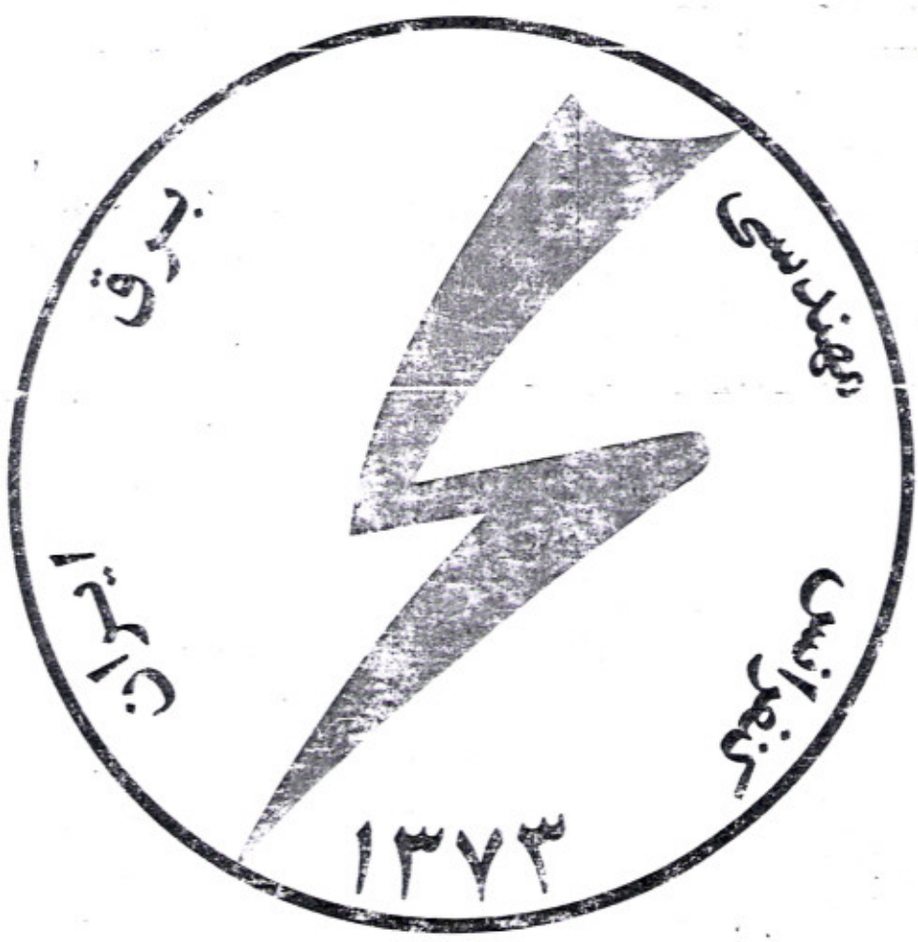
۵/۶

مجموعه مقالات

کنفرانس مهندسی برق ایران

مخابرات



۲۷ الی ۳۰ اردیبهشت ۱۳۷۳

دانشگاه تربیت مدرس؛ تهران - ایران

A Method For Verifying Absence of Deadlock in Networks of Communicating Processes

Mohammad. R. Meybodi
Computer Engineering Department
Amirkabir University
Tehran, Iran

Abstract

This paper presents a method for verifying absence of deadlock in networks of communicating processes in which the communication pattern is independent of the value of the message being communicated.

Index Terms: Deadlock, Networks of Communicating Processes, Concurrency

1 Introduction

A network of communicating processes is a collection of processes which has been programmed to perform an algorithm[1]. In This set up the implementation of the algorithm can be regarded as parallel composition of task performed by each process in the network. These processes are disjoint in their state space (i.e. no shared variable are considered) and communicate with each other along uni-directional channels via synchronized message passing. The algorithm performed by the network can be specified by describing the task performed by each process in the network. This can be modeled using operators on data sequence passed along individual channel in the network. More specifically, the data sequence of the output channels of a particular process are defined as function of the sequences of data items on the input channels of that process.

Unlike the sequential implementation of an algorithm in which there is a predetermined sequential order for various actions performed by the algorithm, in concurrent network, since the processes may act independently, such a predetermined sequential order does not exist. This makes the program harder to understand and analyze, and also can lead to real nondeterminism, or unpredictability. Also, the introduction of concurrency can introduce the possibility of pathological behavior such as deadlock.

