Report
Summer Internship under Prof Yoram Moses

Sharvik Mital
October 9, 2018

Abstract
The summer research under Prof Yoram was focussed on studying different solutions to the Mutual Exclusion Problem, trying to prove them correct and if possible come up with an optimized version. This report tries to prove the Ricart Agrawala solution to ME problem and suggest possible optimized versions of algorithm. The proof is very much along the lines of the proof given for the Bakery algorithm in the Boulangerie paper [1].

1 The Ricart And Agrawala Algorithm
The RA Algorithm [2] as it is called in short is one of the classic solutions to the Mutual Exclusion Problem in a network topography. The algorithm is inspired by the very famous Lamport's Bakery algorithm [3] for Mutual Exclusion in a shared memory scenerio.
The setting of the algorithm is a computer network where nodes can communicate only by messages and do not share memory. Intuitively the algorithm uses the permission based approach to find out for each node if it has priority over all the processes. If the node gets permission from all other nodes, it can enter critical section.
The algorithm is symmetrical across nodes and each node has three processes to implement the mutual exclusion:

1. One is awakened when mutual exclusion is invoked on behalf of this node.
2. Another receives and processes REQUEST messages.
3. The last receives and processes REPLY messages.

The three processes run asynchronously but operate on a set of common variables. A semaphore is used to serialize access to the common variables when necessary. If a node can generate multiple internal requests for mutual exclusion, it must have a method for serializing those requests.
Algorithm 1: PROCESS THAT INVOKES MUTUAL EXCLUSION FOR THIS NODE(i)

/* Request Entry to the Critical Section */
1 P(Shared_Vars)
   /* Choose a sequence number */
2 Requesting_Critical_Section := TRUE;
3 My_Sequence_Number := Highest_Sequence_Number + 1;
4 V(Shared_Vars)
5 Outstanding_Reply_Count := N - 1;
   /* Send a REQUEST message Tᵢₓ(number[i], i) containing number[i]=My_sequence_number and node number=i to all other nodes */
6 for j := 1 UNTIL N STEP 1 IF j ≠ i THEN do
7     Send_Message (REQUEST(My_Sequence_Number, i), j)
8 end
   /* Now wait for REPLY from each of the other nodes */
9 WAITFOR (Outstanding_Reply_Count = 0);
   /* Critical Section Processing can be performed at this point */
   /* Release the Critical Section */
10 P(Shared_Vars)
11 Requesting_Critical_Section := FALSE;
12 V(Shared_Vars)
13 for j := 1 STEP 1 UNTIL N do
14 if Reply_Deferred[j] then
15     Reply_Deferred[j] := FALSE;
16     Send_Message (REPLY, j);
17     /* send a REPLY to node j */
18 end
Algorithm 2: PROCESS THAT HANDLES OTHER NODES’ REQUEST MESSAGES

/* j is the node number making the request */
/* k is the sequence number(number[j]) of the node j making the request */

Boolean: Defer_it: True when this node(i) cannot reply immediately

1. Highest_Sequence_Number := Max(Highest_Sequence_Number, k)

2. P(Shared_Vars)
   Defer_it := Requesting_Critical_Section
   AND ((k > Highest_Sequence_Number)
   OR (k = My_Sequence_Number AND j > i));

   /* Defer_it will be TRUE if this node(i) have priority over node j’s request */

3. if Defer_it then
   Reply_Derected [j] := TRUE
   else
   Send_Message (REPLY, j);

4. V(Shared_Vars)

Algorithm 3: PROCESS WHICH RECEIVES REPLY MESSAGES

1. Outstanding_Reply_Count := Outstanding_Reply_Count − 1;

2 2 Proof of RA Algorithm

We first partition the code of the processes that invoke Mutual Exclusion and that handle other nodes’ request.

