Work Time: 2hours and 30 minutes

If a problem implementation does not compile or does not run you will get 0 points for that problem (that means no default points)!!!

1. (0.5p by default). Problem 1: Implement For statement in Toy Language.

a. (2.75p). 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:

int v; v=exp1;(while(v<exp2) stmt;v=exp3)

- push the new statement on the stack

The typecheck method of for statement verifies if exp1, exp2, and exp3 have the type int.

b. (with a working GUI 1.75p, with a working text UI 0.25p). 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 readable log text file. The following program must be hard coded in your implementation:

Ref int a; new(a,20);

(for(v=0;v<3;v=v+1) fork(print(v);v=v*rh(a)));

print(rh(a))

The final Out should be {0,1,2,20}

2. (0.5p by default). Problem 2:Implement lock mechanism in ToyLanguage.

a. (0.5p). Inside PrgState, define a new global table (global means it is similar to Heap, FileTable 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 a class which implements the interface. *Note that the lookup and the update of the LockTable must be atomic operations, 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 over the LockTable in order to read and write the values of the LockTable entrances.*

b. (0.5p). 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
FileTable1
LockTable1
==>
Stack2={Stmt2|...}
Out2=Out1
Heap2=Heap1
FileTable2=FileTable1
LockTable2 = LockTable1 synchronizedUnion {newfreelocation ->-1}
if var exists in SymTable1 and has the type int *then*
SymTable2 = update(SymTable1,var, newfreelocation)
else print an error and stop the execution.

Note that you must use the lock mechanisms of the host language Java over the LockTable in order to add a new lock to the table.

c. (0.5p). Define the new statement

lock(var)

where var represents an int 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 or has not int type then 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 over the LockTable in order to read and write the values of the LockTable entrances.

d. (0.5p) Define the new statement:

unlock(var)

where var represents an int 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 or has not an int type then 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]== 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 over the LockTable in order to read and write the values of the LockTable entrances.

e.(0.75p). Implement the method typecheck for the statement newLock(var) to verify if var has the type int. Implement the method typecheck for the statement lock(var) to verify if var has the type int. Implement the method typecheck for the statement unlock(var) to verify if var has the type int.

f. (1p) Extend your GUI to suport step-by-step execution of the new added features. To represent the LockTable please use a TableView with two columns location and value.

g. (with a working GUI 0.75p, with a working text UI 0.25p). 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 readable log text file. The following program must be hard coded in your implementation.

```
Ref int v1; Ref int v2; int x;  int q;
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)*10);unlock(x)
);newLock(q);
fork(
  fork(lock(q);wh(v2,rh(v2)+5);unlock(q));
 lock(q);wh(v2,rh(v2)*10);unlock(q)
);
nop;nop;nop;nop;
lock(x); print(rh(v1)); unlock(x);
```

lock(q); print(rh(v2)); unlock(q);
The final Out should be {190 or 199,350 or 305}