Critical resources software and control modeling with finite automata

BOGUSŁAW SCHREYER\textsuperscript{a}, KRZYSZTOF KOSIŃSKI\textsuperscript{b}

\textsuperscript{a}Nipissing University, North Bay, Kanada

\textsuperscript{b}Wyższa Szkoła Informatyki Stosowanej i Zarządzania, Warszawa

Received 20 September 2010, Revised 15 October 2010, Accepted 29 October 2010

Abstract: There are analogies between modeling the transportation systems with critical resources (CR) and a critical section (CS) problem in operating systems. This article is developing this analogy in the direction of using the finite automata. The article analyses Peterson’s algorithm [4], [5] and Lamport’s algorithm [2] using deterministic finite automata (DFA) as well as it takes into consideration the problem of modeling traffic with binary semaphore using nondeterministic finite automata (NFA). Introduction of finite automata makes traffic control modeling much clearer on programming side and brings hardware application closer to such control.

Keywords: critical resources, finite automata, traffic control

1. Introduction

In accordance with [2] the following conditions have to be fulfilled:

1. Mutual exclusion: two or more Transportation Units (TU) cannot use CR simultaneously,
2. Progress: if CR is not currently busy, it can be accessed after limited amount of time, if any TU requests such access, (1)
3. Bounded waiting: if CR is busy with TU, the number of reentries to CR is limited, if other TU requested access to CR.

In order to use finite automata and according to [3], the following definition has been used:

\[ \text{FSA} = (Q, \Sigma, T, q_0, F) \], where

1. \( Q \)– automaton states,
2. \( \Sigma \)– state inputs,
3. \( T \)– transition function,
4. q₀ – initial state,
5. F – final states,
For DFA, transition function T: Q × Σ → Q
For NFA, transition function T: Q × Σ → P(Q) (P(Q) – automaton state subset).

2. Peterson’s algorithm with DFA automaton

Peterson’s algorithm for two arbitrary processes P₀ and P₁, which correspond to two TU units: TU₀ and TU₁ uses three variables: ready[0], ready[1] and turn. Variables Ready are Boolean variables defined on integer variables: 0 and 1 and True or False. The value of ready[k] variable becomes True, if TUₖ (k=0,1) is ready to enter into CR. Otherwise its value is False. Variable Turn is an integer variable with values 0 or 1. Its value is set by P₀ and P₁ control processes. If ready[k]= =True and turn= =k, it means that TUₖ enters CR. nr state (TU not ready) is when ready variable = False. bw state (JTₖ busy waiting) is when both variables ready[1-k]=True and turn=1-k (k=0,1). In other words, TU₀ is in bw state, if ready[1]=True and turn=1. Similar situations is for TU₁.

If the condition is not fulfilled, TU enters CR zone.

Practical application of this algorithm for control involves the introduction of additional variables such as entry, and advance in [1]. Peterson’s algorithm for two JT₀ and JT₁ entries (controlled with P₀ and P₁ processes) is as follows:

```c
    // P₀ process for JT₀
    do {
        while (entry[0] == F);
        ready[0]=True; //stand ready, JT₀ jest gotowa
        turn=1; //be a gentleman
        while (ready[1] && turn==1); //stand bw, JT₀ czeka
        advance[0]=0;
        advance[1]=0;
    }
```

```c
    // P₁ process for JT₁
    do {
        while (entry[1] == F);
        ready[1]=True; //stand ready, JT₁ jest gotowa
        turn=0; //be a gentleman
        while (ready[0] && turn==0); //stand bw, JT₁ czeka
        advance[1]=0;
        advance[0]=0;
    }
```

```
----------------------------------- critical resources (CR) -----------------------------------

ready[0]=False; //nr, JT₀ not ready
remains of section 0 //JT₀ performs its task
while(1)

ready[1]=False; //nr, JT₁ not ready
remains of section 1 // JT₁ performs its task
while(1)
```

