A Valid Algorithm for Judging Liveness of Marked Graph[1]

HanYaojun, Jiang Changjun

Dept. of Computer Sci. and Eng.,Tongji University, Shanghai, 200092

College of Inf. & Electrical Eng.,Shandong University of Sci. & Tech., Jinan,250031

(Email□cjjiang@online.sh.cn)

Abstracts: An algorithm for judging liveness of marked graph is presented in this paper. A proof of correctness and a valuation of complexity for the algorithm are also given. The result shows that the algorithm is of polynomial complexity, hence it is a valid algorithm.

Keywords: Marked Graph, Liveness, circuit, Algorithm

1. Introduction

Liveness of Petri net is one of the earliest properties in net theory. The judgment of liveness for general Petri net has not been resolved so far. But many researchers gave the sufficient and necessary conditions of judgment of liveness for some subclasses of Petri net such as marked graph (also called T-graph), S-graph, free choice net and weighted T-graph[1][2]. Tadao Murata[1] gave the sufficient and necessary conditions : a marked graph is live iff M_0 places at least one token on each directed circuit in G. However, it takes much more time to find all circuits according to [1]. Can we only find a few circuits in G to judge the liveness of marked graph? Tadao Murata[1] didn't give the algorithm to do it. This paper presents a valid algorithm. We needn't find all circuits in G to judge the liveness of marked graph with the algorithm. First, the input arcs and output arcs of places that have at least one initial token are deleted from G. Then, we can determine that the marked graph is not live as long as there exits one circuit in deleted G according to [1]. Otherwise, the marked graph is live. The algorithm greatly reduces the time to find circuits in G.

2. Related Concepts and Conclusions

We assume the reader is familiar with basic concepts and properties of Petri net[1][3]. However, in this section we provide some basic concepts and conclusions to be used.

Definition 1. Let N=(P,T;F) is a net. For $x \in P \cup T$,

${}^{\bullet}x=\{y \in P \cup T|(y,x) \in F\}$

$x^{\bullet}=\{y \in P \cup T|(x,y) \in F\}$

are called the pre-set and post-set of x respectively.

Definition 2. N=(P,T;F) is called marked graph iff every place p has exactly one input transition and exactly one output transition. i.e.

$\forall\ p \in P, |{}^{\bullet}p|=|p^{\bullet}|=1$

---

1 This research is supported by the National Natural Science Foundation of China

A marked graph (N, M_0) can be drawn as a marked directed graph (G, M_0) for each place of marked graph has exactly one incoming arc and exactly one outgoing arc with unit weight, where arcs correspond to places , nodes to transitions, and tokens are placed on arcs.

Definition 3. Let N=(P,T;F) be a Petri net. The incidence matrix A of N is an m×n(m=|P|,n=|T|) matrix of integers and its entry is given by

$$a_{ij} = \begin{cases} 1 & if \quad p_i \in t_j^{\bullet} - {}^{\bullet}t_j \\ -1 & if \quad p_i \in {}^{\bullet}t_j - t_j^{\bullet} \\ 0 & otherwise \end{cases}$$

When a Petri net is pure, the Petri net corresponds to its incidence matrix. The net discussed in the paper is supposed to be pure net. Every row of the incidence matrix corresponding to marked graph has exactly two nonzero elements because every place of marked graph has exactly one input transition and exactly one output transition.

Conclusion 1. For a marked graph (G, M_0), the token count in a directed circuit is invariant under any firing, i.e., $M(d) = M_0(d)$ for each directed circuit d and for any M in $R(M_0)$, where $M(d)$ denotes the total number of tokens on d.

By conclusion 1, if there are no tokens on a directed circuit at the initial marking, then this directed circuit remains token-free. Thus the nodes on this directed circuit will never be enabled. On the other hand, if a node is never enabled by any firing sequence, then by back-tracking token-free arcs, one can find a token-free directed circuit. Therefore, we have:

Conclusion 2. A marked graph (G, M_0) is live iff M_0 places at least on token on each directed circuit in G.

3. Algorithm for Judging The Liveness of Marked Graph

3.1 Data Structures Used in the Algorithm

(1) m×2(m=|P|) matrix: *tran*. The first column of *tran* stores the serial numbers of successor transitions of all places. For example, if t_j is a successor transition of place p_i, then tran[i,1]=j. The second column of *tran* is used during finding circuits. If place p_i is visited, then tran[i,2]=1,otherwise tran[i,2]=0.

