Synthesizing Distributed Implementation from Global Specification

M.Tech Project Report

Submitted in fulfillment of the requirements for the degree of

Master of Technology

by

Rathish Das
Roll No : 09305040

under the guidance of

Prof. Bharat G. Adsul

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

June, 2011
Acknowledgement

I would like to express my deep gratitude to my guide Prof. Bharat G. Adsul, who has always been making things simple to understand. Without his deep insight into this domain and his valuable time for this project, it would not have been possible for me to move ahead properly.

Rathish Das
(09305040)
Contents

1 Introduction

2 Implementation Based on Zielonka Theorem

2.1 Message Sequence Charts and Regularity

2.2 Message Passing Automaton

2.3 Size of message buffer

2.4 Information needed to compare latest knowledge

2.5 Secondary Information

2.6 Designing Scheme

2.7 Building States of each local automaton

3 Implementation of Specification Only with Local Action Using Message Passing

3.1 Distributed automata

3.2 Correctness Issue in transitivity of architecture

3.3 Various Deadlock-free Implementation

3.4 Some properties of the specification

3.5 Implementability when the specification is not Forward Diamond

3.6 Correctness issue related to weakly deadlock-free implementation

3.7 Correctness issue in the remark 2 section of the paper

3.8 Remarks

4 Implementation of Specification Only with Local Actions Using Trace Model

4.1 Distributed automata

4.2 Condition of Implementability

4.3 Breaking the problem into subproblems

4.4 Construction of intermediate automaton

4.5 Constructing the Distributed Automata from the Intermediate Automaton

5 Implementation Using Gossip Protocol

5.1 Construction of intermediate automaton

5.2 Constructing the Distributed Automata from the Intermediate Automaton

6 Implementation Maintaining Parallelism

6.1 Construction of intermediate automaton

6.2 Condition for constructing longest diagonal of diamond

6.3 Condition for removing redundant transition of specification

6.4 Constructing the Distributed Automata from the Intermediate Automaton

6.5 An Example

7 Implementation Maintaining Control

7.1 Condition for constructing longest diagonal of diamond

7.2 Condition for removing redundant transition of specification

7.3 Constructing the Distributed Automata from the Intermediate Automaton

7.4 An Example
7 Bisimulation
7.1 Introduction ........................................ 46
7.2 Largest $\leq_p$ relation ................................. 46
7.3 Largest Bisimulation ..................................... 47
7.4 Construction of local automaton: ..................... 49
7.5 An Example: ............................................ 52

8 Summary and Future Work ............................. 54
8.1 Summary ................................................. 54
8.2 Future Work: .............................................. 54
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>MSC Languages</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Process View</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>MSC Languages</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Synchronization and Architecture</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Local automata of Process (p,q) and (r) with Transitivity Closed Architecture</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Local automata of Process (p,q) and (r)</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Specification and Architecture</td>
<td>19</td>
</tr>
<tr>
<td>3.5</td>
<td>Synchronization and Architecture</td>
<td>20</td>
</tr>
<tr>
<td>3.6</td>
<td>Process 1’s Local Automaton</td>
<td>21</td>
</tr>
<tr>
<td>3.7</td>
<td>Process 2’s Local Automaton</td>
<td>21</td>
</tr>
<tr>
<td>3.8</td>
<td>Process 3’s Local Automaton</td>
<td>22</td>
</tr>
<tr>
<td>3.9</td>
<td>Process 4’s Local Automaton</td>
<td>22</td>
</tr>
<tr>
<td>3.10</td>
<td>Process 5’s Local Automaton</td>
<td>22</td>
</tr>
<tr>
<td>3.11</td>
<td>Process 6’s Local Automaton</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Simple Synchronization</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Gadget State</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Transitivity(Case1)</td>
<td>27</td>
</tr>
<tr>
<td>4.4</td>
<td>Modified scenario of figure 4.3</td>
<td>27</td>
</tr>
<tr>
<td>4.5</td>
<td>Transitivity(Case2)</td>
<td>28</td>
</tr>
<tr>
<td>4.6</td>
<td>Modified scenario of figure 4.5</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Simple Synchronization</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>Gadget State</td>
<td>34</td>
</tr>
<tr>
<td>6.1</td>
<td>A Sample Specification</td>
<td>37</td>
</tr>
<tr>
<td>6.2</td>
<td>Putting Longest Diagonal</td>
<td>37</td>
</tr>
<tr>
<td>6.3</td>
<td>Simple Synchronization</td>
<td>38</td>
</tr>
<tr>
<td>6.4</td>
<td>A Modified Specification</td>
<td>39</td>
</tr>
<tr>
<td>6.5</td>
<td>Gadget State of Figure: 6.4</td>
<td>39</td>
</tr>
<tr>
<td>6.6</td>
<td>A Specification</td>
<td>42</td>
</tr>
<tr>
<td>6.7</td>
<td>After Adding Diagonal</td>
<td>42</td>
</tr>
<tr>
<td>6.8</td>
<td>After Removing Redundant Transition</td>
<td>42</td>
</tr>
<tr>
<td>6.9</td>
<td>After construction of Gadget State</td>
<td>43</td>
</tr>
<tr>
<td>6.10</td>
<td>(p)’s NFA</td>
<td>43</td>
</tr>
<tr>
<td>6.11</td>
<td>(q)’s NFA</td>
<td>43</td>
</tr>
<tr>
<td>6.12</td>
<td>(r)’s NFA</td>
<td>44</td>
</tr>
<tr>
<td>6.13</td>
<td>(p)’s DFA</td>
<td>44</td>
</tr>
<tr>
<td>6.14</td>
<td>(q)’s DFA</td>
<td>44</td>
</tr>
<tr>
<td>6.15</td>
<td>(r)’s DFA</td>
<td>44</td>
</tr>
<tr>
<td>6.16</td>
<td>Local automata of Process (p,q) and (r)</td>
<td>45</td>
</tr>
</tbody>
</table>
6.17 Cross Product of Local Automata of Process \( p, q \) and \( r \) 

7.1 A Specification 

7.2 A Sample Specification 

7.3 Local automata of Process \( p \) 

7.4 Local automata of Process \( q \) 

7.5 Local automata of Process \( r \) 

7.6 Local automata of Process \( p, q \) and \( r \) accepting the specification in Figure-6.17
Chapter 1

Introduction

In a distributed system processes communicate with each other. There are many ways to communicate. Among them two important ways are sending and receiving messages and the other one is via synchronization action, where by a synchronization action \( x \), we mean that some processes \( p \) and \( q \) synchronize with each other through action \( x \). Message sequence chart (MSC) represents exchange of messages among processes in a distributed system along a single partial order execution. Through a collection of MSC, the behavior of the system can be represented. On the other hand traces are the natural model, which describes the communication among processes via synchronization action.

A specification of a system depicts the global behavior of it, whereas the implementation of it consists of a set of local controllers and each local controller has very limited information about other. So, it is a challenging task to implement the system specification correctly. Specification of a system is generally given by a regular finite state machine. Now here comes two varieties of specification. In one type, all state transition are labeled by communication actions and in the other type all state transition are labeled by local actions of processes and there no communication action is given. Each type of specification can be modeled by either message sequence chart (i.e. communication actions are given as sending or receiving messages among processes) or traces (communication are given as synchronization action). [1] gives a nice solution to the problem where all communication actions are given in the specification. As here communication among processes are given in the specification, so the communication channel between processes are implicit. On the other hand, when the specification are given with the local action only, we have to add some communication actions in the implementation, so that it correctly implements the specification. But before adding communication action, we have to investigate the existence of communication channel among processes. Here comes the notion of architecture. Here architecture means a graph where each node represents a process and a edge between two process means that they can communicate. A directed edge from \( p \) to \( q \) means \( p \) can send message to \( q \) and \( q \) can receive the message, whereas an undirected edge between \( p \) and \( q \) means that they can synchronize. Directed edge architecture is used for MSC model and non-directed edge architecture is used for trace model. One important fact is that in case of MSC model, the channel should not contain more than some fixed number of undelivered messages, after all in real life situation, every channel has some fixed capacity. We assume that that the channel is not lossy (i.e. a message that has been sent, must be received, it can not lost in the channel) and the channel satisfies the FIFO condition (messages are received in the same order as they send).

This report mainly concentrate on the problem, where the specifications are consists of local action of processes. One direction to implement this may be by adding some sending or receiving actions between processes which is discussed in [5]. But there are some correctness issues in [5]. In this report we mainly focus on the trace model in which processes communicate via synchronization actions. One algorithm is presented in this report, which implements a specification using trace
The structure of this report is therefore as follows: in the next chapter the implementation based on Zielonka Theorem is described. Chapter 3 describes the implementation based on message passing as discussed in [5] and points some correctness issues. Chapter 4 and 5 give two different solutions to implement the local automata accepting specification over local action of processes in trace model. But the execution mode is sequential. In chapter 6 and 7 we have introduced some notions called diamond’s diagonal and bisimulation which make the execution parallel. In the last chapter summary and scope of future work are discussed.
Chapter 2

Implementation Based on Zielonka Theorem

In a distributed system processes can communicate via exchanging messages. Message sequence chart (MSC) depicts this message exchanging between processes and a collection of MSC (MSC language) defines the system behavior in terms of communication. So the specification, which captures this MSC language is a automaton (later we show about it’s regularity) where every communication action is given. We have to implement it through a distributed system of processes.

2.1 Message Sequence Charts and Regularity

Let \( P = \{p, q, r, \ldots\} \) is a finite set of processes that communicate with each other via message passing in a FIFO channel. Let we define the action \( p!q \) as \( p \) sends a message to \( q \), and \( p?q \) as \( p \) receives a message from \( q \). We also define \( \Sigma_p = \{p!q | p \neq q\} \cup \{p?q | p \neq q\} \) as the set of events where process \( p \) has participated and \( \Sigma = \bigcup_{p \in P} \Sigma_p \). We denote the set of channels by \( C = \{(p, q) | p \neq q\} \).

2.1.1 Labeled Poset

A \( \Sigma \)-labeled poset is a structure \( M = (E, \leq, \lambda) \) where \( (E, \leq) \) is a poset and \( \lambda : E \rightarrow \Sigma \) is a labeling function. We define \( e \downarrow = \{e' | e' \leq e\} \) where \( e, e' \in E \). Lets \( E_p = \{e | \lambda(e) \in \Sigma_p\} \) and \( E_a = \{e | \lambda(e) = a\} \). Now we define the \( \leq \) relation as follows:

\[
e \leq_{pq} e' \iff \lambda(e) = p!q, \lambda(e') = p?q \text{ and } |e \downarrow \cap E_{pq}| = |e' \downarrow \cap E_{qp}|.
\]

Lets say \( \leq_{pp} = (E_p \times E_p) \cap \leq \) and \( <_{pp} \) is the largest irreflexive subset of \( \leq_{pp} \). [1]

Definition of MSC

An MSC(over \( P \)) is a finite \( \Sigma \)-labelled poset \( M = (E, \leq, \lambda) \) that satisfies the following conditions:

1. Each relation \( \leq_{pp} \) is a linear order.
2. If \( p \neq q \) then \( |E_{pq}| = |E_{qp}| \).
3. The partial order \( \leq \) is the reflexive and transitive closure of the relation \( \bigcup_{p,q \in P} \leq_{pq} \). [1]

In the figure 2.1 \( E = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \),
\(<_{pq} = \{(1, 2)\} \), \(<_{pr} = \{(3, 4)\} \),
\(<_{qp} = \{(5, 6), (11, 12)\} \), \(<_{qr} = \{(7, 8)\} \), \(<_{rq} = \{(9, 10)\} \).
Now this is found that the partial order $\leq$ is the reflexive and transitive closure of the relation $\bigcup_{p,q \in \mathcal{P}} p \leq q$.

**MSC Language**

Let $\mathcal{P}$ be the total set of processes and $\mathcal{M}_p$ be the all possible MSCs over $\mathcal{P}$. An MSC language is as subset $\mathcal{L} \subseteq \mathcal{M}_p$. Now we can choose a action sequence (sending or receiving) of a MSC $M$ which is called as a linearization of that MSC. So $\text{lin}(M) = \{\lambda(\pi) | \pi \text{ is a linearization of } (E, \leq)\}$. This linearization can be thought as a word. Same way $\text{lin}(\mathcal{L}) = \bigcup \{\text{lin}(M) | M \in \mathcal{L}\}$.

**Definition:** A language $\mathcal{L}$ is called as regular if there exist a DFA which accept $\text{lin}(\mathcal{L})$. So we define regularity of MSC languages in terms of their linearization.

We know this construction of the DFA from [1]

### 2.2 Message Passing Automaton

**Practical simulation** Now the question comes whether we can simulate the behavior of this DFA with discrete local automaton which individually represents each process.

**Formal Definition of Message Passing Automaton:** Let $\mathcal{P}$ be the set of all processes which communicate over the alphabet $\Sigma$. Then a message passing automaton $A$ of $\mathcal{P}$ over $\Sigma$ can be defined as a structure $A = (\{A_p\}_{p \in \mathcal{P}}, \Delta, s_{in}, F)$ where

1. $\Delta$ is a finite set of message alphabet.
2. Each component $A_p$ is of the form $(S_p, \rightarrow_p)$ where $S_p$ is the set of process $p$'s local states and $\rightarrow_p \subseteq S_p \times \Sigma_p \times \Delta \times S_p$ says all the $p$-local transition.
3. $s_{in} \in \Pi_{p \in \mathcal{P}} S_p$ is the global initial state.
4. $F \subseteq \Pi_{p \in \mathcal{P}} S_p$ is the set of global final states.

**Deterministic Message Passing Automaton:** $A$ is said to be deterministic if $\rightarrow_p$ of each component $A_p$ satisfies the following conditions:
1. \((s, plq, m_1, s'_1) \in \rightarrow_p\) and \((s, plq, m_2, s'_2) \in \rightarrow_p\) imply \(m_1 = m_2\) and \(s'_1 = s'_2\).

2. \((s, p?q, m, s'_1) \in \rightarrow_p\) and \((s, p?q, m, s'_2) \in \rightarrow_p\) imply \(s'_1 = s'_2\).

[1]

The set of global state of \(A\) is \(\Pi_{p \in P} S_p\). A configuration is a pair \((s, \chi)\) where \(s\) is a global state and \(\chi : C \rightarrow \Delta^*\) is the channel state that specifies the queue of messages currently residing in each channel \(c\). The initial configuration of \(A\) is \((s_{in}, \chi_e)\), where \(\chi_e(c)\) is the empty string for every channel \(c\). The final configuration of \(A\) is \(F \times \chi_e\). Let we define the set of reachable configuration \(conf_A\) and the global transition \(\Rightarrow \subseteq conf_A \times \Sigma \times conf_A\) inductively as follows:

- \((s_{in}, \chi_e) \in conf_A\)
- Suppose \((s, \chi) \in conf_A\), \((s', \chi')\) be a global state and \((s_p, plq, m, s'_p) \in \rightarrow_p\) such that the following conditions are satisfied:
  
  \(- r \neq p\) implies \(s_r = s'_r\) for each \(r \in P\).
  