A concurrent network is deadlocked if it can not proceed because it is waiting for an event which will never happen. This is usually because every process in the network is ready to communicate with some process or processes in the network but its communications are blocked by these processes, that is, none of these processes is willing to respond. We say the request made by P_i to communicate with P_j is ungranted if process P_i istrying to communicate with process P_j but process P_j is unwilling to respond to P_i s request for communication. We also say a process is blocked if a request made by that process to communicate is left ungranted. A network is deadlocked if and only if every process in the network is blocked. A cycle of ungranted requests in a network of communicating processes is the building block of deadlock and is generated when there is a subset of processes P_1, P_2, P_3, ..., P_m such that at some point during the operation of the network, for each J, process P_j wants to talk

to process P_{j+1} before it is prepared to talk to P_{j-1}. In this report, we will describe a simple method of deadlock analysis. This method basically checks if there is any cycle of ungranting requests in the network.

2 A Method for Verifying Absence of deadlock

For deadlock analysis it is sufficient to examine the patterns of communications that may arise during the operation of the network. We assume that the pattern of communication is independent of the value of messages being communicated along different channels in the network. We specify the pattern of communication in the network by specifying the input/output behavior of each processing element using an expression called *I/O* expression. An *I/O* expression for a processing element specifies the order in which that processing element communicates with its neighboring processes. In these expressions we use the channel name to denote either an input or an output along that channel. The method proposed here to verify deadlock freedom is based on a graphical representation of communication pattern arises during the operation of the network. This graph is derived from input /output behavior of the processing elements in the network. We call this graph an *I/O* graph.

Before we discuss the method we show how to obtain the *I/O* graph of a network. This is illustrated using the network given in figure 1 .

This is a network of processes which multiplies a 3xn matrix by a fixed vector $V=(v_1, v_2, v_3)$ [2]. These processes are labeled X_1, X_2, X_3, X_4, and X_5. Each process except X_1 and X_5 which are input and output processes reads a value from the left and above. It then multiplies the value from left by the vector element stored in the process, and add this product to partial sum received from above. This partial sum is then sent down to the next process. In the diagram of the network the presence of the name col[j], row[i] or output besides a line indicates the sequence of data items transmitted along the channel corresponding to that line. If we name each channel after the sequence that passes along it then the input/output behavior of each processing element can be stated as follows:

X_1:(col[0])^n

X_2:(col[0];row[1];col[1])^n

X_3:(col[1];row[2];col[2])^n

X_4:(col[2];row[3];col[3])^n

X_5:(col[3];output)^n

The index n is used to denote n-fold repetition, i.e, P^3=P;P;P. A semicolon in *I/O* expression indicates that the communication to the left of it must be completed before the communication to the right can be initiated. For example *I/O* expression for process X_2 indicates that this process will first communicate with process X_1 and then with process X_3. The *I/O* graph for the pattern of communications for this network is given in figure 2. In the *I/O* graph all external communications, i.e. communications along channels row[1], row[2], row[3], and output are omitted.

In this graph the vertical lines are labeled (from left to right) X_1, X_2, X_3, X_4, and X_5. As we move

up the vertical line labeled X_1 we encounter an infinite sequence of edges all labeled col[0]. This represents the fact that

X_1:(col[0])^n

Similarly, as we move up the vertical line labeled X_2 we encounter the following infinite sequence of edges : col[0], row[1], col[1], col[0], row[1], col[1],....This represent the fact that

X_2:(col[0]; row[0]; col[1])^n.

The same is true of the edge sequences encountered as we move up the other vertical lines. In this graph we use pair (X_k,i) to denote the i^{th} vertex on the vertical line labeled X_k . For example the first vertex on the vertical line X_1 will be labeled $(X_1,1)$. In this graph the n^{th} edge labeled col[1] we encounter on moving up vertical line is also the n^{th} edge labeled col[1] that we encounter on moving up vertical line X_3. Thus we can associate this n^{th} edge with the n^{th} communication to occur along the channel col[1]. Of course the same can be said of the other edges in the graph. The position of vertex v along the vertical line X can be interpreted as the earliest possible time that process X is ready to engage in a communication along the channel which is the lable of an edge connecting v to vertex along another vertical line.

In some concurrent networks a process may communicate with a subset of its neighbors in any order. We use symbol || to refer to this situation when specifying the input/output behavior of a network. Therefore if process P can communicate along channels $c_1, c_2, c_3,\ldots,c_k$ with k of its neighbors in any order, then when we specifying the input/output pattern for this process we use $c_1 \| c_2 \| c_3 \| \ldots \| c_k$ to indicate this fact. Process P can not engage in any other communication unless all these k communications have been completed. A network with such a communication pattern together with its *I/O* expression and the corresponding *I/O* graph is given in figure 3 .

