Five Message Handshake Project in Spin

Alexander Steen and Max Wisniewski

Institut für Informatik, FU Berlin

Abstract. The distributed algorithm for the mutual exclusion problem proposed by Suzuki and Kasami [**?**] is checked with the modelchecker *Spin*. We present a modeling for the algorithm in Promela, the properties we want to check for this algorithm and a short error analysis, why the second algorithm of Suzuki and Kasami does not work.

1 Introduction

Skip

2 Problem / Algorithm???

No problem

3 Modeling in Spin

In this part we look at some details of the modeling process.

3.1 Why Spin

Because FUCK YOU! Thats why.

3.2 Channels

The processes send *REQUEST* and *PRIVILIGE* messages which both need to be handeld. For any atempt to enter a process sends *REQUEST* to each of its neighbors. We decided to use a mailbox kind of channel system because this way a process does not have to iterate over many possible channels and check each of them seperatly.

```
chan mailbox[N] = [N] of {mtype, int, int, Queue, Array}
```

3.3 Request Messages

The methode p1 is excuted if a *REQUEST* message is received. In the modeling we decided to prioritize the receiving of *REQUEST* at the waiting points that are the *remainder* and the waiting to enter the critical section.

This is neccessary to satisfy the deadlock freedom and fairness constraints.

3.4 Global Variables

We decided to model all local variables of the processes as a global array of variables. We have done this out of debug reasons. This way we could check on receiving a message if everything was the way we expected it.

3.5 Send and Receive in Spin

There occured an ambiguous error in our implementation when we used a wrong number of matching variables in receiving a message. It happend some times that the received message differed from the send message. This way some of the requesting processes where dropped from the queue and were not considered for execution leading to a state where only one process was possible to enter the critical section.

3.6 Queue

We choose an implementation of a queue without a check for overflow.

```
insert 'insert' here
```

We could do this because in the program a number for a process is at most added once to the queue. If we give the queue an array of size N there could never occure an overflow.

3.7 Next

Write smth.

4 LTL Properties

A Mutual Exclusion Algorithm needs to satisfy the four properties

- Mutual Exclusion,
- Absence of Starvation,
- Fairness,
- No Unneccessary Delay

to be considered as correct.

In Spin we have to model for each of these properties one or more LTL - Properties.

4.1 Mutual Exclusion

We added a variable incs that is incremented before the critical section and decremented aftarwards. If initialized to zero mutual exclusion is expressed by the property

$$\Box\,(\text{incs} \leq 1) \tag{1}$$

which is used in both implementations.

4.2 Absence of Starvation

The algorithm already uses a flag requesting to mark if a process wants to enter the critical section. Using the counter incs from the Mutual Exclusion property we can express deadlock freedom by

$$\Box\,((\exists i\ :\ 0 \leq i \wedge i < N \Rightarrow \text{requesting}[i]) \Rightarrow \diamond \text{incs} = 1) \tag{2}$$

again in both algorithms. For the checking in the bounded example we expanded the existential quantifier explicitly.

$$\Box\,(\text{requesting}[0] \vee ... \vee \text{requesting}[N-1] \Rightarrow \diamond \text{incs} = 1) \tag{3}$$

4.3 Fairness

All processes do not differ except for their Identifier. Therefore we will check the fairness constraint for the first process and the second process. The first one because it has initially the privilage and the second one as a representive for every other process. This time we used a label at the critical section and an array for the process id's.

Fairness can be expressed by

$$\Box\,(\text{requesting}[0] \Rightarrow \diamond \text{p[ids [0]] @critical} \wedge \text{requesting}[1] \Rightarrow \diamond \text{p[ids [1]] @critical}) \tag{4}$$

in both algorithms.

4.4 No Uneccessary Delay

As in the last property we will check this one for one of the precesses.

$$\Box\,(\text{incs} = 0 \wedge \text{requesting}[0] = 1 \wedge \text{requesting—}[1] = 0 \wedge ...\text{requesting}[N-1] = 0 \Rightarrow \diamond \text{incs} = 0) \tag{5}$$

NO IDEAD HERE.

5 Checking the Properties

We saw something

5.1 Communication Diagrams

Insert some.

6 Conclusion

Yes.

7 Bibliography

Not today.