  \(- \chi'((p, q)) = \chi((p, q)).m\) and for \(c \neq (p, q)\), \(\chi'(c) = \chi(c)\)

  Then \((s, \chi, p) \Rightarrow (s', \chi')\) and \((s', \chi') \in conf_A\).

- Suppose \((s, \chi) \in conf_A\), \((s', \chi')\) be a global state and \((s_p, p?q, m, s'_p) \in \rightarrow_p\) such that the following conditions are satisfied:
  
  \(- r \neq p\) implies \(s_r = s'_r\) for each \(r \in P\).
  
  \(- \chi'((q, p)) = m.\chi((q, p))\) and for \(c \neq (q, p)\), \(\chi'(c) = \chi(c)\)

  Then \((s, \chi, q) \Rightarrow (s', \chi')\) and \((s', \chi') \in conf_A\).

Let \(w \in \Sigma^*\). A run of \(A\) over \(w\) is a map \(\rho : prf(w) \rightarrow S_e\) such that \(\rho(\epsilon) = s_{in}\) and for each \(\tau a \in prf(w)\), \(\rho(\tau) \Rightarrow \rho(\tau a)\).

The run \(\rho\) is said to be accepted if \(\rho(w)\) is a final configuration.

**Proposition-1:** If \(L \subseteq \Sigma^*\) is a regular MSC language , then there exists a Deterministic message passing automaton \(A\) such that \(L = A(L)\).

### 2.2.1 Interesting Issues in Simulation

The most interesting issue is that in the message passing automaton none of the process can see full message passing . They can see only a part of this computation . So there comes a notion of process’s View which says all the action that a process \(p\) can see. From which it tries to simulate its position in the global minimal DFA.Now when a process \(p\) receives a message from process \(q\) , naturally its current simulated state in the global minimal DFA can be changed if \(q\) has some new information compared to \(p\). So here comes a notion of latest information . But before going further let us define these new notions which are just introduced. For this we have to at first define proper and Ideal.

**Definition of Proper and Ideal:** Let \(M = (E, \leq, \lambda)\) be a MSC(over \(P\)) over the alphabet \(\Sigma\). Now \(\sigma \in \Sigma^*\) is said to be proper if for every prefix \(\tau\) of \(\sigma\) and every pair of processes \(p, q \in P\) it is seen \(|\tau|_{pq} \geq |\tau|_{q}\). A subset \(X\) of MSC \(M\) is said to be an ideal if \(X\) is closed with respect to \(\leq\) of \(M\) i.e. \(e \in I\) and \(f \leq e\) means \(f \in I\).
**Definition of Process view and Latest Information:** Let $M = (E, \leq, \lambda)$ be a MSC(over $\mathcal{P}$) over the alphabet $\Sigma$ and $I$ is an ideal in $M$. Let $p \in \mathcal{P}$ is a process and in the ideal $I$ the maximal $p-$ event is $e$ i.e. $e = \max_p(I)$, then $p-$ view in $I$ denoted as $\partial_p(I)$ is $e \downarrow$ that means set of all those events which have $\leq$ relation with $e$ in the poset relation of $M$.

The latest information of $p$ in it’s view will be $\max(\partial_p(I))$.

In the figure-2.1 let $I$ be $\{1, 2, 3, 4, 5, 6, 7\}$ then $\max_p(I)$ will be event 6. So $p-$ view will be $\{1, 2, 3, 4, 5, 6\}$ and the latest information that process $p$ know in $I$ will be event 6.

![Figure 2.2: Process View](image)

2.2.2 **Need for Time Stamping**

To capture the latest informations there should be a protocol which stamps a time whenever a message is sent. When a process sends a message, it also gives enough informations that it knows about all the processes along with the message so that the receiving process can compare who knows latest informations about processes.

2.2.3 **Size of message buffer**

In the local automaton the message buffer can contains some fixed number of undelivered messages, So we can call a computation as B bounded if none of the message buffer ever contains more than B undelivered messages. Here comes the notion of message acknowledgment.

$B-$ bounded computation: A word $w$ is called $B-$bounded if for every prefix $\sigma$ of $w$ and for every $(p, q) \in \mathcal{P} \times \mathcal{P}$ the following thing should be happened:

$$|\sigma_{plq}| - |\sigma_{plq}| \leq B.$$ 

A MSC $M$ is called $B-$ bounded if every linearization of $M$ is $B-$ bounded. Likewise a MSC-language is called $B-$ bounded if all the MSCs of that language are $B-$bounded.
Lemma-1: If $L$ is a regular MSC language then there is a bound $B \in \mathbb{N}$ such that $L$ is $B$ bounded. [1]

Message acknowledgment Let $e \in I$ is an event of type $p!q$ then $e$ is said to be acknowledged if the corresponding receive event $f$ such that $e <_p f$ lies in $\partial_p(I)$, otherwise $e$ is called as an unacknowledged message to process $p$.

2.2.4 Information needed to compare latest knowledge

A process $p \in \mathcal{P}$ when sends a message to a process $q \in \mathcal{P}$, it must sends all the latest informations about all processes that it knows and also all the unacknowledged messages that it can see. This informations are now current to process $p$ at the sending moment. Formally we call this current informations as primary information of process $p$.

Primary information Let $M = (E, \preceq, \lambda)$ is an MSC and $I$ is an ideal of $M$. Now the primary information of $I$ is the collection of the following elements:

1. $\text{latest}(I) = \{ \max_p(I) | p \in \mathcal{P} \}$.
2. $\text{unack}(I) = \{ \text{unack}_{p!q}(I) | p, q \in \mathcal{P} \}$ where $\text{unack}_{p!q}(I)$ means set of unacknowledged messages in $I$ of type $p!q$.

In figure-2.1 for ideal $I = \{ 1, 2, 3, 4, 5, 6, 7 \}$ we can find that process $p$’s primary information will be $\{ 3, 5, 6 \}$ as 6 and 5 are latest informations of $p$ and $q$ respectively as far as $p$ knows and 3 is an unacknowledged message to $p$.

Lemma-2: For any non-empty ideal $I$ and $p, q \in \mathcal{P}$, the maximal events in $\partial_p(I) \cap \partial_q(I)$ lie in $\text{primary}_p(I) \cap \text{primary}_q(I)$.

For this we may proof that each maximal event $e$ in $\partial_p(I) \cap \partial_q(I)$ lies either in $\text{latest}(\partial_p(I)) \cap \text{unack}(\partial_q(I))$ or $\text{unack}(\partial_p(I)) \cap \text{latest}(\partial_q(I))$.

Let’s suppose $\partial_p(I) \setminus \partial_q(I)$ and $\partial_q(I) \setminus \partial_p(I)$ are both nonempty and $e$ is a maximal event in $\partial_p(I) \cap \partial_q(I)$. Here $e$ can also be a $p$ or $q$ event or some other process’s event. From figure-2.3 it can be seen.

```
Figure 2.3: MSC Languages
```

Here event $e_3$ is the maximal event in $\partial_q(I) \cap \partial_r(I)$ and $e_3$ is a process $r$’s event.

In general we suppose that $e$ is process $r$’s event. The event $e$ must have $\leq$ immediate successors in both $\partial_q$ and $\partial_r$. Now any event $f$ can have at most two immediate successors if $f$ is a send event.
so \( e \) must be a send event.

Process \( p \) comes to know about event \( e \) through a \(<_{rs} \) successor \( e_r \) and process \( q \) through \(<_{rs} \) successor \( e_s \) for some \( s \in P \). Now let \( e_r \in \partial_q(I)\setminus \partial_p(I) \) and \( e_s \in \partial_p(I)\setminus \partial_q(I) \). Now \( e \) is a \( r \) event and \( r \)—successor of \( e \) is outside \( \partial_p \) so \( e \in \text{latest}(\partial_p(I)) \). On the other hand \( e \) is an unacknowledged \( r \)!s event in \( \partial_p(I) \) so \( e \in \text{unack}(\partial_q(I)) \).

Symmetrically, if \( e_s \in \partial_q(I)\setminus \partial_p(I) \) and \( e_r \in \partial_p(I)\setminus \partial_q(I) \) it can be seen that \( e \) lies in \( \text{unack}(\partial_p(I))\cap \text{latest}(\partial_q(I)) \)

Now we have to consider other cases also. Suppose \( \partial_p(I) \subseteq \partial_q(I) \) then \( e \) must be the latest event of both \( p \) and \( q \). Same thing also happens when \( \partial_q(I) \subseteq \partial_p(I) \).

**More than Primary information** Processes can keep track of which events in the system are current by maintaining one additional level of data, called secondary information.

### 2.2.5 Secondary Information

Let \( I \) be an ideal of the MSC \( M = (E, \leq, \lambda) \), then secondary information is the collection of \( \text{primary}(e) \) for each \( e \in \text{primary}(I) \). Similarly for a process \( p \) its secondary information will be \( \bigcup_{e \in \text{primary}(p)} \text{primary}(e) \).

In figure-2.1 for \( I = \{1, 2, 3, 4, 5, 6, 7\} \) process \( p \)'s secondary information will be \( \{1, 3\} \cup \{1, 3, 5\} \cup \{3, 5, 6\} = \{1, 3, 5, 6\} \). So \( e_3 \) is added in the secondary information of \( p \) which was not in it’s primary information. \( e_3 \) comes from \( \text{primary}(e_5 \downarrow) \).

**Lemma-3** : If a \( p \) event is in primary information of a process \( q \), then that event should be in the Secondary information of \( p \). [1]

Now we have enough basics to design a scheme of simulation.

### 2.2.6 Designing Scheme

Now as computation is going on, local automata (each processes’) of the message passing automaton keep changes their local states. But the problem is to take the decision of accepting a MSC globally and this can be done if the local processes keep track the effect of this computation on the DFA that accept the regular MSC language. So a function \( f_w \) can be associated for each word \( w \) such that after reading \( w \) the function \( f_w \) maps for each state \( s \) of DFA to some state of it. So, \( f_w \): \( S \rightarrow S \) where \( S \) is the set of states of the minimal DFA.

### 2.2.7 Building States of each local automaton:

Actually the states of each local automaton somehow indicate the position in the minimal DFA and to calculate it not only a single process’s information is sufficient but more. So switching to some particular state in a local automaton depends also some other local automaton also. But how? Before going to that we have to understand one important concept called residue which intuitively means what a process knows but some other processes don’t.

**Residues**

Let \( \sigma \) be a proper of the MSC \( M = (E, \leq, \lambda) \). Let \( I_1 \) be the ideal consisting of those events which are corresponds to the actions (send or receive) of the \( \sigma \) with respect to the MSC \( M \) and \( p \in P \) be a process. Then \( R(\sigma, p, I) \) denotes the set \( \partial_p(I_1) \setminus I \) is called the residue of \( \sigma \) at \( p \) with respect to \( I \).
**Process Residues**

Let \( p \in P \) is a process and \( P \subseteq \mathcal{P} \) is a set of processes. Now to get all the events that process \( p \) knows but \( P \) don’t we take the notion of process residue i.e. \( R(\sigma, p, \partial P(I)) \).

**Primary Residue:**

\( R(\sigma, p, I) \) is said to be a primary residue if \( I \) is of the form \( X \downarrow \) for some subset \( X \in \text{primary} p(I) \).

**Computing Process Residues in terms of terms of primary residues:**

From lemma-2 we come to know that \( R(\sigma, p, \partial p_q(I)) \) is nothing but \( R(\sigma, p, e_{\downarrow}) \) where \( e \) is the maximal event in \( \partial p(I) \cup \partial q(I) \) and \( e_{\downarrow} \) means all such event \( f \) such that \( f \leq e \). That means we can find some \( X \in \text{primary} p(I) \) such that \( R(\sigma, p, \partial p_q(I)) \equiv R(\sigma, p, X_{\downarrow}) \).

Same way \( R(\sigma, p, \partial p_{p_1}(I) \cup \partial p_{p_2}(I) \cup ... \partial p_{p_k}(I)) \) can be expressed as \( R(\sigma, p, X_{1\downarrow} \cup X_{k\downarrow} \cup ... \cup X_{n\downarrow}) \).

Finally this also can be expressed as \( R(\sigma, p, X_{\downarrow}) \). [1]

So process residues can be computed as primary residues.

**Use of residue and minimal DFA state mapping function**

Let \( \mathcal{P} \) is the set of \( m \) processes \( \{p_1, p_2, ..., p_m\} \). Now we construct a chain of subsets of processes \( \Phi = Q_0 \subset Q_1 \subset Q_2 \subset ... Q_m = \mathcal{P} \). Here we consider the acceptance condition of a complete word \( \sigma \) where by complete word we mean every send action has a corresponding receive action in that word. Let \( I \) be the ideal consisting of those events which are corresponds to the actions (send or receive) of the complete word with respect to the MSC \( M = (E, \leq, \lambda) \). Now \( \sigma[\partial Q_0(I)] \sigma[\partial Q_1(I) \setminus \partial Q_0(I)] \cdots \sigma[\partial Q_m(I) \setminus \partial Q_{m-1}(I)] \) is a valid set of linearizations of \( \sigma \) where by \( \sigma[X] \) we denote all possible linearizations of \( X \) maintaing the \( \leq \) relation of \( M = (E, \leq, \lambda) \).

If each process locally computes the function \( f_{w_i} \) corresponding to any one linearization \( w_i \) of the partial computation \( \{\partial Q_0(E_\sigma) \setminus \partial Q_0\} \) then at the end from the global state we can reconstruct \( f_\sigma \) by composing \( f_{w_m} \circ f_{w_{m-1}} \circ ... \circ f_{w_1} \) and check whether the global state is a final state or not i.e. if \( f_\sigma(s_{in}) \in F \).

Thus the specification is implemented correctly.
Chapter 3

Implementation of Specification Only with Local Action Using Message Passing

Here by a specification we mean an **deterministic minimal** automaton $S$ where transitions are labeled by local actions of processes. We are given a specification $S$. Let $P = \{p, q, r, \ldots\}$ be a finite set of processes in the specification $S$ and let $\Sigma_p, \Sigma_q, \Sigma_r, \ldots$ be the local alphabets of processes $p$, $q$, $r$ and so on. We denote $\Sigma = \Sigma_p \cup \Sigma_q \cup \Sigma_r \cup \ldots$.

An Architecture $A$ is a directed graph $(P, C)$ where each node represents a process in $P$ and an edge from process $p$ to $q$ means process $p$ can send some message to process $q$ and process $q$ can receive message from process $p$. Let $(P, C^*)$ be the transitive closure of $(P, C)$. We denote the sending action with message $a$ from process $p$ to $q$ by $plq(a)$ and receiving action with message $a$ by process $p$ from process $q$ through $p?q$. We denote $\Sigma_p^2$ be the set of all sending or receiving actions where process $p$ can participate. $\Sigma^2 = \Sigma_p^2 \cup \Sigma_q^2 \cup \Sigma_r^2 \cup \ldots$, $\Sigma_p = \Sigma_p^1 \cup \Sigma_p^2$ is the set of all local and communication action where process $p$ has participated. $\Sigma = \Sigma^1 \cup \Sigma^2$ is the whole alphabet.