In an *I/O* expression we use symbol @ to refer to the situation when a process can choose between several communications. For example if process X can communicate with either process Y along channel c_1 or with process Z along channel c_2 then the *I/O* expression for this process is written c_1 @ c_2 . The *I/O* graph for this *I/O* expression is given in figure 4.

Now we are in a position to present rules for verifying the freedom of deadlock in a concurrent network. The first rule can be applied to networks of communicating processes in which all processes are deterministic and also interleaving of communications is not allowed. The second rule is applicable to networks in which every process is a deterministic process and in which interleaving of communication as described above is allowed. The last rule which is the generalization of the first two rules can be applied to networks in which the processes are allowed to choose between communications.

Theorem 1 *A deterministic network with no interleaving of communications between the processes is deadlock free if there exist a function f such that f(a, i) > max{f((a, j)), f((b,k))}, where (a,i), (a,j), and (b,k) are any three vertices of the network I/O graph such that i > j and (a,i) and (a, j) are two adjacent vertices along vertical line a, and (a,i) and (b,k) are two adjacent vertices along*

vertical line a and vertical line b, respectively.

Proof: A network of communicating processes deadlocks if and only if there is a cycle of ungranted requests. That is, there is a subset of processes P_1, P_2, ...,P_n such that at some point during the operation of network, for each k, P_k wants to talk to process P_{k+1} before it is prepared to talk to P_{k-1}. The graph in figure 5 represents part of the *I/O* graph which corresponds to this cycle of ungranted requests. Let f((p,i)) be a function that returns the earliest possible time that process P is ready to engage in another communication which corresponds to vertex (p, i) in its *I/O* graph. Then for the figure 5 we have

1 . $f((P_l,i_l)) < f((P_l,j_l))$ for $1<= l <= n-2$
2 . $f((P_l,i_l)) < f((P_{l+1},i_{l+1}))$ for $1<= l <= n-2$
3 . $f((P_n,i_n)) < f((P_n,j_n))$
4 . $f((P_n,i_n)) < f((P_1,i_1))$
5 . $f((P_{n-1},j_{n-1})) < f((P_{n-1},i_{n-1}))$
6 . $f((P_n,i_n)) < f((P_{n-1},i_{n-1}))$

Clearly, for the network to be free of dealock process p_n can not engage in a communication with process p_1 unless its communication with process p_{n-1} has been completed. That is $f((p_n , j_n)) > f((p_1, i_1))$. And hence the theorem.

To show that a given network is free of deadlock we must find a function that satisfy the conditions of theorem 1. Such a function for vector- matrix multiplication network described earlier is given below.

$f(x_i,1)=0$ for all $1 <= i <= n$
$f(x_1, i) = 0$ for $i > 1$, $f(x_5, i)=0$ for $i >1$
$f(x_i,j)= \max(f(x_i,j-1), f(x_{i+1}, j-1))$
if process x_i communicates with process x_{i+1}
$f(x_i, j)= \max(f((x_i,j-1),f(x_{i-1},j-1))$
if process x_i communicates with x_{i-1}

where (x_i, j) is the j^{th} vertex on the vertical line x_i and n is the number of processes in the network.

The *I/O* graph of figure 6 shows the value of this function for different vertices in the network. In the *I/O* graph all external communication, i.e. communications along channels row[1], row[2], row[3], and output are omitted.

Theorem 2 *A deterministic network in w in which interleaving of communication is allowed is deadlock free if and only if there exist function f such that for every sequence of vertices (a,i), ((a,j), (b_1,j_1), (b_2,j_2),...., and (b_n,j_n) for some n and i>j we have* $f((a,i)) > \max\{f((a,j)), f((b_1, j_1)), f((b_2,j_2)),..., f((b_n,j_n))\}$, *where (a,i) and (a,j) are two adjacent vertices along vertical line a, and vertices (a,j), (b_1,j_1), $((b_2,j_2),...,$ (b_n,j_n) form a tree rooted at (a,j).*

Proof: Let f((p,i)) returns the time that process p is ready to engage in a communication which correspond to vertex (p,i). A process p is ready to engage in a comunication along channel ((p, i), (q,k)) for some q and some k when all the communications with its children have been completed. And therefor we must have f((a,i))> max {f((b,j)), f((b_1,j_1)), f((b_2,j_2)),....., f((b_n,j_n))}.

