receives a basic message, it updates its logical clock based on the clock value contained in the message. Rule R3 states that when a process becomes idle, it updates its local clock, sends a request for snapshot $R(x, k)$ to every other process, and takes a local snapshot for this request.

Rule R4 is the most interesting. On the receipt of a message $R(x', k')$, the process takes a local snapshot if it is idle and $(x', k') > (x, k)$, i.e., timing in the message is later than the local time at the process, implying that the sender of $R(x', k')$ terminated after this process. In this case, it is likely that the sender is the last process to terminate and thus, the receiving process takes a snapshot for it. Because of this action, every process will eventually take a local snapshot for the last request when the computation has terminated, that is, the request by the latest process to terminate will become successful.

In the second case, $(x', k') \leq (x, k)$, implying that the sender of $R(x', k')$ terminated before this process. Hence, the sender of $R(x', k')$ cannot be the last process to terminate. Thus, the receiving process does not take a snapshot for it. In the third case, the receiving process has not even terminated. Hence, the sender of $R(x', k')$ cannot be the last process to terminate and no snapshot is taken.

The last process to terminate will have the largest clock value. Therefore, every process will take a snapshot for it; however, it will not take a snapshot for any other process.

7.4 Termination detection by weight throwing

In termination detection by weight throwing, a process called *controlling agent*[1] monitors the computation. A communication channel exists between each of the processes and the controlling agent and also between every pair of processes.

Basic idea

Initially, all processes are in the idle state. The weight at each process is zero and the weight at the controlling agent is 1. The computation starts when the controlling agent sends a basic message to one of the processes. The process becomes active and the computation starts. A non-zero weight W $(0 < W \leq 1)$ is assigned to each process in the active state and to each message in transit in the following manner: When a process sends a message, it sends a part of its weight in the message. When a process receives a message, it add the weight received in the message to its weight. Thus, the sum of weights on all the processes and on all the messages in trasit

[1] The controlling agent can be one of the processes in the computation.

Two invariants I_1 and I_2 are defined for the algorithm:

$$I_1\colon W_c + \sum_{W \in (A \cup B \cup C)} W = 1.$$
$$I_2\colon \forall W \in (A \cup B \cup C),\ W > 0.$$

Invariant I_1 states that the sum of weights at the controlling process, at all active processes, on all basic messages in transit, and on all control messages in transit is always equal to 1. Invariant I_2 states that weight at each active process, on each basic message in transit, and on each control message in transit is non-zero.

Hence,

$$\begin{aligned} & W_c = 1 \\ & \Longrightarrow \textstyle\sum_{W \in (A \cup B \cup C)} W = 0 \text{ (by } I_1) \\ & \Longrightarrow (A \cup B \cup C) = \phi \text{ (by } I_2) \\ & \Longrightarrow (A \cup B) = \phi. \end{aligned}$$

Note that $(A \cup B) = \phi$ implies that the computation has terminated. Therefore, the algorithm never detects a false termination.

Further,

$$\begin{aligned} & (A \cup B) = \phi \\ & \Longrightarrow W_c + \textstyle\sum_{W \in C} W = 1 \text{ (by } I_1). \end{aligned}$$

Since the message delay is finite, after the computation has terminated, eventually $W_c = 1$. Thus, the algorithm detects a termination in finite time.

7.5 A spanning-tree-based termination detection algorithm

The algorithm assumes there are N processes P_i, $0 \leq i \leq N$, which are modeled as the nodes i, $0 \leq i \leq N$, of a fixed connected undirected graph. The edges of the graph represent the communication channels, through which a process sends messages to neighboring processes in the graph. The algorithm uses a fixed spanning tree of the graph with process P_0 at its root which is responsible for termination detection. Process P_0 communicates with other processes to determine their states and the messages used for this purpose are called signals. All leaf nodes report to their parents, if they have terminated. A parent node will similarly report to its parent when it has completed processing and all of its immediate children have terminated, and so on. The root concludes that termination has occurred, if it has terminated and all of its immediate children have also terminated.