Let we call a directed edge from $p$ to $q$ as a channel $C_{p,q}$ from $p$ to $q$. We assume that the channel satisfies the FIFO condition that means the order by which messages are sent from $p$ to $q$ they also received by $q$ in the same order.

### 3.1 Distributed automata:

A distributed finite state machine $A = (A_p)_{p \in P}$ is a set of local automaton $A_p$ simulating the behaviour of process $p$ where each $A_p$ is of the form: $(S_p, \Sigma_p, -p, s_{p0})$ where $S_p$ be the set of local state of the local automaton corresponding to process $p$, $\Sigma_p = \Sigma_p^1 \cup \Sigma_p^2$ be the set of alphabet of $A_p$, the transition relation $-p \subseteq S_p \times \Sigma_p \times S_p$ and $s_{p0}$ be the initial state. An action of $A_p$ is either a local action $p(a)$, a send $plq(a)$ or a receive action $p?q(a)$, where $q \in P$. Sending message $plq(a)$ means $a$ is appended to the channel $C_{p,q}$. Receiving message $p?q(a)$ means that, $a$ must be first message in $C_{q,p}$, which will be removed from $C_{q,p}$ after doing the receiving action.

A run of $A_p$ is a word $x$ such that for all $p \in P$, the projection of $x$ on $\Sigma_p$ is a run of $A_p$ and every receive action $p?q(a)$ is enabled, i.e, the first message in the channel $C_{q,p}$ is $a$ before doing $p?q(a)$. A run $x$ is said to be successful if every channel $C(p, q)$ will be empty at the end of $x$.[5]

We make an ordering relation $<_m$ over a send event $s$ and $t$ and say $s <_m t$ if $t$ is the corresponding receive action of the send event $s$. Now we make an ordering relation $<_p$ over the actions of a process $p$. In a run $x$ if there are two action $a$ and $b$ are done by process $p$ then we say that $a <_p b$ with respect to $x$ if $a$ appears before $b$ in $x$. The visual order $<$ in a run $x$ can be defined as the transitive closure of $\bigcup_{p \in P}<_p \cup <_m$. Now applying this relation $<$ we define $\text{past}(a) = \{ x \mid x < a \}$ and $\text{future}(a) = \{ x \mid a < x \}$.[5]
Given a specification $S$ we say distributed automaton $A$ is an implementation if the projection $\pi(L(A))$ of $(L(A)$ on the local actions is equal to $L(S)$. In this paper our intention is to build a deadlock-free implementation. The first step for implementation used in the paper is to reduce the architecture to transitively closed and loop free. But here comes some correctness issues.

### 3.2 Correctness Issue in transitivity of architecture:

In the paper it is stated that we can assume that the architecture is closed by transitivity. Let $(p,q) \in c^*$ but $(p,q) \notin C$. That means, any implementation $A$ of a specification $S$ using $C \cup (p,q)$ can also be translated to another implementation $A'$ using $C$ for $S$.

In the paper the transformation from $A$ to $A'$ is done in the following way: Let $p_0, p_1, ..., p_n$ be some sequence of processes with $(p_ip_{i+1}) \in C$ and $p_0 = p$ and $p_n = q$. Then it suffice to change every action $plq(a)$ into $plp_1fw,a$ (for every $a$, here $(fw,a)$ is a new symbol meaning forward $a$) and $q?p()$ into $q?p_n−1(fw,a)$. Moreover simple loops are added on every state of processes $p_i$, $1 \leq i \leq (n−1)$ labeled by $p_i?p_{i+1}(fw,a)$ and then $p_i?p_{i−1}(fw,a)$. Thus we get the implementation $A'$ not using $(p,q)$.

But let us see the following specification and architecture (Here from $p$ to $r$ there is no direct edge, but we draw $(p,r)$ as a dotted edge because it is transitive closure of $(p,q)$ and $(q,r)$) given in figure 3.1

![Figure 3.1: Synchronization and Architecture](image)

Now see the following implementation in figure 3.2 using the transitive closure of architecture. Figure 3.3 does the transformation from this implementation as described above. But this newly constructed distributed automaton is not the correct implementation of the specification.

**Remark:** This newly constructed distributed automaton is not the correct implementation because, FIFO condition applies on a single channel, not over more than one channel. For example, let suppose $p$ sends two message $a$ and $b$ to $r$ one after another. Let $a$ is sent first, but it first received by process $q$ and then $q$ sends $a$ to $r$, whereas message $b$ goes directly to $r$ from $p$. So, it may be possible that $r$ receives $b$ first and $a$ second.

We can assume that the channel is loop free. If two process $p$ and $q$ are such that $(p,q) \in C$ and $(q,p) \in C$, then any implementation $A$ for $P − \{q\}$ where every action $q(a)$ is replaced by an
Figure 3.2: Local automata of Process \( p, q \) and \( r \) with Transitivity Closed Architecture

Figure 3.3: Local automata of Process \( p, q \) and \( r \)
action \( p(r,a) \), can be translated into an implementation for \( \mathcal{P} \). It suffices to change every action \( p(r,a) \) in \( A \) by two actions \( p!q(a) \) and \( p?q(ack) \). Process \( q \) will be made of loops with transitions \( q?p(a), q(a) \) and \( q?p(ack) \).

### 3.3 Various Deadlock-free Implementation

**Definition:** Very Weakly Deadlock-free Implementation: A implementation is said to be very weakly deadlock-free if every run of it is either a successful run or can be extended to a successful run.

**Definition:** Strongly Deadlock-free Implementation: A very weakly deadlock-free implementation is said to be strongly deadlock-free if for every run \( x \) of the distributed automaton, if there exist an accepting run \( y \) with \( \pi(y) = \pi(x)a \), then \( x \) can be extended into an accepting run \( xz \) such that \( \pi(z) = a \).

**Definition:** Weakly Deadlock-free Implementation: A very weakly deadlock-free implementation is said to be weakly deadlock-free if for every run \( x \) of the distributed automaton, if there exist an accepting run \( y \) with \( \pi(y) = \pi(x)a \), then \( x \) can be extended into an accepting run.

### 3.4 Some properties of the specification:

**Definition:** Diamond Property: A specification satisfies the diamond property if for every states \( r,s,t \) with \( r \xrightarrow{p(a)} s \xrightarrow{q(b)} t \) and \( (p,q) \notin C^* \), there exists a fourth state \( s' \) with \( r \xrightarrow{p(a)} s \xrightarrow{q(b)} s' \) and \( r \xrightarrow{q(b)} s' \xrightarrow{p(a)} t \).

**Definition:** Forward Diamond Property: A specification satisfies the forward diamond property if for every states \( r,s,t \) with \( r \xrightarrow{p(a)} s \) and \( r \xrightarrow{q(b)} t \) and either \( (p,q) \notin C^* \) or \( (q,p) \notin C^* \), there exists a fourth state \( s' \) with \( r \xrightarrow{p(a)} s \xrightarrow{q(b)} s' \) and \( r \xrightarrow{q(b)} t \xrightarrow{p(a)} s' \).

**Lemma 1:** To be implemented, a specification must satisfy diamond property.\footnote{[5]}

**Lemma 2:** Let \( S \) be a deterministic and diamond specification. If \( S \) is implemented by a strongly deadlock-free automaton \( A \), then \( S \) must satisfy forward diamond property. \footnote{[5]}

**Lemma 3:** If a specification satisfies diamond and forward diamond properties, then there is a strongly deadlock-free implementation of that specification. \footnote{[5]}

**Construction of the distributed automaton:**

Let \( S \) be the specification satisfying diamond and forward properties. Now for each process \( p \in \mathcal{P} \) let \( S_p \) be the accessible nodes of \( S \) starting from the initial node by any action of processes \( \{q|(q,p) \in C^*\} \). We now define \( A_p \) from the automaton \( S_p \), where the labels \( a \in \Sigma_q^1 \) and \( q \neq p \) of the transition are replaced by \( p!q(a) \) and every transition labeled by \( a \in \Sigma_p^1 \) is replaced by a sequence of transitions labeled by \( a \) and then by \( plq(a) \) for all \( (p,q) \in C^* \). That means when an action is done by \( p \), then \( p \) informs each of it’s children. We name this distributed automaton as \( A \).

We can show that \( \pi(L(A)) = L(S) \) as described in \footnote{[5]}.
3.5 Implementability when the specification is not Forward Diamond

When the specification satisfies diamond and forward diamond properties, then we have an implementation as discussed earlier. In the implementation we saw that every choice i.e, doing a local action by a process is decided by itself. That means choices were local. Now let us look the specification and architecture given in figure 3.4, where \( s_1 \xrightarrow{p(a)} s_2 \) and \( s_1 \xrightarrow{q(b)} s_3 \) are two state transitions. This specification is not forward diamond. Now process \( p \) can not locally (i.e, by itself only) choose to do \( p(a) \) and process \( q \) also can not locally (i.e, by itself only) choose to do \( q(b) \), because then \( p(a)q(b) \) and \( q(b)p(a) \) also be accepted by the distributed automaton. So, to exactly implement the specification here such choices can not be done locally.

One solution to this problem is that process \( r \) sends bit 0 to it’s two children \( p \) and \( q \) or sends bit 1 to both. Now if process \( p \) gets bit 0, it does action \( p(a) \) and upon getting bit 0 \( q \) does nothing. On the other hand if process \( p \) gets bit 1, it does nothing, whereas upon getting bit 1 \( q \) does \( q(b) \). So this is a correct implementation.

So, these choices have to be decided globally. Here comes the notion of global choice. Before formally giving the definition of global choices let us first know some terminology.

By \( \mathcal{L}(s) \), we denote the language where each word in \( \mathcal{L}(s) \) is accepted by taking \( s \) as the initial state in the specification.

By \( \mathcal{L}_Q \) be the projection of \( \mathcal{L} \) on the set of processes belongs to \( Q \).

**Definition : Global Choice** A transition \( s \xrightarrow{p(a)} t \) is said to be a global choice if \( \mathcal{L}_Q(s) \neq \mathcal{L}_Q(t) \) for \( Q = \{ k | (p,k) \notin C^* \} \). We denote by \( \mathcal{G} \), the set of global choices. [5]

**Observation:** If the specification does not satisfies forward diamond properties, then \( \mathcal{G} \neq \emptyset \)

3.5.1 Weakly Deadlock-free Implementation

In the paper it is shown that the number of global choice in the specification is finite.

Let \( S \) be deterministic specification satisfying diamond property, then there is no loop \( s \xrightarrow{p(a)} t \xrightarrow{y} s \) with \( s \xrightarrow{p(a)} t \in \mathcal{G} \) is a global choice

The idea for weakly deadlock-free implementation is that, as the number of global choice is finite, so these choices can be made once at the beginning as a subset \( G \in \mathcal{G} \). However not every subset can be chosen. Let \( S^G \) the specification obtained from \( S \) by deleting every global choices not in \( G \). We say, \( G \) is compatible if,

1. There exists \( H \supseteq G \) such that \( S^H \) satisfies diamond and forward diamond properties.
2. Every \( x \in \mathcal{L}(S^G) \) that cannot be extended in \( S^G \), cannot be extended in \( S \) either.

We denote \( \mathcal{F} \) be set of compatible sets \( G \subseteq \mathcal{G} \). Let \( \mathcal{F} = \{ F_1, F_2 \ldots F_k \} \) and by \( \pi_p(F_i) \) we mean the projection of \( F_i \) on process \( p \)'s global choices. So, \( \mathcal{F}_p = \bigcup \pi_p(F_i) \). [5]

To help processes making global choices, their ancestor sends information. We now impose some condition under which, there will be a weakly deadlock-free implementation of a specification (when the specification does not satisfying forward diamond property.)

Let \( p_1, p_2, \ldots p_n \) be the minimal processes in the architecture, where by minimal process \( p \) we mean there is no \( q \) such that \( (q, p) \in C^* \). Let \( I_k \) be the information sent by minimal process \( p_k \) and \( I = I_1 \times I_2 \times \ldots \times I_n \). Now we make a function \( \text{choice}(p) \) corresponding to each process \( p \) which maps \( I \) to \( \mathcal{F}_p \), i.e, \( \text{choice}(p) : I \rightarrow \mathcal{F}_p \). Now we say \( S \) will have a weakly deadlock-free implementation with information \( I \), if there is a function \( \text{choice} \) such that-

1. For every tuple \( i \in I \), let \( g(i) = \bigcup_{p \in P} \text{choice}(p)(i) \).
2. For every tuple \( i \in I \), \( g(i) \in \mathcal{F} \), that is it is a compatible set of choices.
3. \( \bigcup \mathcal{L}(S^{\text{choice}(i)}) = \mathcal{L}(S) \)

[5]

3.6 Correctness issue related to weakly deadlock-free implementation:

Let us see the following specification and the architecture.

![Figure 3.5: Synchronization and Architecture](image)

In this example \( \mathcal{G} = \left\{ a \xrightarrow{6(a)} b, a \xrightarrow{5(a)} d, a \xrightarrow{4(a)} f \right\} \).

The set of compatible set i.e, \( \mathcal{F} = \left\{ \left\{ a \xrightarrow{6(a)} b \right\}, \left\{ a \xrightarrow{5(a)} d \right\}, \left\{ a \xrightarrow{4(a)} f \right\} \right\} \). We want to show that we can not find out a function \( g \) as discussed earlier satisfying all properties. For contradiction, let us assume we can find out such a function \( g \).
Let,

1. \( g(a, b, c) = \left\{ a \xrightarrow{4(a)} f \right\} \) where \( a, b, c \) are information sent by minimal processes 1, 2, 3 respectively.

2. \( g(a_1, b_1, c_1) = \left\{ a \xrightarrow{6(a)} b \right\} \)

3. \( g(a_2, b_2, c_2) = \left\{ a \xrightarrow{5(a)} d \right\} \)

From 3) we get choice(5)(a_2, c_2) = \( a \xrightarrow{5(a)} d \) ..........(4)

From 2) we get choice(6)(b_1, c_1) = \( a \xrightarrow{6(a)} b \) ..........(5)

As \( (a_2, b_1, c_2) \in I \), to make \( g(a_2, b_1, c_2) \) compatible \( g(a_2, b_1, c_2) \) will be \( a \xrightarrow{5(a)} d \) as choice(5)(a_2, c_2) = \( a \xrightarrow{5(a)} d \) [from (3)]

and choice(6)(b_1, c_1) = \( \epsilon \) ..........(6)

As \( (a, b_1, c_1) \in I \), so to make \( g(a, b_1, c_1) \) compatible \( g(a, b_1, c_1) \) will be \( a \xrightarrow{6(a)} b \) [from (5)] and

choice(4)(a, b_1) = \( \epsilon \) ..........(7)

As \( (a, b, c_2) \in I \), so to make \( g(a, b, c_2) \) compatible \( g(a, b, c_2) \) will be \( a \xrightarrow{4(a)} f \) and choice(5)(a, c_2) = \( \epsilon \) ..........(8)

Now, let us we take \( i = (a, b_1, c_2) \) and \( i \in I \). But choice(6)(b_1, c_1) = \( \epsilon \), choice(4)(a, b_1) = \( \epsilon \) and choice(5)(a, c_2) = \( \epsilon \).

So \( g(a, b_1, c_2) = \epsilon \) and \( \epsilon \notin \mathcal{F} \).