2.1 Partitioning into regions

<table>
<thead>
<tr>
<th>main region</th>
<th>symbol</th>
<th>lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>doorway</td>
<td>i_d</td>
<td>1-4</td>
</tr>
<tr>
<td>bakery</td>
<td>i_b</td>
<td>5-12</td>
</tr>
</tbody>
</table>

Table 1: Partition of the code for Process invoking Critical section into regions
Table 2: Partition of the code for the process receiving requests

For Table 2, we have used a symbol $\text{CP}_i(T_j)$. This essentially means that node $i$ is processing the request of ticket $T_j$ sent by $j$. By processing we mean that the node $i$ decides whether to send the reply to request now or send it later (by deferring it). Note that the region also includes the sending part in case process decides to reply and not defer.

We now prove the correctness of the algorithm in the sense that it indeed leads to Mutual Exclusion. Briefly, we first come up with a relation to find out which between the two nodes first gets the chance to be in critical section. This relation should have following properties:

1. Anti-symmetric.
2. To make sure that while a node is in critical section, no other node enters it, we show that once a node gets the priority through the relation, it retains the priority at least till it is in the critical section or more strongly till it is in the bakery.

At the end, using the above properties of the relation and the fact that a node enters critical section only if it knows that is has priority (through this relation) over all the other nodes, we show that the algorithm indeed guarantees Mutual Exclusion.

2.2 A Priority Relation for the Ricart Agrawala Algorithm

Let us try to come up with a priority relation that formulates the fact that if node $i$ has priority over node $j$ then node $i$ enters the critical section before node $j$ enters.

**Definition of** $i \prec j$:

The priority relation $i \prec j$ holds at run $r$ at time $t$ if:

1. $(r, t) \models i_b \land j_b \land \langle \text{number}[i], i \rangle \ll _L \langle \text{number}[j], j \rangle$, or

2. $(r, t) \models i_b \land \neg j_b \land \text{was}[\text{CP}_j(T_i)]$

where $T_i$ is $\langle \text{number}[i], i \rangle$ and number[i] = My.Sequence_Number at time $t$ (since $i$ is in bakery at $t$, it must surely have a Ticket number).

Here $\text{was}[\text{CP}_j(T_i)]$ is true at time $t$, if $\exists t' \leq t$ such that $(r, t') \models \text{CP}_j(T_i)$. Intuitively, it means that $j$ is either processing the $T_i$ request of $i$ or has already processed the request.

We now show that as time $t$ increases, the ticket $T_i$ of a node $i$ when it is in bakery is non decreasing lexicographical function of $t$ for a run $r$. 

<table>
<thead>
<tr>
<th>main region</th>
<th>symbol</th>
<th>lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing</td>
<td>CP$_i$(T$_j$)</td>
<td>2-4</td>
</tr>
</tbody>
</table>
Proof. From line 1 in algo 2 it is clear that the value of Highest_Sequence_Number never decreases with time and so from line 3 in algo 1 we get that if \((r, t) \models i_b\) and \(i\) has ticket \(T_i\), \((r, t') \models i_b\) and \(i\) has ticket \(T_i'\) and \(t < t'\) then \(T_i \leq_T T_i'\).

Lemma 2.1. The priority relation \(\prec\) is antisymmetric.

Proof. We need to show that \((r, t) \not\models (i \prec j) \land (j \prec i)\) for all points \((r, t)\). Assume that \((r, t) \models (i \prec j) \land (j \prec i)\), by way of contradiction. Then, by definition, we have that \((r, t) \models i_b \land j_b\). By definition of priority relation, both \(\langle\text{number}[j], j\rangle <_L \langle\text{number}[i], i\rangle\) and \(\langle\text{number}[i], i\rangle <_L \langle\text{number}[j], j\rangle\) hold at \((r, t)\). This is a contradiction, since \(\prec_L\) is an ordering relation.

This shows that priority relation is anti-symmetric. We now show property of priority relation.

Notice, that a node \(i\), enters critical section only when its Outstanding_Reply_Count becomes equal to zero in algo 1. It intuitively seems that for its Outstanding_Reply_Count to become zero, each node must reply to it exactly once. This is indeed true. So we prove the following lemma.

