1 Introduction

This lecture presented the Lehmann-Rabin (LR) randomized algorithm for solving Dining Philosophers for anonymous processors; introduced a probabilistic model of distributed computation with a worst-case adversary; and presented the first part of the proof that the LR Algorithm guarantees progress almost surely.

2 Lehmann-Rabin randomized Dining Philosophers algorithm

This is a simple algorithm, but in order to discuss its “probabilistic correctness,” we need to talk about the computation model. What is a probability distribution over algorithm executions, and how, exactly, are we analysing the algorithm?

We will guarantee Mutual Exclusion with certainty, and progress almost surely (with probability 1). This is an example of a common pattern. Often we will guarantee a safety condition with certainty, and a liveness condition with some probability.

In our current setting, all processors are identical. In particular, processors do not have unique IDs. If they did, we could use those IDs to break symmetry, and simply use a deterministic algorithm we discussed in a previous lecture. Today, we will use coin flips (randomization) to break symmetry. This is an example of a general class of algorithms in which we can use randomization to achieve things that would be impossible using purely deterministic methods.

2.1 The algorithm

Each processor knows its forks by local names $f(right)$ and $f(left)$. We also use the following notation: $j = \text{left}$ if $j = \text{right}$, and $j = \text{right}$ if $j = \text{left}$. The algorithm itself appears on the next page as Algorithm 1.

3 Exclusion and progress of the LR Algorithm

The LR Algorithm guarantees Exclusion. Recall that we are using Read-Modify-Write variables, so we can use an argument that is almost identical to the argument we used to ensure the Right-Left Dining Philosophers Algorithm guaranteed Mutual Exclusion.
Algorithm 1 LR Dining Philosophers Algorithm

**Initialization:** For every $i$, $1 \leq i \leq n$, the Boolean variable $f(i)$ is initially false, and is accessible exactly by the processors $i$ and $i - 1$.

**Algorithm for processor $i$**

<TRYING>

forever do

[F] (Flip)

if coin-flip = 0 then

    first := left

else

    first := right

end if

[W] (Waiting for fork)

wait until $f(first) = false$

$f(first) := true$

[S] (Second fork)

if $f(first) = false$ then

    $f(first) := true$

    goto [L]

else

    [D] (Drop fork)

    $f(first) := false$

end if

end forever do

[L] (goto Line)

<CS>

[E] (Exit)

$f(left) := false$

$f(right) := false$

<REM>
However, the LR Algorithm does not guarantee progress. If everyone gets the same value for the coin flip (e.g., every processor always picks LEFT) we have the same impossibility result we encountered in the deterministic anonymous case. (The argument: processors take steps in round robin order, always making the same random choices.) Therefore, we will show that the LR Algorithm ensures progress with a probability bounded away from 0. We can then amplify that probability to be as high as needed, depending on how many times we are willing to “repeat the experiment.” To state this formally, we will need a new model of distributed computation, and quite a bit of new notation.

3.1 Model and notation

We want to be able to express that the algorithm ensures progress with some probability. In other words, if we start in a system configuration with at least one processor in TRYING, we get to a system configuration in which some processor is in CS with some probability. We capture this formally as follows.

**Definition 1**

1. $\mathcal{T}$ is the set of all configurations where some processor is in the TRYING region.
2. $\mathcal{C}$ is the set of all configurations where some processor is in CS.
3. $\mathcal{T} \xrightarrow{t, p} \mathcal{C}$ asserts that, within time $t$, with probability $p$, if the system starts in a configuration in $\mathcal{T}$ it will reach a configuration in $\mathcal{C}$.

An execution of a probabilistic distributed computation includes both nondeterministic choices and probabilistic choices. We would like to state that, even if the worst-possible nondeterministic choices take place, even so the LR Algorithm guarantees that progress will occur with a probability bounded away from zero. To formalize this statement, we will assume the existence of a worst-case adversary that always selects the worst possible nondeterministic next step. Then we will show that the system has the desired properties regardless of the nondeterministic choices made.

In order to define a worst-case adversary, we need to define a way to pick the order of the processes, and a way to pick a time at which the processes take steps. As with other models we’ve considered, we place time upper bounds on how long a process can wait before taking a step: a process will take a step in $\leq t$ time, and a process will spend $\leq c$ time in the Critical Section. We assume an omniscient adversary with complete knowledge of process states and past random choices.

Formally, adversary $A$ is a function mapping a finite execution to (process, time) pairs. So $A$ indicates the next process to take a step, and the time at which the step is taken.

If $\mathcal{U}$ and $\mathcal{U}'$ are sets of system configurations, the notation $\mathcal{U} \xrightarrow{t, p} \mathcal{U}'$ means: for every adversary $A$, if the LR Algorithm starts in a configuration in $\mathcal{U}$, then in the probability distribution of executions determined by $A$, it will reach a configuration in $\mathcal{U}'$ by time $t$ with probability $p$. 

