Lecture 9: Roadmap

- N-Process Mutual Exclusion: Bakery
- Other Solutions to Mutex
- MuxSem: Mutex with a Semaphore
- Probabilistic Solutions to Mutex
- A Possible Competition
- From Competition to Mutual Exclusion
- Resource Allocation
- The Dining Philosophers Problem (DPP)
- A wrong solution to DPP
- A (asymmetric) solution to DPP
- From DPP to Resource Allocation
N-Process Mutual Exclusion: Bakery

This solution is due to Lamport. The idea is simple: processes that wish to enter the critical region get “numbers” that indicate their position in line. A process with the lowest number enters the critical section. Ties (processes with same numbers) can be created because of the asynchronous nature of the system. However, they are broken using the processes’ ids.

One attractive feature of this solution is the lack of a “real” shared variables. All shared variables are distributed shared variables, i.e., only one process is allowed to write onto them, and all can read them. These are just like the \( y_i \) (and unlike \( \text{turn} \)) of Peterson’s solution.

Using non-IOA terminology, the bakery algorithm can be described by:
The Bakery Algorithm

\[
\begin{align*}
\text{in} & \quad N : \text{natural where } N > 1 \\
\text{local} & \quad y : \text{array } [1..N] \text{ of natural init 0} \\
\text{local} & \quad choosing : \text{array } [1..N] \text{ of boolean init 0} \\
\text{local} & \quad m.y : \text{natural} \\
\text{loop forever do} \\
& \begin{cases}
0 : & \text{NonCritical} \\
1 : & choosing[i] := 1 \\
2 : & m.y := (\max_{j=1}^{N} y[j]) + 1 \\
3 : & y[i] := m.y \\
4 : & choosing[i] := 0 \\
5 : & \textbf{await } \forall j \neq i : \neg choosing[j] \land \\
& \quad (y[j] = 0 \lor (y[i], i) <_{\text{lex}} (y[j], j)) \\
6 : & \text{Critical} \\
7 : & y[i] := 0
\end{cases}
\end{align*}
\]

\[
\prod_{i=1}^{N} P[i] ::
\]
Mutual Exclusion of Bakery

The following are invariants for every $i$:

\[(I1) \quad C_i \rightarrow y[i] > 0 \]
\[(I2) \quad C_i \rightarrow \forall j \neq i : (y[j] = 0 \lor (y[j], j) \prec_{lex} (y[i], i)) \]

The mutual exclusion property now immediately follows from (I1) and (I2), since (I1) implies:

\[C_i \land C_j \rightarrow y[i] > 0 \land y[j] > 0\]

and from (I2) we now get

\[C_i \land C_j \land i \neq j \rightarrow (y[j], j) \prec_{lex} (y[i], i) \land (y[i], i) \prec_{lex} (y[j], j)\]

which is impossible since $\prec_{lex}$ totally orders the $(y[k], k)$ pairs.
Liveness of Bakery

Assume we want to show liveness w.r.t. $P_i^*$, i.e., that $T_i^* \rightsquigarrow \Diamond C_i^*$. Note first that from fairness considerations it’s obvious that $T_i^* \rightsquigarrow \Diamond \text{at}_- \ell_5[i^*]$, so we’ll only establish $\text{at}_- \ell_5[i^*] \rightsquigarrow \Diamond C_i^*$.

To establish the above, we use well founded induction over the domain $(\mathbb{N}^{10}, <_{lex})$, i.e., 10-tuples of naturals with the usual lexicographical ordering. The goal states are, of course, the states where $\text{at}_- \ell_6[i^*]$. In all other states, we assume that $\text{at}_- \ell_5[i^*]$, and rank them by the tuple

$$(N_3^-, N_4^-, N_5^-, N_6, N_7, N_0, N_1, N_2, N_3^+, N_4^+, N_5^+)$$

where $N_\ell$ denotes the number of processes in location $\ell$, $N_3^+$ and $N_4^+$ denote the number of processes $j$ in locations 3 and 4 (respectively) such that $(\text{my}, j) >_{lex} (y[i^*], i^*)$, $N_5^+$ denotes the number of processes $j$ in location 5 such that $(y[j], j) >_{lex} (y[i^*], i^*)$, and similarly for $N_3^-$, $N_4^-$ and $N_5^-$. 
Liveness of Bakery

With the ranking \((N_3^- N_4^- N_5^- N_6 N_7 N_0 N_1 N_2 N_3^+ N_4^+ N_5^+)\) we have:

Processes from location 0 may go to 1, 2, and then to \(3^+\), thus decreasing the ranking. Obviously, once in \(3^+\), they can advance to \(4^+\) and then to \(5^+\) with each step decreasing the rank. From \(5^+\), no process can progress as long as \(at_\ell_5[i^*]\).

