A Generalization of Termination Detection for Diffusing Computations[†]

K. M. Chandy
J. Misra[††]

TR-106 July 1979

[†]Work partially supported by NSF Grant MCS77-09812.

[††]This work was completed while the author was visiting Hahn Meitner Institute in Berlin.

Reproduction of this paper was supported by Hitachi Corporation.

1. Introduction

In [1] Dijkstra and Scholten introduce the notion of diffusing computation in a distributed system of processes and suggest an elegant algorithm for detecting the termination of an arbitrary diffusing computation in an arbitrary network. The generality of the solution makes it suitable for application in a number of problems arising in distributed processing; subsequently Dijkstra [2] has applied these ideas in formulating a distributed algorithm to answer the following question - for an arbitrary node D in a network, does D belong to a knot?*

We consider the following generalization of the termination detection problem. Suppose that every node of the network computes some data value v during a diffusing computation starting with some initial value of v. It is desired on termination to compute the value of some function of the v's of the various nodes of the network. We solve this problem when the value of every v is from a finite bounded set at all points in the computation.

Our solution works in conjunction with the algorithm for termination detection. The typical application of this algorithm is when v denotes the "state" of a node; then our algorithm allows us to compute any function on the final states of the nodes. We use this algorithm to formulate a solution in Section 4, for the problem of knot (described above) in a conceptually simple manner.

2. The Algorithm of Dijkstra and Scholten for Termination Detection in Diffusing Computations.

We provide only a sketch of the problem and its solution here. The reader is referred to [1] for a precise description of the problem and a

* See Section 4 for a precise definition of this problem.

systematic development of the solution.

Given,

(i) a finite directed graph each node of which denotes a process and each directed edge denotes the direction of message flow;

(ii) a special node, called gate, which has an input edge from some external process called the environment;

(iii) that every node in the graph is reachable* from the gate

Initially, every node is in a neutral state. The diffusing computation starts when the gate receives a message from the environment. A node is free to send messages along its outgoing arcs (to its successors) only after receiving the first message. A diffusing computation terminates when no node recieves or sends any more messages. It is required to devise a signalling scheme, to be superimposed on the diffusing computation, such that the gate will send a signal to the environment when the diffusing computation terminates. We assume that a node can send a signal to any of its predecessors.

In a terminating diffusing computation, every node will send a signal for every message it receives from every predecessor. The order in which the signals are sent is crucial for the correct operation of the algorithm. During diffusing computation for any node let,

c = number of messages received (from predecessors) - number of signals sent (to predecessors)

Clearly, $c \geq 0$. Define the state of a node to be neutral, if $c = 0$ and engaged if $c > 0$.

* A node y is reachable from a node x if and only if there exists a directed path from x to y; if x = y, such a path of length 0 always exists.

Initially, after receipt of the message from the environment by the gate node, c = 1 for the gate node and c = 0 for every other node. Diffusing computation terminates when c = 0 for every node. Typically a node will alternate between neutral and engaged state. A node goes from neutral to engaged state when it recieves a message from a predecessor (and just prior to the message receipt, c = 0 for the node). Define father of an engaged node to be that predecessor from which the message received transformed this node to its current engaged state from neutral state. Note that a node will typically have many different fathers during a diffusing computation as it alternates between neutral and engaged state; however it has one unique father at any point in the computation if it is engaged. Initially, the father of the gate node is the environment.

The solution to the termination detection problem is based upon the following invariant.

p1:: Father of every engaged node (except the gate node) is engaged.

It follows from the invariant that every node must remain engaged as long as any son of it is engaged. It is however impossible for a node to decide if some one of its sons is engaged solely from the data about the number of messages sent and the signals received. We therefore add the following protocol constraint.

p2:: A node may go from engaged to neutral state only by sending a signal to its father.

For any node let, D = the total number of messages sent to its successors - the number of signals received from successors.

It follows from the constraint that for any node,

1. $D > 0$ means one or more of its sons is engaged. Hence $D > 0 \Rightarrow c > 0$.
2. $D = 0$ means all of its sons are in neutral state.

If this node has no more messages to send then within a finite time this node should go to neutral state (set its c to 0) by sending signals to all the predecessors from which it had received messages and has not yet sent back signals; the last signal must go the father.

It can be shown from the invariant (by induction on the number of state changes) that the set of engaged nodes form a directed tree by the relationship defined by "father"; the root of the tree is the gate node. The gate node can become neutral (by sending a signal to the environment) only if all other nodes are in neutral state. Conversely, if all the nodes are in neutral state the gate must send a signal to the environment in a finite time.

There are several important aspects of this algorithm.

1. Termination is a global property: A process can not decide if it has terminated since it may receive some message in the future. However the algorithm detects termination without using a global (central) process.
2. The nature of the diffusing computation as well as the structure of the network are general.
3. Very little extra storage is required for the implementation of the algorithm.

3. A Generalization

The algorithm of the previous section can be extended to solve the following problem. Suppose that every node computes a data value v during a diffusing computation starting with some initial value. Assume that for some finite, bounded, positive integer m, $v \in \{1,2,\ldots,m\}$ at all points during the diffusing computation. For ease of exposition, we call v the state of the node.

