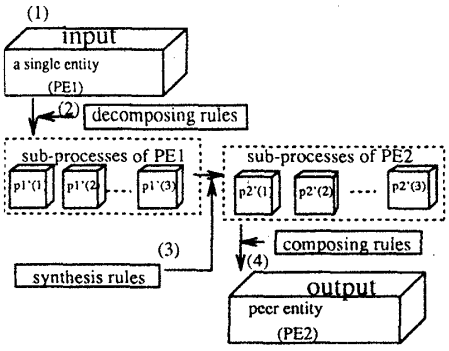


Figure 1: Synthesis Approach

2.2 Definitions and Notation

2.2.1 Definitions

[Definition 2] (Actions of process P):
Actions of process P, Act(P), is defined as a set of all possible actions executed by P.□

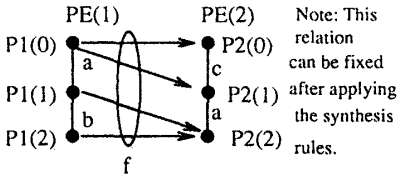
[Definition 3] (Synchronization actions of process P):
Synchronization actions of process P, Sync(P), is defined as a set of actions which P synchronizes with a process Q.□

Example: If P $|[a,c]|$ Q then Sync(P) = Sync(Q) = {a, c}

[Definition 4] (Non-synchronization actions of process P):
Non-synchronization actions of process P,Non-sync(P) is defined as Non-sync(P) = Act(P) - Sync(P).□

[Definition 5] (State Exploration Relation)
Let k be a state of entity PE1, then f(k) is a relation denoting corresponding states of entity PE2.□

Example: Suppose PE1 = a; b; exit and PE2 = c; a ;exit then we have following.



2.2.2 Notations

γ :a non-synchronization action of PE1.
β : a non-synchronization action of PE2.
α: a synchronization action of PE1 and PE2.
P1(i): state i of PE1; P2(j): state j of PE2.
P1(0):the initial state of PE1;P2(0): the initial state of PE2.

2.3 Communication Model

Communication model which we adopted is synchronous and can be summarized in figure 2.

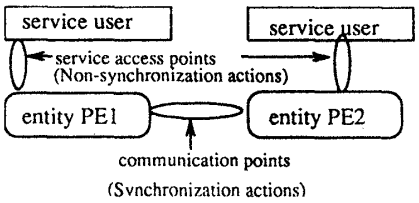


Figure 2: Communication Model

In our communication model, service access points correspond to Service Primitives(SP) and communication points correspond to Protocol Data Units(PDU). The constraints in PE1 and PE2 is that in normal course of actions each SP produces a PDU and vice versa.

2.4 Synthesis Rules for Constructing Sub-processes of Peer Entity

The synthesis rules are constructed from the following two points; (1) whenever an entity interacts first with its user at a service access point (fig.2) in order to obtain a message from the user to send to its peer entity then it interacts with its peer entity at a communication point to send the message, (2) similarly when an entity interacts first at a communication point to receive a message from its peer entity then it interacts with its user at a service access point to deliver the message to the user.

2.4.1 Prefix / Recursion

$$Rule(pr_1) \quad If \quad P1(i) \xrightarrow{\alpha} P1(i+1) \xrightarrow{\gamma} P1(i+2)/P1(k)$$
$$then \quad P2(j) \xrightarrow{\beta} P2(j+1) \xrightarrow{\alpha} P2(j+2)/P2(f(k))$$
$$Rule(pr_2) \quad If \quad P1(i) \xrightarrow{\gamma} P1(i+1) \xrightarrow{\alpha} P1(i+2)/P1(k)$$
$$then \quad P2(j) \xrightarrow{\alpha} P2(j+1) \xrightarrow{\beta} P2(j+2)/P2(f(k))$$

2.4.2 Choice / Recursion

$$Rule(ch_1) \quad If \quad P1(i) \xrightarrow{\alpha 1} P1(i+1) \xrightarrow{\gamma 1} P1(i+2)/P1(k)$$
$$[] \; P1(i) \xrightarrow{\alpha 2} P1(i+1) \xrightarrow{\gamma 2} P1(i+2)/P1(k)$$
$$then \quad P2(j) \xrightarrow{\beta 1} P2(j+1) \xrightarrow{\alpha 1} P2(j+2)/P2(f(k))$$
$$[] \; P2(j) \xrightarrow{\beta 2} P2(j+1) \xrightarrow{\alpha 2} P2(j+2)/P2(f(k))$$
$$Rule(ch_2) \quad If \quad P1(i) \xrightarrow{\gamma 1} P1(i+1) \xrightarrow{\alpha 1} P1(i+2)/P1(k)$$
$$[] \; P1(i) \xrightarrow{\gamma 2} P1(i+1) \xrightarrow{\alpha 2} P1(i+2)/P1(k)$$
$$then \quad P2(j) \xrightarrow{\alpha 1} P2(j+1) \xrightarrow{\beta 1} P2(j+2)/P2(f(k))$$
$$[] \; P2(j) \xrightarrow{\alpha 2} P2(j+1) \xrightarrow{\beta 2} P2(j+2)/P2(f(k))$$
$$Rule(ch_3) \quad If \quad P1(i) \xrightarrow{\gamma 1} P1(i+1) \xrightarrow{\alpha 1} P1(i+2)/P1(k)$$
$$[] \; P1(i) \xrightarrow{\gamma 2} P1(i+1) \xrightarrow{\gamma 2} P1(i+2)/P1(k)$$
$$then \quad P2(j) \xrightarrow{\alpha 1} P2(j+1) \xrightarrow{\beta 1} P2(j+2)/P2(f(k))$$
$$[] \; P2(j) \xrightarrow{\beta 2} P2(j+1) \xrightarrow{\alpha 2} P2(j+2)/P2(f(k))$$