Processes from \(3^-\) will (fairness) progress to \(4^-\), and then (fairness again) to \(5^-\) with, from which they’ll progress to 6, 7, and then 0. Thus, each transition decreases the ranking. Moreover, from each state there are some enabled transitions.

We can therefore conclude the eventually \(P_{i^*}\) reaches location 6.
Other Solutions to Mutex

Since mutual exclusion is all about symmetry breaking, as we saw time and again, there are no deterministic solutions to the problem that are fully symmetric and use only shared distributed variables, or any r/w shared variables.

There are, however, solutions that use semaphores. Semaphores are shared variables that support two types of operations, \texttt{REQ} and \texttt{REL}. A \texttt{REQ}(y) operation performs, in a single indivisible step:

\[ \text{await } y > 0 \text{ and then } y := y - 1 \]

thus, both read and write are performed atomically.

A \texttt{REL}(y) operation performs

\[ y := y + 1 \]

Together with semaphore we often need strong fairness, that requires that if somebody request the semaphore, and the semaphore is available infinitely often, then eventually the request will be granted.
MuxSem: Mutex with a Semaphore

\[\begin{align*}
\text{in} & \quad N : \text{natural where } N > 1 \\
\text{local} & \quad y : \text{boolean where } y = 1 \\
\text{loop forever do} & \\
\begin{array}{c}
\begin{array}{c}N \quad \text{Non-Critical} \\
T \quad \text{REQ}(y) \\
C \quad \text{Critical} \\
E \quad \text{REL}(y)
\end{array} \\
\end{array} \\
\begin{array}{c}
\begin{array}{c}N \parallel P[i] :: \\
\end{array} \\
\end{array} \\
\end{align*}\]

We prove the mutual exclusion using the auxiliary invariant

\[|\{i : C_i \lor E_i\}| + y = 1\]

(assuming, of course, we know that \(y \in \{0, 1\}\) is invariant.)

Note that it’s necessary to prove that the auxiliary invariant holds at the intimal step, that it is inductive, and that it implies (together with the invariant \(C_i \rightarrow y = 0\)) the mutual exclusion property.

As to liveness, for livelock freedom, it suffices to assume weak fairness on the semaphore. For individual liveness, it is necessary to assume the stronger fairness.
Probabilistic Solutions

As usual, when we can’t find fully symmetric deterministic solutions, we break symmetry by introducing randomization. For mutual exclusion, we propose the following:

Consider a competition that no process can join once it starts. In the competition, participants flip coins. The losers, that draw tail, wait until there are no winners. The winners, that draw head, continue to compete until there is a single winner, who enters the critical section, or until there are no winners, in which case they start again.

Since no process can join the competition, processes that leave it (and the only way to leave it is by entering the critical section) cannot re-join it as long as there are other competitors. The competition is controlled by a doorway led to by a waiting room. The door from the waiting room to the competition is open only when the competition is not active or when the last competitor leaves its critical section—it then opens the doorway and awaits until all those in the waiting room enter the competition.
A Possible Competition

Consider the following competition section that guarantees *livelock freedom* with probability 1. That is, if we assume a fair coin-tosser whose actions are independent on the environment, or, alternatively, an infinite sequence of coin flips that are generated fairly, which nobody but those in need of a coin-toss can sample, then almost all the computations of the system guarantee livelock freedom.

