1 Lower bound on the number of States of memory

Theorem 1 Any algorithm that uses RMW (Read-Modify-Write) variables and provides Mutual Exclusion and no starvation must have at least \( n \) shared memory states (and consequently \( \lceil \log n \rceil \) bits to represent shared memory). Here \( n \) is the number of processors.

We shall now prove a slightly modified version of the above theorem.

Definition 1 An algorithm satisfies \( k \) bounded waiting if for every valid schedule there is no processor which remains in the entry while another processor enters critical section more than \( k \) times.

Note: The requirement of \( k \) bounded waiting is not as strong as the requirement of no starvation. Consider this scenario: All processes are in entry section forever. In this case the requirement of \( k \) bounded waiting is satisfied but there is still starvation. So, We need to add the no deadlock requirement to the \( k \) bounded waiting to make it at least as strong as the condition of no starvation.

Theorem 2 Any algorithm that uses RMW (Read-Modify-Write) variables and provides Mutual Exclusion, \( k \) bounded waiting and no deadlock must have at least \( n \) shared memory states (where \( n \) is the number of processors).

Proof: Consider the initial state \( C_0 \). All processes are in remainder section. Let the following successive states in the diagram be reached when each processor enters the entry section one by one. Let \( P_1 \) go into critical section first. In the following figure, annotations about states are shown in blue. Schedules and state labels are written in black.
Suppose that we do not need \( n \) different shared memory states. In the above figure we have \( n \) configurations that are not quiescent. By pigeon hole principle, at least two configurations will have the same shared memory state. Let them be \( C_i \) and \( C_j \). In \( C_i \) the processor \( P_1 \) is in CS and the processors \( P_2 \) to \( P_i \) are in Entry and the rest are in REM. In \( C_j \) the processor \( P_1 \) is in CS and the processors \( P_2 \) to \( P_j \) are in Entry and the rest are in REM. Consider the processors \( P_1 \) through \( P_i \). For these processors \( C_i \) is similar to \( C_j \). This is because the state of the processors from \( P_1 \) through \( P_i \) is same in both \( C_i \) and \( C_j \). And the shared memory state is also same.

Any steps they allowed to take in \( C_i \), they are able to take in \( C_j \). Let \( \alpha \) be the schedule where \( P_1 \) exited critical section. Let \( D \) be the configuration that is reached from \( C_i \) after apply schedule \( \alpha \) to \( C_i \). Let \( D' \) be the state reached after applying the schedule \( \alpha \) to \( C_j \). In \( D \), \( P_2 \) to \( P_i \) are in Entry and all others are in REM. In \( D' \), \( P_2 \) to \( P_j \) are in Entry and all others are in REM. Note that \( D \) and \( D' \) have same shared memory state and appear similar to \( P_1 \).

In \( D \), those processors that are in entry section will have to enter CS eventually. So, by
no deadlock condition there exists a state say, $E$, where all of $P_2$ to $P_i$ will have entered and exited the CS exactly once. Let $\beta$ be the schedule such that $\beta(D) = E$. $E$ is quiescent and all processors are in REM. Let $E'$ be the configuration achieved by applying $\beta$ to $D'$. $E'$ appears quiescent to all processors except $P_{i+1}$ through $P_j$. For all processors other than $P_{i+1}$ through $P_j$, $E'$ is similar to $E$ because the processor states and shared memory is same as that in $E$. Note that $\beta$ will have steps only by $P_2$ to $P_i$.

Let $\gamma$ be a $P_1$ only schedule where $P_1$ enters the critical section $k + 1$ times. In configuration $E$ (where all processors are in REM) $\gamma$ can be applied because $E$ is quiescent and $\gamma$ is $P_1$ only (other processors do not go into Entry and so no violation of k-bounded waiting). Note that $E$ and $E'$ are $P_1$ quiescent. So, in $E'$ also, $P_1$ can follow the steps in $\gamma$. However in $E'$, unbeknownst to $P_1$, processors $P_{i+1}$ through $P_j$ are in Entry. This violates $k$ bounded waiting and this contradiction arises from our assumption that less than $n$ shared states are sufficient for Mutual Exclusion with $k$ bounded waiting and no deadlock.

