CS_213 System Specification

(Attempt 2 questions out of 3)

Question 1.

(a.) i) Write a module `QUEUE` which specifies queues containing elements of a sort `Elt`. The constructors are:

`<> : -> Queue`,	representing the empty queue,
`<_> : Elt -> Queue`,	representing a queue with one element,
`_#_ : Queue Queue -> Queue`,	concatenating two queues.

Your module should further contain an operator `isEmpty`, which tests whether or not a queue is empty, and operators `first` and `last`, which return the first element, and last element respectively.

ii) Extend your module by an operator `reverse` which computes the reverse of a queue.

[9 marks]

(b.) The following is a Maude module specifying binary trees.

```
fmod BIN-TREE is
    sorts BinTree Elt .

    op null : -> BinTree [ctor] .
    op [_<-_->_] : BinTree Elt BinTree -> BinTree [ctor] .

    ops rightTree leftTree : BinTree -> BinTree .
    op getData : BinTree -> Elt .
    op isLeaf : BinTree -> Bool .

    vars L R : BinTree .
    var D : Elt .

    eq rightTree([L <- D -> R]) = R .
    eq leftTree([L <- D -> R]) = L .
    eq getData([L <- D -> R]) = D .
    ceq isLeaf([L <- D -> R]) = true if (L == null) and (R == null) .
endfm
```

i) Extend the module `BIN-TREE` by an operator specifying the height of a tree.

ii) Introduce a sub-sort `BalTree` consisting of trees where, in each node, the difference between the height of the left subtree and the height of the right subtree is at most 1.

[8 marks]

(c.) i) Explain what is meant by the term *formal methods*.

ii) What does it mean for an implementation of a specification to be *correct*? Give an explanation using the correctness model given in the lecture.

[8 marks]

Question 2.

(a.) i) Specify a stream transformer that takes two boolean streams as input and returns a stream which, at each time `T`, is `true` if at least one of the input streams is `true`.

ii) Consider the following Maude specification:

```
fmod ST is protecting INT .
   sort BoolStr IntStr .
   var B  : BoolStr .
   var I  : IntStr .
   var T : Int .

   op _(_) : BoolStr Int -> Bool [prec 1] .
   op _(_) : IntStr Int -> Int [prec 1] .
   op st : BoolStr IntStr -> IntStr .

   eq  st(B,I)(0) = 0 .
   ceq st(B,I)(T) = I(T - 1) + st(B,I)(T - 1)
                    if B(T - 1) == true .
   eq st(B,I)(T) = st(B,I)(T - 1) [owise] .
 endfm
```

Briefly explain what `ST` is doing.

iii) Write a module `ST-RUN` that executes the stream transformer with some sample input of your own choice. State the output value of the stream transformer at time 2 when using your sample input data.

[10 marks]

b.) A microprocessor specification is typically built on a module defining the binary numbers such as the following.

```
fmod BINARY is
   protecting INT .
   sorts Bit Bits .
   subsort Bit < Bits .

   ops 0 1 : -> Bit .
   op __ : Bits Bits -> Bits [assoc prec 1 gather (e E)] .
   vars S T : Bits .
   vars B C : Bit .
endfm
```

i) Extend the module `BINARY` by an operator `isZero` testing whether or not a given binary number is zero (i.e., is of form `0 0...0`).
(You may use further operators introduced in the course, however you should briefly explain what they are doing.)

ii) Extend the module `BINARY` by an operator `_<<_ : Bits Bits -> Bits` which shifts the bits in its first argument by a required number of positions, given by the second argument, to the left, and fills with '0's from the right. (Example: `red 1 1 1 0 1 << 1 0` yields `1 0 1 0 0`).

[7 marks]

(c.) i) Explain the difference between local and global confluence in a term rewriting system.

ii) Let `A,B,C,D` be constants and consider the reduction system given by the following set of rules:

```
A = B
B = A
A = C
B = D
```

Is this reduction system locally/globally confluent? Justify your answers.

iii) Newman's Lemma says that, in a reduction system, termination and local confluence implies global confluence. Is the example above (Question 2, c, ii) contradicting Newman's Lemma?

[8 marks]

Question 3.

Consider the following informal description of a simple register based microprocessor.

- a memory consisting of 16-bit words, memory addresses are 16-bits long,
- 16 16-bit registers,
- a 16-bit program counter

Instructions are 16 bits long: the first four bits are the opcode, the remaining bits are used for three register addresses. There are four instructions:

- **Add Ra, Rb, Rc:** Add register `Ra` to register `Rb` and write the result to register `Rc`. Increment the program counter by **1**. The opcode for this instruction is `0 0 0 0`.
- **SLL Ra, Rb, Rc:** Shift `Ra` by `Rb` positions to the left and write the result to `Rc`. Increment the program counter by **1**. The opcode is `0 0 0 1`.
- **ST Ra, Rb:** Store `Ra` in memory location `Rb`. Increment the program counter by **1**. The opcode is `0 0 1 0`.
- **JMP Ra, Rb, Rc:** If `Ra == 0` then store the current program counter, incremented by **1**, in `Rb` and set the program counter `PC` to `PC + Rc`; otherwise increment the program counter by **1**. The opcode is `0 0 1 1`.

You may assume the following are available: the module `BINARY` (+ and `<<` as operators), the sorts `Word` (16 bits) and `Opfield` (4 bits) with operators `opcode, rega, regb, regc : Word -> Opfield`, returning the according parts of an instruction, and appropriate constants, denoting **0** and **1**, of sort `Word`. Minor syntactic errors will not be penalised.

(a.) Formally specify the memory of the microprocessor.
What is the size of the memory? **[5 marks]**

(b.) Write a module `STATE` that specifies the state of the microprocessor. Your module should contain appropriate tupling and projection operators.
[4 marks]

(c.) Write a Maude module defining the behaviour of the microprocessor as an iterated map. Your module should contain the iterated map state function, as well as the next-state function.
[12 marks]

(d.) Write a module that executes your specification. Your sample run should first execute an addition, then jump to a subroutine, and finally write the result of the addition to the memory. **[4 marks]**