Deadlock Avoidance Based on Banker's Algorithm for FMS

Xu Gang   Wu Zhiming
Shanghai Jiaotong University

Keywords Flexible Manufacturing System, Deadlock, Banker's Algorithm

Abstract

This paper presents a method for deadlock avoidance algorithm used in Flexible Manufacturing System(FMS). This method is an improvement of the Banker algorithm. The Banker algorithm is commonly used in the Operating System (OS), but some improvement will have to be made on the algorithm if this algorithm is used in FMS. The difference between the process in operating system and the job in the FMS is fully discussed. Based on this difference, the improvement is made. In order to improve the algorithm, formal methods are adopted to the manufacturing systems. The simulation model is translated into a format suitable for model checking. That is, the model is written into PROMELA, the input language of the popular model checker SPIN. After that, SPIN is used to verify that the model does not have deadlock. This algorithm proves to be highly effective in practice.

1.Introduction.

When designed and operated effectively, FMS can be of assistance to manufacturer according to a defined production and process plans, which specify the activities and resources as well as sequence related conditions, such as precedence relations and synchronization. Alternative resources and routing for some of the jobs in FMS may be specified previously. This results in an increasing flexibility in job scheduling. Some resource allocations may lead to deadlock situations, in particular when the resources in system are limited. Deadlock problems can cause unnecessary costs (e.g. long down-time and low use of some critical and expensive resources), which are particularly important to be solved in Flexible Manufacturing Systems. Therefore, to develop efficient algorithm to improve and optimize the system performances while preventing deadlock situations become a basic requirement in running an FMS.

A deadlock is a state where a set of parts is in "circular waiting" i.e. each part in the set waits for a resource held by another part in the same set. In general, three strategies are used for dealing with deadlocks.

1. Detection and recovery. Every time a resource is requested or released, a check is made to see if any deadlocks exist. If exist, some policy will be adopted to recover the system from deadlock.
2. Deadlock prevention, by structurally negating one of the four required deadlock conditions. deadlocks will be impossible.
3. Dynamic avoidance by careful resource allocation: deadlock avoidance. Make judicious choices to assure that the deadlock point is never reached.

In this paper, we focus on the problem of deadlock avoidance. There are quite a lot research papers devote to deadlock avoidance. In [1] a deadlock avoidance algorithm is proposed for a class of Petri net models formed for flow shop manufacturing where a set of sequential processes are executed without alternating the order of using resources in each case. The algorithm controls the input flowing of new tokens in a local area, ensuring that token evolutions in system are always possible. Abdallah in [5] use structure theory of Petri nets to develop efficient deadlock prevention and deadlock avoidance methods for FMS. In [8] Naiqi Wu et al point out that, if an Automated Manufacturing System(AMS) operates at the deadlock boundary, i.e., under the maximally permissive control policy, it will not be deadlocked but a blocking(the process is stopped temporarily, and will go on after a peried of time) may occur more likely. Wu presents an AMS that works near but not at the deadlock boundary in order to gain the highest productivity. For the first time Wu presents such a policy: Liveness-policy. Without being too conservative, it can effectively reduce or even eliminate the blocking possibility that exists under a maximally permissive control policy. In [10] both deadlock prevention and avoidance control policies are proposed. The first part is based on the petri net reachable graph, while the second part is based on a look-ahead procedure that searches for deadlock situations by simulating the evolution process of system for a preestablished number of steps. Due to the fact that the avoidance policy does not assure that deadlocks are not reachable in future, they propose to combine this policy with a deadlock recovery system. CSP is also well suited to model information and control flows in manufacturing system

Foundation item：Project supported by the National Natural and Science Foundation(Grant No. 59889505,60074011,70071017)

usually exhibiting concurrency and non-determinism. Holzmann in [4] give an overview of the design and structure of the verifier: SPIN, reviews its theoretical foundation and gives some practical application. Havelund in [2] presents an application of the finite state model checker SPIN to formally analyze a multithreaded plan execution module. J.M. van et al in [6][7] proposed a specification of a heterarchical control system of a flexible production cell. The CSP-based language χ is used in the specification. V. Bos in [9] investigates whether automatic verification can be applied successfully for the analysis of industrial manufacturing systems models specified in χ, and applies the approach to a model of a manufacturing system consisting of a turntable, a drill, and a testing device.