So, we never find out such function \( g \). That means according to the paper we say this specification does not have a weakly deadlock-free implementation.

But let us see the following implementation(Figure 3.6,3.7,3.8,3.9,3.10,3.11)-

![Figure 3.6: Process 1’s Local Automaton](image1)

![Figure 3.7: Process 2’s Local Automaton](image2)

### 3.7 Correctness issue in the remark 2 section of the paper:

It is written in the paper that transition like \( s \xrightarrow{\tau} t \) with \( L_Q(s) \neq L_Q(t) \) are the only reason, so that a specification may not be very weakly deadlock free. Then it is written that If there is no such transition, each minimal process guesses a subset \( G \subseteq \mathcal{G} \). But when there are no such transition then there will be no such global choices, so here this statement is contradictory.
3.8 Remarks:

As we have seen that there are some correctness issues arise in this paper. But, the problem statement is no doubt very interesting. So we try to make an implementation in the next chapter using simple trace language.
Chapter 4

Implementation of Specification Only with Local Actions Using Trace Model

Here by a specification we mean an deterministic minimal automaton $S$ where transitions are labeled by local actions of processes. We are given a specification $S$. Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes in the specification $S$ and let $\Sigma_p, \Sigma_q, \Sigma_r, \ldots$ be the local alphabets of processes $p$, $q$, $r$ and so on. We denote $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \ldots$.

An Architecture $A$ is an undirected graph $(\mathcal{P}, C)$ where each node represents a process in $\mathcal{P}$ and a edge between process $p$ and $q$ means process $p$ and $q$ can synchronize among themselves. Let we denote this synchronization action by $pq(x)$ or $qp(x)$ which means process $p$ and $q$ can synchronize through this action. This $pq(x)$ and $qp(x)$ are actually same. We denote $\Sigma_2 = \Sigma_2^p \cup \Sigma_2^q \cup \Sigma_2^r \cup \ldots$; $\Sigma_2^p$ be the set of all synchronization actions where process $p$ can participate. $\Sigma_2 = \Sigma_2^p \cup \Sigma_2^q \cup \Sigma_2^r \cup \ldots$ is the set of all local and synchronization action where process $p$ has participated. $\Sigma = \Sigma_1 \cup \Sigma_2$ is the whole alphabet. Let $a \in \Sigma_2^p \cap \Sigma_2^q$ then it means that $a$ is a synchronization action between $p$ and $q$.

Here architecture can be thought as a set of connected components which are disjoint to each other, that means a process $p$ can not reside in two distinct connected component $A_i$ and $A_j$, otherwise $A_i$ and $A_j$ will be merged in a single component. Let $\{A_1, A_2, \ldots, A_n\}$ be the set of connected components and let $\mathcal{P}_i \subseteq \mathcal{P}$ be the set of processes resides in the connected component $A_i$. So for two distinct $\mathcal{P}_i$ and $\mathcal{P}_j$, it is the case that, $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset$.

4.1 Distributed automata :

A distributed finite state machine $A = (A_p)_{p \in \mathcal{P}}$ is a set of local automaton $A_p$ simulating the behaviour of process $p$ where each $A_p$ is of the form : $(S_p, \Sigma_p, \rightarrow_p, s_{p0})$ where $S_p$ be the set of local state of the local automaton corresponding to process $p$; $\Sigma_p = \Sigma_1^p \cup \Sigma_2^p$ be alphabet of $A_p$; the transition relation $\rightarrow_p \subseteq S_p \times \Sigma_p \times S_p$ and $s_{p0}$ be the initial state.

Note 1: Here each local automaton is deterministic.

Note 2: When a process $p$ performs some synchronization action, say $pq()$ with some process $q$, process $p$'s state transition depends upon it’s current state and the synchronization letter. Whereas in the standard case such transition depends on both $p$ and $q$’s current state and the synchronization letter.

Here $s_{in} = (s_{p0}, s_{q0}, \ldots, s_{r0})$ be the global initial state. The set of global state of $A$ is $\Pi_{p \in \mathcal{P}} S_p$. However every global state is not reachable. Let we define the set of reachable global states $S_r$ and the global transition $\Rightarrow \subseteq S_r \times S_r$ inductively as follows:

- $s_{in} \in S_r$
• Suppose \( s \in S_r, s' \) be a global state and \( (s_p, p(a), s'_p) \in \rightarrow_p \) such that the following condition is satisfied:
  \[ r \neq p \text{ implies } s_r = s'_r \text{ for each } r \in P. \]
  Then \( s \xrightarrow{p(a)} s' \) and \( s' \in S_r. \)
• Suppose \( s \in S_r, s' \) be a global state, \( (s_p, pq(), s'_p) \in \rightarrow_p \) and \( (s_q, pq(), s'_q) \in \rightarrow_p \) such that the following condition is satisfied:
  If \( r \) is some other process different from \( p \) and \( q \), then \( s_r = s'_r \) for each \( r \in P. \)
  Then \( s \xrightarrow{pq()} s' \) and \( s' \in S_r. \)

Let \( w \in \Sigma^* \). A run of \( A \) over \( w \) is a map \( \rho : prf(w) \rightarrow S_r \) such that \( \rho(\epsilon) = s_{in} \) and for each \( \tau a \in prf(w), \rho(\tau) \xrightarrow{a} \rho(\tau a). \)

A word \( x \) is said to be accepted if \( x \) is a run of \( A \).
If a word \( x \) is accepted by \( A \), the projection of \( x \) on \( \Sigma_p \) is accepted by \( A_p. \)

Problem Statement : Given a specification \( S \) over the alphabet \( \Sigma^l \) and an architecture \( A \) we have to find out if \( S \) is implementable or not and if implementable, we have to design a trace language \( L \) over \( \Sigma^l(\Sigma^2) \) where \( \Sigma^2 \) is the set of synchronization actions between processes and also design a distributed finite state machine which accept \( L \) with the condition that the projection of \( L \) on \( \Sigma^l \) is equal to \( L(S) \).

4.2 Condition of Implementability:

First we discuss two properties of specification.

**Definition : Diamond Property:** A specification satisfies the diamond property if for every states \( r, s, t \) with \( r \xrightarrow{p(a)} s \xrightarrow{q(b)} t \) such that \( p \in P \) and \( q \in Q \) where \( P \) and \( Q \) are sets of processes of two distinct connected components, there exists a fourth state \( s' \) with \( r \xrightarrow{p(a)} s \xrightarrow{q(b)} t \) and \( r \xrightarrow{q(b)} s' \xrightarrow{p(a)} t. \)

**Definition : Forward Diamond Property:** A specification satisfies the forward diamond property if for every states \( r, s, t \) with \( r \xrightarrow{p(a)} s \) and \( r \xrightarrow{q(b)} t \) such that \( p \in P \) and \( q \in Q \) where \( P \) and \( Q \) are sets of processes of two distinct connected components, then there exists a fourth state \( s' \) with \( r \xrightarrow{p(a)} s \xrightarrow{q(b)} s' \) and \( r \xrightarrow{q(b)} t \xrightarrow{p(a)} s' . \)

If \( p \) and \( q \) resides in two different connected component, then they can not synchronize by any means.

**Proposition 1-** To have a Implementation specification should satisfy both diamond and forward diamond property.

The specification is deterministic and minimal. If the specification does not satisfy diamond property, then we must find a word \( w = up(a)q(b)v \) such that \( up(a)q(b)v \in S \) but \( uq(b)p(a)v \notin S \) (because of the minimal property). But we have an implementation \( A \). Let \( u'p(a)wq(b)v' = up(a)q(b)v \), where by \( u(X) \) we mean projection of \( X \) on the local actions. Let \( p \in P \) and \( q \in Q \) where \( P \) and \( Q \) are sets of processes of two distinct connected components. \( w_p \) and \( w_Q \) be the projection of \( w \) over the actions of processes in \( P \) and \( Q \) respectively. Now, \( u'w_Qq(a)w_PP(a)v' \in L(A). \) So, \( u'w_Qq(a)w_PP(a)v' = uq(b)p(a)v \) should be accepted by \( S \). That means, the specification should satisfy diamond property.
To have a implementation $S$ also has to satisfy forward diamond property. Let $up(a) \in S$ and $uq(b) \in S$ but $up(a)q(b), uq(b)p(a) \notin S$. But we have a implementation $A$. Let $vp(a), wq(b) \in \mathcal{L}(A)$ where $\Pi(up(a)) = up(a)$ and $\Pi(wq(b)) = uq(b)$. Let $p \in P$ and $q \in Q$ where $P$ and $Q$ are sets of processes of two distinct connected components. Let in $v$ and $w$, processes which have done some actions belongs to the connected component $P_1, P_2....P_k, P$ and $Q$. As $\Pi(v) = \Pi(w) = u$, so $\Pi(v_{P_i}) = \Pi(w_{P_i}) = u_{P_i}$ for all $P_i$ where by $v_{P_i}$, we mean projection of $v$ over the actions of processes in $P_i$. Let $z = v_{P_i}P_i.....v_{P_i}P_iP_iq(a)q(b)$ and $x = v_{P_i}P_i.....v_{P_i}P_iq(b)p(a)$ is a run of $A$, as the projection of $z$ and $x$ over $P_i$ is a run in corresponding local automaton. Now $\Pi(z) = up(a)q(b)$ and $\Pi(x) = uq(b)p(a)$. As the specification is minimal, so the specification should satisfy forward diamond property.

### 4.3 Breaking the problem into subproblems:

Now from the original specification $S$, we construct $S_{P_i}$ for each $P_i$ as defined above($P_i$ is the set of processes of the connected component $A_i$). We say, $S_{P_i}$ be the accessible states of original $S$ starting from start state by any action of processes in $P_i$. As this minimal specification $S$ holds diamond and forward diamond properties, so $S = S_{P_1} \times S_{P_2} \times \cdots \times S_{P_k}$, where $k$ is the total number of distinct connected component.

Construction sketch: If we can implement each $S_{P_i}$ as described above, then we ultimately implement the original specification $S$. We can now assume that we reduced the problem to implement a specification over a connected architecture. So without loss of generality we name this new reduced specification $S_{P_i}$ as $S$ and the connected architecture consisting with processes belong to $P_i$ as $A$. Let $\Sigma^1$ and $\Sigma^2$ be the local actions and synchronization actions respectively of all the processes belong to $A$. From $S$ we construct an intermediate automaton $A_2$ over alphabet $\Sigma^1 \cup \Sigma^2$ and then we take the projection over $\Sigma_{P_i}$ from $A_2$ for all $p_i \in P$ and do some refinements. Thus we get the distributed set of automaton $A_3$. Although it may seems not such straight forward(Zielonka Theorem), but due to special construction of $A_2$ this is possible. Later we prove that $L(A_2)$ and $L(A_3)$ are indeed same.

### 4.4 Construction of intermediate automaton:

If in the specification $S$, the start state has any incoming transition towards it , we modify $S$ to $S_1$ in the following way such that the start state of $S_1$ has no incoming transition towards it and $\mathcal{L}(S) = \mathcal{L}(S_1)$.

1. Create a new state $s'$.

2. For every state $s_k$ if there is a transition $p(a)$ from start state $s$ to $s_k$, then make a transition $p(a)$ from $s'$ to $s_k$.

3. For every state $s_i$ if there is a transition $p(b)$ from state $s_i$ to start state $s$, then delete that transition and make a new transition $p(b)$ from $s_i$ to $s'$.

**Lemma 1 - $\mathcal{L}(S)$ and $\mathcal{L}(S_1)$ are same**:

If there is a run $w$ in $S$, say $s \xrightarrow{p(a)} s_1 \xrightarrow{q(a)} s_2 \ldots \xrightarrow{r(a)} s \xrightarrow{p(b)} s_{j \ldots m}$, then except the first occurrence of start state $s$ change every occurrence of $s$ with $s'$. This will be a run of $S_1$.

Other way, if there is a run $w'$ in $S_1$, say $s \xrightarrow{p(a)} s' \xrightarrow{q(a)} s' \xrightarrow{r(a)} s \xrightarrow{p(b)} s' \ldots \xrightarrow{r(a)} s' \xrightarrow{p(b)} s' \xrightarrow{r(a)} s' \ldots \xrightarrow{r(a)} s' \xrightarrow{p(b)} s' \xrightarrow{r(a)} s'$, then change every occurrence of $s'$ with $s$. This will be a run of $S$. 

25
Some Notation:
We will now introduce two functions \( \phi : Q \times \Sigma^1 \rightarrow 2^P \) and \( f : \Sigma^1 \times Q \times \Sigma^1 \rightarrow 2^P \) where \( Q \) is the set of state of \( S' \) and \( \Sigma^1 \) is the set of local action of processes.

- If \( s \) has only one outgoing transition \( q(a) \) from it, then \( \phi(s, q(a)) = \{q\} \).
- If \( s \) has other outgoing transition from it besides \( q(a) \) then \( \phi(s, q(a)) \) means all those processes which has participated in some action along the path(s) starting from \( s \) with the first transition \( q(a) \).
- Let \( s \) has a incoming transition \( p(a) \) towards it and a outgoing transition \( p(b) \) from it, then \( f(p(a), s, p(b)) = \{q\} \) if \( \phi(s, p(b)) = \{p\} \).
- \( f(p(a), s, q(b)) = \phi(s, q(b)) - \{p\} \).

Construction of Intermediate Automaton from the Modified Specification:

1. If there is a state transition say \( s \xrightarrow{p(a)} t \) and \( t \) has only one incoming and one outgoing state transition say \( t \xrightarrow{q(b)} r \), then replace the label \( p(a) \) by \( p(a) \) then \( pq(q, b) \).

\[
\begin{array}{c}
\text{s} \\
\downarrow^{p(a)} \quad \downarrow^{q(b)} \\
\text{t} \\
\downarrow \qquad \downarrow \\
\text{r}
\end{array}
\]

Figure 4.1: Simple Synchronization

2. Let suppose there is a state \( t \), now if \( t \) has either more than one incoming transition towards it or more than one outgoing transition from it then we form a gadget state \( G_t \) which simulates state \( t \). Let us suppose there are \( y \) different incoming edges incident to \( t \) labelled as \( q_1(a), q_2(b) \ldots q_y(c) \) and there are \( x \) number of outgoing edges from \( t \) labelled as \( p_1(a), p_2(b) \ldots p_x(d) \). Then our gadget state construction will be following -

In the gadget state \( G_t \) there will be two column of states. The first column is the set of states each corresponds to a incoming transition towards state \( t \). The second column consists of the set of states each corresponds to one outgoing transition from state \( t \). Let in the first column, state \( s_1 \) corresponds to incoming action \( x \) of process \( p \), say \( p(a) \) and in the second column, state \( s_2 \) corresponds to outgoing transition \( y \), say \( q(b) \), then between \( s_1 \) and \( s_2 \) process \( p \) will synchronize with all process \( p_i \in f(x, t, y) \) sequentially where each synchronization action will be of the form \( pp_i(q, b) \). One important fact is that, if \( p \) and \( q \) are different processes, then \( p \) will synchronize with \( q \) after finishing synchronization action with all other process \( p_i \in f(x, t, y) - \{q\} \). This is done, because the next transition after \( pq(q, b) \) will be \( q(b) \), so every two consecutive action has a common process participator - this condition will be still remained true.
3. Let \( p \) and \( q \) synchronize among themselves. Then the next transition will be, either \( q \) does some local action or \( p \) synchronizes with some other process. For the first case the scenario will be like figure 4.3. Process \( p \) has done action \( a \) and goes to state \( s \) then it synchronizes with \( q \) via \( pq(q,a) \) and goes to \( t \) and then process \( q \) has done \( a \). Now let suppose \( p \) and \( q \) are not directly connected. But there is a chain of process \( p = p_0, p_1, p_2...p_k, q = p_{k+1} \) such that \( p_i \) and \( p_{i+1} \) are directly connected (\( 0 \leq i \leq k \)).

