1 Time-Stamp Systems

In many computer applications, it is important to establish the ordering among items. For example, in a database system, one could have a set of records, say A and B, and want to continuously record which item was last updated. A simple way to do this is to add to each item a "label," a natural number that is increased every time the record is updated. For items A and B, we can denote their respective labels $l(A)$ and $l(B)$, and the "update time" order among labels as $<_T$. If at some point $l(A) <_T l(B)$, and A is updated, then $l(A)$ can be set to $l(B) + 1$ changing the order to $l(B) <_T l(A)$, and indicating that A was the last item to be updated. Unfortunately, this kind of scheme introduces a problem when we try and implement it. The problem is that on a computer, the labels are stored each in a register, and registers are of bounded size. If the labels grow too big, the registers will overflow and the order of their values will no longer describe the order of updates to the items. For example, if the register has only two bits, $l(A) = 2$ and $l(B) = 3$. Then if A is updated, it will get set to $l(B) + 1$ which overflows the register so $l(B) + 1 = 4 mod 4 = 0$ so $l(A) <_T l(B)$, contrary to what we actually wanted to achieve.

Is this problem solvable without labels that are of unbounded size? To solve this problem, we need to be able to design a *finite* set of labels and an ordering among them so that the order among labels reflects the real order of updates at any time. Such a system is called a bounded time-stamp system.

A time-stamp system is an abstract data type that allows two types of operations:

1. Label.

2. Scan (read the labels and return the order between them).

Label operations assign a new higher label, and scans return an order among all labels.

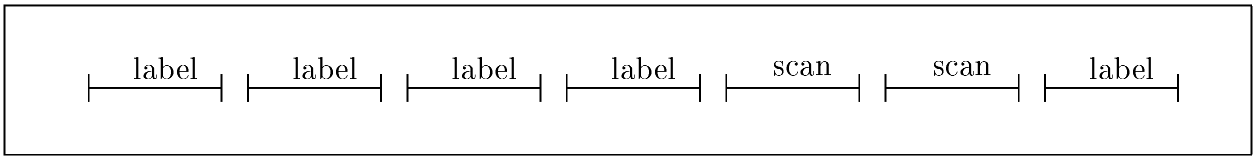

Figure 1: An execution of labels and scans.

We can mathematically model time-stamp systems as graphs. The "unbounded time-stamps" we use the directed acyclic graph $T = (G, <_T)$, where G is the set of vertices which represent the various labels and $<_T$ is the set of edges which represent the order between them. The unbounded time-stamp system described above is represented by the infinite tournament graph in labels Figure 2. A tournament graph is a directed graph that has a single directed edge between any two vertices.

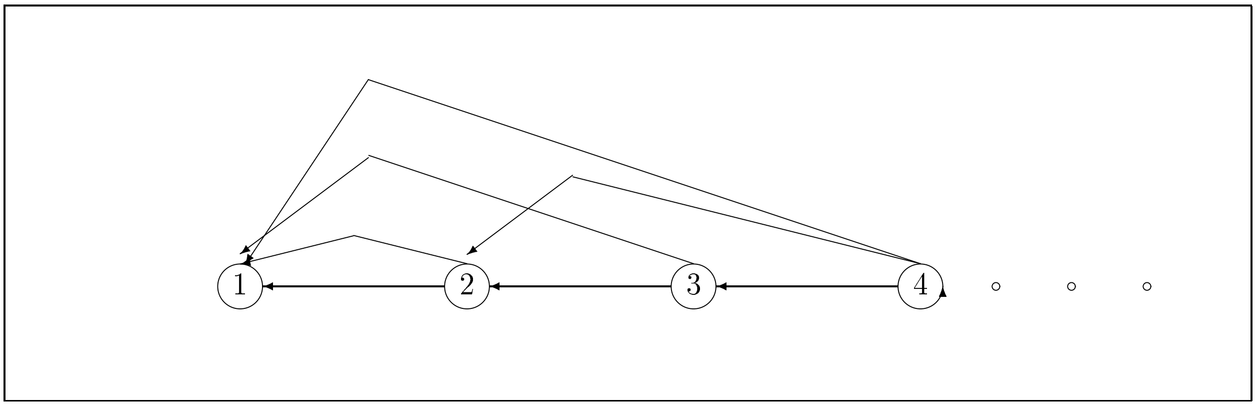

Figure 2: The unbounded natural number based time-stamp system.

