1 Introduction

In the last lecture, we discussed a Mutual Exclusion algorithm based on token passing and full
link reversal. Today, we will state he algorithm formally and sketch a proof that it is correct.

One thing to note: this algorithm uses the idea of link reversals, but doesn’t use the techniques
we have seen before—no heights, vectors, etc. Our presentation will only consider static systems;
we won’t be able to use this simple idea on dynamic graphs.

2 Link Reversal Mutual Exclusion Algorithm (LRME)

2.1 Variables

Each processor $i$ has the following local variables:

- $\text{hasToken}_i$: set to either 0 or 1
- $\text{OG}_i$, $\text{IC}_i$: sets of $i$’s outgoing or incoming links, respectively
- $\text{N}_i$: the neighbors of $i$
- $\text{RQ}_i$: the request queue of $i$

2.2 The algorithm

Pseudocode for the Link Reversal Mutual Exclusion algorithm appears in Algorithm 1.

2.3 Proof of correctness

Recall that an execution is a sequence of system configurations and events, starting with the
initial configuration $C_0$, and proceeding $C_0$, $e_1$, $C_1$, $e_2$, $C_2$, . . . , where the $e_i$ are “message sent”
events or “message received” events. We will write $e_1(C_0) = C_1$ to say that the application of
event $e_1$ to configuration $C_0$ produces configuration $C_1$. More generally, $e_{i+1}(C_i) = C_{i+1}$.

We assume that the system is initialized with exactly one token in the network, and all proces-
sors start in Remainder status.
Algorithm 1 LRME code for processor \(i\)

1: when Request message received from node \(j\)
2: Enqueue \((RQ, j)\)
3: if \(\text{hasToken} = 1\) then
4:   if \(\text{status} \neq \text{CS}\) then
5:     Handle-Token()
6: else
7:   if \(|RQ| = 1\) then
8:     send Request message to some \(k \in OG\)
9: when Token message received from node \(j\)
10: set \(\text{hasToken} := 1\)
11: send MakeOG message to all \(k \in OG\)
12: wait for \(\text{ackOG}\) message from all \(k \in OG\)
13: set \(IC := B;\) set \(OG := \emptyset\)
14: if \(|RQ| > 0\) then
15:   Handle-Token()
16: when MakeOG message received from node \(j\)
17: remove \(j\) from \(IC\); add \(j\) to \(OG\)
18: send \(\text{ackOG}\) message to \(j\)
19: when Application requests access to \(\text{CS}\) (Request \(CS\))
20: set \(\text{status} := \text{CS}\)
21: Enqueue \((RQ, i)\)
22: if \(\text{hasToken} = 1\) then
23:   Handle-Token()
24: else
25:   if \(|RQ| = 1\) then
26:     send Request message to some \(k \in OG\)
27: when Application releases \(\text{CS}\) (Release \(CS\) message)
28: set \(\text{status} := \text{REM}\)
29: if \(|RQ| > 0\) then
30:   Handle-Token()
31: Procedure Handle-Token()
32: set \(p := \text{Dequeue}(RQ)\)
33: if \(p \neq i\) then
34:   \(\text{hasToken} := 0\)
35: send Token message to \(p\)
36: if \(|RQ| > 0\) then
37:   send Request message to \(p\)
Lemma 1 In every configuration, the following is true.

(a) At most one process has $\text{hasToken} = 1$.

(b) If no process has $\text{hasToken} = 1$ then there is exactly one Token message in transit.

(c) If any process has $\text{hasToken} = 1$ then there is no Token message in transit.

(d) If process $i$ has status $= \text{CS}$ then $\text{hasToken}_i = 1$.

(e) If $\text{status}_i = \text{ES}$ then $i$ appears exactly once in $\text{RQ}_i$.

(f) If $\text{status}_i \neq \text{ES}$ then $i$ is not in $\text{RQ}_i$.

Proof: Proof by induction. We first show all conditions are true for the initial configuration, and then show that if the conditions are true for configuration $C_i$, they will be true for $C_{i+1} = e_{i+1}(C_i)$, by an argument by cases, where each possible event type of $e_{i+1}$ is one of the cases.

In the initial configuration $C_0$, $\text{hasToken} = 1$ is true for exactly one process, no messages are in transit, all processes are in status REM, and $\text{RQ}_i = 0$ for all $i$. So all conditions hold in the initial configuration.

Case 1: Node $i$ receives a Request message. There are two subcases to consider: either $\text{hasToken}_i = 0$ or $\text{hasToken}_i = 1$. If $\text{hasToken}_i = 0$, $i$ makes no changes to the system except possibly for sending a Request message on one of its outgoing links. So all of the Lemma’s conditions continue to hold. Therefore, suppose $\text{hasToken}_i = 1$. If $\text{status}_i = \text{CS}$, Lemma conditions (b)-(e) are immediately satisfied, and execution proceeds as though $\text{hasToken}_i = 0$, which we have already shown to satisfy conditions (a) and (f).

So let’s consider the situation where $\text{hasToken}_i = 1$ and $\text{status}_i \neq \text{CS}$. In that situation, the algorithm executes $\text{Handle – Token}()$, which dequeues the ID $p$ of a processor from $\text{RQ}_i$. Since $\text{hasToken}_i = 1$, there is no Token message in transit. There are two possibilities: either $p = i$, or $p \neq i$. Suppose $p = i$. This is only possible if $\text{status}_i = \text{ES}$, as otherwise (by conditions (e) and (f)) $i$ would not be in $\text{RQ}_i$ in configuration $C_i$. When $p = i$, $\text{Handle – Token}()$ takes no further steps, so (e) and (f) remain true in configuration $C_{i+1}$. Moreover, since (a)-(d) were true in $C_i$, and $\text{Handle – Token}()$ takes no action other than the one Dequeue command, (a)-(d) remain true in $C_{i+1}$. Therefore, let’s consider the possibility $p \neq i$ when $\text{Handle – Token}()$ dequeues $p$. $\text{Handle – Token}()$ sets $\text{hasToken} = 0$, which will preserve (a) in $C_{i+1}$. Then $\text{Handle – Token}()$ sends a Token message to $p$. Since now no process in the network has $\text{hasToken} = 1$, (b) will be true in $C_{i+1}$. Conditions (c)-(f) will be true by default.

Therefore, if node $i$ receives a Request message, no matter the state of the network or of node $i$, the conditions of the Lemma will remain true once $i$ has processed that message. ■