1 Formal Model for Message Passing Systems

In this particular model, there are $n$ processors/states. Each pair of processors $(i, j)$ contains an out buffer and an in buffer.

**Definition 1** The contents of each out buffer (outbuf) and in buffer (inbuf), and the state of the processors comprise the *configuration*.

1.1 Events

1. **Delivery Event** - environment/message delivery system - Removes message $m$ from outbuf[$i,j$], and places message $m$ into inbuf[$i,j$]

2. **Computation event** - processor event - Processes messages in inbuf, empties inbuf, changes state, and passes message to outbuf.

1.2 Execution

**Definition 2** *Accessible* - The only data accessible to processor $i$ are the processor’s internal state, and the contents of the processor’s inbuffers.

**Definition 3** *Execution* - Sequence of alternating configurations and events starting at the initial configuration $C_0$: $C_0, \phi_1, C_1, \phi_2, C_2, \ldots$. If $C_0$ is not an initial configuration, then the sequence is an *execution fragment*.

**Theorem 1** Event $\phi_k$ is *applicable* to $C_{k-1}$ (event can happen at that configuration) if:

1. $\phi_k = del(i,j,m)$ (It's a delivery event - $m$ must be in outbuf[$i,j$] - $\phi_k(C_{k-1}) = C_k$ (m is removed from outbuf[$i,j$] and placed in inbuf[$i,j$] - No other changes)

2. $\phi_k = comp(i)$ (Its a computation event - must be determined by portion of $C_{k-1}$ accessible to $P_i$ and program of $P_i$. $C_k$ and $C_{k-1}$ differ only in the state of $P_i$, $\forall j$ inbuf[$j,i$] is emptied in $C_k$, and $\forall j$ outbuf[$i,j$] has more messages.
A schedule is a sequence of events in execution (but not every sequence is a schedule). i.e., $\phi_1, \phi_2, \phi_3 : \exists C_1(C) = C_1, \phi_2(C_1) = C_2, \phi_3(C_2) = C_3$

An execution is valid if $\forall k, \phi_k$ is applicable to $C_{k-1}$.

The Safety Condition is the idea that “nothing bad ever happens”. It can be checked in every prefix, and is a property of a finite execution. Validity is a safety condition. The Liveness Condition is the idea that “Something good eventually happens”. We cannot check in a finite execution, because it must hold over the entire execution and is a property of infinite execution. Admissability is a liveness condition.

## 2 Comparison Between Models

### Definition 4 Admissability

In the asynchronous model, an execution is admissible if each processor takes an infinite number of steps and every message is eventually delivered. In the synchronous model, in each round all messages are delivered, then processor events happen, and then messages are placed in outbuffers. (In the asynchronous model, there are many possible executions (non-deterministic) and in the synchronous model, there is only one possible).

1. Synchronous vs. Asynchronous Processors
2. Synchronous vs. Asynchronous Communication

If both the processors and the communication system are synchronous, this is the strongest model because it allows for the weakest possible algorithms. If both are asynchronous, this is the weakest model, and needs the strongest possible algorithms.

An algorithm in the asynchronous model implies an algorithm in the synchronous model. But, an impossibility result in the synchronous model implies the same in the asynchronous model. Hence, we can perform general transformations of an algorithm from one model to another.

Given an algorithm, a synchronous system allows a set of executions $S$ and an asynchronous system allows a set of executions $A$. An algorithm that terminates successfully in all executions $e \in A$ will terminate successfully in $S$ since $S \subseteq A$. A problem is solvable if $\exists a \forall e \in S$, algorithm $a$ terminates correctly in execution $e$. Conversely, a problem is unsolvable if $\forall a \exists e \in S$ such that algorithm $a$ terminates unsuccessfully in execution $e$.

We will show that even if a processor is synchronous, the communication method remaining asynchronous allows a worst-case execution equivalent to that of a processor that is asynchronous with asynchronous communication. Consider the following two models:

**Model A:** Asynchronous Processor, Asynchronous Communication

**Model B:** Synchronous Processor, Asynchronous Communication

Note the following two observations:
1. Any execution of B is an execution of A

2. Any execution of A is not necessarily an execution of B

**Definition 5** Executions $e$ and $e'$ are equivalent if $\forall i, P_i$ cannot distinguish between $e$ and $e'$ (states are same).

**Definition 6** Model A is at least as strong as model B, if for every execution $e$ of model A, there is an equivalent execution $e'$ of model B such that the processors can’t tell them apart.

Because $E_B \subset E_A$, clearly B is at least as strong as A. However, it turns out B is no stronger.

**Theorem 2** Model A is at least as strong as model B.

**Proof:** Consider any execution $e$ of model A. Construct $e'$ for B such that $e'$ is equivalent to $e$ in the following manner. Consider any computation event $c \in e$. Let message $m_i$ be an arbitrary message arriving before $c$, and let $t_i$ be the time of the computation event $c$ processing message $m_i$ in $e$. Delay the delivery of message $m_i$ in $e'$ until $t_i$. Since the processor is synchronous, the message is processed at the same time $t_i$ as in $e$. Each message delivery can be delayed in $e'$ such that the processors process the message at the same time as in $e$. These executions cannot be distinguished from one another, and hence by definition 5, they are equivalent. Therefore, by definition 6, Model A is at least as strong as Model B. □