Lemma 2.2. For each node \(i\), if it enters critical section \(n^{th}\) time at time \(t\), then between time \(t'\) when is starts sending \(n^{th}\) request(start executing in algo 1) and \(t\), it receives exactly one reply from each node.

Proof. Assume by contradiction, that there exists a run \(r\), where node \(i\) gets more than one reply from some node between the time period \(i\) sends request and enters its critical section. Then in that run \(r\), consider the first such instance of such critical section entrance, where it happens and let that be the \(n^{th}\) invocation of request of critical section of \(i\). Also let \(j\) be such node which sends two replies. Now \(i\) sends exactly one request to each node in invocation of request for critical section. So when \(i\) enters critical section \(n^{th}\) number of time, it has sent exactly \(n\) requests to \(j\) since time \(t = 0\). Since initial value of Outstanding_Reply_Count = \(n - 1\) and there are \(n - 1\) nodes except \(i\) therefore \(i\) must get reply from all other nodes exactly once for all \(k < n\) instances of such critical section. For this invocation, \(j\) sent two replies, while for all invocations before \(j\) sent exactly one reply(Since this is the first such instance where a node is sending two replies). So \(j\) has sent \(n + 1\) replies since time \(t = 0\). But observe that a node replies to a request exactly once in either line 16 in algo 1 or 3 in algo 2 which leads to contradiction.

We now show that when \(i\) receives a reply from \(j\) then at that time \(i\) is in bakery with the same ticket for which it recieves the reply from \(j\). Formally,

Corollary 2.3. \(((r, t) \models \text{receives}_i(j, T'_i)) \Rightarrow ((r, t) \models i_b \land (T_i = T'_i))\) where \(T_i\) is the ticket present with node \(i\) at \(t\).
Proof. We first show that $i$ is in bakery when it receives a reply from $j$.
Assume, by way of contradiction, there exists a run $r$, where node $i$ receives a reply from $j$ when $i$ is not in bakery. Consider an instance of such reply received and let it be the $n^{th}$ reply received by $i$ from $j$. Since it is $n^{th}$ reply, $i$ has already entered bakery $n$ times. Moreover because $i$ is not in bakery now it has exited bakery $n$ times. We know that before exiting bakery, $i$ must enter critical section (def of bakery section) exactly once. From lemma 2.2, we know that $i$ must receive reply from $j$ exactly once between sending requests and entering critical section. So, since $i$ has exited bakery $n$ times before it received the $n^{th}$ reply, it must have received $n$ replies before it received the $n^{th}$ reply, a contradiction. So $(r, t) = i_b$.

We now show that the request ticket for which $i$ receives reply is same as the ticket present with it at that time.
Again, assume by way of contradiction, there exists a run $r$ where the Ticket $T_i$ present with $i$ when it receives the reply from $j$ is different from Ticket $T'_i$ for which it receives the reply. Consider the first such instance in run $r$. Let $i$ have entered the bakery section for the $n^{th}$ time. Clearly the request of $T'_i$ must have been sent by $i$ before it entered bakery for $n^{th}$ time (because $i$ sends exactly one request to each node in each invocation request for critical section). Let that invocation be $n'$. So, $n' < n$. Since this is the first instance when $i$ receives a reply for a different ticket, so at the $n'$ invocation, $i$ received the reply for $T'_i$. So, $i$ received the reply for $T'_i$ from $j$ both at $n'$ and $n$ invocation, a contradiction, which completes the proof.

The above results now directly leads to the following corollary:

**Corollary 2.4.** A node $i$ remains in bakery between all the time $t_1$ (sends its $n^{th}$ request to $j$) and $t_2$ (receives the reply for the $n^{th}$ request from $j$).

