1. Let \( A \parallel B \) represent concurrency between two events \( A \) and \( B \), which we define formally as

\[
A \parallel B = \neg(A \rightarrow B) \land \neg(B \rightarrow A),
\]

where \( \rightarrow \) means that event \( A \) happens before \( B \) (note that we are overloading the operator \( \rightarrow \) depending on whether it acts on predicates or events). Prove that concurrency is transitive:

\[
(A \parallel B) \land (B \parallel C) \implies (A \parallel C)
\]

2. [Sivilotti, Section 5.4.3] Consider the timeline shown below:

Assign vector timestamps to the events (indicated by circles) in accordance with the algorithm listed in Sivilotti, Section 5.4.3. Initially, the clocks begin with the values indicated.

3. A distributed system \( D \) is represented by a directed graph in which the vertices represent agents and directed edges represent directed channels. For this problem we use the standard model of channels: every message sent on a channel will be delivered eventually, and messages are delivered in the order sent.

An agent is in one of two states: \textit{active} or \textit{idle}. An active agent may send and receive messages. An idle agent remains idle until it receives a message at which point it becomes active. Initially, no messages have been sent, and so all the channels are empty; also all the agents are active.
The system is said to have terminated execution if and only if all the agents are idle and all the channels are empty. Let \( \text{terminated} \) be the predicate that holds exactly when the system has terminated execution. You can prove that:

\[
\text{stable}(\text{terminated})
\]

Associated with the system is a single OS process. The OS process is outside the system, i.e., the OS process is not one of the agents in \( D \). We want an algorithm by which the OS process detects when the system has terminated. The OS process has a Boolean variable \( \text{claim}_\text{terminated} \), and the specification of the problem is:

- **Safety**: The OS claims the system has terminated only if the system has terminated.

  \[
  \text{invariant}(\text{claim}_\text{terminated} \Rightarrow \text{terminated})
  \]

- **Progress**: If the system terminates execution then eventually the OS will claim that the system has terminated execution.

  \[
  \text{terminated} \Rightarrow \text{claim}_\text{terminated}
  \]

(a) Here is an algorithm that assumes an idealized OS. Initially \( \text{claim}_\text{terminated} \) is false.  
Assume that the OS can record the state of the entire distributed system \( D \), i.e., the states of all the agents and the states of all the channels in \( D \), at each instant in time. (We are not concerned here with whether this OS is implementable.) The OS checks the state of \( D \) periodically with some finite period \( T \). When the OS checks the state of \( D \), if the OS determines that all the agents are idle and all the channels are empty — i.e. that \( \text{terminated} \) holds — then the OS changes the value of \( \text{claim}_\text{terminated} \) from false to true.

Is the algorithm correct? Prove or give a counterexample.

(b) Here is a suggestion for an implementable algorithm.

Each agent has a channel from the agent to the OS process. Each agent records the total number of messages it has sent to other agents and the total number of messages that it has received from other agents. When an agent \( r \) transits state from active to idle, it sends a message containing the tuple \((r, r.\text{number\_sent}, r.\text{number\_received})\) to the OS along the channel from \( r \) to the OS where \( r.\text{number\_sent} \) and \( r.\text{number\_received} \) are the total number of messages that agent \( r \) has sent to other agents, and the total number of messages that agent \( r \) has received from other agents (respectively). Note that messages between an agent and the OS are not counted in the number of messages sent/received among agents.

The OS maintains variables \( r.\text{count\_sent} \) and \( r.\text{count\_received} \) for each agent \( r \). Initially, the values of these variables are 0. Initially \( \text{claim}_\text{terminated} \) is false.

When the OS receives a message \((r, r.\text{number\_sent}, r.\text{number\_received})\), it sets the values of the counters \( r.\text{count\_sent} \) and \( r.\text{count\_received} \) to \( r.\text{number\_sent} \) and \( r.\text{number\_received} \), respectively. The OS sets \( \text{claim}_\text{terminated} \) to true if the OS has received at least one message from each agent, and:

\[
\sum_r r.\text{count\_sent} = \sum_r r.\text{count\_received}
\]

Is the algorithm correct? Prove or give a counterexample.
(c) Here is a variant of the algorithms given in parts (a) and (b).

Each agent maintains a logical clock. Assume that the logical clock never stops, i.e., the value of the logical clock always increases eventually. Also, assume that the value of the logical clock is an integer that starts at 0 and ticks forward by 1 at each tick. This is a slight variant of the algorithm in the paper and the book, because in the book, an agent could have a logical clock with value 10 and when it receives a message with timestamp 20 the agent sets its clock to 21 in a single step; here we assume that the agent moves its clock from 10 to 21 in the sequence of steps: 10, 11, 12, ... 21.

This algorithm is exactly like the one given in part (a) except that we are using logical clocks as opposed to idealized perfect clocks. Each agent records its own state periodically with some period $T$. Here the period is with respect to the agent’s logical clock. Agent $r$ sends the message $(r, \tau, r.state, r.number\_sent, r.number\_received)$ to the OS, where $\tau$ is the time, according to its logical clock, that $r$ sends the message, and $r.state$ is $r$’s state at that point.

The OS maintains a table for each value of $\tau$ where the table contains the states and counts it has received from each agent for that value of $\tau$. If for any $\tau$, the table has all agents idle, and the total numbers of messages sent equal to the total number of messages received, then the OS sets $claim\_terminated$ to true.

Is the algorithm correct? Prove or give a counterexample.