Computer Science 2221	**Homework 2**
University of Toronto	**Due: Friday, October 24, 2003**

This problem set is about the round-model synchrony assumptions, early-stopping TRB, uniform TRB, causality and logical time.
The first question concerns the round-model assumption that clocks are perfectly synchronized, and how one can weaken this assumption.

1. The round-model of computation (assumed by the $t+1$-rounds TRB algorithm that we gave in class) is based on the assumption of perfect synchrony:

   - The processes' clocks are perfectly synchronized with one another and with real-time.
   - there is a known bound δ such that a message sent at real-time t is received by real-time $t+\delta$.

   Thus, all processes start a round at the same real-time and end it simultaneously δ real-time units later. However, perfect clock synchronization is not achievable.

   Suppose clocks are only ϵ-synchronized, i.e. at all times the difference between two clocks is bounded by ϵ. For example, a process may "think" that the current time is c, while another may "think" that the current time is $c-\epsilon$. Thus, processes do not necessarily start a round at exactly the same real-time.

   (a) Show (by giving a counter-example) that the $t+1$-rounds TRB algorithm given in class, where the length of a round is δ, does not work anymore.

   (b) Modify the algorithm so that it works when the clocks are ϵ-synchronized. (Note: This is a very simple modification.)

The next questions concern Uniform TRB — *a stronger version of TRB, where the* Agreement *requirement is replaced with:*
Uniform Agreement: *If* any *process (whether correct or faulty) delivers a message m, then all correct processes eventually deliver m.*

2. Consider the $t+1$-rounds TRB algorithm given in the class notes. Show that this algorithm does not solve the Uniform TRB problem in a system with process crashes (i.e., give an explicit execution of this algorithm that violates Uniform Agreement).

   Modify the algorithm so that it *does* solve Uniform TRB in systems with process crashes, and prove it correct (you only need to give the parts of the proof that changed from the original proof).

3. Consider the problem of solving Uniform TRB in a round-based system with *general omission* failures.

   (a) Assume that a *majority* of the processes are correct, i.e., $n > 2t$. Describe a round-based algorithm that solves the above problem and prove it correct.

(b) Prove that this problem cannot be solved if $n \leq 2t$. (Hint: partition the n processes into two sets of size at most t, and consider several scenarios.)

The next question concerns early-stopping TRB.

4. Consider the early-stopping algorithm that we saw in class for TRB.

   Replace the statement:

   "**else if** $|quiet^i| < i$ **then** deliver SF" with

   "**else if** $|quiet^i| = |quiet^{i-1}|$ **then** deliver SF"

   In this protocol, a process halts if it does not see any *new* failures in a round. Thus, if all f failures occur in the first r rounds, every correct process decides by round $r+2$ and halts by round $r+3$.

   (a) Prove this is a t-tolerant early-stopping TRB protocol for crash failures.

   (b) Prove it is *not* a correct t-tolerant TRB protocol for send-omission failures.

The last question regards causality and an extension of Lamport's clocks.

5. In class, we defined the *happens before* (also called *causal precedence*) relation denoted $\rightarrow$. We also described a simple distributed algorithm to implement clocks (*Lamport Clocks*) that satisfy the following property:

   $$\text{If } \; a_i \rightarrow b_j \; \text{ then } \; C(a_i) < C(b_j)$$

   In this implementation, each clock is just an integer variable.

   (a) Prove that Lamport's Clocks do **not** satisfy the converse property:

   $$\text{If } \; C(a_i) < C(b_j) \; \text{ then } \; a_i \rightarrow b_j$$

   Suppose we know the Lamport's Clock times $C(a_i)$ and $C(b_j)$ of two events a_i and b_j. Can we use these times to determine whether (i) $a_i \rightarrow b_j$, or (ii) $b_j \rightarrow a_i$, or (iii) a_i and b_j are concurrent? (Recall that two events a_i and b_j are *concurrent* if neither $a_i \rightarrow b_j$ nor $b_j \rightarrow a_i$.)

   (b) Give an algorithm that implements logical clocks that satisfy:

   $$a_i \rightarrow b_j \text{ if and only if } C(a_i) \prec C(b_j)$$

   where $\prec$ is a partial order relation on clock values.

   Hint: Each clock will now be a *vector* of integers (instead of a single integer), one component for each process. You must describe how to compare vectors $C(a_i)$ and $C(b_j)$ to determine whether $C(a_i) \prec C(b_j)$ (and thus $a_i \rightarrow b_j$).