Proof. By way of contradiction, let us assume that $\exists t$ such that $t_1 \leq t \leq t_2$ and $(r, t) = \neg i_b$, then at some $t'$, $t \leq t' < t$, $i$ received a reply of request ticket $T'_i$ from $j$ (For $i$ to enter critical section it must receive exactly one reply from all the nodes (lemma 2.2). But from the corollary 2.3, $T'_i$ should be the $n^{th}$ request. Now node $j$ can reply to a request exactly once, so $i$ cannot receive a reply from $j$ to its $n^{th}$ request both at time $t'$ and $t_2$. So contradiction.

We now define another symbol $\text{reply}(i, T_j)$. This essentially means $i$ sending a reply message to $j$’s request of ticket $T_j$. Notice that node $i$ sends reply to $j$ in either of the two lines 16 in algo 1 or line 5 in algo 2.

The above corollary trivially leads to a following useful result-

**Corollary 2.5.** $(r, t) = \text{reply}(j, T'_i) \Rightarrow ((r, t) = i_b \land (T_i = T'_i))$ where $T_i$ is the ticket present with node $i$ at $t$.

Proof. Clearly, a node $j$ replies to $T_i$ between the time $i$ sends this request and receives the reply for this ticket. From corollary 2.4, $i$ remains in bakery between the whole time waiting for the same ticket. So trivially the corollary holds.
Now, we show that \( i \) gets the priority over \( j \) somewhere near the time when \( j \) replies to a \( T_i \) present with \( i \). More formally:

**Lemma 2.6.** \((r, t') \models \text{reply}(j, T_i) \Rightarrow (\exists t \leq t' \text{ such that } (r, t) \models i \triangleleft j \text{ and } (r, t') \models i_b S t)\)

**Proof.** After receiving the request and processing it, node \( j \) replies to \( i \) in two scenarios:

1. \( \text{defer}_{\text{it}} = \text{False} \Rightarrow (\text{Requesting\_Critical\_Section}_j = \text{False} \text{ or } (\text{Requesting\_Critical\_Section}_j = \text{True} \text{ and } \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle))\)

2. \( \text{reply\_defer}_{j}[i] = \text{True} \text{ when } j \text{ is exiting bakery at time } t'' \text{ (Exiting bakery means the node } j \text{ is executing the last statement which is the part of the bakery section } L_i \text{ and the request sent by } i)\)

In case [i] if Requesting\_Critical\_Section\_j = False, then \( j \) was not in bakery when it was processing the request. Now since sending reply\_\_\_\_\_3 in algo\_\_\_\_\_2 is part of the same region as \( L_i \) while crossing doorway is a part of another and both regions are locked by the same semaphore, so \( j \) cannot enter bakery till it sends the reply. So \( j \) cannot be in bakery at time \( t' \) also when it is sending the reply. From lemma 2.5 we know that at time \( t' \), \( i \) is in bakery. Also was[C\_Pj(T_i)] is true at time \( t' \) by the definiton of was. So from the second condition of priority relation, we get \((r, t') \models i \triangleleft j\).

In case [ii] at time \( t \) when \( j \) has completed exiting bakery section \( L_i \text{ and the request sent by } i \) is in bakery(as it has not yet received reply for the Ticket \( T_i \) so from corollary 2.4, \( j \) is not in bakery and was[C\_Pj(T_i)] is true. So from second condition of priority relation, \((r, t) \models i \triangleleft j\).

Moreover from corollary 2.4 till the time \( j \) sends the reply at the time \( t \), \( i \) is in the bakery and hence \((r, t') \models i_b S t\). \( \square \)

Now we show that if priority holds at some point in time, it will hold till the process having priority exits the bakery.