Corollary 1 *A non- deterministic network of communicating processes is deadlock free if there exists function f such that for every vertex (a,i) and (a,j), where i > j, and (a,i) and (a,j) are two adjacent vertices along vertical line a, in its I/O graph we have f((a,i)) >max(lk), where lk is the maximum of the values of f computed at both ends of k^{th} edgeemanating from vertex (a,j).*

3 Conclusion

A pictorial method for verifying absence of deadlock in networks of communicating processes is presented. The methods is applicable to the situation where the communication pattern is independent of the value of the message being communicated.

Acknowledgement: The author would like to thank prof. T. Y. Kong for his constructive inputs on the early version of this report.

4 References

1. C. A. R. Hoare, Communication Sequential Processes, Prentice-Hall International, 1985.
2. J. D. Ullman, Computational Aspect of VLSI, Computer Science Press, 1984.

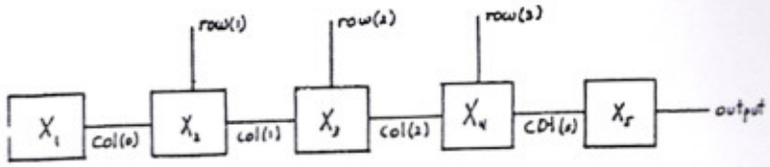


figure 1

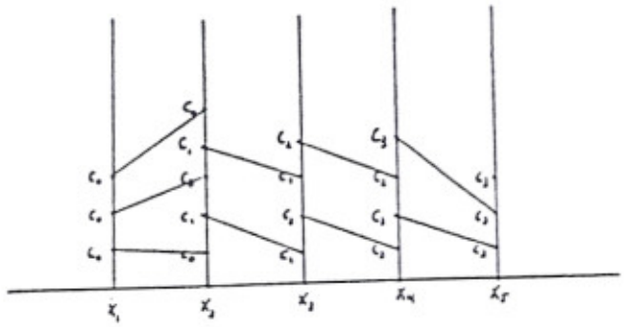


figure 2

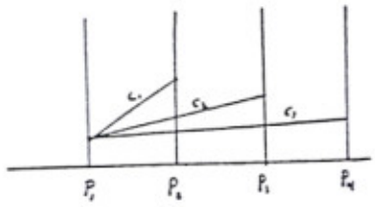


figure 3

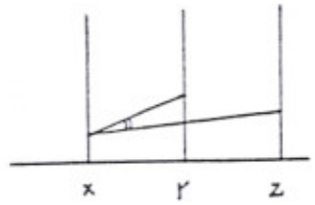


figure 4

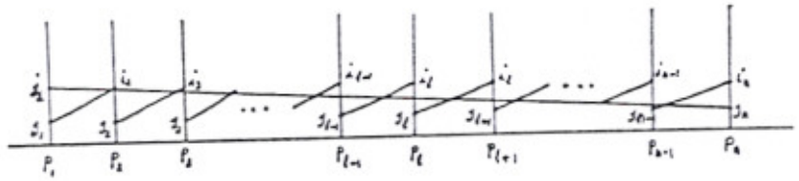


figure 5

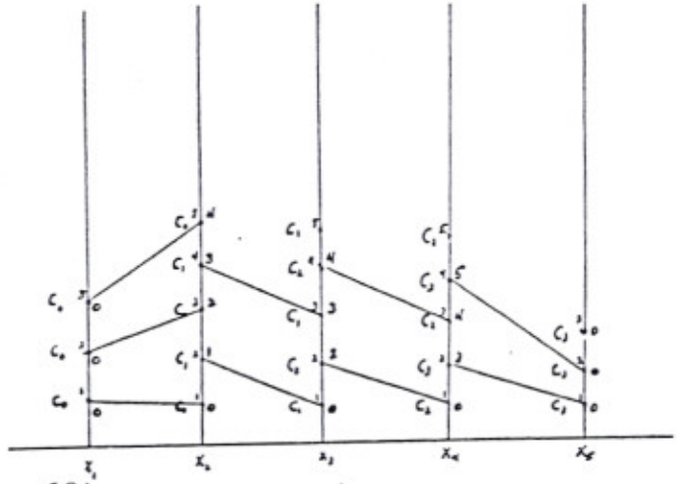


figure 6