\[
\begin{align*}
\text{in } & \quad N : \text{ natural where } N > 1 \\
\text{in my : array } [1..N] \text{ of } \{h, t, l\} \text{ init } t \\
\quad & \begin{cases}
0 : & \{0.5 : \text{my}[i] := h; \text{goto 1}, 0.5 : \text{my}[i] := l; \text{goto 1}\} \\
1 : & \text{if } \forall j \neq i : \text{my}[j] \notin \{h\} \text{ goto 6} \\
2 : & \text{my}[i] := t \\
3 : & \text{await } \forall j \neq i : \text{my}[j] \notin \{h\}; \text{goto 0} \\
4 : & \text{await } \forall j \neq i : \text{my}[j] \notin \{t, h\} \\
5 : & \text{my}[i] := t; \text{ goto 0} \\
6 : & \text{await } \text{false}
\end{cases}
\end{align*}
\]
Liveness of Competition

We show that $\exists i : at_\cdot l_0[i] \leadsto \diamond \exists j : at_\cdot l_6[j]$ with probability 1, using the well founded domain $(\{0, 1, 2, 3\} \times \mathbb{N}^5, \leq_{lex})$. The goal states are all ranked $(0, -, -, -, -, -)$. Unlike the deterministic case, where we required that from each state, each transition leads to the same or to a better rank (with a guaranteed transition that leads to a better ranked state), here we allow some transitions to lead to a worse rank. However, we allow this to happen only for probabilistic transitions that can also lead to a better ranked state.

Thus, for every transition, we require that the transition leads to a better ranked state, that the transition leads to a state with the same rank preserving the set of of helpful transitions; or that the transition may lead to a worse ranked state, but then it must have a mode leading to a better ranked state.

In addition, we also require that each state has a helpful transition that can lead it to a better ranked state.
Liveness of Competition: Ranking

states satisfying \( N_1 = 1 \land N_2 = 0 \) are ranked helpful transitions
\( N_1 = 0 \land N_2 = 0 \) \( 1, N_5 + N_3, N_0, N_4, -, - \)
\( N_1 > 1 \land N_2 > 0 \) \( 3, N_1, N_2, N_3 + N_5, N_0, N_4 \) (*)

(*): in states where there are no processes in locaitons 3, 5 or 0, then the helpful transition are those of processes in locaiton 4; in all other states, the helpful transitions are those of processes in locations 3, 5 or 0.

Note that in type-1 states, the “rank decreasing” probabilistic transitions are those that draw \( l \), from type-1 states thare are those that draw \( h \), and from type-3 states they are those that draw \( l \).

Suppose we are given a type-3 state with one process in location 2, and consider the states after the process at location 2 takes a step. If there are no processes in location 1, the resulting state is a type-2 state. If there is a sigle process in location 1, it’s a type-1 state. Otherwise, it’s (a better ranked, but still) a type-3 state.
Example: Type-2 states

Consider a state where there are no processes with $h$ value. Such a state is ranked $(2, N_4, N_3 + N_5, N_0, -, -)$. (If there are no processes in locations 3,5 or 0, then the rank is $(2, N_4, 0, 0, -, -)$. If there is a process in location 0, it may draw $h$ and bring the system into a type-1 state. If there is a process in location 3, it will enter location 0 and decrease the rank. If there is a process in location 4, then if there are no processes in locations 3, 5 or 0, this process will enter location 5 and decrease the rank. Else, it will stay in place and leave the rank, and the set of helpful transitions, intact. If there is a process in location 5 it will progress to location 0 and decrease the rank. Thus, every transition satisfies our requirements, and helpful transitions may lead to better ranked states.
From Competition to Mutual Exclusion: I

Note first that we can add a new value, \( a \), to the system, and rename locations, not impacting its correctness:

\[
\begin{align*}
\text{in} & \quad N : \text{natural where } N > 1 \\
\text{in} & \quad my : \text{array } [1..N] \text{ of } \{ h, t, l, a \} \text{ init } t \\
N \parallel \sum_{i=1}^{N} P[i] :: & \begin{cases} 
D & : \{ 0.5 : my[i] := h; \text{goto } H, 0.5 : my[i] := L \} \\
H & : \text{if } \forall j \neq i : my[j] \not\in \{ h, a \}; \text{goto } C \\
H1 & : my[i] := t \\
H2 & : \text{await } \forall j \neq i : my[j] \not\in \{ h, a \}; \text{goto } D \\
L & : \text{await } \forall j \neq i : my[j] \not\in \{ t, h, a \} \\
L1 & : my[i] := t; \text{goto } D \\
C & : my := \{ h, a \}
\end{cases}
\end{align*}
\]