Formally,

**Lemma 2.7.** \( i \triangleleft j \Rightarrow (i \triangleleft j) U i_b \)

**Proof.** This is equivalent to \( i_b S (i \triangleleft j) \Rightarrow i \triangleleft j \). Let us prove the later. So, assuming \((r, t) \models i_b S (i \triangleleft j) \), we show that that \( i \triangleleft j \) holds at \((r, t) \). By definition of \( S \) we have that \((r, t) \models i_b \) and for some time \( t' \leq t \) both (i) \((r, t') \models i_b \) and (ii) \((r, m) \models i_b \) holds for all times \( m \) in the range \( t' \leq m \leq t \). Let’s prove the claim by induction on \( k = t - t' \). The claim is immediate if \( t - t' = 0 \) since then \( t = t' \) and \((r, t') \models i_b \) and (i) \((r, m) \models i_b \) holds for all times \( m \) in the range \( t' \leq m \leq t \). Let \( t - t' = k > 0 \), and assume inductively that the claim is true for \( k - 1 \). Since \((t - 1) - t' = k - 1 \geq 0 \), we have by the inductive assumption that \((r, (t - 1)) \models i \triangleleft j \). By definition of the ‘\( \triangleleft \)’ relation, we consider two cases:
Proof. From lemma 2.2, for a node \( j \) throughout the RA algorithm, (b) \((r, t) \models \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle \). This proves that \( j \)'s request of the ticket \( T_j \) present at \((r, t - 1)\) at \( j \) and replied. From lemma 2.5 this implies that for some \( t'' \leq t' \), \( j < i \) holds with \( j \) having ticket \( T_j \).

Now \( j < i \) can hold because of two conditions from the definition of priority relation. If condition one holds this implies at \( t'' \), \( \langle \text{number}[j], j \rangle <_L \langle \text{number}[i], i \rangle \).

As at \( t \), \( \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle \) holds, this implies that \( \langle \text{number}[i], i \rangle <_L \langle \text{number}[i''], i'' \rangle \) which is not possible (proved earlier).

If condition two holds i.e \((r, t'') \models \langle j_b \wedge \neg j_b \wedge \text{was}[CP_i(T_j)] \rangle \), this implies Highest_Sequence_Number \( \ni \langle \text{number}[j] \rangle \) at \( t'' \) (Highest_Sequence_Number is updated in line 7 in algo 2 happening before \( i \) enters processing region). Now, since at \( t - 1 \), \( \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle \) holds, so the ticket \( T_i \) with \( i \) at \( t \) had to be chosen before \( t'' \). This implies that the order of the events should be \( i_d \) (crossing doorway or choosing Ticket \( T_i \)) followed by \( CP_i(T_j) \) (holds at \( t'' \)) strictly followed by \( i_b \) (does not hold at \( t'' \) but holds at \( t - 1 \)) (followed by means that the second event cannot happen before the first whereas strictly followed by means that second event has to happen strictly afterwards). But observe that, the there is a binary semaphore associated with doorway and \( CP_i(T_j) \). So these have to happen sequentially. And this is not possible as the end of doorway marks the beginning of bakery section. So \( CP_i(T_j) \) cannot happen strictly in between the two events.

This proves that \((r, t) \models j_b \).

\( (r, t - 1) \models i_b \wedge \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle \). In this case if \((r, t) \models j_b \) then \((r, t) \models i_b \wedge \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle \) holds and \( i < j \) holds. Now we show that if \((r, t) \models j_b \) then \( \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle \) holds. Assume by way of contradiction that \( \langle \text{number}[j], j \rangle <_L \langle \text{number}[i], i \rangle \) at \((r, t) \). Since at \( (r, t - 1) \models \text{was}[CP_i(T_j)] \) holds, so we know that at \( t - 1 \), Highest_Sequence_Number \( \ni \langle \text{number}[i] \rangle \) of ticket \( T_i \) present with \( i \) at \( t \). So the ticket \( T_j \) with \( j \) at \( t \) had to be chosen before \( t - 1 \). This implies that the order of the events should be \( j_d \) followed by \( CP_j(T_j) \) (holds at \( t - 1 \) as well as \( t \)) strictly followed by \( j_b \) (holds at \( t \) but not at \( t - 1 \)). And following the same argument as used before in case a) condition two, this is not possible.