Fig. 1. Peterson’s algorithm adopted for TU control

The following definition of DFA automaton has been assumed for Peterson’s algorithm: DFA=(Q, Σ, T, q₀, F), where
1. Q – automaton states: {(nr, nr), (nr, CR), (CR, nr), (CR, bw), (bw, CR)}
2. Σ – state inputs: (ready[0],ready[1],turn)= {(F,F,0), (F,F,1), (F,T,0), (F,T,1), (T,F,0),
   (T,F,1), (T,T,0), (T,T,1)}
3. T– Transition function (in accordance with the Table 1 and a graph in Fig. 2),
4. q₀ – initial state: (nr,nr)
5. F– final stages: = {Ø, (nr,nr), (nr,CR), (CR,nr), (CR,bw), (bw,CR)}

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Q states</th>
<th>(F,F,0)</th>
<th>(F,F,1)</th>
<th>(F,T,0)</th>
<th>(F,T,1)</th>
<th>(T,F,0)</th>
<th>(T,F,1)</th>
<th>(T,T,0)</th>
<th>(T,T,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(nr, nr)</td>
<td>Ø</td>
<td>Ø</td>
<td>(nr,CR)</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>(bw,CR)</td>
</tr>
<tr>
<td>(nr, CR)</td>
<td>(nr,nr)</td>
<td>(nr,nr)</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>(bw,CR)</td>
</tr>
<tr>
<td>(CR, nr)</td>
<td>(nr,nr)</td>
<td>(nr,nr)</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>(CR,bw)</td>
</tr>
<tr>
<td>(CR, bw)</td>
<td>Ø</td>
<td>Ø</td>
<td>(nr,CR)</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>(bw, CR)</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>(CR,nr)</td>
<td>(CR nr)</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
</tbody>
</table>

*Σ= (ready[0], ready[1],turn)

Tab. 1. TU₀ and TU₁ final stages as a function of ready[0], ready[1] and turn variables

DFA finite automaton state graph corresponding to the Table 1 has been presented in Fig. 2:

![Fig. 2. State graph of two units: TU₀ and TU₁](image-url)
As it is seen from Table 1 and Fig. 2, all three conditions of smooth and collision-free use of CR (1) are provided:

1. Does not have state (CR,CR),
2. After (nr,nr) state, that is, when CR is not busy with any TU, (CR,**) or (**,CR) state immediately follows, where **= nr,bw.
3. (CR,**) state follows (**,CR) state immediately or after two subsequent CR request by TU₀. Similarly in case of TU₁.

### 3. Lamport’s algorithm with DFA automaton

Lamport’s algorithm is applicable to any number of processes, and therefore also to any number of TU, that also request to handle a CR. As in [6], it is convenient to number the algorithm code blocks for the process \( i \) and to treat those blocks as finite automaton states. *Choosing* block (state 1) is subsequently attributed to a higher ’ticket’ number \([i]\) that sets the \( i \) process (or TU\( i \)) in a queue to CR.

*Busy waiting* block (state 2) causes that the current process \( i \) (TU\( i \)) is waiting to enter CR, if either the next ticket is being assigned to another process, or when a given \( i \) process has a higher ticket number than any other process, or if the \( i \) process and any other process have the same ticket numbers, but the given \( i \) process and is ’younger’ than the other process. The process ’age’ is determined by next \( i \) cardinal number that is being assigned during the creation of the process. Thus, a younger process (TU) has a higher ID number.

*Critical section* block (state 3) does not require comments.

Not ready block (state 0) assigns 0 ticket number to the given \( i \) process (TU\( i \)). When the first and temporarily the only process (TU) requests access to CS, it will be granted the first ticket number.

<table>
<thead>
<tr>
<th>State:</th>
<th>Code block:</th>
</tr>
</thead>
</table>
| Choosing (ch) or 1 | choosing\([i]\) = true;  
  number\([i]\) = max(number\([0]\),  
  number\([1]\), ..., number\([n-1]\))+1;  
  choosing\([i]\) = false; |
| busy-waiting (bw) or 2 | for (j = 0; j < n; j++)  
  do  
  while choosing\([j]\) do no-op;  
  while number\([j]\) ≠ 0 and (number\([j]\), j)<(number\([i]\), i) do no-op;  
  end; |
| critical resource or 3 | /*** CR /***/ |
| not ready (nr) or 0 | number\([i]\) = 0; |