3
4 Main result

Definition 2 1. $\mathcal{T}$ is the set of reachable system configurations in which some process is in TRYING.

2. $\mathcal{C}$ is the set of reachable system configurations in which some process is in CS.

Now we are able to state the main result we want to prove.

Theorem 1 For $n \geq 3$, the LR algorithm satisfies $\mathcal{T} \xrightarrow{14l}{1/16} \mathcal{C}$.

We prove the theorem by a series of lemmas. To conclude this lecture, we will state five lemmas, and prove two of them. First, some additional notation: let $F, W, S, D, L$ be the set of process states when the process is at the beginning of the respective line of code. So TRYING $= F \cup W \cup S \cup D \cup L$, and those subsets partition TRYING, i.e., the sets are pairwise disjoint.

We can subdivide $W, S, D$ further, based on the value of the variable $\text{first}$. We let $\overrightarrow{W}$, $\overrightarrow{S}$, and $\overrightarrow{D}$ be the subsets of $W, S$ and $D$ in which $\text{first}$ equals $\text{right}$; and $\overleftarrow{W}$, $\overleftarrow{S}$ and $\overleftarrow{D}$ be the subsets in which $\text{first}$ equals $\text{left}$. We also “abbreviate” with $\overrightarrow{=} = \overrightarrow{W} \cup \overrightarrow{S} \cup \overrightarrow{D}$ and $\overleftarrow{=} = \overleftarrow{W} \cup \overleftarrow{S} \cup \overleftarrow{D}$.

We let $L$ be the set of reachable configurations in which some process is in $L$, that is to say, within time $l$ a process will enter CS.

Definition 3 1. $\mathcal{RT} \subseteq \mathcal{T}$ is the set of configurations in which all processes are either in TRYING or in REM.

2. $\mathcal{F} \subseteq \mathcal{RT}$ is the set of configurations in which some processor is in $F$.

3. $\mathcal{G} \subseteq \mathcal{RT}$ is the set of configurations such that there is some process $i$ such that either
   (a) $i \in \overrightarrow{W} \cup \overrightarrow{S}$ and $i - 1 \in \overrightarrow{=} \cup R \cup F$; or
   (b) $i \in \overleftarrow{W} \cup \overleftarrow{S}$ and $i + 1 \in \overleftarrow{=} \cup R \cup F$.

Note that $\mathcal{G}$ is the set of “good” system configurations, in which two neighboring processes are in a position to obtain both forks with high likelihood.

Lemma 1 1. If $\mathcal{U} \xrightarrow{t}{p} \mathcal{U}'$ and $\mathcal{U}' \xrightarrow{t'}{p'} \mathcal{U}''$ then $\mathcal{U} \xrightarrow{t+t'}{pp'} \mathcal{U}''$.

2. If $\mathcal{U} \xrightarrow{t}{p} \mathcal{U}'$ then $\mathcal{U} \cup \mathcal{U}'' \xrightarrow{t}{p} \mathcal{U}' \cup \mathcal{U}''$.

Lemma 2 $\mathcal{T} \xrightarrow{3l}{1} \mathcal{RT} \cup \mathcal{C}$
Lemma 3 \( \mathcal{RT} \xrightarrow{3l}{1} \mathcal{F} \cup \mathcal{L} \)

Lemma 4 \( \mathcal{F} \xrightarrow{2l}{1/4} \mathcal{G} \cup \mathcal{L} \)

Lemma 5 \( \mathcal{G} \xrightarrow{5l}{1/4} \mathcal{L} \)

Lemma 6 \( \mathcal{L} \xrightarrow{l}{1} \mathcal{C} \)

We now provide two simple proofs (of Lemmas 6 and 2).

First, Lemma 6: \( \mathcal{L} \xrightarrow{l}{1} \mathcal{C} \). Proof: if some processor in \( \mathcal{L} \) is ready to enter CS, within time \( l \) it will indeed enter CS, so with certainty (and hence with probability 1) the system configuration will be in \( \mathcal{C} \).

Now, Lemma 2: \( \mathcal{T} \xrightarrow{3l}{1} \mathcal{RT} \cup \mathcal{C} \).

Proof: [Lemma 2] If any process is in CS, or enters CS in time \( \leq 3l \), we are done. So assume all processors are not in CS and do not enter CS within \( 3l \) time. Then all processors are in \( R \cup T \cup E \) for time at least \( 3l \). Also, no process enters \( E \) during this time, because it would have had to pass through CS. But any process initially in \( E \) should return to \( R \) within time \( 3l \) So all processors reach \( R \cup T \) within \( 3l \) (because it has to execute at most two lines of code, dropping both forks), and the statement of the lemma is true.