If the FMS is modeled with CSP, many algorithms used in OS can be easily modified and applied in FMS. In the model of CSP, the problems of FMS correspond to the ones of OS. For example, the Job scheduling in FMS to the Process scheduling in OS. So, modeling FMS with CSP is valuable. The objectives of this paper are threefold. Firstly, the Banker algorithm is improved to be suitable to the FMS. Secondly, model FMS in CSP, and ground the future work in the analysis of FMS. Thirdly, verify the deadlock property of the improved algorithm with SPIN.

The Banker algorithm is used in OS. OS and FMS are all concurrent systems, they are similar in some way. So, some processes scheduling algorithm in OS can be applied in FMS after modified. But the two systems are different. In this paper, the difference between OS and FMS is discussed

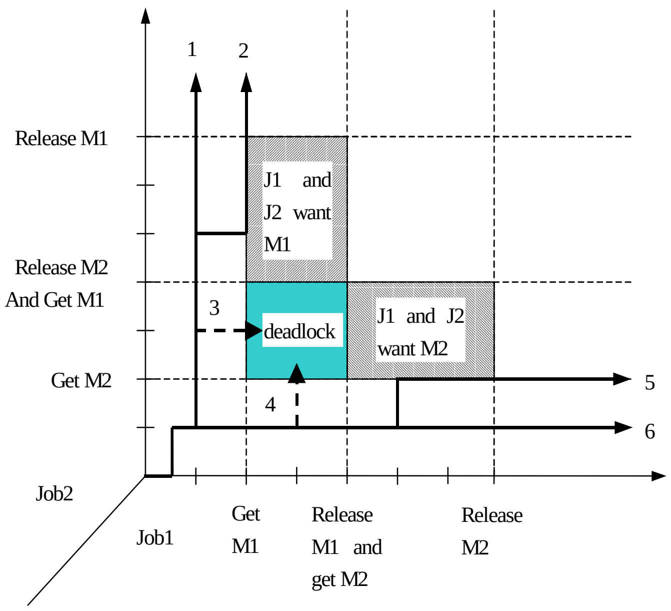

Fig.1 Example of deadlock

in detail. Based upon this difference, the Banker algorithm is modified and is applied in the control of FMS. This paper is organized as follows: the problem is presented in Section 2, the improvement of the algorithm is introduced in Section 3, the algorithm is applied in a case in Section 4 and Section 5 is a conclusion.

2. Modeling FMS with CSP

In this paper, readers are supposed to be familiar with CSP, and the definition of CSP can be found in Hoare [3]. A job is modeled as a process (in section3, the operations of Job is taken as process). The behaviour of each job is described by a process. Individual processes that operate concurrently communicate with each other by means of channels. All channels are one-to-one connections between processes. The interleaving of processes forms the behaviour of the whole system, that is, the scheduling of the two Jobs. The fig.1 describes two jobs competing for two resources. Each job needs exclusive use of resource for a certain period of time. The component of the system can be depicted using CSP as following:

Machine = (get M->maching-> releaseM->Machine);
Job1=(getM1->maching->releaseM1->getM2->maching->releaseM2->skip);
Job2=(getM2->maching->releaseM2->getM1->maching->releaseM1->skip);
The whole system can be described as following:

M1:Machine // M2:Machine // (Job1 ||| Job2)

"|||" represents that the interleaving of process Job1 and Job2. "//" represents that the processes M1 and M2 are subordination process of (Job1 ||| Job2). Job1 and Job2 interleave and share the resource M1 and M2

Deadlock is a kind of system status, in which a set of parts enter into a waiting loop, each part in the set waits the resource occupied by another part in the set. Just like what the shadow in Fig.1. has demonstrated, the x-axis represents progress in the execution of Job1 and y-axis represents progress in the execution of Job2. The joint progress of the two processes is therefore represented by a path that progresses from the origin in a northeasterly direction.

Process Job1 and Job2 share the resource M1 and M2, which is different from processes in OS.