(2) n-linked list(shown in Fig. 1). *First* is an array of head of list. It is used to store the first node address of every linked list, where the field *place* stores the serial numbers of successor places of transition t_j (sorted in ascending order). For example, if p_i is a successor place of transition t_j, then there is a node that stores the serial number i of place p_i in the linked list t_j.

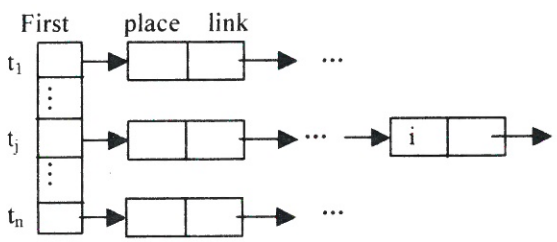


Figure 1

(3) m-vector: *mark*. It is used to store initial tokens.

(4) stack: *lt*. It is used to store the serial numbers of transitions visited during finding circuits.

3.2 Fundamental Idea of the Algorithm

First, the algorithm deletes the input arcs and output arcs of places with at least one initial token from marked graph. Namely, if $M_0(p_i)>0$ and t_j is the successor transition of place p_i,then let tran[i,1]=0 and delete node that stores the serial number i of place p_i from the linked list t_j.Then, for every place p_i that tran[i,1]$\neq$0, the algorithm finds directed circuit by the way of breadth-first search in ascending order of the serial numbers of places. The marked graph is not live as long as there exists one circuit in deleted G. Because the input arcs and output arcs of all places with at least one initial token are deleted from G, the marked graph is not live according to conclusion 2. The algorithm ends. If there is not any directed circuit in deleted G, the marked graph is live.

The process of finding directed circuit is given as follows.

For 1≤i≤n, if tran[i,1]$\neq$0 and tran[i,2]=0, then let tran[i,2]=1□According to the value of tran[i,1](i.e. j--the serial number of successor transition t_j of all place p_i), locates the linked list t_j. If the address of the first node of the linked list is not empty, then push j of the serial number of transition t_j into stack lt. Then locate the *k*th element of tran according to the value of field *place* of first node of linked list t_j (supposed k). Repeat the above process. At last, there exist two cases: the first case is tran[k,2]=1, which shows that a circuit is found, and the algorithm ends. The second case is that the address of first node of some linked list is empty, which needs to go back to previous transition. Pop the serial number of transition t_j out stack lt, then locate next node of the linked list t_j, and repeat above process till stack lt is empty. It is shown that place p_i is not on the circuit. Repeat the above process for the next element of tran till the algorithm ends either in the first case(the marked graph G is not live) or when i>m(the marked graph G is live).

3.3 Description of Algorithm

Step1. For 1≤i≤n, if mark[i]>0, then let j=tran[i,1], tran[i,1]=0, and the node storing the serial number of place p_i from linked list t_j.

Step 2.

(1) Let i=1;

(2) If i>m , then print "The marked graph is live" and the algorithm ends, else go to(3);

(3) If tran[i,1]=0 or tran[i,2]=1, then let i=i+1 and go to (2),else go to (4);

(4) Let k=i;

(5) If tran[k,2]=1, then print "The marked graph is not live" and the algorithm ends, else go to(6);

(6) Let tran[k,2]=1;

(7) Let j= tran[k,1];

(8) Let s=first[j];

(9) If s=NIL, then go to(10), else let k=s↑ .place , push j into stack lt and go to(5);

(10) If stack lt is empty , then let i=i+1 and go to(2), else go to (11);

(11) Pop the number out stack lt into j. Let first[j]= first[j]↑ .link;

(12) Let s=first[j]. Go to (9).

4. Proof of Correctness and Analysis of Complexity of Algorithm

4.1 Proof of Correctness of Algorithm

Theorem 3. Let N=(P,T;F, M_0) be marked graph. Then algorithm 3.3 for judging the liveness of marked graph is correct.

Proof. Step 2 of algorithm 3.3 is obviously correct according to conclusion 2. It is only necessary to prove the correctness of step 1 of algorithm 3.3, i.e., after the input arcs and output arcs of places with at least one initial token being deleted from G, conclusion 2 also holds.

