Theory Subjects – 19 January 2016

Work Time: 2 hours and 50min
Please implement one problem in Java and the other problem in C#.
You are free to choose which problem will be implemented in Java and which problem will be implemented in C#.
If a problem implementation does not compile or does not run you will get 0 points for that problem (that means no default points)!!!
If for one problem you have only a text interface to display the program execution you are penalized with 1 point for that problem. The GUI to input the ToyLanguage programs is not necessary, you can hard code the input programs in your implementations.

1. **(0.5p by default). Problem 1:Implement lock mechanism in ToyLanguage.**
   a. (0.5p). Inside PrgState, define a new global table (global means it is similar to Heap and Out tables and it is shared among different threads), LockTable that maps integer to integer. LockTable must be supported by all previous statements. It must be implemented in the same manner as Heap, namely an interface and the class which implements the interface.
   b. (0.75p). Define a new statement
   newLock(var)
   which creates a new lock into the LockTable. The statement evaluation rule is as follows:
   Stack1={newLock(var)| Stmt2|...}
   SymTable1
   Out1
   Heap1
   LockTable1
   ==>
   Stack2={Stmt2|...}
   Out2=Out1
   Heap1
   LockTable2 = LockTable1 synchronizedUnion {newfreelocation ->-1}
   if var exists in SymTable1 *then*
   SymTable2 = update(SymTable1,var, newfreelocation)
   else SymTable2 = add(SymTable1,var, newfreelocation)
   Note that you must use the lock mechanisms of the host language (Java/C#) over the LockTable in order to add a new lock to the table.
   c. (0.75p). Define the new statement
   lock(var)
   where var represents a variable from SymTable which is mapped to an index into the LockTable. Its execution on the ExeStack is the following:
   - pop the statement
   - foundIndex=lookup(SymTable,var). If var is not in SymTable print an error message and terminate the execution.

- *if* foundIndex is not an index in the LockTable *then*
  print an error message and terminate the execution
  elseif LockTable[foundIndex]==-1 *then*
  LockTable[foundIndex]=Identifier of the PrgState
  else push back the lock statement(that means other PrgState holds the lock)

Note that the lookup and the update of the LockTable must be an atomic operation, that means they cannot be interrupted by the execution of the other PrgStates. Therefore you must use the lock mechanisms of the host language (Java/C#) over the LockTable in order to read and write the values of the LockTable entrances .

d. (0.75p) Define the new statement:

unlock(var)

where var represents a variable from SymTable which is mapped to an index into the LockTable. Its execution on the ExeStack is the following:

- pop the statement
- foundIndex=lookup(SymTable,var). If var is not in SymTable print an error message and terminate the execution.
- *if* foundIndex is not an index in the LockTable *then*
  do nothing
  elseif LockTable[foundIndex]== Identifier of the PrgState *then*
  LockTable[foundIndex]= -1
  else do nothing

Note that the lookup and the update of the LockTable must be an atomic operation, that means they cannot be interrupted by the execution of the other PrgStates. Therefore you must use the lock mechanisms of the host language (Java/C#) over the LockTable in order to read and write the values of the LockTable entrances .

e. (0.5p) Extend your GUI to suport step-by-step execution of the new added features. It is not necesary to have a GUI to input the ToyLanguage programs, therefore please hard code the examples to be run by your implementation.

f. (1.25p). Show the step-by-step execution of the following program. At each step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a text readable log file.

The following program must be hard coded in your implementation.

```
new(v1,20);new(v2,30);newLock(x);
fork(
  fork(
    lock(x);wh(v1,rh(v1)-1);unlock(x)
  );
  lock(x);wh(v1,rh(v1)+1);unlock(x)
);
fork(
  fork(wh(v2,rh(v2)+1));
 wh(v2,rh(v2)+1);unlock(x)
```

);
skip;skip;skip;skip;skip;skip;skip;skip;skip
print (rh(v1));
print(rh(v2))
The final Out should be either {20,32} or {20,31}

2. **(0.5p by default) Implement For statement in Toy Language.**
   a. (2.25p). Define the new statement:
   for(v=exp1;v<exp2;v=exp3) stmt
   Its execution on the ExeStack is the following:
   - pop the statement
   - create the following statement: v=exp1;(while(v<exp2) stmt;v=exp3)
   - push the new statement on the stack

   b. (1p). Extend your GUI to suport step-by-step execution of the new added features. It is not necesary to have a GUI to input the ToyLanguage programs, therefore please hard code the examples to be run by your implementation.
   c. (1.25p). Show the step-by-step execution of the following program. At each step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a text readable log file.
   The following program must be hard coded in your implementation:
   v=20;
   (for(v=0;v<3;v=v+1) fork(print(v);v=v+1));
   print(v*10)
   The final Out should be {0,1,2,30}