1. Job1 and Job2 do not need both resources at the same time, and instead have the following forms:

Job1	**Job2**
…	…
Get M1	Get M2
…	…
Release M1	Release M2
…	…

```
Get M2                  Get M1
……                      ……
Release M2              Release M1
```

2. One Job represents one part flowing in the FMS. Part will occupy corresponding resource when it is processed in the FMS. So, if Job1 finish its first operation, that is, its operation on M1 has been done, Job1(part1) still occupied the M1 (suppose no buffer in this system) and wait to enter into M2. Releasing M1 occurs only when Job1 get M2. When all the operations of Job have been finished, the Job will be unloaded from the machine automatically. The same is to Job2. So, if the Job1 and Job2 run as the execution of 3 or 4, the system will deadlock.

Fig.1. shows six different execution paths. These can be summarized as follows:

Path 1. Job2 gets M2 and then releases M2. Get M1 and then release M1. When Job1 resumes execution, it will be able to get both resources.

Path 2. Job2 gets M2 and then releases M2. Get M1 and then Job1 executes and blocks on a request for M1. Job2 release M1 and finish. Job1 will get M1 and then M2

Path 3. Job2 gets M2 and then Job1 gets M1, deadlock will occur. Because as execution proceeds Job1 will block Job2 on M1 and Job2 will block Job1 on M2.

Path 4. Job1 gets M1 and then Job2 gets M2, deadlock will occur. Because as execution proceeds Job1 will block Job2 on M1 and Job2 will block Job1 on M2.

Path 5. Job1 gets M1 and then release M1. Get M2 and then Job2 executes and blocks on a request for M2. Job1 release M2 and then Job2 will get M2 and then M1.

Path 6. Job1 gets M1 and then releases M1. Get M2 and then release M2. When Job2 resumes execution, it will be able to get both resource.

The structure shown in Fig.1. can be extended to more machines and jobs with different routing procedures to describe a FMS. The approximate Gannt chart of Fig.1 can be seen in Fig.2,3,4,5

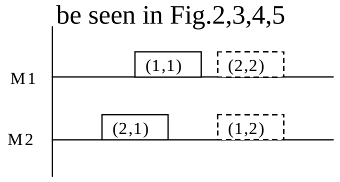

Fig.2. Gannt for deadlock in routing 3

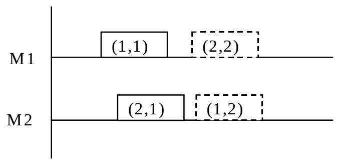

Fig.3. Gannt for deadlock in routing 4

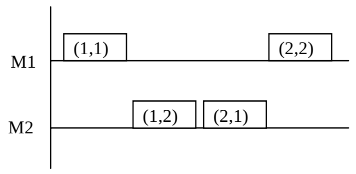

Fig.4. Gannt for routing 5,6

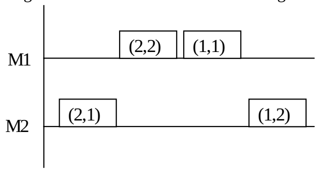

Fig.5. Gannt for routing 1,2

SPIN is a tool for analyzing models expressed in the modeling language PROMELA[4]. PROMELA was chosen for the study because of its focus on the interaction between processes. It is loosely based on Hoare's language of CSP and on Dijkstra's guarded command. A PROMELA program consists of processes, message channels, and variables. Processes are global entities, which specify behavior. They communicate with one another on message channels. Communication can be synchronous or asynchronous.

Given a PROMELA model, SPIN can perform simulations or exhaustive verifications of the system state space, during which it checks for the absence of deadlocks and for un-executable code. It can also verify linear time temporal constraints. Exhaustive verification can show conclusively whether a model contains errors. The model simulation tool provided in SPIN allows users to interactively simulate execution of PROMELA models. This tool is invaluable in the initial development and refinement of the model for the controller.

Using XSPIN (XSPIN is a graphical front-end of SPIN), the simulation result of routing 4 can be seen in fig6. The CSP model is written as PROMELA. Every machine is represented by a process, two jobs is combined into controllers, and the controller is in charge of scheduling the two Jobs. The channel is used by processes to communicate with each other. In the figure6, the vertical line represents the time line for instantiations of a process, and the line between processes represents the channel between processes. It can be seen that the system is deadlocked: process M1 (corresponding to M:2 process) and M2 (corresponding to M:3 process) wait for each other for ever, and the system can't evolve.

