CS4231 Homework 2

Laurence Putra Franslay (U096833E)

12th Apr 2013

1 Question 1

```
1 getAndSet(newValue, curVar){
2         while(true){
3                 a = [curVar]*;
4                 tmp = a;
5                 if([curVar]* = newValue){
6                         return tmp;
7                 }
8         }
9 }
```

2 Question 2

2.1 Description of ABA

The ABA problem happens when 2 or more threads are working on the same data structure, and while 1 thread has read the information it needs to get the job done, it is interrupted, and another thread comes in and modifies the data structure. The data structure is modified such that when the first thread verifies it later on, it's seems to that thread that the value it will be working with has been left untouched, when it already has been, and results in the first thread working on the current data set based on previous assumptions, and may potentially lead to problems later on.

2.2 Can ABA happen to MCS? Why?

No, it will not affect MCS. This is as only the main thread can decide when to let the next thread become the main thread and allow it do do more than just adding itself to the back of the queue.

2.3 If get-and-set is not truly atomic, does that change things

No, it will not change anything. This is as while get-and-set is not truly atomic, the way it has been designed forces it to act in such a way that's similar to how an atomic operation would, hence, it would

not make any difference.

3 Question 3

If an interrupt happens at that point, there are 3 things that can happen, 1) the other thread proceeds to remove successfully pop the top element out, 2) the other thread pushes a new element in and 3) the other thread does absolutely nothing.

If the other thread does absolutely nothing, then this thread will continue, and compare will be successful, and it will set the top to the next node, and return the node that was popped.

However if the other thread either pushes a new node onto the stack, or pops the node, compare-and-set will fail. In this case, the process will then have to restart.

The hidden assumption of course, is that compare-and-set is and atomic operation.

4 Question 4

The parallel version of this dynamic programming problem will run all the recursive calls in parallel, and store the value of the paths in an array.

The runtime of this algorithm sequentially is $O(n^2 2^n)$.

The depth of the tree generated by the algorithm is of size n. It takes a constant $O(1)$ time to recurse, and an additional $O(n)$ time on each level to find the smallest value. Since it has a depth of n, and a runtime of $O(n)$ on each level, given an infinite amount of processors, it has a total runtime of $O(n^2)$.

5 Question 5

The metric b for this case would be the difference between the number of tokens in the system, and the actual number of tokens that's supposed to be on the system (in this case 1). The number of nodes in the system is N.

5.1 Stable State

In stable state, as there is only 1 token in the system, the $metric == 0$.

5.2 Proof of closure [When it's at stable state, state will be preserved]

When the $metric == 0$, there is only 1 token in the entire system. Every time the machine that owns the token fires, it loses the token, and ONLY it's successor will recieve the token, and no one else will recieve it. That one and only token is then passed around the ring.

5.3 Proof of convergence [It will ultimately result in stable state]

5.3.1 After N firings at the boss node, it will definitely reach stable state

Each time the boss node fires, the predecessor of the boss node would have to have the same value as the boss node before the boss node can fire. So, if the boss node fires x times, the predecessor of the boss node would have gone through x changes in value. Other than in the first iteration, where the value is generated by the node, these values would have to be copied from the other nodes in the ring. Since there is N nodes, x has a upper limit of N.

5.3.2 The boss node will eventually get the token

Assume first that the boss node does not have the token, and that the token is somewhere in the token ring. The node with the token is bound to fire sooner or later, and move down the ring, eventually reaching the boss node.

5.3.3 There is no point where it will have zero or less tokens

Firstly, having a negative amount of tokens is a mathematical impossibility, simply because there is no way to represent a negative token in this system.

Assume that the values of all the nodes on the system is identical. In this case, the boss node would then have the token.

Now, we change the value of 1 node on the system. Unless we change the predecessor of the boss node to a value that's not the same as the value of the boss node, it becomes 2 tokens. If we continue changing the values of the nodes, we will always have a minimum of one token, and sometimes even more,

5.3.4 Conclusion

Since the boss node will eventually get the token, and after n firings, the system will reach stable state, and under no circumstances can the number of tokens drop below 1, any action by any node will eventually move the system closer to stability, and not the other way round.