2 Resource Allocation

Processors often need more than one resource for execution. Resource allocation is a generalization of Mutual Exclusion problem. Before proceeding to define how a resource allocation problem can be formulated, we will look at some simple assumptions related to resource allocation:

Processors can not substitute alternate combination of resources. Certain processors are not allowed to share resources.

There are two methods for specifying how resources can be shared among multiple users.

I. Explicit Resource Specification: Explicit resource specification $\mathcal{R}$ for $n$ users is defined by

1. A set $R$ of resources.
2. For every user $i$, $1 \leq i \leq n$, a set $R_i \subseteq R$ is needed by user $i$.

Two users $u_i$ and $u_j$ conflict if $R_i \cap R_j \neq \emptyset$

Example: There are 4 users: $u_1$, $u_2$, $u_3$ and $u_4$. Let $R = \{r_1, r_2, r_3, r_4\}$. Let $R_1 = \{r_1, r_2\}$, $R_2 = \{r_1, r_3\}$, $R_3 = \{r_2, r_4\}$, $R_4 = \{r_3, r_4\}$. Here, users $u_1$ and $u_2$ conflict because they both need $r_1$. $u_1$ and $u_3$ also conflict because both need $r_2$. In this manner we can find other conflicting pairs.

II. Exclusion Specifications

This specification does not mention anything about resources. Instead, it specifies a collection, $\mathcal{B}$, of bad sets of processors that can not work simultaneously. $\mathcal{B}$ must be closed under superset.

$\mathcal{B}$ for Mutual exclusion problem can be defined as:
Exclusion specification for dining philosophers problem is given by:

$$B = \{ E | E \subseteq \{1, ..., n\}, |E| > 1 \}.$$  

K-Exclusion problem:  

$$B = \{ E | E \subseteq \{1, 2, ..., n\}, |E| > k \}.$$  

Note: We can derive exclusion specs from explicit resource specification. However, we cannot derive explicit resource specification from exclusion specs. Consider the K exclusion problem. Suppose K = 2. Suppose we have 3 users, u1, u2, u3. Then, by exclusion specs defined by the 2-Exclusion problem: u1 and u2 can run together; u2 and u3 can run together; u3 and u1 can run together. If we bring the notion of a “set of resources” into the picture, then it means u1 and u2 do not have common resources and the same applies to the following user pairs: (u2, u3) and (u3, u1). That is, R1, R2 and R3 are pair wise disjoint. This means, all of them can run together. But this contradicts the condition that no more than two processors can run together. So, we can’t find an explicit resource specification in this case.

A single exclusion specs can give rise to more than one explicit resource specification. As an example, consider the exclusion specs of mutual exclusion problem:  

$$ME = \{ E | E \subseteq \{u_1, ..., u_n\}, |E| > 1 \}.$$  

Let there be 3 users, u1, u2, u3. Both the following resource specifications agree with the given exclusion specs:

1. $R = \{ r \}$ and $R_1 = R_2 = R_3 = \{ r \}.$  
2. $R = \{ r_1, r_2, r_3 \}$ and $R_1 = \{ r_1, r_2 \}, R_2 = \{ r_2, r_3 \}$ and $R_3 = \{ r_1, r_3 \}.$

So, we cannot derive explicit resource specification from exclusion specs. In a sense, exclusion specs is stronger than explicit resource specification.

To derive the exclusion specs from explicit resource specification, we first list the pairs of users that can not run together. Then any superset of any pair is a member of exclusion specs. More formally, given an explicit specification $\mathcal{R}$ that consists of users $u_1, u_2, ..., u_n$ and resource sets $R_1, R_2, ..., R_n$. Then the exclusion specs is:

$$B = \{ E | \exists x, y \text{ such that } u_x, u_y \in E \text{ and } R_x \cap R_y \neq \emptyset \}.$$  

2.1 Resource Allocation Problem

**Definition 2** Exclusion condition: There is no reachable configuration where the set of users in CS is a set in $B$.

**Definition 3** Progress condition: If some processor is in Entry then at some later point some processor must enter CS.
Definition 4  Independent Progress condition: If processor $u_i$ is in entry then at some later point in time either $u_i$ or some conflicting user $u_j$ must enter CS. Note that 4 is stronger than 3.