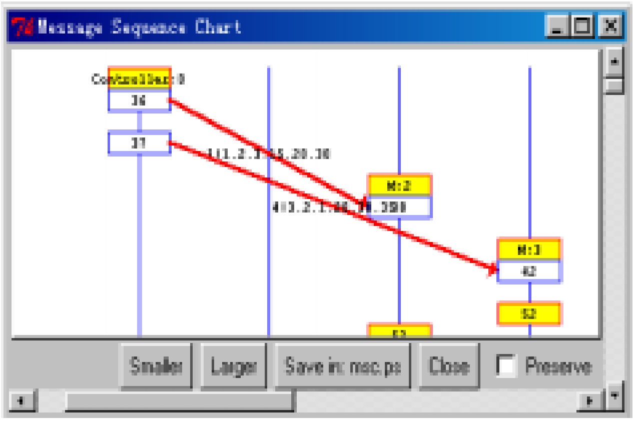

Fig 6 XSPIN for rout 4

3. Improvement of Banker algorithm

The Dijkstra Banker's algorithm operates commonly in the Deadlock-Avoidance of the Operating System's process scheduling. Because of the difference between OS and FMS, it will be too restrictive when this algorithm is directly applied to FMS. Some improvement had to be made.

3.1 why the banker can't be used in FMS directly?

The following table lists the basic difference between FMS and OS.

	FMS	OS
Scheduling object	Job	Process
Deadlock avoidance policy	schedule algorithm	Banker algorithm
Basic scheduling unit	Operation	None

The basic idea of the Banker's algorithm is to assure at all times that all possible future requests for resources can be satisfied with the current set of free resources.

1. Banker's algorithm requires that the number of resources in system should be fixed, the total resource number that the process requires must be declared before the process starts. The resource requirement that will make the system transfer to safe state can only be satisfied. However, the scheduling algorithm in FMS takes advantage of the production routing information. The total resource number that a Job requires can only be determined after the routing of a Job is determined. Because a Job can have one or more routing in FMS, and there will be different number of operations in different routing, the resource requirement number is different in different routing. So, the total number of resource requirement for a Job can only be determined in the processing, and can't be declared before a Job starts.
2. The Banker's algorithm assumes nothing about the order in which resources will be requested and released, that is, the operations order of a Job aren't be taken into account. In OS, the process will not start until its total resource requirement are satisfied, and it will not release the total resource until it finish. But in FMS, with the progress of the Job, its operation releases some resource, at the same time, the next operation will occupied some resource. So, the resources will not be fully used if the Banker's algorithm is directly applied in FMS.

It can be seen from the above description that it will be too restrictive and the analysis of deadlock will be too conservative if the Banker's algorithm is directly used in FMS. Some improvement had to be made.

3.2 Modification of Banker's algorithm

The Jobs are represented by controller process. Every resource is represented by a resource process. The information transferring between processes represents parts flowing in the FMS. In the following description, the process is Job, ie, controller. Some notation are shown as following:

Safe State, at this state there is at least one resource allocation sequence that can make all processes finish safely, and no deadlock produce;

Unsafe State, the state which is not satisfied with the condition of Banker;

Deadlock State, the state in which no process can go on.

Suppose one system with n processes, and m resource. For convenience, the array addition, subtraction, et al are abbreviated as following (***X*** and ***Y*** are arrays with subscript $[n][m]$)

$X <= Y$, if and only if $X[n][m] \le Y[n][m]$

$X \pm Y$, if and only if $X[n][m] \pm Y[n][m]$

$X := Y$, if and only if $X[n][m] := Y[n][m]$

Resource = ARRAY [$1\cdots m$] OF integer. Total amount of each resource in the system

Claim = ARRAY [$1\cdots n, 1\cdots m$] OF integer. Requirement of each process for each resource.

