PROVING TERMINATION FOR DISTRIBUTED COMPUTATIONS ON DAGs

Mohamed G. Gouda

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

TR-197 March 1982

ABSTRACT

We consider a computation where each node of a DAG is assigned a state. If the states of two adjacent nodes (i.e., those placed at the ends of a directed edge) do not satisfy some predefined condition, then these two states can be changed in a predefined manner to satisfy the condition. The computation terminates if the states of any two adjacent nodes in the DAG satisfy the predefined condition. Many algorithms in computer networks and VLSI systems can be stated in this fashion. We discuss two sets of sufficient conditions which guarantee that such a computation terminates on any DAG. Some practical examples, e.g., sorting on a DAG and computing the longest path of a DAG are used to illustrate the results.

KEYWORDS: Computer network, DAG, distributed computation, termination, termination metric, VLSI systems.

I. INTRODUCTION

A computation in a computer network or a VLSI array is usually associated with a directed graph. The graph establishes an adjacency relationship between the different sites in a computer network or between the different cells in a VLSI array. The computation progresses in parallel along each edge of the graph until a global termination state is reached, if at all, where every two adjacent nodes satisfy some predefined condition. Examples of such computations are adaptive routing in a computer network [3], and sorting in a VLSI array [2].

In this paper, we present a simple model to specify a class of these computations associated with connected, directed, acyclic graphs (or DAGs for short). We also discuss two sets of sufficient conditions that these computations do terminate. The resulting specifications and the sufficient conditions are all independent of the underlying DAGs; thus the considered computations can be correctly implemented on any DAG regardless of its topology or size.

II. THE MODEL

A distributed computation C is a tuple of four components:

$$C = (Q, p, r, s)$$

where Q is a set of states,

P is a function, called the predicate of C, whose domain and range are as follows

$$p : Q \times Q \longrightarrow \{\text{true}, \text{false}\},$$

r is a function, called the backward next state of C, whose domain and range are as follows

r : Q x Q --> Q, and

s is a function, called the forward next state of C, whose domain and range are as follows

s : Q x Q --> Q.

The functions p, r, and s must satisfy the following one step condition for any x and y in Q.

$$\neg\, p(x,y) \supset p(r(x,y),\ s(x,y))$$

where $\neg$ is the logical "not" operator, and
$\supset$ is the logical implication operator.

Let C = (Q,p,r,s) be a distributed computation; and let G = (N,E) be a DAG where N is its set of nodes and E is its set of directed edges. The state q of C on G is a function whose domain and range are as follows

$$q : N \text{ --> } Q$$

An edge (i,j) in G is said to be satisfied at state q if p(q(i),q(j)) = true; otherwise it is unsatisfied at q.

Let q and q´ be two states of C on G; and let (i,j) be a directed edge in G. q follows q´ satisfying (i,j), denoted q---(i,j)-->q´, if the following four conditions hold.

1. Edge (i,j) is unsatisfied at q.
2. q´(i) = r(q(i), q(j)).
3. q´(j) = s(q(i), q(j)).
4. For any node k other than i or j in G, q´(k) = q(k).

Let q and q´ be two states of C on G. q´ <u>follows</u> q, denoted q-->q´, if there is a directed edge (i,j) in G such that q---(i,j)-->q´.

Let q and q´ be two states of C on G. q´ is <u>reachable from</u> q if either q´=q or there exist states $q_1,\ldots,q_r$ of C on G such that $q=q_1$, $q'=q_r$, and for i=1,...,r-1, $q_i \text{-->} q_{i+1}$.

Example 1. Sorting: Consider the distributed computation C whose four components are as follows.

Q = the set of real numbers, and
for any x and y in Q, p(x,y) = <u>true iff</u> $x \geq y$,
r(x,y) = y, and
s(x,y) = x.

Note that since
$$\neg p(x,y) = x < y,$$
$$p(r(x,y), s(x,y)) = y \geq x, \text{ and}$$
$$x < y \supset y \geq x,$$

then the one step condition is satisfied for any x and y in Q.

Starting from any state q_o of C on a DAG G, a state q_f must be reached such that for any directed edge (i,j) in G, $q_f(i) \geq q_f(j)$. Notice that for some G, there are two or more states which satisfy this condition; and any one of them can be reached nondeterministically. []

Example 2. Longest Path Computation: Consider the distributed computation C whose four components are as follows.

Q = the set of natural numbers, and
for any x and y in Q, p(x,y) = true iff $y = o \vee x \geq y + 1$,
r(x,y) = $y + 1$, and
s(x,y) = y.