We are interested in computing a certain network function from the various v_i's upon termination of the diffusing computation. Since we consider arbitrary networks, we restrict the function to be of the form $v_1 \oplus v_2 \oplus,\ldots,\oplus v_m$ where $\oplus$ is a binary, commutative, associative operator. It is therefore sufficient for us to count q_s, the number of nodes that are in state s at termination, for $1 \leq s \leq m$, in order to compute any such function. We show how this counting may be done in conjunction with the algorithm for termination detection.

We will assume that,

(i) an engaged node may send data values (instead of simply a signal) to its father and,

(ii) n, the number of nodes in the network, is known to the gate node. (This assumption is unnecessary in many instances; see Section 4 for example.)

We maintain with every engaged node an m-tuple of values $<z_1,z_2,\ldots,z_m>$ satisfying the following invariant.

p3:: $\displaystyle\sum_{\substack{\text{over all}\\ \text{engaged nodes}}} z_s$ = number of neutral nodes that are in state s, $1 \leq s \leq m$.

Assume that initially every node is in a particular state, state 1, for instance. Upon receiving the message from the environment, the gate node sets - in order to establish p3 -

z_1:=n-1; z_i=0, $2 \leq i \leq m$.

Observe that,

(i) A node can not change its state as long as it is neutral. Hence the right hand side of p3 does not change in value if the set of neutral nodes remain invariant.

(ii) Any change in state of an engaged node does not affect the invariant.

Thus the invariant can only be affected when some node goes from neutral to engaged or vice versa. We next consider these cases.

When a node goes from neutral to engaged and it is in state s, the right hand of the equation corresponding to z_s in p3 reduces by 1. This is compensaged by the node in setting,

z_s:=-1; z_i:=0, $1 \leq i \leq m$, $i \neq s$.

When a node goes from engaged to neutral in state s, it transmits the m-tuple $<z_1, z_2, \ldots, z_{s+1}, \ldots, z_m>$ to its father. The father must add this m-tuple component wise to its own m-tuple to satisfy p3.

When the gate node changes from engaged to neutral, it transmits an m-tuple $<q_1, q_2, \ldots, q_m>$ to the environment. It follows from p3 and the fact that every node is then neutral that,

q_s = number of nodes in the network which are in state s at termination, $1 \leq s \leq m$.

4. The Problem of Knots [2]

A knot in a directed graph is a strongly connected component of more than one node such that all succcessors of all nodes in the knot are in the knot. It is required to design a diffusing computation, to be fired by an arbitrary node D, that will enable node D to detect whether or not it belongs to a knot. Since the answer is "no" when D has no successor, we assume without loss in generality that D has at least one successor. It follows then that D is in a knot if and only if every node reachable from D can reach D.

We therefore decompose the problem into the following three parts.

(i) Mark every node reachable from D, i.e. set the variable reachable = true if and only if this node is reachable from D.

(ii) Mark every node that can reach D; i.e. set the variable canreach = true if and only if this node can reach D.

(iii) Find if there is any node for which reachable = true and canreach = false, at the termination of parts (i) and (ii).

We next discuss how each of these parts is implemented. Our description is informal; however it presents the major ideas from which the formal description of the algorithm may be readily constructed. Dijkstra [2] has given algorithms for parts (i) and (ii), which we sketch below. We use algorithm of section 3 to implement part (iii) which results in considerable simplification.

Part (i) is implemented by starting a diffusing computation from D. Initially reachable is false for every node except D. D sends a message of "type 1" to all its successors. Any node receiving a message of type 1 for the first time sends a type 1 message to all its successors and sets its own _reachable_ to _true_. It then ignores every other type 1 message received. This computation terminates since no more than one message is sent along any edge.

Part (ii) is implemented in a similar fashion. Substitute _predecessor_ for _successor_, _canreach_ for _reachable_ and _type 2_ for _type 1_ in the above description of the implementation of part (i).

Note that the two computations are distinguished by the types of the messages transmitted. Hence there is no interference in execution of Parts (i) and (ii); therefore the two parts may run simultaneously. We next superimpose the execution of Part (iii) on the execution of parts (i) and (ii). Part (iii) uses the algorithm of section 3. Note that both types of messages are treated alike as "messages" and hence there is a single tree of engaged nodes for this part.

We associate two states with each node.

state 1:: reachable = true and canreach = false

state2:: otherwise

Initially every node is in state 2. Using the algorithm of section 3, upon termination of computation D sends q_1, q_2 to the environment where,

q_i = number of nodes in state i, $1 \leq i \leq 2$.

D is in a knot if and only if $q_1 = 0$. The computation of D will need to be slightly modified so that D does not actually send q_1, q_2 to the "environment," but simply tests q_1.

Note::

1. Since q_1 is the only value needed, it is sufficient for every node to maintain a single value z_1.
2. Since initially every node is in state 2, z_1 for D may be set to 0.
3. The entire algorithm of three parts runs concurrently.
4. A node may go at most once from state 2 to state 1 to state 2 during the computation. The total number of state changes in the network is thus bounded by 2n, where n is the number of nodes in the network.

References

1. E. W. Dijkstra, C. S. Scholten, "Terminal Detection for Diffusing Computations, EWD 687a, Plataanstraat 5, 5671 Al Neunen, The Netherlands.

2. E. W. Dijkstra, "In Reaction to Earnest Chang's Deadlock Detection," Plataanstraat 5, 5671 Al Neunen, The Netherlands.