The places with at least one initial token are in the two cases. First, they are on the directed circuits. Suppose these directed circuits are found. The purpose in deleting the input arcs and output arcs of places with at least one initial token is to avoid finding this directed circus again. So, to delete the input arcs and output arcs of places on the directed circuits with at least one initial token does not destroy the condition of conclusion 2. Second, they do not lie on any directed circuit. According to conclusion 1, if there are no tokens on a directed circuit at the initial marking, then this directed circuit remains token-free. Thus, initial tokens of places that do not lie on any directed circuit do not flow into the directed circuit with any initial tokens. Therefore, deleting the input arcs and output arcs of places with at least one initial token that do not lie on any directed circuit does not destroy the condition of conclusion 2.

4.2 Analysis of Complexity of Algorithm

Theorem 4. Let N=(P,T;F, M_0) be marked graph. Then time complexity of algorithm 3.3 is $O(m^2+n)$, where m=|P|,n=|T|.

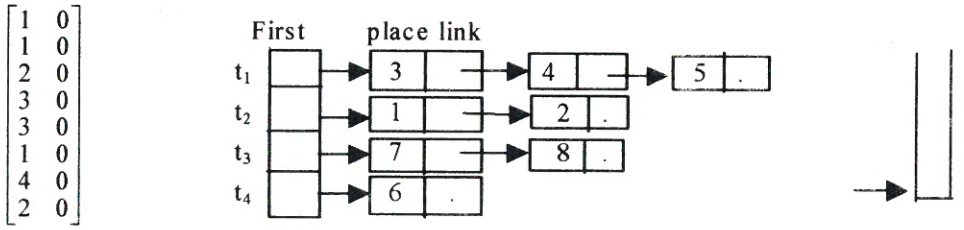
Proof. In step 1, the algorithm does m comparisons for the elements of array tran and at best m assignments. For every place p_i with at least one initial token, the algorithm does at best m-1 comparisons when deleting the serial number of place p_i from linked list t_j. So the time of step is $O(m^2)$.

In step 2, the algorithm does m +n comparisons when finding circuits, n-1 push operations and pop operations. So the time of step is O(m+n). Therefore, the time complexity of algorithm 3.3 is $O(m^2+n)$.

5. Example

Example 1. A marked graph is given in Fig. 2 below, where m=8,n=4 and M_0=(1,1,0,0,0,1).

The initial states of array *tran*, linked list and stack *lt* are given as follows.

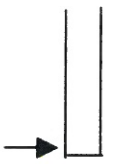


array *tran* linked list stack *lt*

The process of resolving the example is given as follows according to algorithm 3.3.

Step 1. Since $M_0(p_1)=M_0(p_2)=M_0(p_6)=1>0$, the input arcs and output arcs of places p_1, p_2 and p_6 are deleted from the Fig. 2, i.e. let tran[1,1]=0,tran[2,1]=0 and tran[6,1]=0, and delete two nodes from linked list t_2 and one node from linked list t_4. The states of array *tran*, linked list and stack *lt* are given as follows after step 1.

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 0 \\ 3 & 0 \\ 3 & 0 \\ 0 & 0 \\ 4 & 0 \\ 2 & 0 \end{bmatrix}$$



array tran linked list stack lt

Step 2. Finding a directed circuit in deleted G.

(1) Let i=1. Since tran[1,1]=0, let i=i+1 and go to (2);

(2) Since tran[2,1]=0, let i=i+1 and go to (3);

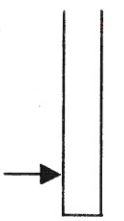
(3) Since tran[3,1]=2≠0 and tran[3,2]=0≠1, go to (4);

(4) Let tran[3,2]=1 . Locate linked list t_2. Go to (5);

(5) Since first[2]=NIL and stack lt is empty, let i=i+1 and go to (6);

(6) Since tran[4,1]=3≠0 and tran[4,2]=0≠1, go to (7);

(7) Let tran[4,2]=1> Locate linked list t_3. Take the value 7 from field *place* of first node of linked list t_3 and assign to k. Push the serial number 3 of transition t_3 into stack lt(the state of stack lt is shown in Fig on the right.). Go to (8);



The state of lt after (7)

(8) Since tran[7,1]=4≠0 and tran[7,2]=0≠1, go to (9);

(9) Let tran[7,2]=1. Locate linked list t_4. Go to (10);