It is straightforward to show that the one step condition is satisfied.

Define a state q_o of C on a DAG G such that $q_o(i)=1$ if node i has no output edges in G, and $q_o(i)=0$ otherwise. From q_o, a state q_f must be reached such that for any node i in G, $q_f(i)$ = the length of the longest path which starts from node i. []

Example 3. Multiplication: Consider the distributed computation C whose four components are as follows.

Q = the set of integers, and
for any x and y in Q, p(x,y) = true iff $y = 1$,
r(x,y) = $x * y$, and
s(x,y) = 1.

It is straightforward to show that the one step condition is satisfied.

A DAG G is called rooted if it has exactly one node i_o without input edges and any node in G can be reached by a directed path from i_o; node i_o is called the root of G. From any state q_o of C on a rooted DAG G, a state q_f must be reached such that

$q_f(i) = \prod_{j \text{ in } G} q_o(j)$ ... if i is the root of G, and
$= 1$ ... otherwise. []

Let C be a distributed computation; and G be a DAG. Let $\bar{Q}$ be the set of all states of C on G. A termination metric m for C on G is a function which satisfies the following two conditions:

1. $m : \bar{Q} \rightarrow N$, where N is the set of nonnegative integers
2. For any two states q_1 and q_2 of C on G, if $q_1 \rightarrow q_2$ then $m(q_1) > m(q_2)$.

A distributed computation C terminates on a DAG G if there is a termination metric for C on G. C terminates if for any DAG G, C terminates on G. Notice that this definition of termination for a distributed computation is similar to that of sequential program termination [1]. In the next two sections, we discuss two sets of sufficient conditions which guarantee that a distributed computation terminates on any DAG.

III. WAVE TERMINATION

Let C = (Q,p,r,s) be a distributed computation; and assume that p,r, and s satisfy the following three conditions for any x, y, and z in Q.

$$\neg p(x,y) \wedge p(x,z) \supset p(r(x,y),z) \quad (1)$$

$$\neg p(x,y) \wedge p(y,z) \supset p(s(x,y),z) \quad (2)$$

$$\neg p(x,y) \wedge p(z,y) \supset p(z, s(x,y)) \quad (3)$$

In this section, we prove that conditions (1), (2), and (3) are sufficient to establish that C terminates on any DAG G. First, we prove the following three lemmas.

Lemma 1: If $q_1 \text{---}(i,j)\rightarrow q_2$ then for any edge (i,k) in G, if (i,k) is satisfied at q_1 then (i,k) is satisfied at q_2.

Proof: Assume that (i,k) is satisfied at q_1 but not at q_2. Thus node k is different from node j.

since (i,j) is unsatisfied at q_1, $p(q_1(i), q_1(j))$ = false.
since (i,k) is satisfied at q_1, $p(q_1(i), q_1(k))$ = true.
since (i,k) is unsatisfied at q_2, $p(q_2(i), q_2(k))$ = false.
since $q_2(i) = r(q_1(i), q_1(j))$ and $q_2(k) = q_1(k)$, we have
$\neg p(q_1(i), q_1(j)) \wedge p(q_1(i), q_1(k)) \wedge \neg p(r(q_1(i), q_1(j)), q_1(k))$ = true
which contradicts (1). []

The following two lemmas have similar proofs to that of Lemma 1.

Lemma 2: If q_1---(i,j)-->q_2, then for any edge (j,k) in G, if (j,k) is satisfied at q_1 then (j,k) is satisfied at q_2. []

Lemma 3: If q_1---(i,j)-->q_2, then for any edge (k,j) in G, if (k,j) is satisfied at q_1 then (k,j) is satisfied at q_2. []

Let q be a state of C on G; and let M(q) denote the set of all node pairs <i,j>, in G, which satisfy the following condition. There is a directed path d, with at least one edge, from node i to node j such that the last edge in d is unsatisfied at state q. Let |M(q)| be the number of node pairs in set M(q). For any state q, $0 \leq |M(q)| \leq$ the total number of node pairs in G. Therefore, to show that |M(q)| is a termination metric for C on G, it is sufficient to prove the following theorem.

Theorem 1: If q_1-->q_2 then $|M(q_1)| > |M(q_2)|$

Proof: Since q_1-->q_2, then there is a directed edge (i,j) in G such that q_1---(i,j)-->q_2. The set $M(q_1)$ can be partitioned into two disjoint subsets:

$$M(q_1) = M_0 \bigcup M_1$$

where $M_1 = \{\langle k,j\rangle |$ there is a directed path from node k to node i in G$\}$.

