CS_213 System Specification – May/June 2006

(Attempt 2 questions out of 3)

Question 1.

(a.) Consider the following specification of lists consisting of the sorts `Int, NeList` (for non-empty lists) and `List`. The constructors are `nil` and `__` (for concatenating two lists or a list and one integer element). `1 nil` , `1 2 3 nil` are examples for lists.

The tag `id: nil` means that `nil` is an 'identity' element. That is, for instance, `1 2 3` is the same as `1 2 3 nil` or `nil 1 2 3` .

```
fmod LIST is
   protecting INT .
   sorts List NeList .
   subsorts Int < NeList < List .
   op nil : -> List [ctor] .
   op __ : List List -> List [ctor id: nil] .
   op __ : List NeList -> List [ctor id: nil] .
   op __ : NeList List -> List [ctor id: nil] .
endfm
```

i) Explain in general the meaning of the tags `assoc` and `comm` and discuss whether they should be added to the specification of `LIST`.

ii) Extend the module LIST by operators

`head, tail, length, reverse, isEmpty`, and `isOrdered`

as well as their defining equations (with the usual meaning). `isOrdered(L)` should be `true` if (and only if) all elements in the list `L` are strictly ascending.

[12 marks]

(b.) A palindrome is a list that reads the same from the front and from the back. For instance, `4 3 4` and `2 4 3 3 4 2` are palindromes, but `2 4 5 5 2` is not.

Show two different ways of specifying palindromes, first by extending the module LIST by an appropriate operator which tests whether or not a given list is a palindrome, second by defining an appropriate subsort with a (non-conditional) membership axiom.

[6 marks]

(c.) Let `A,B` be constants and consider the reduction system given by the following set of rules:

```
A = B
AB = AAB
```

Decide whether the given reduction system is

- terminating,
- locally confluent,
- globally confluent.

Discuss whether or not each term has a normal form.

Justify all your answers.

[7 marks]

Question 2.

(a.) What do we mean by formal methods? Give reasons why algebras are particularly useful for writing specifications. What is the role of initial algebras? [Two marks for each of these three questions.]

[6 marks]

(b.) A multiplexor has as input three streams of booleans and produces an output stream of booleans. At each time T the value on the output stream is either the current value on the first or on the second stream, depending on the value of the third input stream, the control stream. If a `true` element arrives on the control stream, the current element of the first stream can 'pass', otherwise, that is, if a `false` element arrives, then the current element on the second stream becomes the new value on the output stream.

Complete the following module which specifies the behaviour of the multiplexor.

```
fmod MUX is
  protecting Int .
  sort Boolstr .
  op mux :
  op _(_) :
  vars X Y C :
  var T :
  ...
```

[5 marks]

(c.) Consider the following convolving function which takes as input an integer stream and produces as output a new stream which depends on the input stream. At each time `T` the output stream shows the sum of the last three elements which have arrived on the input stream up to (and including) time `T`.

Write a module that specifies the behaviour of the convolver.

[5 marks]

(d.) The following is an implementation of the convolver specification given in part (b). It uses an inner product step processor.

```
fmod IPS is
    protecting INT .

    sort IntStr .
    sort IPSTuple .
    sort IPSstr .

    op _(_) : IPSstr Int -> IPSTuple [prec 1] .
    op _(_) : IntStr Int -> Int [prec 1] .
    op U : -> Int .
```

(Question continues on the next page)

```
    op _,_,_ : Int Int Int -> IPSTuple .
    op _,_,_ : IntStr IntStr IntStr -> IPSstr .
    op ips : IPSTuple -> IPSTuple .

    vars A B C : Int .
    var T : Int .

    ceq ips(A,B,C) = (U,U,U) if (A == U) or (B == U) or (C == U) .
    eq ips(A,B,C) = (A, B, A * B + C) [owise] .
endfm

fmod CONV-IMP is
    protecting IPS .

    sort ConvState .

    op _,_,_ : IPSTuple IPSTuple IPSTuple -> ConvState .
    op convImp : Int IntStr -> ConvState .
    ops conv1 conv2 conv3 : Int IntStr -> IPSTuple .

    op C : IPSTuple -> Int .

    var X : IntStr .
    var T : Int .
    vars I J K : Int .

    eq C(I,J,K) = K .

    eq convImp(T,X) = (conv1(T,X), conv2(T,X), conv3(T,X)) .

    eq conv1(T,X) = ips(1,X(T - 2),0) .
    eq conv2(T,X) = ips(1,X(T - 1), C(conv1(T - 1, X))) .
    eq conv3(T,X) = ips(1,X(T), C(conv2(T - 1, X))) .
endfm
```

Explain, for example by drawing a diagram, how this implementation works. Define a module CONVIMP-RUN containing a sample stream which can be used to test the implementation.

[4 marks]

(e.) Explain the concepts of a data abstraction map and retiming. Define the data abstraction map for the convolver example given in (c) and (d).

[5 marks]

Question 3.

In this question, you should specify a microprocessor which consists of

- a memory containing 16-bit words (where memory addresses are 12 bits long),
- an accumulator (16 bits), and
- a program counter (12 bits).

Instructions are 16 bits long: the first four bits are the opcode and the remaining bits form an address. The first three instructions are

Add L: Add the accumulator to memory location **L** (that means: add the content of the accumulator to the content of the memory field with address **L**) and write the result to the accumulator. Increment the program counter. The opcode for this instruction is `0 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 0 1`.

SkipIfZero L: If the memory location **L** is **0**, then increment the program counter by **2**; otherwise increment the program counter by **1**. The opcode is `0 0 1 0`.

You may assume that the following are available: the module `INT`, and the module `BINARY` (containing sorts `Bits` and `Bit` as well as the usual range of logical and arithmetic operations). Further you have sorts `Word`(16 bits) and `Address`(12 bits), defined in terms of `Bits` and `Bit`, as well as the operation `_+_` and appropriate constants (denoted by **0, 1** and **2**) for the sorts `Word` and `Address`. Finally, you also have a module `MEM` specifying the memory of the microprocessor.

(a.) Define a new sort `opfield` which may be used to specify the opcode of an instruction. Specify operators `opcode` and `opfield` which determine the opcode and the address part of a given instruction. (Note you do not have to define the full module.)

[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.

[9 marks]

(d.) The current instruction set of the microprocessor is inadequate because there are no flow control instructions. Informally, define and explain an indirect jump and a jump to a subroutine analoguously to the one in the PDP8. (Remember the PDP8 stores the current PC at the first memory address of the subroutine.) Finally, add a formal specification of these two jump instructions. **[7 marks]**