Need = ARRAY [$1\cdots n, 1\cdots m$] OF integer. Equal to (*Claim* – *Allocation*)

Allocation = ARRAY [$1\cdots n, 1\cdots m$] OF integer. Current allocation

Available = ARRAY [$1\cdots m$] OF integer. Total amount of each resource not allocated to a process.

Request = ARRAY [$1\cdots n, 1\cdots m$] OF integer. The current resource requesting of processes.

Route_change OF integer. The flag of the other routing of process.

Need_change = ARRAY [$1\cdots n, 1\cdots m$] OF integer. The Need array on another routing of the process.

Request_change = ARRAY [$1\cdots n, 1\cdots m$] OF integer. The Request array on another routing of the process.

Modified Banker's algorithm

Resource Allocation Algorithm is shown as following:

```
(1) If Request[i] ≤ NEED[i],then goto(2),else ERR // too
    many process request,return.
(2) If Request[i] ≤ Available, then(3), else waiting
(3) Tentative allocation, run as following:
Allocation[i]:= Allocation[i] + Request[i];
Need[i]:= Need[i] - Request[i];
Available[i]:= Available[i] - Request[i];
(4) If Need[i] ≤ Available[i],
   then if (Route_change==1)
Allocation[i]:= Allocation[i] - Request[i];
Need[i]:= Need[i] + Request[i];
Available[i]:= Available[i] + Request[i];
{Need[i]:=Need_change[i];
```

Request[i]:=***Request_change***[i]; goto(1)}

else stop;

else goto(5);

(5) Safety checking. If safe，then allocate，***Request***[i]:=0; else run as following (abort the allocation):

Allocation[i]:= ***Allocation***[i] - ***Request***[i];

Need[i]:= ***Need***[i] + ***Request***[i];

Available[i]:= ***Available***[i] + ***Request***[i].

P_i wait.

Safety checking algorithm is shown as following:

Work : ARRAY [1⋯*m*] OF integer. Record the resources which are currently idle or supposed to be released.

Finish : ARRAY [1⋯*n*] OF bool. The flag for the finishing of process.

(1) ***Work*** := Available;***Finish*** [j] := False (1≤j≤*n*);

(2) Search for process P_j ,whose (***Finish***[j]==False and ***Need***[j]≤*Work*).

If not found, goto (4);

(3) ***Work*** := ***Work*** + ***Allocation***[j];

Finish[j] := True; goto (2).

(4) if ***Finish***[j]==True (1≤j≤*n*),

then the state is safe,

else the state is not safe.

Additional explaining:

Check every process to see if ***Available*** + ***Allocation*** ≥ ***Claim***. Dynamic change of the ***Claim*** array is taken into account in the modified Banker's algorithm. In the original algorithm, the ***Claim*** array is determined at the initial of the system. It is not changed while the whole procedure. This is feasible in OS. But in FMS, it doesn't work. In FMS, there are operations in the Job. If an operation of a job finish, it will release the resource which it occupied, and go on to the job's next operation. The operation is taken as a process here. The process can finish firstly, it releases the resource which it occupies. Thus, the resource can be used by other processes. That is, the operations of a job can release resources orderly when the operations of the job has finished. So, the resource releasing don't need to wait until the job finishes. In this case, the ***Claim*** array is designed to change with the job progressing. The definition of ***Allocation*** array and ***Available*** array are not changed. The ***Claim*** array varies with the information flowing in channels.

The routing selection of Jobs is involved in the modified algorithm. When one routing is not satisfied with the condition of the Banker's algorithm, another routing is selected. If this routing is still not satisfied with the condition of the Banker's algorithm, then the next one, or waiting.

4. Case study

The CSP model of an example with three machines and two jobs is shown in Fig.7. The job information is shown as following: *P*1 = {M1, M2, M3}, *P*2 = {M3, M2, M1 or M3, M1}. The CSP model for Fig.7. is shown as following:

Machine = (getM->maching-> releaseM-> Machine);

Job1=(getM1->maching->releaseM1->getM2->maching->release M2->get M3->maching ->release M3->skip);

Job2=(getM3->maching->releaseM3->(getM2->maching ->release M2->getM1->maching->releaseM1->skip | get M1->maching->releaseM1->skip));

Process Job1 and Job2 share the resource M1 and M2, the whole system can be described as following:

M1:Machine // M2:Machine // M3:Machine // (Job1 ||| Job2)

In this model, there are five processes and nine channels. Transform the model into the PROMELA, the main part of the model is shown as following.