From Lemmas 1, 2, and 3, any directed edge which is satisfied at q_1 but not at q_2 must be of the form (k,i). Therefore, the set $M(q_2)$ can be partitioned into two not necessarily disjoint subsets.

$$M(q_2) = M_0 \bigcup M_2$$

where $M_2 = \{\langle k,i\rangle |$ there is a directed path d, with at least one edge, from node k to node i in G such that the last edge in d is unsatisfied at $q_2\}$.

For any node pair $\langle k,i\rangle$ in M_2, there is a corresponding node pair $\langle k,j\rangle$ in M_1. On the other hand, the node pair $\langle i,j\rangle$ in M_1 has no corresponding pair in M_2. Then, $|M_1| > |M_2|$ and $|M(q_1)| > |M(q_2)|$. []

It is straightforward to verify that the distributed computations of examples 2 and 3 satisfy (1), (2), and (3); then by Theorem 1, each of them terminates on any DAG starting from any initial state.

From Lemmas 1 to 3, the three conditions (1), (2), and (3) guarantee that when an edge becomes satisfied, only edges preceding it can become unsatisfied. In other words, a "satisfaction wave" can proceed backwards along the DAG edges until all edges become satisfied, i.e., a termination state. A satisfaction wave which proceeds forward along the DAG edges will also lead the computation to a termination state at which all edges are satisfied. To guarantee this forward wave, condition (2) should be replaced by the following condition for any x, y, and z in Q.

$$\neg p(x,y) \wedge p(z,x) \supset p(z,\ r(x,y)) \qquad (4)$$

Using arguments similar to those discussed earlier, it can be shown that conditions (1), (3), and (4) are sufficient to establish that a distributed computation terminates on any DAG.

The distributed sorting of example 1 violates both conditions (2) and (4). Therefore to establish its termination, another set of sufficient conditions for termination is needed. This is discussed next.

IV. SORTING TERMINATION

Let C = (Q,p,r,s) be a distributed computation, and assume that p, r, and s satisfy the following four conditions for any x, y, and z in Q:

$$\neg p(x,y) \wedge p(z,x) \supset p(z,r(x,y)) \vee [\neg p(z,y) \wedge p(z,s(x,y))] \qquad (5)$$

$$\neg p(x,y) \wedge p(x,z) \supset p(r(x,y),z) \qquad (6)$$

$$\neg p(x,y) \wedge p(z,y) \supset p(z,\ s(x,y)) \qquad (7)$$

$$\neg p(x,y) \wedge p(y,z) \supset p(s(x,y),z) \vee [\neg p(x,z) \wedge p(r(x,y),z)] \qquad (8)$$

In this section, we prove that these four conditions are sufficient to establish that C terminates on any DAG G.

Let q be a state of C on G. A node pair $\langle i,j \rangle$ in G is said to be potentially satisfied at q if there is a directed path from node i to node j and p(q(i), q(j)) = true. In the following lemmas, let q_1---(i,j)-->q_2; and let k be any node other than i and j in G.

Lemma 4: If $\langle k,i \rangle$ is potentially satisfied at q_1, then either $\langle k,i \rangle$ is potentially satisfied at q_2 or $\langle k,j \rangle$ is potentially satisfied at q_2 but not at q_1.

Proof: (by contradiction) Assume that $\langle k,i \rangle$ is potentially satisfied at q_1 but not at q_2 and that $\langle k,j \rangle$ is potentially satisfied at q_1.

since (i,j) is unsatisfied at q_1, $p(q_1(i), q_1(j)) = \underline{\text{false}}$.

since $\langle k,i \rangle$ is potentially satisfied at q_1, $p(q_1(k), q_1(i)) = \underline{\text{true}}$.

since $\langle k,i \rangle$ is potentially unsatisfied at q_2, $p(q_2(k), q_2(i)) = \underline{\text{false}}$.

since $\langle k,j \rangle$ is potentially satisfied at q_1, $p(q_1(k), q_1(j)) = \underline{\text{true}}$.

since $q_2(k) = q_1(k)$, and $q_2(i) = r(q_1(i), q_1(j))$, then

$$\neg p(q_1(i), q_1(j)) \wedge p(q_1,(k), q_1(i)) \wedge \neg p(q_1(k), r(q_1(i), q_1(j))) \wedge p(q_1(k), q_1(j)) = \underline{\text{true}}.$$

contradicting (5). Similarly, if $\langle k,j \rangle$ is potentially unsatisfied at q_2, then (5) is contradicted. []

The following three lemmas can be proved using proofs similar to that of Lemma 4.

