CS_213 System Specification

(Attempt 2 questions out of 3)

Question 1.

(a.) The following is a simple Maude module.

```
fmod BASIC-NAT is
        sort Nat .

        op 0 : -> Nat .
        op s : Nat -> Nat [iter] .
        op _+_ : Nat Nat -> Nat .

        vars N M : Nat .

        eq 0 + N = N .
        eq s(M) + N = s(M + N) .
endfm
```

i) Extend the module by an operator `_*_` defining multiplication on `Nat`.

ii) Explain how the following expression would be re-written in Maude, using your definition of multiplication.

```
s^2(0) * s(0) .
```

[6 marks]

(b.) The following is a (partial) Maude representation of a stack.

```
fmod STACK is
  sorts Stack Elt .

  op EmptyStack : -> Stack .
  op ErrorElt : -> Elt .
  op push : Stack Elt -> Stack .
  op top : Stack -> Elt .
  op pop : Stack -> Stack .

  var S : Stack .
  var E : Elt .
  eq top(EmptyStack) = ErrorElt .
  eq pop(EmptyStack) = EmptyStack .
endfm
```

i) The module `STACK` is not complete. Write down the missing equations that define the behavior of `top, pop` and `push`.

ii) The specification of `STACK` used an extra operator `ErrorElt` in order to describe the behavior of the operator `top`. This can be avoided by introducing a sub sort `NeStack` of non-empty stacks. Write a new module `BETTER-STACK` specifying a stack that is built on the sorts `Stack, Elt` and `NeStack`.

[6 marks]

(c.) i) Explain informally what it means for a term rewriting system to be terminating and confluent.

ii) The properties 'termination', 'local confluence' and 'global confluence' apply to abstract reduction systems as well. Here is an example of such an abstract reduction system on the set $\mathbb{N} \cup \{\texttt{true, false}\}$.

$n \rightarrow n{+}1$	(Each number reduces to its successor.)
$2n \rightarrow$ `true`	(Each even number reduces to `true`.)
$2n{+}1 \rightarrow$ `false`	(Each odd number reduces to `false`.)

Decide (and briefly explain) whether or not this reduction system is terminating, locally confluent, globally confluent?

[8 marks]

(d.) How would you specify a 64K memory consisting of bytes in Maude? You may assume the presence of the module `BINARY` as well as sorts `Byte` and `Word` (16 bits). Your specification should contain appropriate read and write operations.

Either give an informal explanation or write down a Maude module.

[5 marks]

Question 2.

(a.) The following is a (partial) module specifying the binary numbers.

```
1. fmod BINARY is
2.   protecting INT .
3.   sorts Bit Bits .
4.   subsort Bit < Bits .

5.   ops 0 1 : -> Bit .
6.   op __ : Bits Bits -> Bits [assoc prec 1 gather (e E)] .
7.   op |_| : Bits -> Int .
8.   vars S T : Bits .
9.   vars B C : Bit .
10.  eq | B | = 1 .
11.  eq | S B | = | S | + 1 .
12. endfm
```

i) Briefly, explain what each numbered line means.

ii) Extend the module by an operator Bits2Int that transforms a binary number into an integer. (You may assume the usual arithmetical operations on `Int` and you may also extend the module `BINARY` by further operations if needed.)

[10 marks]

(b.) Consider the following short Maude module:

```
fmod BOOLSTREAM is
   protecting INT .
   sort BoolStr .
   var T : Int .

   op _(_) : BoolStr Int -> Bool .
endfm
```

Briefly explain what is intended by the operator `_(_)`. Use it to define a stream that is constant `true`. **[2 marks]**

(c.) The following is an informal description of a stream transformer that has as inputs a data stream of integers and a control stream (i.e., a stream of booleans). The output is a stream of integers.

At time `T` the output of the stream transformer is the total number of 0-elements that have arrived on the data stream of integers, up to and including time `T`, such that when each 0 arrived the boolean control stream was `true`.

For example:

time	0	1	2	3	4	5
integer stream	1	0	3	0	4	0
control stream	`true`	`true`	`false`	`false`	`true`	`true`

At time 5, the outputstream of the stream transformer has value 2. The 0 at time 3 is not counted because the boolean stream was `false`.

Write a Maude module formally defining the behavior of the stream transformer.

[8 marks]

(d.) Formal methods usually involve a specification and an implementation. We use a data abstraction map that maps the implementation state to a specification state and a time abstraction map (called a retiming) that maps the implementation clock time to the specification clock time. Which properties must be fulfiled by these maps (with respect to the correctness of the implementation)?

[5 marks]

Question 3.

Consider a simple microprocessor with a memory of 8K 16-bit words (i.e. memory addresses are 13 bits long), an accumulator (16 bits) and a program counter (13 bits). The memory is shared by programs and data. Instructions are 16 bits long: the first three bits are the opcode and the remaining bits serve as an address. There are four instructions:

- **Add L:** Add the accumulator to memory location **L** and write the result to the accumulator. Increment the program counter. The opcode is `0 0 0`.
- **Store L:** Write the accumulator to memory location **L**. Set the accumulator to the word **0**. Increment the program counter. The opcode is `0 0 1`.
- **Eq L:** If the accumulator is equal to memory location **L**, then write the word **0** to the accumulator; otherwise set the accumulator to the word **-1**. Increment the program counter. The opcode is `0 1 0`.
- **Jump L:** If the accumulator is the **0**-Word, then set the program counter to the least significant 13 bits of memory location **L**, otherwise increment the program counter. The opcode is `0 1 1`.

You may assume the following are available:

- The module `BINARY`, containing sorts `Bits` and `Bit` as well as the usual range of logical and arithmetic operations.
- A module `MEM` representing the memory, including read and write operations.
- Sorts `Word`(16 bits), `Address`(13 bits), `Opfield` (3 bits), defined in terms of `Bits` and `Bit`, as well as the operation `_+_` for the sorts `Word` and `Address`.
- The built-in module `INT`.

(a.) Write a module `INSTRUCTION-FORMAT` that defines the precise instruction format (i.e., determines opcode and address of a given instruction) and a second module `STATE` defining the state of the microprocessor. Your module `STATE` should contain appropriate tupling and projection operators. **[7 marks]**

(b.) Write a Maude module defining the behavior of the microprocessor as an iterated map. Your module should contain the iterated map state function, as well as the next-state function.

Explain briefly what the iterated map state function does. **[12 marks]**

(c.) Write a sample program that adds two numbers stored in the memory (for instance, in the 10th and 11th address field) and writes the result back to the memory (e.g., to the 12th address field). **[3 marks]**

(d.) How would you modify the jump instruction (in your specification) in order to be able to conveniently jump to a subroutine (and return from it)? Use it in a sample program. **[3 marks]**