Let’s create a doorway to the competition, so that processes cannot enter at unless the doorway is open. The door will be controlled by the new value \( a \). That is, as long as there is no process with \( a \) value and there are some in the competition, nobody will be able to enter it. To do this, it suffices to add, just before location \( D \), and new location \( W \) that just has the instruction

\[
W : \text{await } \forall j \neq i : my[j] \not\in \{ l, t, h \} \text{ or } \exists j \neq i : my[j] \in \{ a \}
\]
From Competition to Mutual Exclusion: II

But, when should the door be opened? Assume that before they enter competition, processes have a $e$ (for “enter”) value denoting their wish to enter the competition and a $u$ (for “uninterested”) value denoting processes are in idle state.

When the last process from the competition enters the critical section, it should open it! We thus need a more elaborate exit region, replacing $C$ with:

\[
C : \text{ Critical} \\
E1 : \text{ if } \exists j \neq i : \text{my}[j] \in \{h, t, l\} \text{ goto } E2 \\
O1 : \text{my}[i] := a \\
O2 : \text{await } \forall j \neq i : \text{my}[j] \not\in \{e\} \\
E2 : \text{my}[i] := u; \text{ goto I}
\]

Finally, we need to control the doorway, to add, before the competition:

\[
I : \text{ non-critical} \\
T1 : \text{my}[i] := e \\
W : \text{await } \forall j \neq i : \text{my}[j] \not\in \{l, h, t\} \text{ or } \exists j \neq i : \text{my}[j] \in \{a\} \\
T2 : \text{my}[i] := t
\]
The Full Protocol

\begin{align*}
\text{in } N & : \text{ natural where } N > 1 \\
\text{in } \text{my } & : \text{ array } [1..N] \text{ of } \{h, t, l, a, e, u\} \text{ init } t \\
I & : \text{ non-critical} \\
T1 & : \text{ my}[i] := e \\
W & : \text{ await } \forall j \neq i : \text{my}[j] \notin \{l, h, t\} \text{ or } \exists j \neq i \text{ my}[j] = l \\
T2 & : \text{ my}[i] := t \\
D & : \{0.5 : \text{my}[i] := h; \text{goto } H, 0.5 : \text{my}[i] := l \}
\\
H & : \text{ if } \forall j \neq i : \text{my}[j] \notin \{h, a\} \text{ goto } C \\
H1 & : \text{ my}[i] := t \\
H2 & : \text{ await } \forall j \neq i : \text{my}[j] \notin \{h, a\}; \text{ goto } D \\
L & : \text{ await } \forall j \neq i : \text{my}[j] \notin \{t, h, a\} \\
L1 & : \text{ my}[i] := t; \text{ goto } D \\
C & : \text{ Critical} \\
E1 & : \text{ if } \exists j \neq i : \text{my}[j] \in \{h, t, l\} \text{ goto } E2 \\
W1 & : \text{ my}[i] := a \\
W2 & : \text{ await } \forall j \neq i : \text{my}[j] \notin \{e\} \\
E2 & : \text{ my}[i] := u; \text{ goto } I
\end{align*}
Resource Allocation

The mutual exclusion problem is an example of the more general resource allocation problem, in which a set of processes share a set of resources. We assume that with each process we can associate a (fixed) set of resources it uses, and that no resource can be used by more than one process at the time.

There are two alternatives (that are not equivalent!) to specify the requirements on users/resources. In the explicit resource specification one specifies, for each user \( i \), a set \( R_i \) or resources, so that users \( i \) and \( j \) conflict if they share a resource, i.e., if \( R_i \cap R_j \neq \emptyset \). E.g., in Mutex there is one resource (or set of resources) \( R \) and assumed \( R_i = R \) for every \( i \). Alternatively, we can specify an exclusion set that describes the set of indices of processes that not obtain. For mutual exclusion, this set will include all subsets of processes with more than one process.