The unbounded tournament graph is acyclic, which is what it is easy to keep the order among labels. One can think of the execution of a sequence of label and scan operations as a *pebble game* on this graph. For n objects that we wish to label, we have n different colored pebbles, and since this is a tournament graph, there is an order among any set of pebbles, determined by the order of the edges. If there is a directed edge (A, B) (from A to B) then $l(B) <_T l(A)$. As can be seen, on the graph of the unbounded tournament graph of Figure 2, it is easy to pick a vertex which dominates all others with pebbles on it. This choice always gives a correct total ordering of labels since there are no cycles in the graph.

However, if the values of the labels are bounded, this property may well be lost, and we need a more sophisticated graph and pebbling game. One time stamps system method due to Israeli and Li, defines a domain of labels (vertexes) and an ordering among them $<_T$ for some k possible pebbles as follows. Define the following (cyclic) tournament graph T^k inductively: T^1 is the single vertex, T^2 is the graph in Figure 3, and $T^k = T^2 \otimes T^{k-1}$. The $\otimes$ is the graph multiplication operator. Graph multiplication $G_1 \otimes G_2$ is the operation that creates a new graph G_3 by taking G_2, replacing every node by a graph of type G_2, and then directing all edges among the newly inserted vertexes according to the order in G_1, creating a new tournament graph.

The pebble game among two pebbles on T^2 is straightforward: In order to illustrate how the pebble game played let us consider the following run of three pebbles in a $T^3 = T^2 \otimes T^2$ graph which is shown in Figure 4:

The notation $l(A) = (i, j)$ means that process A is in graph i at vertex j. This would correcpond to a label with two numbers, "ij", lexicoraphically ordered. At the beginning there are three objects A and B with the following labels: $l(A) = (1, 3)$, $l(B) = (1, 1)$, and $l(C) = (2, 1)$. This implies that $l(B) <_T l(A)$ and both are greater than $l(C)$. A and B can continuously assign new labels by cycling around inside the number 2 T^2 structure (note that if $l(B) <_T l(A)$ and A wishes to change to a higher label, staying with the same label

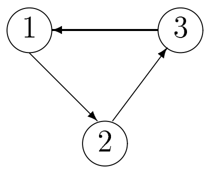

Figure 3: A Sequential Time Stamp System graph for two processes.

suffices to dominate all others). If C wishes to get a new label, it cannot pick a node on T^2 graph number 2, since this is a time-stamp sub-graph for 2 pebbles and it is full with A and B. Thus, C picks $l(C) = (3,1)$ which dominates both A and B. If again A wishes to move, it picks $l(A) = (3,3)$ which dominates C and B. If B now moves, its pebble will go to $l(B) = (2,1)$ and so on. The time-stamp system for k labels extends in the same way to a T^k graph.

2 Bibliography

- A. Israeli and M. Li: "Bounded Time-Stamps", 28th FOCS, 1987.
- D. Dolev and N. Shavit: "Bounded Concurrent Time-Stamps systems are constructible",STOC 1989.