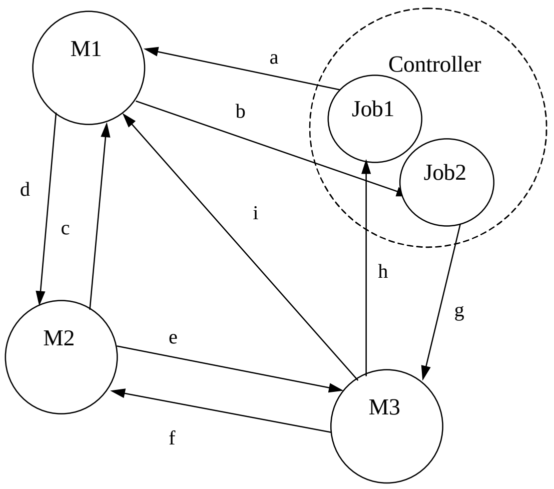

Fig.7. Components of an FMS

```
/*routing information*/
routing definition of Job1,Job2_1,Job2_2;
resources Flag definition;
/*channel information*/
definition of channels among different process;
a,b,c,d,e,f,g,h,i;
/*main function and process*/
inline function banker()
inline function safecheck()
Job1's process template definition, Job1()
Job2's process template definition, Job2()
Machine template definition M
```

The Job routing information is transmitted in the channel. While the Job progresses in the system, the routing information will be modified. In order to make the resources mutual exclusion use, Flag for the resources M1, M2, M3 is set: resourceFlag, which is an user-defined structured data type. When the resource process (such as M2 process) receives message from its upper process (such as M1 process), the Flag.M1flag will be set to 1, which represents the releasing of resource M1. At the same time, Flag.M2flag will be set to 0, which represents the resource M2 being occupied. And the corresponding part of routing information JRInfo will be set to zero, for example, J1Rinfo.OP1=0 if this is Job1. When the Job process (such as the Job1) receives message from other resource process (such as M1 process), the Flag.M1flag will be set to 1. When the Job process send message to other resource (such as M1 process), the Flag.M1flag will be set to 0. When the Job finishes, the corresponding routing information will become all zero. The routing selection is determined in the execution, and it is dynamic. The sequence of the Job process execution and the communication between processes guarantee that the part is processed according to its routing.

Simulation with SPIN

In the Message Sequence Chart fig.8, there are six vertical lines that represents the six processes in the system (list from the left to the right): Job1, Job2, init process (which is used to initialize the M1, M2, M3 process), M1, M2, M3. The arrows connecting the processes are channels. Boxes correspond to steps in the simulation. Communication between Job1, Job2, M1, M2, M3 is executed through channels. Job1 process sends J1RInfo to M1 process (corresponding to process M:3 in the graph; M2 process to M:4 process; M3 process to M:5 process), M1 process finishes its operation, sends the J1Rinfo to M2 process. Finally, M2 process to M3 process. Thus, all operations of Job1 are finished and M3 Process sends the final J1Rinfo to Job1 process. The same to the progressing of J2Rinfo. The graphical output only highlights the interaction between processes. Steps are numbered sequentially and described in a separate Simulation Output window (not shown).

Above system should meet the following requirement:

1.No deadlock;

2.every product should be processed according to their operation requirement.

Verification with SPIN

1.The linear-time temporal logic (LTL) formulae for this system is written as “<>S”. “<>” is the temporal operator “eventually”. S is defined in a macro, represents the final state of the successful execution: it is the *logical and* of the Jobs’ routing information and the final resource state.

```
#define p (J1RInfo. OP1== 0) && (J1RInfo.OP2==0) && (J1RInfo.OP3==0)
#define q (J2RInfo. OP1== 0) && (J2RInfo.OP2==0) && (J2RInfo.OP3==0)
#define r (p && q)
#define r1 (Flag.M1flag==1) && (Flag.M2flag==1) && (Flag.M3flag==1) && (AGV_Flag==1)
#define S (r && r1)
```

The verification result of the LTL statement on the model is shown below.