For the second case the scenario will be like figure 4.5. Process \( p \) has done some synchronization action with process \( p_1 \), say \( pp_1(q,a) \) and goes to state \( t \) then it again does some
synchronization action with process \( p_2 \), say \( pp_2(q,a) \) and goes to state \( r \). Now let’s suppose \( p \) and \( p_1 \) are not directly connected. But there is a chain of process \( p = p_1^0, p_1^1, p_1^2, \ldots, p_1^k, p_1 = p_1^{k+1} \) such that \( p_i \) and \( p_i^{i+1} \) are directly connected \( (0 \leq i \leq k) \).

![Figure 4.5: Transitivity(Case2)](image)

We rearrange this as in figure 4.6.

![Figure 4.6: Modified scenario of figure 4.5](image)

The important observation is that as in the original cases every two consecutive sequential actions were done by some common process, in the modified version also this constraint is maintained.

**An Example:** In figure 4.2 state \( s \) has more than one incoming transition towards it and more than one outgoing transition from it. So, in the intermediate automaton there will be a gadget state \( G_s \) which simulate state \( s \). Here, \( \phi(s,q(b)) = \{q,r\}; \phi(s,p(b)) = \{p\} \) So, \( f(p(a),s,p(b)) = \{q\} \) because, \( \phi(s,p(b)) = \{p\} \); \( f(p(a),s,q(b)) = \{q,r\}; \phi(q(a),s,q(b)) = \phi(s,q(b)) - \{q\} = \{q,r\} - \{q\} = \{r\} \) and \( f(q(a),s,p(b)) = \{p\} \).

**Observation1:** From the construction we can say that \( \Pi_{\Sigma_1}(L(A_2)) = L(S) \).

**Observation2:** In the intermediate automaton every two consecutive transitions have a common process participator.

**Lemma 2-** In the \( A_2 \) from a state \( s \) if there exists \( k \) different paths \( x_1, x_2, \ldots, x_k \) such that the first transition along path \( x_i \) goes to state \( s_i \) and the first transition along path \( x_j \) goes to state \( s_j \) where \( s_i \) and \( s_j \) are different, then for all process \( q \) the first \( q \) transition along path \( x_i \) will be different from the first \( q \) transition along path \( x_j \) and there exist some process \( p \) such that all processes \( p_i \) in the above paths will synchronize directly with \( p \) i.e., \( pp_i() \) or will synchronize with \( p \) transitively, that means \( pp_1(), p_1p_2(), \ldots, p_1\ldots p_k() \). We say \( p_i \)’s this first action as **transitive synchronization action with** \( p \).

This is because of gadget state. If from a state \( s \) there are more than one outgoing path then \( s \) must belongs to a gadget state \( G \). And in the gadget state from a component state \( s \),
for all process \( q \) the first \( q \)-transition along two different paths \( x \) and \( y \) must be synchronization actions and they are different in the argument as the specification is deterministic (for the synchronization action \( pq(r, a) \) the argument is \( (r, a) \)). So the first \( p \) transition along path \( x \) will be different from the first \( p \) transition along path \( y \). More over in the gadget state we must find out a process \( s \) which will synchronize directly or transitively with all processes.

4.5 Constructing the Distributed Automata from the Intermediate Automaton:

Now we are going to construct the local automaton of each process \( p \in \mathcal{P} \). In the intermediate automaton we replace all non-\( p \) action by epsilon. Lets call it as \( \epsilon \)-NFA of process \( p \).

**Construction of non-\( \epsilon \)-Automaton of process \( p \) from \( \epsilon \)-NFA of process \( p \)**

We convert the \( \epsilon \)-NFA of process \( p \) to a non-\( \epsilon \)-Automaton of process \( p \) as follows.

(a) We take the \( \epsilon \)-closure of every state of the \( \epsilon \)-NFA of process \( p \).

(b) Let state \( s \) has the \( \epsilon \)-closure \( Cl_s \). Now let suppose that \( t \in Cl_s \) and \( t \) has an outgoing transition \( x \) to \( r \). Then we make a new transition \( x \) from \( s \) to \( r \).

(c) If a state \( s \) has a \( \epsilon \)-transition to state \( t \) then we delete that \( \epsilon \)-transition.

**Observation 3**

Like the \( \epsilon \)-NFA in this newly constructed automaton every state is a final state.

**Lemma 3:** The language accepted by this \( \epsilon \)-NFA and non-\( \epsilon \)-automaton of process \( p \) are same

Let there is a run \( r \) in the \( \epsilon \)-NFA of \( p \), say \( s \rightarrow s_1 \rightarrow s_1 \rightarrow s_1 \ldots \rightarrow s_1 \). Now let \( s_i \) be the first state in the run from which there is a \( p \) action \( x \) to \( s_{i+1} \), then in the newly constructed automaton there will also be a transition \( x \) from \( s \) to \( s_{i+1} \). Now in \( \epsilon \)-NFA from \( s_{i+1} \) we find out the first state \( s_j \) along the run \( r \), from which there is a \( p \) action \( y \) to \( s_{j+1} \), then in the newly constructed automaton there will also be a transition \( y \) from \( s_{i+1} \) to \( s_{j+1} \). In this way we will reach the state \( s_k \) such that there is a \( p \) action towards \( s_k \) but after \( s_k \) there is no \( s_k' \) along the run \( r \), from which there is a \( p \) action \( z \). So in the newly constructed automaton there is a run \( r_1 \) up to \( s_k \) such that the word accepted by \( r \) and \( r_1 \) are same.

Let there is a run \( r \), say \( s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{c} s_3 \ldots \xrightarrow{d} s_k \) in the newly constructed non-\( \epsilon \)-automaton of \( p \). Then for each \( s_i \xrightarrow{\epsilon} s_{i+1} \) \((0 \leq i \leq k-1)\) in the non-\( \epsilon \)-automaton there will be direct transition \( x \) from \( s_i \) to \( s_{i+1} \) or there will be a path \( s \xrightarrow{\epsilon} s_1 \xrightarrow{\epsilon} s_1 \ldots \xrightarrow{\epsilon} s_{i+1} \xrightarrow{\epsilon} t \) in the \( \epsilon \)-NFA from \( s_i \) to \( s_{i+1} \). So in the \( \epsilon \)-NFA there is a run \( r_1 \) such that the word accepted by \( r \) and \( r_1 \) are same.

**Lemma 4**

This newly constructed non-\( \epsilon \)-Automaton is indeed a DFA

Let suppose that it is not a DFA. So in the non-\( \epsilon \)-Automaton, we must find out a state \( s \) such that from \( s \) there is a transition \( x \) to \( s_1 \) and \( s_2 \) which are distinct. Then in \( \epsilon \)-automaton there is a path from \( s \) to \( s_1 \) with the only \( p \) transition \( x \) and there will also be a distinct path from \( s \) to \( s_2 \) with the only \( p \) transition \( x \). But this does not satisfy lemma-2. So we cannot find such a state \( s \). This newly constructed non-\( \epsilon \)-automaton is indeed a DFA.

We name this newly constructed DFAs of processes as \( A_3 \).
Lemma 5- All the processes’ local DFA will follow the same path of $A_2$
Let process $p$ follows a path as $s \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \ldots \rightarrow s_k$ and $q$ follows a path $s \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \ldots \rightarrow t_k$ in their local DFA, such that they don’t follow the same path in $A_2$. Let in $A_2$ $p$ follows the path $s \rightarrow s^1 \rightarrow s^2 \rightarrow s^3 \ldots \rightarrow s^k$ and this path will be unique as $p$-projection of two different paths in $A_2$ will be different (from lemma-2). Same way $q$ follows the path $s \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \ldots \rightarrow t^k$. Let the last common state in this two sequences will be $s_i$. That means from $s_i$ $p$ and $q$ follow different paths. So in $A_2$ from state $s_i$ there are more than one paths. According to the construction of gadget state if a state $s_i$ has more than one outgoing transition then along all those paths starting from $s_i$ the first action of all processes would be a direct or transitive synchronization action with a common process, say $s$.

For example, let us see figure 4.2. Here from state $s_1$ there are more than one outgoing transition. Now starting from state $s_1$, along two outgoing paths there are actions of process $q$ and $r$ besides $p$. We can see that the first actions of $q$ and $r$ are synchronization action with same process $p$.

Now let us back to the proof. As up to $s_i$ both $p$ and $q$ follows the same path so the first transition for which $p$ distracts the path followed by $q$ is a direct or transitive synchronization action with a process $s$ and same way the first transition for which $q$ distracts the path followed by $p$ is a direct or transitive synchronization action with same process $s$ (according to lemma-2) and obviously this two transition are in different paths starting from $s_i$. Now process $s$’s local automaton can not follow two different paths starting from $s_i$ in a single run. So it is not possible that $p$ and $q$ follows different paths.

Now let suppose between $p$ and $q$ anyone is $s$ itself. Let $p = s$. Then from $s_i$ the first transition of $s$ for which $s$ distracts the path followed by $q$ is a direct or transitive synchronization action with some process $k$, say $sk(x_1, y_1)$ and same way the first transition of $q$ for which $q$ distract the path followed by $s$ is a direct or transitive synchronization action with process $s$ and obviously this two transition are in different paths starting from $s_i$. Now process $s$’s local automaton can not follow two different paths starting from $s_i$ in a single run. So it is not possible that $s$ and $q$ follow different paths.

Definition: Direct and Jumped transition of local DFA of a process
Let from state $s$ to $t$ there is a transition $x$ in the local DFA of process $p$. Then we call this transition a direct transition if in the corresponding $\epsilon$-automaton of the DFA there is a transition $x$ from $s$ to $t$, else we call this transition a jumped transition.

Lemma 6- A transition $x$ from state $s$ to $t$ in the DFA of $p$ can not be direct and jumped transition both.
For contradiction let us suppose, it is possible. Then there are two state $s$ and $t$ such that from $s$ to $t$ there is a direct and jumped transition $x$ in the DFA. Then in the corresponding $\epsilon$-Automaton of the DFA there is a transition $x$ from $s$ to $t$ and there is a path $s \xrightarrow{\epsilon} s_1 \xrightarrow{\epsilon} s_1 \ldots \xrightarrow{\epsilon} s_1 \xrightarrow{x} t$ from state $s$ to $t$. So there are multiple outgoing transitions from $s$ and there are multiple incoming transitions towards state $t$. Thus $s$ and $t$ must be a component of some gadget states. In a gadget state between two state components there are at most one path is possible. So $s$ and $t$ must reside in two different gadget states. But between two state components of two different gadget state, if there is a path then the path consists of more than one sequential transition. But we have already said there is a single transition $x$ from $s$ to $t$ and $s$ and $t$ both are components of two different gadget states. So what we assumed was wrong. A transition $x$ from state $s$ to $t$ in the DFA of $p$ can not be Direct and Jumped transition both.
Lemma 7- A jumped transition must be a synchronization action
Let suppose from \( s \) to \( t \) there is a jumped transition \( x \). That means there is a path \( s \xrightarrow{\epsilon} s_1 \xrightarrow{\epsilon} s_{1\ldots s_{k-1}} \xrightarrow{\epsilon} s_k \xrightarrow{x} t \) from state \( s \) to \( t \) in the corresponding \( \epsilon \)-automaton of the DFA . Let \( x \) be a local action of process \( p \). Then from \( s_{k-1} \) to \( s_k \) there would be a \( p \) transition, otherwise two consecutive transitions would not have a common participator. That means there would not be a jumped transition form \( s \) to \( t \) but it would be \( s \) to \( s_{k-1} \). So, \( x \) can not be a local action. It must be synchronization action.

Lemma 8- \( L(A_2) = L(A_3) \)
First we want to prove that if a word \( w \) is accepted by \( A_3 \) then that is also accepted by \( A_2 \) also. Let \( w \) is not accepted by \( A_2 \). We can break \( w \) as \( w_1w_2 \) such that up to \( w_1 \) is accepted by \( A_2 \) and the first transition of \( w_2 \) is not accepted by \( A_2 \). Now the corresponding run in \( A_2 \) which accept \( w_1 \) be \( s_1 \rightarrow s_2 \rightarrow \ldots s_{k-1} \rightarrow s_k \). All processes’ DFA follow same path in \( A_2 \). As \( A_2 \) is deterministic so all DFA follow the path \( s_1 \rightarrow s_2 \rightarrow \ldots s_{k-1} \rightarrow s_k \). Now we need to understand some observations:

Observation 4: Let us suppose that \( w_1 \) is accepted by \( A_3 \). Then process \( p \) resides at state \( s \) such that at state \( s \) it accepts \( \Pi_p(w_1) \). We can show that \( s \in \{s_1,s_2\ldots s_k\} \). Let us suppose that it is not the case, that means when \( p \) accepts \( \Pi_p(w_1) \), it will be in some state \( s_j \notin \{s_1,s_2\ldots s_k\} \). But as all processes follow the same path as proved in lemma-5 \( s_j \) must be in the path \( s_1 \rightarrow s_2 \rightarrow \ldots s_{k-1} \rightarrow s_k \). So \( p \) follows the path \( s_1 \rightarrow s_2 \rightarrow \ldots s_{k-1} \rightarrow s_k \) such that \( \Pi_p(s_1 \rightarrow s_2 \rightarrow \ldots s_{k-1} \rightarrow s_k) = \Pi_p(w_1) \), i.e., \( \Pi_p(s_1 \rightarrow s_2 \rightarrow \ldots s_{j-1} \rightarrow s_j) = \Pi_p(s_1 \rightarrow s_2 \rightarrow \ldots s_{k-1} \rightarrow s_k) \) and \( \Pi_p(s_1 \rightarrow s_2 \rightarrow \ldots s_{k-1} \rightarrow s_k) \neq \Pi_p(w_1) \). That means it can not be the case that \( p \) accepts only \( \Pi_p(w_1) \), and it will be in some state \( s_j \notin \{s_1,s_2\ldots s_k\} \). End of observation-4’s proof