Solutions to the resource allocation problem should be protocols (IOAs) for the processes that guarantee well-formed executions, exclusion, and progress, under the same assumption and conditions we had for Mutex.
The Dining Philosopher Problem (DPP)

There are $N > 2$ philosophers sitting around a table numbered counter-clockwise $P_1, \ldots, P_N$. Every two adjacent philosophers, $P_i$ and $P_{i \oplus 1}$, share a common fork, $y[i \oplus 1]$.

Philosophers spend most of their lives thinking (non-critical), however, occasionally a philosopher may become hungry. In order to eat, a philosopher needs to obtain both its adjacent forks. A solution to the dining philosophers problem is a program for the philosophers, that guarantees that no two adjacent philosophers eat simultaneously, and that every hungry philosopher eventually eats.

If the system is fully symmetric, there are no deterministic solutions to the problem.
A Wrong Solution to DPP

\[ \text{local } y : \text{array } [1..n] \text{of natural initially } y = 1 \]

\[
\begin{align*}
\frac{n}{j=1} \quad P[j] : & \quad \begin{cases} 
I : \text{Non-Critical} \\
T_1 : \text{REQ } (y[j]) \\
T_2 : \text{REQ } (y[j \oplus n 1]) \\
C : \text{Critical} \\
E_1 : \text{REL } (y[j]) \\
E_2 : \text{REL } (y[j \oplus n 1])
\end{cases}
\end{align*}
\]

The mutual exclusion property is

\[ \forall i : \neg (C_i \land C_{i\oplus 1}) \]

Which follows from the invariant \( C_i \rightarrow \neg (y[i]) \land \neg y[i \oplus 1]. \)
A Wrong Solution to DPP

\[
\begin{align*}
\text{local} \quad y & : \text{array} [1..n] \text{of natural initially} \quad y = 1 \\
\end{align*}
\]

\[
\begin{align*}
\begin{array}{|c|}
\hline
I & : \text{Non-Critical} \\
T1 & : \text{REQ (y[j])} \\
T2 & : \text{REQ (y[j \oplus n 1])} \\
C & : \text{Critical} \\
E1 & : \text{REL (y[j])} \\
E2 & : \text{REL (y[j \oplus n 1])} \\
\hline
\end{array}
\end{align*}
\]

As for liveness, \( T_i \rightsquigarrow \Diamond C_i \), we have a problem... 

Consider an execution in which all the philosophers become hungry and progress to location \( T1 \), i.e., after \( n \) steps the system reaches a state where \( \forall i : T1_i \). Since all the left forks are available, they can all pick them up (one by one), that is, after another \( n \) steps the system reaches a state where \( \forall i : T2_i \). This is a deadlock state: No philosopher can ever take a move now!!
A (asymmetric) solution to DPP

\[
\text{local } y : \text{array } [1..n] \text{ of natural initially } y = 1
\]

\[
\begin{align*}
\forall j = 2^n \quad P[j] \& \quad \begin{cases}
I & \quad \text{Non-Critical} \\
T1 & \quad \text{REQ } (y[j]) \\
T2 & \quad \text{REQ } (y[j \oplus n 1]) \\
C & \quad \text{Critical} \\
E1 & \quad \text{REL } (y[j]) \\
E2 & \quad \text{REL } (y[j \oplus n 1])
\end{cases}
\end{align*}
\]

\[
\begin{align*}
P[1] \& \quad \begin{cases}
I & \quad \text{Non-Critical} \\
T1 & \quad \text{REQ } (y[2]) \\
T2 & \quad \text{REQ } (y[1]) \\
C & \quad \text{Critical} \\
E1 & \quad \text{REL } (y[2]) \\
E2 & \quad \text{REL } (y[1])
\end{cases}
\end{align*}
\]
Correctness of the New DPP Solution

Mutual exclusion remains that same as in the previous (wrong) solutions. However, liveness now holds. Informally, the “contrary” philosopher cannot join the deadlock state. It will thus eat, and release both its forks. Its left neighbour now can pick up their joint fork and eat.

Formally, since there are two types of protocols here, one for a “regular” \( P_2 \ldots P_N \) philosopher, and one for the contrary \( P_1 \) philosophers, we need to prove liveness for each separately.