Fig. 3. Lamport’s algorithm with automaton states
The following definition of DFA automaton has been assumed for two states:

\[ \text{DFA} = (Q, \Sigma, T, q_0, F), \]

1. \( Q = \{(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2), (2, 3), (3, 0), (3, 1), (3, 2), (3, 3)\} \) (state pairs for two processes)
2. \( \Sigma = \{(0, 1), (1, 0), (1, 1)\} \) (state inputs)
3. \( q_0 = (0, 0) \) initial state
4. \( F = \emptyset \cup Q \) \((3,3)\) final states
5. \( T \) the same as in Tab. 2:

\[
\begin{array}{c|ccc}
\text{Inputs} & (0,1) & (1,0) & (1,1) \\
\hline
Q \text{ states} & (0,0) & (0,1) & (0,2) \\
 & (0,2) & (0,3) & (0,0) \\
 & (1,0) & (1,1) & (1,2) \\
 & (1,2) & \emptyset & (2,2) \\
 & (1,3) & (1,0) & (2,0) \\
 & (2,0) & (2,1) & (3,0) \\
 & (2,1) & (2,2) & \emptyset \\
 & (2,2) & (2,3) & \emptyset \\
 & (2,3) & (2,0) & \emptyset \\
 & (3,0) & (3,1) & (0,0) \\
 & (3,1) & (3,2) & (0,1) \\
 & (3,2) & \emptyset & (1,0) \\
 & (3,3) & \emptyset & \emptyset \\
\end{array}
\]

Tab. 2. Transition function (T) of DFA automaton according to Lamport’s algorithm

State inputs indicate the order of state changes in two processes. The (0,0) input indicates no change in the state, i.e. two processes perform the same program blocks. The (0,1) input points to the fact that the first process remains in its state (in its code block), while the second process proceeds to the next state (code block). Similarly, the (1,0) input and the (1,1) input.

As it is seen in Tab. 2, there is no (3,3) final state. An empty symbol \( \emptyset \) is inserted in its place. The lack of this state is a necessary condition to solve the CS problem, namely mutual exclusion. At the same time, it can be seen that there is no transition from the
(1,2) state to the (1,3) (2,1) state to the (3,1) state and from all the states leading in theory to (3,3).

Also, there are no transitions from (3,3) state to another state.

There is no final state for (1,2) state at input (0,1), and for (2,1) state at input (1,0) because such a transition is not possible. For those cases, blank final states Ø have been placed in the table of T function. DFA definitions can be extended to any number of k processes. Then, instead of ordered pairs of elements Q, Σ, q₀, F, an orderly sequence of k elements would occur.

Below is attached a graph of DFA automaton states for Lamport’s Algorithm for the two processes (TU).

k,l m = (0,1,2,...) with exceptions according to the definition of T function. The k and l variables sum up the (1,0) and (0,1) inputs respectively, while m sums up the (1,1) inputs. The mod function provides values from a set of {0,1,2,3}.

![Fig. 4. Automaton state graph according to Lamport’s Algorithm](image)

As it is seen from Tab. 2 and Fig. 4, all three conditions (1) of smooth and collision-free use of CR (1) are provided:

1. No state (CR,CR)=(3,3).
2. State (3,**) follows states (1,0) and (2,0). State (**,3) also follows states (0,1) and (0,2). **= \{nr,ch,bw\}=(0,1,2)
3. State (CR,**)=(3,**) follows shortly after state (**,CR)=(**,3) and the other way round, if TU request entry to CR.
4. NFA automaton that is modeling control using a binary semaphore

If there is no TU in the CR, the semaphore is raised (s=1). Otherwise, s=0. JT control using an s semaphore is as follows:

```c
do {
    P(s);
    // CR
    V(s);
    //
    } while (1);
```

In this algorithm, P(s) is a wait operation and it is placed immediately before CR, while the V(s) operation is a signal operation, placed exactly after CR. P operation checks and sets the s semaphore. If s=1 and TU is waiting for CR, P sets TU in CR and lowers the semaphore (s=0). V operation checks the queue and sets the semaphore. If the queue before CR does not exist (Q=0), it raises the semaphore, but if the queue exists (Q=1), next TU is placed in CR.