Observation 5: Let the first transition in \( w_2 \) be \( x \). Our claim is that \( x \) must be a direct transition of some process \( p \) from state \( s_k \).
For contradiction let us suppose, it is not true. We assume that after accepting \( \Pi_p(w_1) \), process \( p \) will be in some state \( s_j \notin \{s_1,s_2\ldots s_k\} \) in it’s local DFA and there is a \( x \) transition from state \( s_j \). As in \( A_2 \) the last transition of \( s_1 \rightarrow s_2 \rightarrow \ldots s_k \) is a non-\( p \) action and \( p \) follows this path, so the \( x \) transition from \( s_j \) can not be a direct transition, otherwise \( p \) would follow a different path, not same with the process involved in the last transition of \( s_1 \rightarrow s_2 \rightarrow \ldots s_k \). So it must be jumped transition. From lemma-7 we can say this jumped transition must be a synchronization action. Let it is a synchronization action \( pq(a,b) \) between \( p \) and \( q \). Let the corresponding jumped transition is \( s_j \xrightarrow{pq(a,b)} s_m \). Now after accepting \( \Pi_q(w_1) \), process \( q \) will be in some state \( s_i \in \{s_1,s_2\ldots s_k\} \) in it’s local DFA and there is a \( pq(a,b) \) transition from state \( s_i \).
Let the transition \( x \) from \( s_j \) to \( s_m \) is corresponds to the transition \( x \) from \( s_{m-1} \) to \( s_m \) along the path \( s_1 \rightarrow s_2 \rightarrow \ldots s_k \) in the \( \epsilon \)-NFA of \( p \) and \( q \). As \( p \) has a jumped transition to \( s_m \), so towards \( s_{m-1} \) there is no \( p \) transition along the path.
As in the \( \epsilon \)-NFA of \( p \) there is no \( p \) transition towards \( s_{m-1} \), so in the \( \epsilon \)-NFA of \( q \) there must be a \( q \) transition towards \( s_{m-1} \), otherwise two consecutive actions(one is \( pq() \) and other is the transition towards \( s_{m-1} \)) don’t have a common participator - this will not be valid.
In the path \( s_1 \rightarrow s_2 \rightarrow \ldots s_k \) the last \( p \) transition is towards \( s_j \), so \( s_m \notin \{s_1,s_2\ldots s_k\} \). As in the \( \epsilon \)-NFA of \( q \) there is a \( q \) transition towards \( s_{m-1} \) and from \( s_{m-1} \) to \( s_m \) there is a \( pq() \).
transition, so before doing synchronization action q’s local DFA must be in \( s_{m-1} \) because q follows the path \( s_1 \rightarrow s_2 \ldots s_k \rightarrow \ldots s_{m-1} \rightarrow s_m \) and from there will be the direct transition(synchronization action). As from \( s_{m-1} \) there is a direct transition, so \( s_{m-1} \) is either \( s_k \) or some state \( s_k \notin \{s_1,s_2,\ldots,s_k\} \) in the path \( s_1 \rightarrow s_2 \ldots \rightarrow s_k \). But after accepting \( \Pi_q(w_1) \) process q have to stay \( s_i \in \{s_1,s_2,\ldots,s_k\} \) according to observation-4. So to make \( pq() \) q have to stay \( s_k \) i.e \( s_i = s_{m-1} = s_k \). Thus \( pq() \) must be a direct transition of a process (here q) from state \( s_k \).

Let process p resides in state \( s_k \) after accepting only \( \Pi_p(w_1) \). Let there is a jumped transition from \( s_k \) to state \( s_m \). As p has a jumped transition to \( s_m \), so towards \( s_{m-1} \) there is no p transition along the path i.e \( s_{m-1} \notin \{s_1,s_2,\ldots,s_k\} \). So towards \( s_{m-1} \) there must be a q transition along the path \( s_1 \rightarrow s_2 \ldots s_k \rightarrow \ldots s_{m-1} \rightarrow s_m \). So before doing synchronization action q must be in \( s_{m-1} \). But \( s_{m-1} \notin \{s_1,s_2,\ldots,s_k\} \) So, after accepting only \( \Pi_q(w_1) \) process q can not stay \( s_{m-1} \). That means from \( s_k \) a jumped transition can not be taken. So from the above arguments it can be said that the first transition \( x \) in \( w_2 \) must be a direct transition of some process p from state \( s_k \).

End of observation-5’s proof

Now, in local DFA of p, from \( s_k \) if there is a direct transition, then in \( A_2 \) from \( s_k \) there is also a transition from \( s_k \). So the first transition of \( w_2 \) is accepted by \( A_2 \). That means, our assumption was wrong. So w will be accepted by \( A_2 \).

If a word \( w \) is accepted by \( A_2 \). Then \( \Pi_p(w) \) is accepted by all process p’s \( \epsilon \)-NFA. And from lemma-3 if \( \Pi_p(w) \) is accepted by \( \epsilon \)-NFA of p then it is also accepted by DFA of p also. So, \( w \) is accepted by \( A_3 \). So, \( L(A_2) = L(A_3) \)

From Observation2 \( \Pi_{\Sigma^1}(L(A_2)) = L(S) \) and from lemma-8 \( L(A_2) = L(A_3) \). So \( \Pi_{\Sigma^1}(L(A_3)) = L(S) \). Thus we solve each subproblem over a connected architecture, which actually implements the original overall specification.
Chapter 5

Implementation Using Gossip Protocol

5.1 Construction of intermediate automaton:

If in the specification $S$, the start state has any incoming transition towards it, we modify $S$ to $S_1$ in the following way such that the start state of $S_1$ has no incoming transition towards it and $L(S) = L(S_1)$.

(a) Create a new state $s'$.

(b) For every state $s_k$ if there is a transition $p(a)$ from start state $s$ to $s_k$, then make a transition $p(a)$ from $s'$ to $s_k$.

(c) For every state $s_i$ if there is a transition $p(b)$ from state $s_i$ to start state $s$, then delete that transition and make a new transition $p(b)$ from $s_i$ to $s'$.

Lemma 1 - $L(S)$ and $L(S_1)$ are same:

If there is a run $w$ in $S$, say $s \xrightarrow{p(a)} s_1 \xrightarrow{q(a)} s_2 \ldots \xrightarrow{r(a)} s \xrightarrow{p(b)} s_j \ldots s_m$, then except the first occurrence of start state $s$ change every occurrence of $s$ with $s'$. This will be a run of $S_1$.

Otherwise, if there is a run $w'$ in $S_1$, say $s \xrightarrow{p(a)} s_1 \xrightarrow{q(a)} s_2 \ldots \xrightarrow{r(a)} s' \xrightarrow{p(b)} s_j \ldots s_m$, then change every occurrence of $s'$ with $s$. This will be a run of $S$.

Construction of Intermediate Automaton from the Modified Specification:

(a) If there is a state transition say $s \xrightarrow{p(a)} t$ and $t$ has only one incoming and one outgoing state transition say $t \xrightarrow{q(b)} r$, then replace the label $p(a)$ by $p(a)$ then $pq(q,b)$.

(b) Let suppose there is a state $t$, now if $t$ has either more than one incoming transition towards it or more than one outgoing transition from it then we form a gadget state $G_t$ which simulates state $t$. Let us suppose there are $y$ different incoming edges incident to $t$ labelled as $q_1(a), q_2(b) \ldots q_y(c)$ and there are $x$ number of outgoing edges from $t$ labelled as $p_1(a), p_2(b) \ldots p_x(d)$. Then our gadget state construction will be following -

In the gadget state $G_t$ there will be two column of states. The first column is the set of states each corresponds to an incoming transition towards state $t$. The second column consists of the set of states each corresponds to one outgoing transition from state $t$.

Let in the first column, state $s_1$ corresponds to incoming action $x$ of process $p$, say $p(a)$ and in the second column, state $s_2$ corresponds to outgoing transition $y$, say $q(b)$, then between $s_1$ and $s_2$ process $p$ will synchronize with $q$ as $pq(q,b)$. This is done, because the
next transition after $pq(q, b)$ will be $q(b)$, so every two consecutive action has a common process participator- this condition will be still remained true.

**Lemma 2** - $\Pi_{\Sigma_1}(L(A_2)) = L(S_1)$ :
If we take the projection of any word $w$ over local alphabet $\Sigma_1$, then clearly $\Pi_{\Sigma_1}(w)$ is a valid word in $S_1$. Similarly if a word $u$ is accepted, then we can construct following the rules of constructing $A_2$ a word $w$ such that $\Pi_{\Sigma_1}(w) = u$

### 5.2 Constructing the Distributed Automata from the Intermediate Automaton:

Now we are going to construct the local automaton of each process $p \in \mathcal{P}$. Here we apply gossip protocol and the corresponding trace theory as discussed in [3], as commutative property holds $A_2$ and thus it is now applicable for this. But here no parallel action will be occured, everything is sequentialized. So only seeing the local final state of some
process \( p \), we can decide whether a word \( w \) is globally accepted or not by the Distributed Automata. Thus we solve each subproblem over a connected architecture, which actually implements the original overall specification.
Chapter 6

Implementation Maintaining Parallelism

In the previous chapter we built a sequential distributed automata. So in some sense, that was not an ideal case. So, here we mainly focus on parallelism. Let's first modify the specification, so that there will be no incoming transition towards start state.

6.1 Construction of intermediate automaton:

If in the specification $S$, the start state has any incoming transition towards it, we modify $S$ to $S_1$ in the following way such that the start state of $S_1$ has no incoming transition towards it and $L(S) = L(S_1)$.

i. Create a new state $s'$.

ii. For every state $s_k$ if there is a transition $p(a)$ from start state $s$ to $s_k$, then make a transition $p(a)$ from $s'$ to $s_k$.

iii. For every state $s_i$ if there is a transition $p(b)$ from state $s_i$ to start state $s$, then delete that transition and make a new transition $p(b)$ from $s_i$ to $s'$.

Lemma 1 - $L(S)$ and $L(S_1)$ are same:

If there is a run $w$ in $S$, say $s \xrightarrow{p(a)} s_1 \xrightarrow{q(a)} s_2 \ldots s_k \xrightarrow{r(a)} s \xrightarrow{p(b)} s_j \ldots s_m$, then except the first occurrence of start state $s$ change every occurrence of $s$ with $s'$. This will be a run of $S_1$.

Otherway, if there is a run $w'$ in $S_1$, say $s \xrightarrow{p(a)} s_1 \xrightarrow{q(a)} s_2 \ldots s_k \xrightarrow{r(a)} s \xrightarrow{p(b)} s_j \ldots s_m$, then change every occurrence of $s'$ with $s$. This will be a run of $S$.

Some Notation:

We will now introduce two function $\phi : Q \times \Sigma^1 \rightarrow 2^P$ and $f : \Sigma^1 \times Q \times \Sigma^1 \rightarrow 2^P$ where $Q$ is the set of state of $S'$ and $\Sigma^1$ is the set of local action of processes.

- If $s$ has only one outgoing transition $q(a)$ from it, then $\phi(s, q(a)) = \{q\}$.
- If $s$ has other outgoing transition from it besides $q(a)$ then $\phi(s, q(a))$ means all those processes which has participated in some action along the path(s) starting from $s$ with the first transition $q(a)$.
- Let $s$ has a incoming transition $p(a)$ towards it and a outgoing transition $p(b)$ from it, then $f(p(a), s, p(b)) = \{q\}$ if $\phi(s, p(b)) = \{p\}$
- $f(p(a), s, q(b)) = \phi(s, q(b)) - \{p\}$. 

36
6.2 Condition for constructing longest diagonal of diamond:

Here we come with a notion called diamond. By saying a diamond from state $s$ to $t$ ($s \diamond t$) involving processes $p, q, ..., r$, we mean that from state $s$ to $t$ there is a hypercube, so that $p, q, ..., r$ can do their action parallel from $s$ and reach to state $t$.

Let from state $s$ to $t$ there is a diamond where processes $p_1, p_2, ... p_k$ are involved. Here the directed arc from $s$ to $t$ is the longest diagonal of this diamond.

i. If there is one or more than one outgoing transition from $t$, then the following condition must be satisfied -

$\exists$ a outgoing transition from $t$, say $t \xrightarrow{p(a)} t_1$ such that it is not the case that $s \diamond t_1$ via $t$ involving $p_1, p_2, ... p_k$ and $p$.

ii. If there is one or more than one incoming transitions to $s$, then the following condition must be satisfied -

$\exists$ a incoming transition to $s$, say $s_1 \xrightarrow{p(a)} s$ such that it is not the case that $s_1 \diamond t$ via $s$ involving $p_1, p_2, ... p_k$ and $p$.

Figure 6.1: A Sample Specification

Figure 6.2: Putting Longest Diagonal
6.3 Condition for removing redundant transition of specification:

Here, a transition in specification is called redundant, if deleting that transition from specification does not change the accepting language of specification. Below, we give the conditions, which transitions will be remained. Others will be deleted.

Let from state \( s \) to \( t \) there is a \( p \) transition. Now this directed edge will remain iff the following conditions hold:

i. If there is one or more than one outgoing transition from \( t \), then the following condition must be satisfied -
\[ \exists a \text{ outgoing transition from } t, \quad t \xrightarrow{q(a)} t_1 \text{ such that it is not the case that } s \Diamond t_1 \text{ via } t \text{ involving } p \text{ and } q. \]

ii. If there is one or more than one incoming transition to \( s \), then the following condition must be satisfied -
\[ \exists a \text{ incoming transition to } s, \quad s_1 \xrightarrow{q(a)} s \text{ such that it is not the case that } s_1 \Diamond t \text{ via } s \text{ involving } p \text{ and } q. \]

If there is a transition, which can’t be reached from start state, then delete that transition.

We name this modified specification as \( S' \).

Construction of Intermediate Automaton from the Modified Specification:

![Simple Synchronization](image)

Figure 6.3: Simple Synchronization

i. If there is a state transition say \( s \xrightarrow{p(a)} t \) where \( x \) is either some \( p(a) \) or of the type \( \text{par}(p(a), q(b), r(c)) \), .... such that \( P_1 \) be the set of processes which have participated in \( x \) and there is one outgoing state transition, say \( t \xrightarrow{q(a)} r \) where \( y \) is either some \( p(a) \) or of the type \( \text{par}(p'(a), q'(b), r'(c)) \), ...., then replace the label \( x \) by \( x \), then from every process \( p \in P_1 \) put a \( pq(q, b) \) or \( pq(\text{par}(p'(a), q'(b), r'(c)) \), ....) for every \( q(b) \) action involved in \( y \).

ii. Let suppose there is a state \( t \), now if \( t \) has either more than one incoming transition towards it or more than one outgoing transition from it then we form a gadget state \( G_t \) which simulates state \( t \). Let us suppose there are \( y \) different incoming edges incident to \( t \) labelled as \( q_1(a), q_2(b) \), ...., \( q_y(c) \) and there are \( x \) number of outgoing edges from \( t \) labelled as \( p_1(a), p_2(b) \), ...., \( p_x(d) \). Then our gadget state construction will be following -
We now identify all those paths starting from \( t \) such that the first transition will be \( p_1(a) \). Along these path let the set of processes which have done some action be \( P_1 \). Like this way we find out all such \( P_i \) for all outgoing transition \( p_i(a) \) from state \( t \). In the gadget state \( G_t \) there will be two column of states. The first column is the set of states each corresponds to an incoming transition towards state \( t \). The second column consists of the set of states each corresponds to one outgoing transition from state \( t \). Let in the first column, state \( s_1 \) corresponds to incoming action \( x \) which is either some \( p(a) \) or of the type \( \text{par}(p(a), q(b), r(c), ....) \), then replace the label \( x \) by \( x \) then from every process \( p \in P_1 \) put a \( pq(q, b) \) for every \( q(b) \) action involved in \( y \). Here between \( s_1 \) and \( s_2 \) every process \( p \in P_1 \) will synchronize with all process \( p_i \in f(x, t, y) \) sequentially where each synchronization action will be of the form \( pp_i(q, b) \).

\[ \Pi_{\Sigma_1}(L(A_2)) = L(S'') \]

If we take the projection of any word \( w \) accepted by \( A_2 \) over local alphabet \( \Sigma_1 \), then clearly \( \Pi_{\Sigma_1}(w) \) is a valid word in \( S'' \). Similarly if a word \( u \) is accepted, then we can construct following the rules of constructing \( A_2 \) a word \( w \) such that \( \Pi_{\Sigma_1}(w) = u \)

### 6.4 Constructing the Distributed Automata from the Intermediate Automaton:

Now we are going to construct the local automaton of each process \( p \in P \). In the intermediate automaton we replace all non-\( p \) action by epsilon and \( \text{par}(p(a), q(b), r(c), ....) \) by \( p(a) \) as \( p \) has participated in \( \text{par}(p(a), q(b), r(c), ....) \). Let’s call it as \( \epsilon \)-NFA of process \( p \).