3 Question for Tutorial

- The sequential time-stamp system of Israeli-Li above, has a range of $O(3^n)$ different label values for n processes (each possible label value is a node in the T^n graph). Devise a sequential time-stamp system that uses less than $O(3^n)$ values (hint – try for $O(n2^n)$ labels. Second hint – think of each label as an array of $n + \log n$ bits)

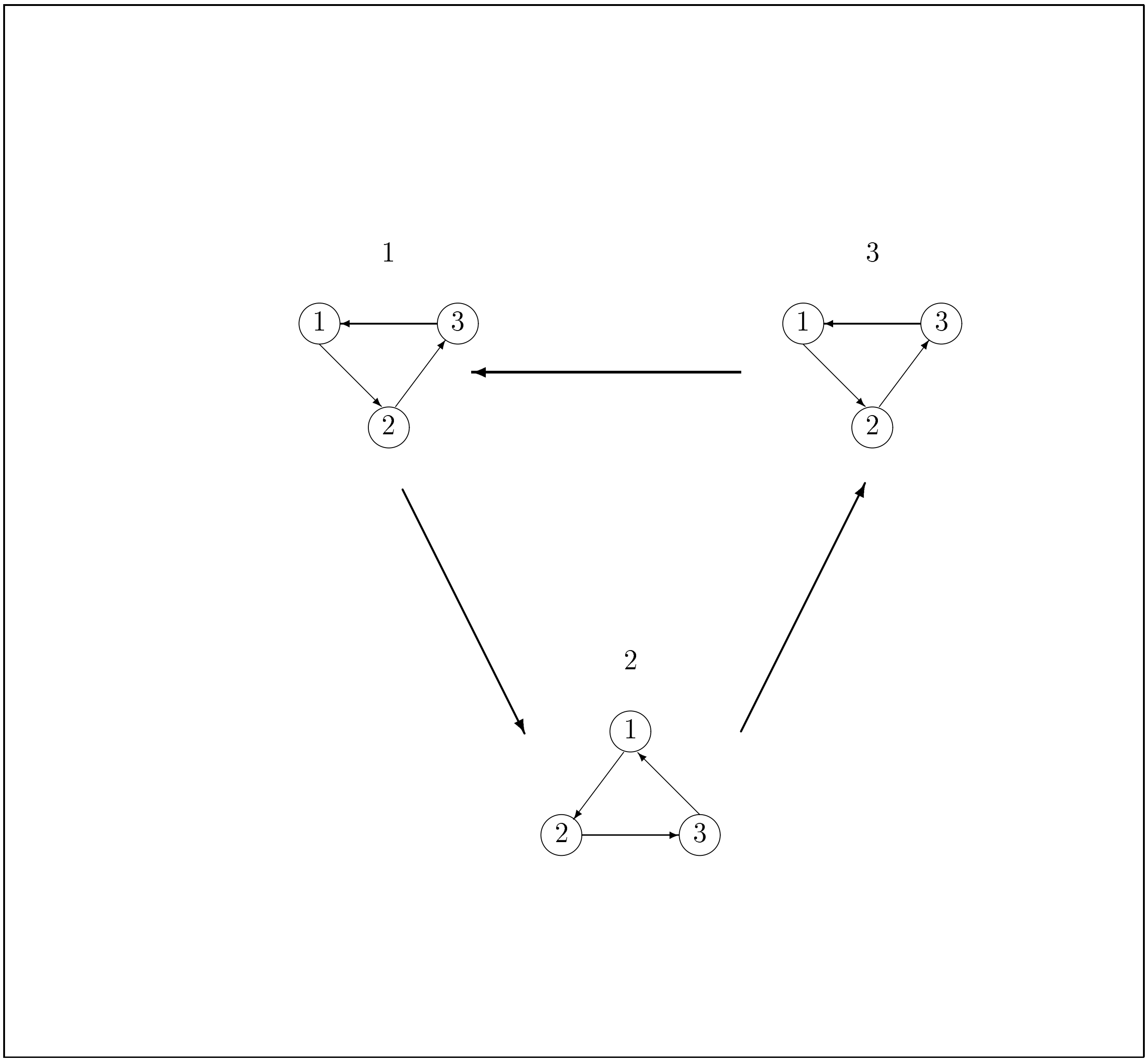

Figure 4: A Sequential Time Stamp System graph for three processes.