Concurrent Programming

Exercise Booklet 3: Atomic Actions

1. Consider the following code fragment (where the variable `x` is initialized with zero):

```
thread T1: {          thread T2: {          thread T3: {
  while(enter());       while(enter());       while(enter());
  x = x + 1;            x = x + 2;            x = x + 3;
  exit();               exit();               exit();
}                     }                     }
```

   Answer the following questions assuming that assignment is atomic:

   a) Given the trivial implementations below of the functions `enter` and `exit` determine all possible values of the variable `x` at the end of the execution of the threads.

```
enter() {             exit() { }
  return false;
}
```

   b) Consider now the implementation of the functions `enter` and `exit` that use the global boolean variable `occupied` (initialized with `false`) and indicate if, by means of their use, we solve the MEP. If not, indicate what properties fail and a trace justifying your answer.

```
enter() {             exit() {
  if (occupied)         occupied = true;
    return true;      }
  occupied = true;
  return false;
}
```

   c) Analyze the problem considering that the functions `enter` and `exit` are atomic. Does this solve the MEP? Justify your answer.

2. Consider the atomic operation fetch-and-add defined as follows:

```
atomic int fetch-and-add(ref, x) {
  local = ref.shared;
  ref.shared = ref.shared + x;
  return local;
}
```

   and the following algorithm to solve the MEP with shared variables `ticket` and `turn`, both initialized with 0.

```
  // non-critical section
  int myTurn = fetch-and-add(ticket, 1);
  while (turn != myTurn) {}
  // critical section
  fetch-and-add(ticket, -1);
  // non-critical section
```

   Indicate whether the algorithm solves the MEP. Justify your answer.

3. Consider the following operation

```
obtainFlag(mine, other) {
  flags[mine] = !flags[other];
}
```

The following algorithm is proposed to solve the MEP between two processes that share and array.

```
global boolean[] flags = {false, false};

thread P: {
  while (!flags[0])
    obtainFlag(0,1);
  // critical section
  flags[0] = false;
}

thread Q: {
  while (!flags[1])
    obtainFlag(1,0);
  // critical section
  flag[1] = false;
}
```

Answer the following questions.

a) Assume that the operation `obtainFlag` is not atomic. Does the proposed algorithm solve the MEP? Justify your answer. Hint: recall that the only atomic operations are assignment of scalar values.

b) If `obtainFlag` is atomic, is the MEP solved? Justify your answer.

4. Consider the following code fragment:

```
global int ticket = 0;
global int turn = 0;


thread T1: {
  // non-critical section
  int myTurn = getTurn();
  while (myTurn != turn);
  // critical section
  releaseTurn();
  // critical section
}

thread T2: {
  // non-critical section
  int myTurn = getTurn();
  while (myTurn != turn);
  // critical section
  releaseTurn();
  // critical section
}
```

Given the following implementations of the functions `getTurn` and `releaseTurn`.

```
getTurn() {
  return ticket++;
}

releaseTurn() {
  turn++;
}
```

Answer:

a) Determine if using functions `getTurn` and `releaseTurn` you solve the MEP. Justify your answer.

b) Now assume that the functions `getTurn` and `releaseTurn` are atomic.

c) If this solution is executed in an environment where integers are represented using one byte (8 bits), that is, the largest number without sign that is representable is 512. Does this affect your previous answer?