**Construction of non-\( \epsilon \)-Automaton of process \( p \) from \( \epsilon \)-NFA of process \( p \)**

We convert the \( \epsilon \)-NFA of process \( p \) to a non-\( \epsilon \)-Automaton of process \( p \) as follows.
A. We take the $\epsilon$-closure of every state of the $\epsilon$-NFA of process $p$.
B. Let state $s$ has the $\epsilon$-closure $Cl_s$. Now let suppose $t \in Cl_s$ and $t$ has an outgoing transition $x$ to $r$, then we make a new transition $x$ from $s$ to $r$
C. If a state $s$ has a $\epsilon$-transition to state $t$ then we delete that $\epsilon$-transition

**Lemma 2** - In the $A_2$ from a state $s$ if there exists $k$ different paths $x_1, x_2, \ldots, x_k$ such that the first transition along path $x_i$ goes to state $s_i$ and the first transition along path $x_j$ goes to state $s_j$ where $s_i$ and $s_j$ are different, then for all process $q$ the first $q$ transition along path $x_i$ will be different from the first $q$ transition along path $x_j$ and there exist some process $p$ such that all processes $p_i$ in the above paths will synchronize directly with $p$ i.e, $pp_i()$ or will synchronize with $p$ transitively, that means $pp_1(), p_1p_2, \ldots, p_{i-1}p_i()$. We say, $p_i$’s this first action as **transitive synchronization action with $p$**.

This is because of gadget state. If from a state $s$ there are more than one outgoing path then $s$ must belongs to a gadget state $G$. And in the gadget state from a component state $s$, for all process $q$ the first $q$-transition along two different paths $x$ and $y$ must be synchronization actions and they are different in the argument as the specification is deterministic (for the synchronization action $pq(r, a)$ the argument is $(r, a)$). So the first $p$ transition along path $x$ will be different from the first $p$ transition along path $y$. More over in the gadget state we must find out a process $s$ which will synchronize directly or transitively with all processes.

**Observation 1** - Like the $\epsilon$-NFA in this newly constructed automaton every state is a final state.

**Lemma 3**: The language accepted by this $\epsilon$-NFA and non-$\epsilon$-automaton of process $p$ are same

Let there is a run $r$ in the $\epsilon$-NFA of $p$, say $s \rightarrow s_1 \rightarrow s_1 \rightarrow s_1 \ldots \rightarrow s_1$. Now let $s_i$ be the first state in the run from which there is a $p$ action $x$ to $s_{i+1}$, then in the newly constructed automaton there will also be a transition $x$ from $s$ to $s_{i+1}$. Now in $\epsilon$-NFA from $s_{i+1}$ we find out the first state $s_j$ along the run $r$, from which there is a $p$ action $y$ to $s_{j+1}$, then in the newly constructed automaton there will also be a transition $y$ from $s_{i+1}$ to $s_{j+1}$. In this way we will reach the state $s_k$ so that there is a $p$ action towards $s_k$ but after $s_k$ there is no $s_j$ along the run $r$, from which there is a $p$ action $z$. So in the newly constructed automaton there is a run $r_1$ up to $s_k$ such that the word accepted by $r$ and $r_1$ are same.

Let there is a run $r$, say $s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{c} \ldots \xrightarrow{d} s_k$ in the newly constructed non-$\epsilon$-automaton of $p$. Then for each $s_i \xrightarrow{\epsilon} s_{i+1}$ (0 $\leq i$ $\leq k - 1$) in the non-$\epsilon$-automaton there will be direct transition $x$ from $s_i$ to $s_{i+1}$ or there will be a path $s \xrightarrow{\epsilon} s_1 \xrightarrow{\epsilon} s_1 \ldots \xrightarrow{\epsilon} s_1 \xrightarrow{\epsilon} t$ in the $\epsilon$-NFA from $s_i$ to $s_{i+1}$. So in the $\epsilon$-NFA there is a run $r_1$ such that the word accepted by $r$ and $r_1$ are same.

**Lemma 4** - **This newly constructed non-$\epsilon$-Automaton is indeed a DFA**

Let suppose that it is not a DFA. So in the non-$\epsilon$-Automaton, we must find out a state $s$ such that from $s$ there is a transition $x$ to $s_1$ and $s_2$ which are distinct. Then in $\epsilon$-automaton there is a path from $s$ to $s_1$ with the only $p$ transition $x$ and there will also be a distinct path from $s$ to $s_2$ with the only $p$ transition $x$. But this does not satisfy lemma-2. So we can’t find such a state $s$. This newly constructed non-$\epsilon$-automaton is indeed a DFA.

We name this newly constructed DFAs of processes as $A_3$
Lemma 5- All the processes’ local DFA will follow the same path of $A_2$

Let process $p$ follows a path as $s \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \ldots \rightarrow s_k$ and $q$ follows a path $s \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \ldots \rightarrow t_k$ in their local DFA, such that they don’t follow the same path in $A_2$. Let in $A_2$ $p$ follows the path $s \rightarrow s^1 \rightarrow s^2 \rightarrow s^3 \ldots \rightarrow s^k$ and this path will be unique as $p$-projection of two different paths in $A_2$ will be different(from lemma-2). Same way $q$ follows the path $s \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \ldots \rightarrow t^k$. Let the last common state in this two sequences will be $s_i$. That means from $s_i$, $p$ and $q$ follow different paths. So in $A_2$ from state $s_i$ there are more than one paths. According to the construction of gadget state if a state $s_i$ has more than one outgoing transition then along all those paths starting from $s_i$ the first action of all processes would be a direct or transitive synchronization action with a common process, say $s$.

Now let us back to the proof. As up to $s_i$ both $p$ and $q$ follows the same path so the first transition for which $p$ distracts the path followed by $q$ is a direct or transitive synchronization action with a process $s$ and same way the first transition for which $q$ distracts the path followed by $p$ is a direct or transitive synchronization action with same process $s$(according to lemma-2) and obviously this two transition are in different paths starting from $s_i$. Now process $s$’s local automaton can not follow two different paths starting from $s_i$ in a single run. So it is not possible that $p$ and $q$ follows different paths.

Now let suppose between $p$ and $q$ anyone is $s$ itself, let $p = s$. Then from $s_i$, the first transition of $s$ for which $s$ distracts the path followed by $q$ is a direct or transitive synchronization action with some process $k$, say $sk(x_1,y_1)$ and same way the first transition of $q$ for which $q$ distract the path followed by $s$ is a direct or transitive synchronization action with process $s$ and obviously this two transition are in different paths starting from $s_i$. Now process $s$’s local automaton can not follow two different paths starting from $s_i$ in a single run. So it is not possible that $s$ and $q$ follow different paths.

Lemma 8- $L(A_2) = L(A_3)$

Here all process follows the same path and the ordering of local actions are also maintained, as if $s$ to $t$ there is a $p(a)$ and from $t$ or in the forward path of $t$, there is a $q(b)$ transition, then $q(b)$ can’t occur before $p(a)$,as there is a $pq(q,b)$ or some $px(x,c)$ which must happen before $q(b)$. Thus if a word $w$ is accepted by $A_3$, it is also accepted a word $w$ is accepted by $A_2$, then from the construction of $\epsilon-$ NFA, it is clear that $w$ is also accepted by this $\epsilon-$ NFA. And we have also seen that the language accepted by $\epsilon-$ NFA and non-$\epsilon-$ DFA are same. Thus $w$ will also be accepted $A_3$. So, $L(A_2) = L(A_3)$

We have already seen that $\Pi_{\Sigma^1}(L(A_2)) = L(S)$ and from lemma-8 $L(A_2) = L(A_3)$. So $\Pi_{\Sigma^1}(L(A_3)) = L(S)$. Thus we solve each subproblem over a connected architecture, which actually implements the original overall specification.

6.5 An Example:
Figure 6.6: A Specification

Figure 6.7: After Adding Diagonal

Figure 6.8: After Removing Redundant Transition
Figure 6.9: After construction of Gadget State

Figure 6.10: p’s NFA

Figure 6.11: q’s NFA
Figure 6.12: r’s NFA

Figure 6.13: p’s DFA

Figure 6.14: q’s DFA

Figure 6.15: r’s DFA
6.6 Scope of improvement:

Here we make the specification by doing cross product of \( p, q \) and \( r \)'s automata and get the specification as shown in figure- 16 and 17. But we find out that, no transition can be deleted from specification as redundant as no transition satisfies the condition as discussed in section-3. Then there much more synchronization actions will be required. But actually figure is a right implementation. So in the next chapter we will introduce a new concept called bisimilarity, which will solve this type of problem.

![Figure 6.16: Local automata of Process \( p, q \) and \( r \)](image1)

![Figure 6.17: Cross Product of Local Automata of Process \( p, q \) and \( r \)](image2)
Chapter 7

Bisimulation

7.1 Introduction

As in figure-6.17, we can find that state 1, 2, 3, 4 are somehow equivalent. So, here we want to club the states in terms of equivalence. But here we choose bisimulation equivalence instead of language equivalence. Instead of taking the ordinary bisimulation relation we modify the definition of bisimilarity. Here we do so, because in the transition system, more than one process are involved.

A relation $B_p \subseteq S \times S$ between $S$ and $S$ itself is a bisimulation relation with respect to $p$ iff for all $s$ and $s'$, if $B_p(s, s')$ then $s \leq_p s'$ and $s' \leq_p s$

Let $s, s'$ be two state. We say $s \leq_p s'$ iff the following conditions hold:

A. For every $s_1$ if $s \xrightarrow{p(a)} s_1$ with $P$ be the set of process such that for every process $q$ in $P$, $s \leq_q s_1$ and $s_1 \leq_q s$, then there exists a $s_2$ such that $s' \xrightarrow{p(a)} s_2$ with $Q$ is the set of process where for every process $q$ in $Q$ $s' \leq_q s_2$, $s_2 \leq_q s_1$ and $P = Q$.

B. Now any one of following conditions must be hold-

- there exists a $s_2$ such that $s' \xrightarrow{r(b)} s_2$, $\leq_p (s, s_2)$ and $\leq_p (s, s_2')$.
- For every $s_3$ if $s \xrightarrow{q(a)} s_3$, then either $\leq_p (s_3, s')$ and $\leq_p (s_3, s_2)$
  or there exists a $s_2$ such that $s' \xrightarrow{r(b)} s_2$, $\leq_p (s_3, s_2)$ and $\leq_p (s_2, s_3)$.

7.2 Largest $\leq_p$ relation

A $\leq_p$ relation $\leq_p \subseteq S \times S$ is called the largest $\leq_p$ (less than with respect to $p$) relation between $S$ and $S$ itself, iff for all less than relation $\leq_p \subseteq S \times S$, it is the case that $\leq_p \subseteq \leq_p$.

Algorithm for computing largest $\leq_p$ relation:

We define a sequence of relations $\leq_0^p, \leq_1^p, \leq_2^p, \ldots$ on $S \times S$ as follows:

- $\forall p \in P$, $\leq_0^p = S \times S$ where $P$ is the set of processes.
- $\leq_{i+1}^p (s, s')$ iff
  - $\leq_i^p (s, s')$
  - For every $s_1$ if $s \xrightarrow{p(a)} s_1$ with $P$ is the set of process such that for every process $q$ in $P$ $\leq_i^p (s, s_1)$ and $\leq_i^p (s_1, s)$, then there exists a $s_2$ such that $s' \xrightarrow{p(a)} s_2$,
We define a sequence of relations that $P = Q$ and for every process $q$ in $Q$
$\leq^i_q (s', s_2)$, $\leq^i_q (s_2, s')$, $\leq^i_p (s_1, s_2)$ and $\leq^i_p (s_2, s_1)$.

C. Now any one of following conditions must be hold-
- there exists a $s_2$ such that $s \xrightarrow{r(b)} s_2$, $\leq^i_p (s, s_2)$ and $\leq^i_p (s_2, s)$.
- For every $s_3$ if $s \xrightarrow{q(a)} s_3$, then either $\leq^i_p (s_3, s')$ and $\leq^i_p (s', s_3)$ or there exists a $s_2$ such that $s \xrightarrow{r(b)} s_2$, $\leq_p (s_3, s_2)$ and $\leq^i_p (s_2, s_3)$.

As the specification is finite, so this algorithm will end for some $n$ and we get $\leq_{n+1}^n \leq_{n+1}$.

We name this relation as $\leq_p^i$. This is indeed a $\leq_p$ relation, as when $\leq_{n+1}^n \leq_{n+1}$ then the definition of $\leq_{n+1}^n$ is similar as ordinary $\leq_p$ relation.

Lemma-1 : $\leq_p^i$ is the largest $\leq_p$ relation.

Now we want to prove that it is largest. Let take any $\leq_p$ relation $\leq_p^i$. We want to show that if $\leq_p^i (s, s')$ then $\leq_p (s, s')$. We show this by proving that $\leq_{p+1}^i \leq_p$ for all $i$. So we do an induction on $i$.

Clearly, $\leq_p^i \leq_{p+1}^i$. Let assume $\leq_p^i$ is contained in $\leq_{p+1}^i$. Now we take any pair $(s, s')$ such that $\leq_p^i (s, s')$. According to the first condition, let there is a $p$ transition from $s$ to $s_1$ with $P$ is the set of process such that for every process $q$ in $P$ $\leq^i_q (s, s_1)$ and $\leq^i_q (s_1, s)$, then there exists a $s_2$ such that $s \xrightarrow{p(a)} s_2$ with $Q$ is the set of process such that for every process $q$ in $Q$ $\leq^i_q (s_q(s), s_2)$, $\leq^i_q (s_2, s_q(s'))$, $\leq^i_q (s_2, s_1)$ and $\leq^i_p (s_2, s_1)$. But $\forall p \in P$, $\leq_{p+1}^i \leq_{p+1}^i$, so $\leq_{p+1} (s, s')$. Similarly all the conditions hold. Thus $\leq_{p+1}^i (s, s')$.