Lemma 5: If $\langle i,k \rangle$ is potentially satisfied at q_1, then it is potentially satisfied at q_2. []

Lemma 6: If $\langle k,j \rangle$ is potentially satisfied at q_1, then it is potentially satisfied at q_2. []

Lemma 7: If $\langle j,k \rangle$ is potentially satisfied at q_1, then either $\langle j,k \rangle$ is potentially satisfied at q_2 or $\langle i,k \rangle$ is potentially satisfied at q_2 and not at q_1. []

Let N(q) be the set of all potentially satisfied node pairs at state q of C over G; and let P be the set of all node pairs in G. Since for any q, N(q) is a subset of P; then $0 \leq |P|-|N(q)| \leq |P|$. To show that $|P|-|N(q)|$ is a termination metric for C on G, it is sufficient to prove the following theorem.

Theorem 2: If $q_1 \text{-->} q_2$ then $|P|-|N(q_1)| > |P|-|N(q_2)|$.

Proof: Since $N(q_1)$ and $N(q_2)$ are subsets of P, it is sufficient to prove that $|N(q_1)|<|N(q_2)|$. This can be done by showing that every element in $N(q_1)$ corresponds to a distinct element in $N(q_2)$, and that there is an element in $N(q_2)$ which has no corresponding distinct element in $N(q_1)$. Since $q_1 \text{-->} q_2$, then let (i,j) be the directed edge such that $q_1 \text{---}(i,j)\text{-->} q_2$.

Part 1. Every element in $N(q_1)$ corresponds to a distinct element in $N(q_2)$: Let $\langle k,l \rangle$ be an element in $N(q_1)$. There are five cases to consider.

a. Neither k nor l is i or j:
Since $\langle k,l \rangle$ is in $N(q_1)$, then $p(q_1(k), q_1(l)) = \underline{\text{true}}$. Then $p(q_2(k), q_2(l)) = \underline{\text{true}}$ since $q_2(k) = q_1(k)$ and $q_2(l) = q_2(l)$. Then $\langle k,l \rangle$ is in $N(q_2)$.

b. l is i:

$\langle k,i\rangle$ is in $N(q_1)$. If $\langle k,j\rangle$ is also in $N(q_1)$, then $\langle k,i\rangle$ is in $N(q_2)$, else $\langle k,j\rangle$ is in $N(q_2)$ by lemma 4.

c. k is i:

$\langle i,l\rangle$ is in $N(q_1)$. Then $\langle i,l\rangle$ is in $N(q_2)$ by lemma 5.

d. l is j:

$\langle k,j\rangle$ is in $N(q_1)$. Then $\langle k,j\rangle$ is in $N(q_2)$ by lemma 6.

e. k is j:

$\langle j,l\rangle$ is in $N(q_1)$. If $\langle i,l\rangle$ is also in $N(q_1)$, then $\langle j,l\rangle$ is in $N(q_2)$, else $\langle i,l\rangle$ is in $N(q_2)$ by lemma 7.

Part 2. Element (i,j) in $N(q_2)$ has no corresponding element in $N(q_1)$. Therefore $|N(q_1)|<|N(q_2)|$ and $|P|-|N(q_1)|>|P|-N(q_2)|$. []

It is straightforward to verify that the distributed sorting of example 1 satisfies conditions (5) to (8). Then by Theorem 2 it is guaranteed to terminate on any DAG. Also, the distributed multiplication of example 3 satisfies these conditions and is guaranteed to terminate on any DAG; this is our second proof that this computation terminates.

V. CONCLUDING REMARKS

This paper demonstrates that ideas (such as termination metrics) which were originally developed for sequential computations can be applied to distributed computations given a "convenient" model for the latter.

The proposed model in this paper separates between a distributed computation and its underlying DAG encouraging one to reason about the computation independently of its DAG. This has led to sufficient conditions for termination which are independent of the underlying DAG and the initial state.

The model can be extended to represent VLSI arrays with different types of interconnections between cells. In this extension, the underlying DAG has two (or more) types of edges. Each edge type has a different predicate and different backward and forward next state functions. With this extension, a distributed computation becomes dependent on the topology of its DAG but not necessarily on its size.

ACKNOWLEDGEMENT: The author is thankful to K. F. Carbone for her careful typing.

REFERENCES

[1] S. Alagic and M. Arbib, The design of well-structured and correct programs, Springer Verlag, 1978.

[2] T. Chen, V. Lum, and C. Tung, "The rebound sorter: an efficient sort engine for large files," 4th VLDB Proceedings 1978.

[3] D. Davies, et. al., Computer networks and their protocols, John Wiley & Sons Ltd., 1979.