So \((r, t) \models j_b \) \implies \((r, t) \models \langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle \) which implies that \((r, t) \models i < j \)

\( \Box \)

We can now prove that a node \( i \) enters or is in its critical section only when it knows that it has priority have all other nodes. Formally:

**Lemma 2.8.** Throughout the RA algorithm, \( \text{CS}_i \models \bigwedge_{j \neq i} K_i(i < j) \).

**Proof.** From lemma 2.2, for a node \( i \) to enter its critical section, it must receive the reply from all the nodes. Now we know that if a node \( j \) replies to \( i \) at time \( t \), then for
some $t' \leq t$, $(r, t') \models i_b \land i \triangleleft j$ (lemma 2.6). From lemma 2.7, $i \triangleleft j \Rightarrow (i \triangleleft j) \cup \neg i_b$. From corollary 2.5, $i$ is in bakery since time $t'$ (clearly $t'$ is later than time $t$ when $i$ sends the request for the $T_i$ because of which it holds the priority over $j$), so when node $i$ receives $j$'s reply, $i$ knows that it is has priority over $j$.

This implies that once node $i$ has received replies from everyone, it knows that it has priority over all other nodes. Also, this priority holds till it exits bakery section which happens later than it exits critical section, so throughout the critical section, $i$ knows that it has priority over all other nodes.

This result directly leads to Mutual Exclusion. Due to the anti-symmetric property of priority relation, at most one node can have priority over all other nodes and therefore at most one node can be in its CS at any time.

Can nodes do better following this protocol? By better we mean, can a node know that it has priority over another node sooner. It turns out that this is possible.

3 Making the nodes do better

Consider a scenario in which node $i$ sends a request to node $j$ at $(r, t)$ and before receiving the reply from $j$, it receives $j$’s request such that $\langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle$. Using this information, node $i$ can know that node $j$ cannot enter critical section before $i$ enters and exits its critical section. So, in a sense $i$ can know that it has priority over $j$ even before receiving the reply and therefore need not wait for the reply.

To prove the correctness of this reasoning, we first prove the following lemma.

**Lemma 3.1.** If at $(r, t)$, $i_b$ is true and nodes $i$ recieves request from $j$ such that $\langle \text{number}[i], i \rangle <_L \langle \text{number}[j], j \rangle$ then $j_b$ is also true. This implies $i \triangleleft j$ is true at $(r, t)$

**Proof.** Let node $j$ send the request $T_j$ to $i$ at $(r, t')$. For $j$ to enter the critical section, it must get a reply from every other node which can happen only after the other node receives $j$’s request. So this implies that when node $i$ receives $j$’s request, clearly $j$ has not received $i$’s reply and is waiting for reply. This means that $j$ is still in the bakery section. And by the first condition of priority relation, $i \triangleleft j$ holds. Moreover since this reasoning can be found using the local information of $i$, this also implies $K_i(i \triangleleft j)$.

From lemma 2.7, we also know that, this priority relation holds till $i$ is in the bakery. And therefore because of the antisymmetric property of the relation, $j \triangleleft i$ cannot hold till $i$ is in bakery and so node $j$ cannot enter the critical section till $i$ enters and exits the critical section.

Hence node $i$ can enter critical section without waiting for the reply of the $j$ if it receives an inferior request from $j$ than its own when it is in the bakery.

But what to do about the reply sent by node $j$ later for the present request of $i$?

There are a number of possible solutions...
1) Change the algorithm, so that node $j$ does not send the reply if it knows that node $i$ can know that $i < j$ holds without $j$ sending the reply. This solution works correctly only if there is an inorder message delivery guaranteed.

2) Node $i$ ignores the reply sent by node $j$. For this, $i$ keeps a count for each node of how many times it entered critical section before receiving reply from that node. So at a later time, if it receives a reply from a node, and count is greater than zero than it ignores the reply and decrease the count by one. Otherwise if count is zero this implies this reply is of the present request.

3) Node $j$ sends a reply which has the value of the $\text{number}[i]$ for which it sending the reply. Since the value of $\text{number}[i]$ does not decrease each time node $i$ enters critical section, $i$ will know on reviewing the reply if it is of the present request.

References

