1. Consider two UNITY program $F$ and $G$ and let $F \parallel G$ represent the union of the two programs. We write $P in F$ to say that a property $P$ holds in a program $F$. Prove the following statements formally or provide counterexamples with formal program definitions.

(a) $P \text{ ensures } \neg P$ in $F$ $\implies$ $P \text{ ensures } \neg P$ in $F \parallel G$ for any $G$. [5 points]

(b) $(P \rightsquigarrow Q \text{ in } F) \land (\text{stable}(P) \text{ in } G) \implies P \rightsquigarrow Q$ in $F \parallel G$ [5 points]

(c) $(P \rightsquigarrow Q \text{ in } F) \land (P \rightsquigarrow Q \text{ in } G) \implies P \rightsquigarrow Q$ in $F \parallel G$ [5 points]

2. Consider the following program:

```
Program GCD
var x, y : integers
initially x = X \land y = Y \land x > 0 \land y > 0
assign
  x > y -> x := x - y
  x < y -> y := y - x
```

(a) Write a program that uses superposition to count the number of actions that are executed before the $GCD$ program reaches a fixed point. Write down your program formally similar to the one in the Lecture 7.1 slide 9. [5 points]

(b) Show that the program you write down is correct. In other words, show that your program terminates and the count is correct. You do not need to prove that the original program $GCD$ terminates. [10 points]

3. A key idea in conflict resolution algorithms, as illustrated by the dining philosophers problem, is to maintain a priority graph structure among agents which has the following properties:

1. The graph is acyclic, and
2. The graph changes in a fair way, so that every agent that needs a resource gets higher priority than its neighbors eventually. In the case of the dining philosophers problem: every hungry agent gets to eat eventually.

One class of algorithms for conflict resolution uses each agent’s local clock time. The local clocks are not synchronized. If $t[i]$ is the time for agent $i$ and $T$ is the real time, then the maximum drift is a constant value, $\epsilon$, i.e.,

$$\forall i : |t[i] - T| \leq \epsilon$$
We don’t know the value of $\epsilon$ but we know it exists. So, clocks cannot drift apart by arbitrarily large amounts.

Each agent’s clock moves forwards (never backwards and never stays still). Assume that clock values and $\epsilon$ are integer so that we can carry out induction on their values (If clock values were real numbers then we can have a situation where a clock ticks forward by $0.5^n$ on the $n$-th step, and thus the clock never increases by more than 2).

Also, when an agent makes a state transition (e.g., thinking to hungry) the agent’s clock ticks forward. An agent does not make a state transition while its clock remains unchanged.

This question is about the relationship, if any, between acyclic priority graphs and time-based conflict resolution.

(a) Which of the following methods would you use to define the priority in terms of local clock values so that the two conditions (listed above) for priority graphs are met? In particular, could you solve the dining philosophers problem by allowing a hungry philosopher with higher priority than all its neighbors to eat, where priority was defined in terms of local clock time?

1. An agent’s priority is the ordered pair [agent’s current clock value, agent id].
2. An agent’s priority is the agent’s current clock value.
3. An agent’s priority is defined as follows. If the agent is waiting to enter a critical section (i.e., hungry) or in a critical section (eating) then the priority is the ordered pair [the agent’s local clock value at the point at which the agent last started waiting, agent id]. If the agent is executing outside the critical section (thinking) then the priority is [agent’s current clock value, agent id].

Note: Lower numbers indicate higher priority.

(b) Show that the method you chose satisfies the safety property that the priority graph is acyclic.

(c) Show that the method you chose satisfies the progress property that every hungry agent in dining philosophers gets to have higher priority than all its neighbors eventually (and therefore gets to eat eventually).