P(s) and V(s) definitions are as follows:

```c
P(s):
    if (s = =1)
        s=0;
        CR=1; //if s=1, TU is placed in CR
    else
        {Q=1; //if s=0, TU is placed in a queue };
}
```

```c
V(s):
    if( Q= =0) s= =1; // if there is no queue, s=1
    else CR= =1; // TU is placed in CR
    
```

The automaton state is an ordered three (s, Q, CR), where

s= 0, 1, i.e. if there is no TU in CR, the semaphore is raised and s=1, otherwise s=0
Q= 0, 1, i.e. if there is a queue to CR, Q=1, otherwise Q=0
CR=0,1, i.e. if any TU is in CR, CR=1, otherwise CR=0.

State (1,0,0) is the initial state. The semaphore is raised (s=0), there is no queue (Q=0), there is no TU in CR (CR=0).

If a TU requests access to CR, it will be accepted and the semaphore will be lowered (s=0), there is no queue. Automaton state is (0,0,1). In the graph it corresponds to P(s) input causing a transition from state (1,0,0) to state (0,0,1).

The next state can be either a (1,0,0), if TU leaves CR before the other TU appears (input V(s) is working), or state (0,1,1), if the next TU appears before the first TU leaves CR (P(s) input). Simultaneously, a queue is being created.
In the second case, the next arriving TUs do not change it, but the queue is growing (P(s) operates repeatedly, but does not change the status (0,1,1)). If TU leave CR simultaneously, V(s) is in operation. As long as there is a queue, the state is (0,1,1) until the last TU in the queue (state (0,1,1)).

When the last TU leaves CR, V(s) changes the automaton state from (0,1,1) to (0,0,1). When the last TU leaves CR, V(s) input changes state (0,0,1) to the initial state (1,0,0).

As it can be seen from the control algorithm, semaphore definition, and Fig. 5 graph, all three conditions (1) are met:

1. Two TU can not be present in CR at the same time,
2. Access to CR is controlled by the Q queue with the FCFS protocol,
3. TU that left CR will be placed at the end of the queue. Other TU will be admitted to CR in the order of their placement in a finite Q queue.

Since the V(s) operation may cause two different states: (0,1,1) and (0,0,1), it is a nondeterministic automaton.

5. Conclusion

This article analyzes the three algorithms in the critical section of operating systems used to control the transport units (TU) in the control of the allocation of critical resources (CR). The analysis uses Deterministic Finite Automata (DFA) and Nondeterministic Finite Automata (NFA). It can be clearly seen from this analysis that all three conditions (1) of proper TU control have been fulfilled. At the same time, finite automata allowed for concise and clear modeling of the CR problem in transport systems.
References


Automaty skończone w modelowaniu sterowania ruchem w systemach transportowych z zasobami krytycznymi

Streszczenie

Problem poruszany w artykule dotyczy modelowania ruchu w systemie transportowym ze środkami krytycznymi (CR). Jest oparta na analogii z podobnym problemem sekcji krytycznej (CS) w systemach operacyjnych.

Niniejszy artykuł jest rozwinięciem tej idei w naturalnym kierunku uogólnienia tj. z zastosowaniem automatów skończonych. Artykuł analizuje algorytm Petersona oraz algorytm Lamporta przy użyciu deterministycznych automatów skończonych, jak również problem modelowania ruchu przy pomocy semaforu binarnego przy użyciu niedeterministycznych automatów skończonych.

Wprowadzenie automatów skończonych czyni modelowanie sterowania ruchem na drodze programowej bardziej przejrzystym jak również przybliża zastosowanie hardwaru do tego sterowania.

W artykule dokonano analizy trzech algorytmów krytycznej sekcji w systemach operacyjnych w zastosowaniu do sterowania jednostkami transportowymi w kontroli przyznawania środków krytycznych. W analizie zastosowano skończone automaty deterministyczne oraz niedeterministyczne. Jak wynika z tej analizy, wszystkie trzy warunki (1) poprawnej kontroli JT zostały spełnione. Jednocześnie automaty skończone pozwoliły na zwarte i przejrzyste modelowanie problemu CR w systemach transportowych.