(10) Since first[4]=NIL and stack lt is not empty , pop 3 out stack lt and go to (11);

(11) Move the pointer of linked list t_3 into next node. Take the value 8 from field *place* of second node of linked list t_3 and assign to k. Push the serial number 3 of transition t_3 into stack lt. Go to (12);

(12) Since tran[8,1]=2≠0 and tran[8,2]=0≠1, go to (13);

(13) Let tran[8,2]=1. Since first[2]=NIL and stack lt is not empty , pop 3 out stack lt (the states of tran, linked list and lt is shown in following Fig.) and go to (14);

(14) Move the pointer of linked list t_3 into next node. Since the node that pointer points is the last node of linked list t_3, the value of pointer is NIL. Since stack lt is empty , let i=i+1 and go to (15);

(15) Since tran[5,1]=3≠0 and tran[5,2]=0≠1, go to (16);

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 3 & 1 \\ 3 & 0 \\ 0 & 0 \\ 4 & 1 \\ 2 & 1 \end{bmatrix}$$

array tran linked list stack lt

(16) Since first[3]=NIL and stack lt is empty , let i=i+1 and go to (17);

(17) Since tran[6,1]=0, let i:=i+1 and go to (18);

(18) Since tran[7,2]=1, let i=i+1 and go to (19);

(19) Since tran[8,2]=0, let i=i+1 and go to (20);

(20) Since i>8, the marked graph given in Fig. 2 is live and the algorithm ends.

REFERENCES

(1) T. MURATA, **Petri Nets: Properties, Analysis And Application** , Proceeding of IEEE,1989.

(2) XU ANGUO & WU ZHEHUI, **Analysis of Liveness for Weighted T-graph**, J. Software,1993 (12).

(3) J. L. PETERSON, **Petri Net Theory and the Modeling of System**, Englewood Cliffs, New Jersey, Prentice Hall, Inc.,1981.

(4) WU ZHEHUI & JIANG CHANGJUN, **The Algorithm for converting Reachable Graph to Net Graph for Bounded Petri Net**, J. Software, 1992(1).

(5) C.J.JIANG, S.G.SHU & Y.P.ZHENG, **Logical Properties Analysis and Stochastic Performances Estimated of Petri Nets Under Restrictive Concurrent Machine**, Acta. Auto, 4 (1996).

(6) C.J.JIANG & Z.H.WU, **Two Fast Algorithms for Matrix Multiplication and Vector Convolution**, Int. J. of Comp. Math., 1(1997).

(7) C.J.JIANG, et. al., **Replacement Operation of Petri Net and Its Application in Systems Hierarchical Modeling**,[J. of Syst. Sci. & Syst. Eng.], 1(1998).

(8) C.J.JIANG et. al, **A Method to Detect the Abnormal Phenomenon in PVM Program Based on Petri Net**, J. of Syst. Sci. & Syst. Eng., 1 (1999).

(9) H.Q.WANG, C.J.JIANG & S.Y.LIAO, **Behaviour Relations in Synthesis Process of Petri Net Models**, IEEE Trans. on RA, 4 (2000).

(10) C.J.JIANG, Testing of Functions of Complex Systems Based on Synchronous Composition Nets, Studies in Information Control, 4 (2000).

(11) H.Q.WANG, C.J.JIANG & S.Y.LIAO, **Concurrent Reasoning of Fuzzy Logical Petri Nets Based on Multi-Task Schedule,** IEEE Trans. on Fuzzy Syst., 3 (2001).

(12) C.J.JIANG, **A Polynomial-time Algorithm for the Legal Firing Sequences Problem of A Type of Synchronous Composition Petri Nets**, Science in China (Series E), 2 (2001).

(13) Y. Y. DU, C. J. JIANG, **Formal analysis of an online stock trading system by temporal Petri nets**, In: Proc. Int. Workshop on Computer Networks and Mobile Computing (published by IEEE Computer Society Press). Oct. 2001, Beijing, China.

(14) Y. Y. DU, M. Q. ZHAO, C. J. JIANG, **Study on method of Petri nets for stock trading systems**, In: Proc. Int. Workshop on IFAC-CEFEES, Oct. 2001, Tianjin, China.

(15) C.J.JIANG, **PN Machine Theory of Discrete Event Dynamic Systems**, Science Press of China, Beijing, 2000.