7.3 Largest Bisimulation

Here we want to construct the largest bisimulation relation, so that the resultant local automaton of a process $p$ would be as much as succinct.

A bisimulation relation $B_p \subseteq S \times S$ is called the largest bisimulation relation between $S$ and $S$ itself iff for all bisimulation $B_p' \subseteq S \times S$ it is the case that $B_p' \subseteq B$.

Algorithm for computing largest bisimulation:

We define a sequence of relations $B_{p_0}^0, B_{q_0}^0, B_{p_1}^1, B_{q_1}^1, \ldots$ on $S \times S$ as follows:

- $\forall p \in P, B_{p_0}^0 = S \times S$.
- $B_{p_{i+1}}^i (s, s')$ iff
  - $B_{p_i}^i (s, s')$.
  - For every $s_1$ if $s \xrightarrow{p(a)} s_1$ with $P$ is the set of process such that for every process $q$ in $P$ $B_{q_i}(s, s_1)$, then there exists a $s_2$ such that $s \xrightarrow{p(a)} s_2$ with $Q$ is the set of process such that $P = Q$ and for every process $q$ in $Q$ $B_{q_i}(s', s_2)$, $B_{p_i}(s_1, s_2)$.
  - Now any one of following conditions must be hold-
    - there exists a $s_2$ such that $s \xrightarrow{r(b)} s_2$ and $B_{p_i}^i (s, s_2)$.
    - For every $s_3$ if $s \xrightarrow{q(a)} s_3$, then either $B(s_3, s')$ or there exists a $s_2$ such that $s \xrightarrow{r(b)} s_2$ and $B_{p_i}^i (s_3, s_2)$.
• Vice versa for s and s′

As the specification is finite, so this algorithm will end for some n and we get \( B_p^n = B_p^{n+1} \). We name this relation as \( B_p^* \). This is indeed a bisimulation, as when \( B_p^n = B_p^{n+1} \) then the definition of \( B_{n+1} \) is similar as ordinary bisimulation relation.

**Lemma-2 :** \( B_p^* \) is the largest bisimulation relation

Now we want to prove that it is largest. Let's take any bisimulation \( B_p \). We want to show that if \( B_p(s, s′) \) then \( B_p^*(s, s′) \). We show this by proving that \( B_p \subseteq B_i^p \) for all i. So we do an induction on i.

Clearly, \( B \subseteq B_0^0 \). Let's assume \( B_p \) is contained in \( B_i^p \). Now we take any pair \((s, s′)\) such that \( B_p(s, s′) \). According to the first condition let there is a \( p \) transition from \( s \) to \( s_1 \), then there will also be a \( p \) transition from \( s′ \) to some \( s_2 \) such that \( B_p(s_1, s_2) \). But \( B \subseteq B_i^p \), so \( B_p^i(s_1, s_2) \). Similarly all the conditions hold. Thus \( B_i^p(s, s′) \).

**Proposition-1 :** If \( B \) and \( B′ \) are two bisimulation on \( S \times S \), then \( B \circ B′ \) is also a bisimulation.

We define \( B \circ B′ = \{(s, s_2) | \exists s_1 [B(s, s_1) \land B(s_1, s_2)]\} \). We rename this \( B \circ B′ \) as \( B_1 \)

First we want to prove that \( s \leq s_2 \)

Let there is a \( p \) transition from \( s \) to \( s′ \), then there is some \( s_1 \) such that there is a \( p \) transition from \( s_1 \) to \( s′ _1 \) and \( B(s′_1, s′_1) \). As \( B_1(s_1, s_2) \) and there is a \( p \) transition from \( s_1 \) to \( s_1′ \), so there will also be a \( p \) transition from \( s_2 \) to \( s_2′ \) and \( B′(s_1′, s_2′) \). So, \( B_1(s′, s_2′) \)

Let's suppose from \( s \) there is a non-\( p \) transition to \( s′ \) and \( B(s′, s_1) \). Now \( B′(s_1, s_2) \). So there is a transition from \( s \) to \( s′ \) and \( B_1(s′, s_2) \).

Let's suppose from \( s \) there is a non-\( p \) transition to \( s′ \) and there is a non-\( p \) transition from \( s_1 \) to \( s′_1 \) such that \( B(s′_1, s′_1) \). Now there are three possibility to make \( B′(s_1, s_2) \). One is \( B′(s_1′, s_2) \), another is- there is a non-\( p \) transition from \( s_2 \) to \( s′_2 \) and \( B′(s′_1, s′_2) \) and the last one is - there is a non-\( p \) transition from \( s_2 \) to \( s′_2 \) and \( B′(s_1′, s_2′) \). So either \( B_1(s_1, s_2) \) or \( B_1(s′_1, s_2) \) or \( B_1(s, s_2) \).

Now suppose there is a non-\( p \) transition from \( s_1 \) to \( s′_1 \) such that \( B(s, s′_1) \). Now there are three possibility to make \( B′(s_1, s_2) \). One is \( B′(s_1′, s_2) \), another is- there is a non-\( p \) transition from \( s_2 \) to \( s′_2 \) and \( B′(s′_1, s′_2) \) and the last one is - there is a non-\( p \) transition from \( s_2 \) to \( s′_2 \) and \( B′(s_1′, s_2′) \). So \( s \leq s_2 \)

Similar argument for to proof that \( s_2 \leq s \). Thus \( B_1 \) is a bisimulation.

**Lemma-3 :** \( B^* \) is an equivalence relation

If we can divide the states of the specification in some equivalence classes, then it will be nice to treat such equivalence classes as a single state in local automaton. Let's suppose \( B^* \) is not transitive. Let \( B^*(s, s_1) \) and \( B^*(s_1, s_2) \) but \( (s, s_2) \) does not belongs to \( B^* \). Then we compose \( B^* \) with itself and get a new bisimulation. Now \( (s, s_2) \) will belong to \( B^* \). But we have just proved that \( B^* \) is the largest bisimulation. So our assumption was wrong and \( B^* \) is a transitive relation.

By definition any bisimulation relation is reflexive and symmetric. We have just shown that \( B^* \) is a bisimulation relation. Thus it is reflexive and symmetric. Hence \( B^* \) is a equivalence
relation.

An Example

![Figure 7.1: A Specification](image)

Here the $t_1$ and $t_2$ are bisimilar with respect to process $p$, as from $t_1$ there is a $p$ transition to $t_4$ and $(t_1,t_4)$ are only bisimilar with respect to process $q$, similarly from $t_1$ there is a $p$ transition to $t_4$ and $(t_1,t_4)$ are only bisimilar with respect to process $q$ and also $(t_1,t_4)$ are $p$-bisimilar. Same way we can see that other equivalence classes of $p$ is $(t_3,t_4,t_5)$ and $t_6$.

7.4 Construction of local automaton:

We denote the equivalence class of process $p$ corresponding to a state $s$ in the specification as $[s]_p$.

(a) Let from state $s$ to $t$ there is a $p(a)$ transition. Suppose $q$ is a process such that $s_q t$ or $s_q t'$ where $s \equiv_p s'$ and $t \equiv_p t'$ and there are $n$ number of such processes. Now create a sequence of $n$ intermediate states $s_1, s_2, ..., s_n$ between class(state) $[s]_p$ and $[t]_p$.

(b) Put a $p$ transition after the above stated synchronization actions.

(c) Suppose $q$ is a process such that $t_q s$ or $t_q s'$ where $s \equiv_p s'$ and $t \equiv_p t'$ and there are $m$ number of such processes. Now create a sequence of $m$ synchronization action of such process with after the $p$ local action.

(d) If there is a process $q$ such that, there is a synchronization action with $p$ before the local action but not after that, then put a $pq$ synchronization action after that.

(e) If from state $s$ to $t$ there is a $q(a)$ transition and $sn \equiv_p t$, then look from class(state) $[s]_q$ to $[t]_q$ in the local automaton of $q$, if there is a single synchronization action with $p$, put a $pq(a)$ synchronization transition from $[s]_p$ to $[t]_p$, else two $pq$ between $[s]_p$ and $[t]_p$.

Mapping function from global state to states of specification:

Here we introduce a mapping function $f$ from global state space to states of specification.

(a) The start state is $S = ([s]_p, [s]_q, [s]_r, ...)$ where $s$ is the start state in the specification and this global state is simulating state $s$, so $f(S) = s$.

(b) From a global state $S$, simulating $s$, if we reach a global state $T$ only by doing a synchronization action, then say $T$ is simulating $s$, so $f(T) = s$. 

49
(c) From a global state $S$, simulating $s$, if we reach a global state $T$ only by doing a local action $p$, then there must be a $p$ action from $s$ to some state $t$ in specification. Say $T$ is simulating $t$, so $f(T) = t$. (We will see this proof in proposition-2)

(d) Every reachable global state is a final state.

**Proposition-2 :** Whenever some process $q$ is not in $[t]_q$, where $t$ is the current simulating state, there can't be a $q$ local transition from current $q$ local state.

Let $s$ is the start state of specification, so initially all process $p$ will resides in $[s]_p$. Now lets suppose from global state $S_k$ which simulates $s$ there is a $p$ transition from it's $p$- component $[s]_p$ to $[t]_p$ which simulates $t$. Now what will be the possible state of process $q(q \neq p)$ after doing $p$ and before occuring any other local action?

It can be in -

(a) $[s]_q$
(b) $[t]_q$
(c) some state $[t_1]_q$ where $t_qt_1$
(d) some state $s_1$ where $s \equiv_p s_1$ and $s_qs_1$.

In general when the global state is simulating state $t$ accepting $w$, following are the possibilities, in which local state a process $q$ can reside after accepting $w$ globally.

![Figure 7.2: A Sample Specification](image)

(a) There is a intermediate state $t'$ in the path from start state to $t$ such that in that path the outgoing transition from $t'$ is to state $t_1$ where $t_1$ and $t'$ are not bisimilar with respect to $q$ and $q$ will be in $[t']_q$.

Now we want to show that from $[t']_q$ no $q$ local action can be triggerred. We assume that the transition from $t'$ to $t_1$ is a $p$ transition. Here $p$ has already reached to $t_1$, but $q$ is still $t'$. From $t'$ there is a $q$ transition. As after $p$ action, $q$ is still in $t'$, so we can say $t' \leq_q t_1$. that means as in figure-7.2 from $t'$ there is a $q$ transition to $t_2$ and there is also a $q$ transition from $t_1$ to $t_3$ and $t_2 \equiv_q t_3$. Now from $t'$ there is a $p$ transition to $t_1$ which
is not $q$ similar, but from $t_2$ there is no $p$ transition which is not $q$ similar as we have already said that $t_2 \equiv_q t_3$. Moreover from $t_2$ there can’t be a $p$ transition which is not $q$ similar as in that case there would be same transition to two different state. But the specification is deterministic. So it is not the case that $t' \leq_p t_2$. So before doing a local $q$ action, $q$ has to communicate with $p$, but $p$ has already gone to $t_1$. So from $[t']_q$ $q$ local action can’t be possible.

(b) Now lets suppose that current global state is simulating state $t$ and $t_q t'$, then process $q$ may be in $[t']_q$ by doing a $pq$ action from $[t]_q$. But from $[t']_q$, $q$ can’t do a $q$ action because after doing first $pq$ there will be another $pq$ which will happen after doing the local action $p$ from $[t]_p$. So, from $[t']_q$ there no $q$ local action will be fired, while current simulated state is $t$.

(c) Another possibility is that $q$ is in the state $t'$ by doing a $pq$ action from $t_1$ such that $t_1 q t'$ and there is a direct path from start state via $t_1$ to $t$. But from $t'$ no $q$ action can be fired, because from $t'$ to $t_1$ there must happen a $p$ before $q$.

(d) The last possibility is that $q$ resides in $[t]_q$. If there is a $q$ transition from $[t]_q$, then from $t$ there will also be a $q$ transition.

So, whenever some process $q$ is not $[t]_q$, where $t$ is the current simulating state, there can’t be a $q$ local transition from current $q$ local state.

**Proposition-3 :** $L(S) = \Pi_{\Sigma_1}(L(A))$ where $\Sigma_1$ is the local action alphabet.

Let $w$ is a valid word in specification. Then there is an accepting path of $w$, say $s \rightarrow s^1 \rightarrow s^2 \rightarrow \ldots s^k$. As the local automata are constructed from the specification, so there is a valid word $u$ ($\Pi_{\Sigma_1}(u) = w$) such that every process $p$ will reach to $[s^k]_p$ after accepting $u$ globally. Thus $L(S) \subseteq \Pi_{\Sigma_1}(L(A))$.

Let $u$ is accepted by $A$. Then there is an accepting path of $u$, say $S \rightarrow S^1 \rightarrow S^2 \rightarrow \ldots S^k$. Now according to proposition 2 we can map these states to the corresponding mapped states of specification such that from any $S^i$ mapping $s^i$, if there is a local $p$ transition to $T^i$ mapping $t^i$, then from $s^i$ there will also be a $p$ transition to $t^i$ in specification. Thus $\Pi_{\Sigma_1}(L(A))(S)$. So $L(S) = \Pi_{\Sigma_1}(L(A))$. 

51
7.5 An Example:

Figure 7.3, 7.4, 7.5 shows an implementation of specification as depicted in figure 7.1

Figure 7.3: Local automata of Process $p$

Figure 7.4: Local automata of Process $q$

Figure 7.5: Local automata of Process $r$
Another example:
In the figure, the bisimilar equivalence classes of $p$ are $\{1, 2, 3, 4\}; \{5, 6, 7, 8\}; \{9, 10, 11, 12\}; \{13, 14, 15, 16\}

Figure 7.6: Local automata of Process $p, q$ and $r$ accepting the specification in Figure-6.17
Chapter 8

Summary and Future Work

8.1 Summary

Chapter 2 and 3 are literature survey part. In chapter 2 the implementation based on Zielonka Theorem is discussed from [1]. It is seen that [1] gives a nice solution to this problem. Later in chapter 3 the implementation based on message passing is discussed from [5]. Here the correctness issues related to transitive closure property of architecture is pointed out. There was also correctness issue in condition of weakly deadlock free implementation. This is also pointed out.

This report mainly focuses on the implementation using trace language. In chapter 4, the two condition namely diamond and forward diamond condition are imposed on the specification, which are necessary to implement. A algorithm for implementation is presented then. The final outcome of this algorithm is set of local automata, where each automaton is deterministic. But the execution is sequential. Like chapter 4 chapter 5 also give a sequential solution with the help of gossip protocol. In the next chapter we have come with parallelism. But there we found further scope of improvement. In the last chapter we have introduced a notion called bisimulation, which gives a solution with taking care of parallelism.

8.2 Future Work:

(a) Here we give the solution for trace model. Future research can be done on the message passing system.

(b) In the solution discussed in this report, there comes some situation where parallel execution is possible, but this solution does not apply all of them. Future investigation can be done to have a more better solution.

(c) Deadlock free implementation can be viewed as one important condition to implement. Further work can be done in doing this.

(d) In [5], the channel buffer between two processes can grow boundlessly. But in real life such thing is not possible, as buffer can store up to some finite amount of messages. So further investigation in this issue can be done.
Bibliography


55