State vector	Depth reached	Output
392	1300	success

2. We used PROMELA's event trace mechanism to verify that every product is processed. Event traces define admissible sequences of communications. SPIN checks if each trace of the model adheres to the event trace. That is, of each trace of the model only the events occurring in the event trace are considered and these events have to occur in the order defined by the event trace. Event traces are defined with communication statements and normal control flow structures of PROMELA (for example, do...od and if...fi), however no variables are allowed. If a channel occurs in a statement of an event trace, the verifier checks if every communication along the channel matches an event of the event trace. For a thorough description of SPIN's event traces, refer to [20]. In the event trace, we define possible/allowable communications that identify the Job operations. We chose communications over channels, and we can make the following observations:

```
trace
{
    do
    ::Job1’s operation channels sequence: Job1->M1->M2->M3->Job1;
    ::Job1’s operation channels sequence: Job2->M3->
            if
            ::M2 is idle:->M2->M1->Job2;
            ::M1 is idle:->M1->Job2;
            fi
    od;
}
```

The verification result shows that every product will be processed according to the operation requirement.

It can be seen that after simulation and verification, the algorithm is right. The complexity of the algorithm is linear $O(|C_n|)$, C_n is the number of channels. The channel number is related to the machines and operations number. The algorithms in Banaszak[1], Abdallah[5], Wu[8], Viswanadham[10] put more emphasis on the

Petri net theoretical analysis. Limited by the Petri net, the size of the FMS in which it can be applied is small, and has little practical use. After simulation and verification, the algorithm in this paper has been applied in the FMS controller of the virtual workshop platform in Shanghai Jiaotong University. The example in [11] is adopted to demonstrate the effectiveness of the algorithm. It is supposed that the problem size can be represented with M×N(M is the machine numbers, N is the job numbers). The size of this problem is 33×127. The CPU time for this problem on PC with Celeron533, RAM 64M is 7.7 minutes. All jobs can be completed, and no deadlock occurring.

5. Conclusion

In this paper, a deadlock avoidance method applied in FMS is proposed. This method is modified upon the classical Banker's algorithm. The different between OS and FMS is discussed in detail. Based on this difference, some improvement is made. FMS is modeled with CSP, so the algorithm can be verified with SPIN. Form the result of the experiment, it can be seen that the model is deadlock-free.

Reference

[1] Banaszak, Z.A.; Krogh, B.H. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows[J]. IEEE Transactions on Robotics and Automation, 1990, 6(6), 724 –734.

[2] Havelund K; Lowry M; Penix J. Formal analysis of a space-craft controller using SPIN[J].IEEE Transactions on Software Engineering, 2001, 27(8), 749-765.

[3] Hoare C A R, Communicating Sequential Processes[M]. Prentice--Hall, 1985, 9.

[4] Holzmann G J. The model checker SPIN[J]. IEEE Transactions on Software Engineering, 1997, 23(5), 279–295.

[5] I.BenAbdallah ; H.ElMaraghy. Deadlock Prevention and Avoidance in FMS:A Petri Net-Based Approach[J]. International Journal of Advanced Manufacturing Technology, 1998, 16(1).

[6] J.M. van de Mortel-Fronczak and J.E. Rooda. A Case Study in the Design of Control Systems for Flexible Production Cells[A]. Proceedings of MIM'97[C], Vienna Austria:1997,243-248.

[7] J.M. van de Mortel-Fronczak, J.E. Rooda, and N.J.M. van den Nieuwelaar. Specification of a Flexible Manufacturing System Using Concurrent Programming[J]. The International Journal of Concurrent Engineering: Research & Applications, 1995,3 (3), 187--194.

[8] Naiqi Wu, MengChu Zhou. Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems[J]. IEEE Transactions on Robotics and Automation, 2001, 17(5), 658 –669.

[9] Bos and J.J.T. Kleijn. Automatic verification of a manufacturing system[J]. Robotics and Computer Integrated Manufacturing, 2001, 17(3):185-198.

[10] Viswanadham, N.; Narahari, Y.; Johnson, T.L. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models[J]. IEEE Transactions on Robotics and Automation, 1990, 6(6), 713–723.