3 SYNTHESIS ALGORITHM

Our algorithm supports the following operators in LOTOS:prefix, choice, **stop**, **exit**, disable and recursion.

3.1 Algorithm

step 1 Design a single entity in basic LOTOS.

step 2 Starting from the initial state of PE1 decompose PE1 into sub-processes such that each sub-process with prefix only, has one SP and one PDU and choice has n SP and n PDU ($n \geq 2$).

step 3 Use the synthesis rules in Section 2.4 to construct sub-processes of PE2. Each α action will have one β action which can be renamed for semantical meaning.

step 4 Combine sub-processes of PE2 to construct the complete peer entity. Note: this is inverse operation of step 1.

3.2 Properties

Our algorithm and the synthesis rules hold the following properties.
(1) PE1 |[*synchronization actions*]| PE2 is deadlock-free.
(2) Interactions between PE1 and PE2 are complete.
(3) PE1 and PE2 are well terminated.

4 APPLICATION EXAMPLE

The example shown in figure 3 is for connection establishment, data transfer and disconnect phase of Transport Layer.

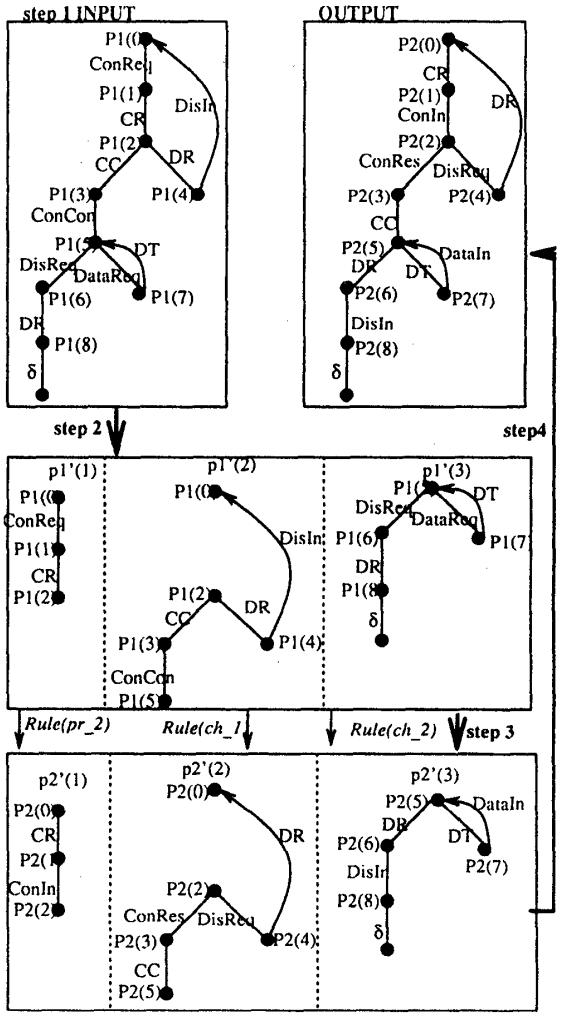


Figure 3: Application Example

5 CONCLUSIONS

When designing a new protocol, our algorithm and synthesis rules creates a peer entity, if a single entity is given in basic LOTOS. Our future research is to extend our present algorithm for full LOTOS and n entities.

6 References

[1] C.V Ramamoorthy, S.T. Dong and Y.Usuda, *An Implementation of an Automated Protocol Synthesizer(APS) and its Implementation to the X.21 Protocol*, IEEE Trans. Software Engineering, Vol SE-11,NO. 9 Sept. 1985.
[2] Y. Kakuda and Y. Wakahara, *Component-based Synthesis of protocols for unlimited number of Processes*, Proc. IEEE COMPSAC'87, OCT.1987 PP721-730.
[3] ISO, *LOTOS- a formal description technique based on the temporal ordering of observational behaviour*, ISO8807(Feb.1989).