Distributed Algorithms

Nancy Lynch

June 26, 2024

Contents

1 Mutual Exclusion

1.1 Asynchronous Shared Memory Model

The system is modelled as a collection of processes and shared variables, with interactions. Each process i is a kind of state machine, with a set statesi of states and a subset $start$ of $states_i$ indicating the start states, just as in the synchronous setting. However, now process i also has labelled $actions$, describing the activities in which it participates. These are classified as either $input$, $output$, or $internal$ actions. We further distinguish between two different kinds of internal actions: those that involve the shared memory and those that involve strictly local computation. If an action involves the shared memory, we assumethat it only involves one shared variable.

There is a transition relation $trans$ for the entire system, which is a set of (s, π, s') triples, where and s' are **automaton states**, that is, combinations of states for all the processes and values for all the shared variables, and where π is the label of an input, output, or internal action. We call these combinations of process states and variable values "automaton states" because the entire system is modelled as a single automaton. The statement that $(s, \pi, s') \in trans$ says that from automaton state s it is possible to go to automaton state s' as a result of performing action π.

We assume that input actions can always happen, that is, that the system is input-enabled. Formally, this means that for every automaton state

s and input action π, there exists s' such that $(s, \pi, s') \in trans$. In contrast, output and internal steps might be enabled only in a subset of the states. The intuition behind the input-enabling property is that the input actions are controlled by an arbitrary external user, while the internal and output actions are controlled by the system itself.

1.2 The Problem

The mutual exclusion problem involves the allocation of a single, indivisible, nonshareable resource among n **users**, $U_1, \dots, U_n$.

A user with access to the resource is modelled as being in a **critical region**, which is simply a designated subset of its states. When a user is not involved in any way with the resource, it is said to be in the **remainder region**. In order to gain admittance to its critical region, a user executes a **trying protocol**, and after it is done with the resource, it executes an (often trivial) **exit protocol**. This procedure can be repeated, so that each user follows a cycle, moving from its *remainder region* (R) to its *trying region* (T), then to its *critical region* (C), then to its *exit region* (E), and then back again to its remainder region.

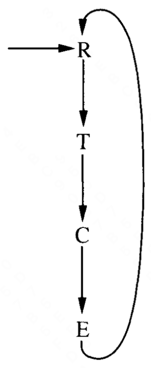

Figure 1: The cycle of regions of a single user

Each