[11] Hoitomt D J, Luh P B, Pattipati K R, A practical approach to job-shop scheduling problems,IEEE Transactions on Robotics and Automation, 1993, 9(1), 1-13.

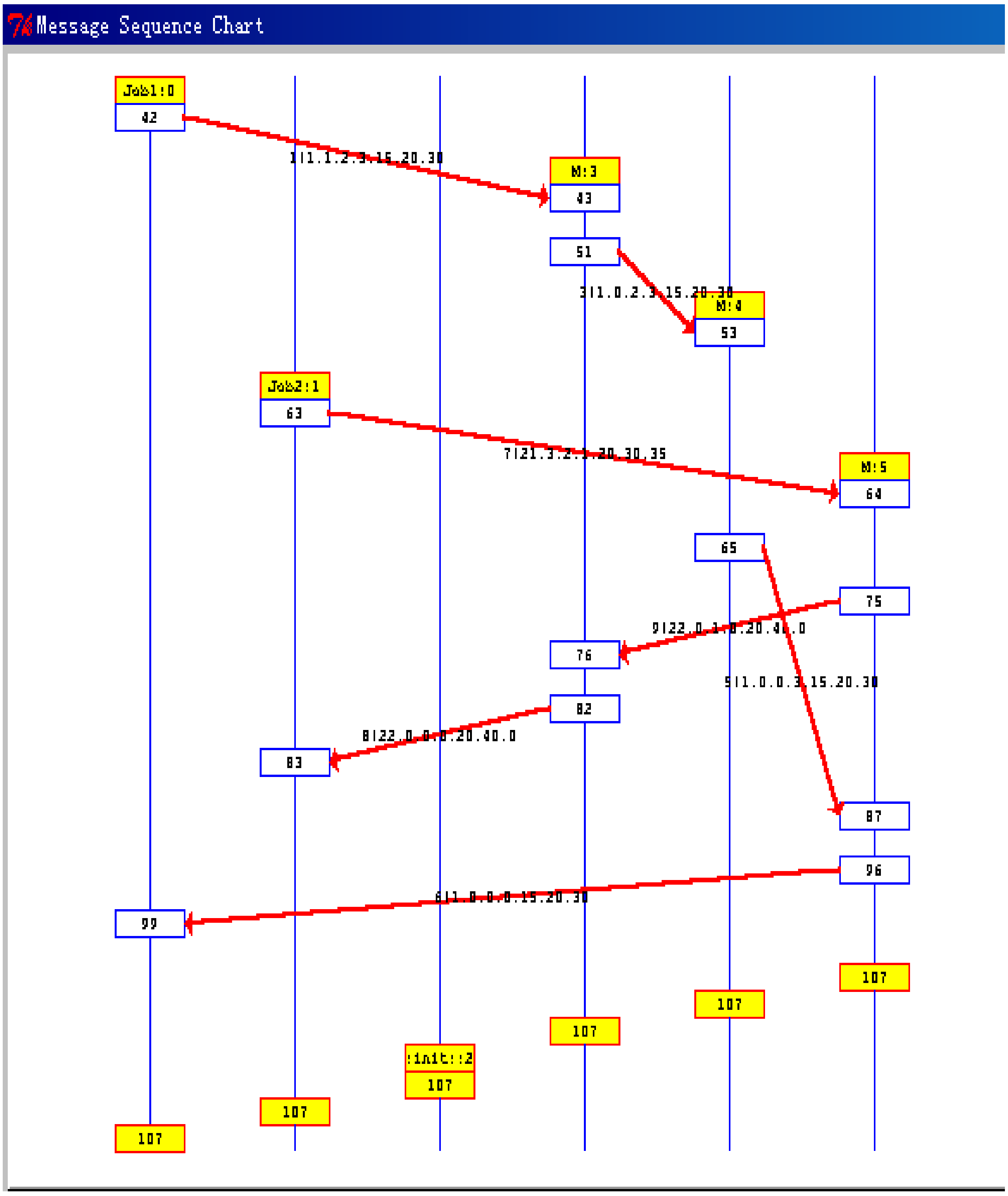

Fig.8. Simulation for controller (random seed=1)