Distributed Algorithms **M.Tech., CSE, 2016**

Lecture 5: Asynchronous shared memory Model.

Faculty: K.R. Chowdhary *: Professor of CS*

5.1 Introduction

An asynchronous shared memory system consists of a finite collection of processes interacting with each other by means of a finite collection of shared variables. It is assumed that each process has a port through which it can interact with the outside world using input and output actions (see figure 5.1).

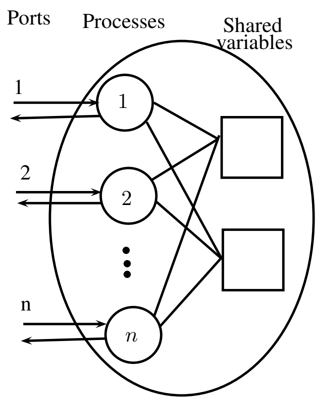

Figure 5.1: Asynchronous shared memory system.

We model a shared memory system using I/O automata A, i.e., it has only two actions, *input* and *outputs*. As in synchronous network model, we assume that the processes in the system are indexed by integer vales $1, \ldots, n$. Let each process has associated set of states $states_i$, with some start states $start_i$. Let each shared variable x has set of values $values_x$, and initial values as $initial_x$. For each process i there is set of states $state_i$ and set of values for each shared variable x.

Also there is action $act(A)$ associated with one of the process, interactions occur for process i on port $port_i$.

The set of transitions are represented by π, which are generated by set of tripples (S, π, s'), where $s, s' \in states_i$. An action i may also be associated with procees i and a variable x. That is, π transitions are generated from some set of tripples of the form $((s, v), \pi, (s', v'))$, where $s, s' \in states_i$, and $v, v' \in values_x$.

Example 5.1 *Shared memory system.*

Let V be a fixed value set. Consider a shared memory system A, consisting of n processes: $1, \ldots, n$, and a single shared variable x with values in $V \cup \{unknown\}$, initially x is *unknown*. The inputs are of the form $init(v)_i$, where $v \in V$, and i is process index. The outputs are of the form $decide(v)_i$. The internal actions are of the form $access_i$. All actions with subscript i are associated with process i, and the access actions are associated with variable x.

After process i receives an input $init(v)_i$, it accesses x. If it finds $x = unknown$, then it writes its value v into x and decides v. If it find $x = w$, where $w \in V$, then it does not write anything into x, but decides w. Finally, each $states_i$ consists of local variables.

States of i:
$status \in \{idle, access, decide, done\}$, initialy $idle$
$input \in V \cup \{unknown\}$, initially $unknown$
$output \in V \cup \{known\}$, initially $unknown$

The transitions are:
Transitions of i:

$init(v)_i$
Effect:
$input \ := v$
$if \ status = idle \ then$
$status := access$

$decide(v)_i$
Precondition:
$status = decide$
$output = v$
Effect:
$status := done$

$access_i$
Precondition:
$status = access$
Effect:
$if \ x = unknown \ then \ x := input$
$output := x$
$status := decide$

There is one task per process, which contains all the *access* and *decide* actions for that process.

It is not hard to see that in every fair execution α of A, any process that receives *init* input eventually performs a *decide* output. Also, every execution (fair or not, and with any number of *init* events occurring anywhere) satisfies the "agreement property" that no two processes decide on different values, and the "validity property" that every decision value is the initial value of some process.

$\square$

References

[1] Nancy A. Lynch, "Distributed Algorithms," *Elsevier*, 2013.

[2] Allen B. Tucker, Jr., "The computer Science and Engineering Handbook," *CRC Press*, 1997.