CS-M43
Algebraic Specification of Software and Hardware
(Answer 2 questions out of 3)

Question 1

(A) Give a brief explanation for the following:

(a) Formal Method

(b) Specification

[4 Marks]

(B) Consider the data type 'stack of natural numbers':

There is a sort `Stack`. As an initial value, the stack can be `empty`. A test `isEmpty` checks if a stack contains no elements. The operation `push` changes a stack by putting a natural number on it. The operation `pop` removes the top element of an non-empty stack. The operation `top` returns the top element of an non-empty stack without changing its contents.

(a) Write a CASL specification of the signature of this stack of natural numbers.

You may use the following specification fragment in order to import the specification of natural numbers from the CASL standard library:

```
spec Stack =
  Nat
then
  ...
```

A specification fragment of `Nat` can be found at the end of this paper.

[4 Marks]

(b) Complete your above CASL specification by adding axioms in three different ways:

(i) use only first order logic.

(ii) use first order logic and the CASL `generated` construct.

(iii) use first order logic and the CASL `free` construct.

[9 Marks]

...

(c) Define three Σ algebras A, B, C, where

(i) A is the intended model,

(ii) B has confusion in the carrier set of `Stack`, and

(iii) C includes junk in the carrier set of `Stack`.

The carrier sets $A(Nat) = B(Nat) = C(Nat) = \mathbf{N}$ shall be the natural numbers. Don't define the interpretations of the signature elements of the CASL specification `Nat`.

State which of your algebras A, B, C is a model of which of your specifications.

[8 Marks]

Question 2

(A) Give experiments that check if a specification is

(a) precise

(b) consistent

(c) complete

[6 Marks]

(B) Specify a multiplexer in CASL.

A multiplexer is a hardware element, that merges two incoming bit streams A and B into one outgoing bit stream O, where the choice between the incoming bitstreams is trigged by a third incoming bit stream T: whenever T is true, O is set to the current value of A, whenever T is false, O is set to the current value of B.

You may use the following specification fragment in order to import the specifications of natural numbers and the specification of booleans from the CASL standard library:

```
spec Multiplexer =
  Nat
then
  Boolean
then
  ...
```

Specification fragments of `Nat` and `Boolean` can be found at the end of this paper.

[9 Marks]

...

(C) Specify the Quicksort-Algorithm for natural numbers in CASL.

A short reminder of Quicksort: Quicksort is based on the divide-and-conquer paradigm. Here is the three step divide-and-conquer process for sorting a list L:

Divide: An element x of the list L is chosen. Then the list L is partitioned (rearranged) into two new lists L_1 and L_2 such that each element of L_1 is less than or equal x, and such that each element of L_2 is greater than x (i.e. x belongs to L_1).

Conquer: The two lists L_1 and L_2 are sorted by recursive calls to quicksort.

Combine: The result is obtained by appending the sorted version of the list L_1 and the sorted version of the list L_2.

You may use the following specification fragment in order to import the specifications of lists from the CASL standard library:

```
spec QuickSort =
  List[Nat fit sort Elem |-> Nat] with sort List[Nat] |-> List
then
  ...
```

Specification fragments of `Nat` and `List` can be found at the end of this paper.

[10 Marks]

Question 3

(A) Give the definitions of

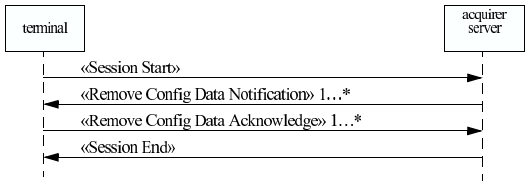
(a) a signature Σ.

(b) a Σ-algebra A.

[8 Marks]

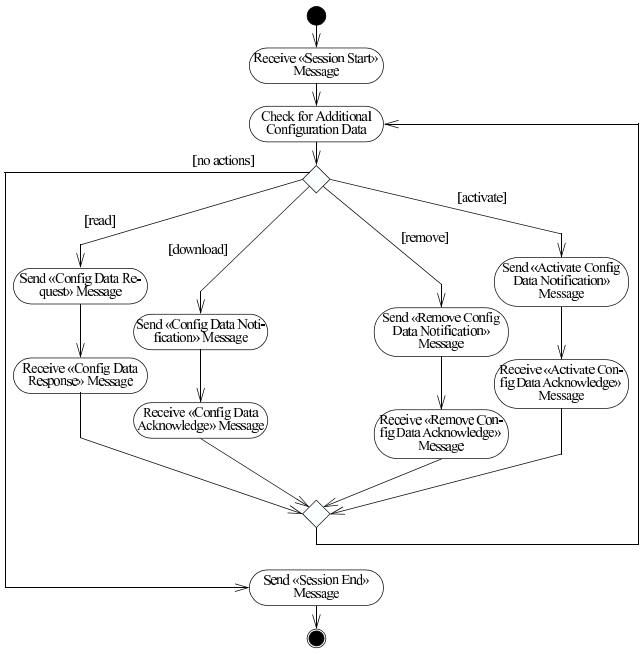
(B) Consider the following two UML like pictorial descriptions of a message interchange between two components of a system named 'Terminal' and 'Acquirer'.

Picture 1:



The text inbetween `<< ...>>` names a message type, the notation "1...*" says that there are one or more instances allowed.

Picture 2:



Picture 1 describes the system view of what happens if data shall be removed. Picture 2 shows the possible behaviour of the component named 'Acquirer', including the case that data shall be removed (see the path labeled `[remove]`).

...

Describe the inconsistency between these two pictures.

Remark This example stems from an actual specification document which is currently sold as a new international standard.

[7 Marks]

(C) Consider the arithmetic-logic unit (ALU) of a processor that consists of

- two registers R_1 and R_2,
- an accumulator A, and
- a program counter PC.

For simplicity, we assume that all these registers can deal with natural numbers.

The ALU can perform three instructions:

(a) `add`: this operation adds the contents of the registers R_1 and R_2 to the content of the accumulator A; then it adds 1 to the program counter.

(b) `mult`: this operation multiplies the contents of the registers R_1 and R_2 and adds them to the content of the accumulator A; then it adds 1 to the program counter.

(c) `jmpZ`: this operation sets the program counter PC to the value found in register $R1$ if the current value of the accumulator A is zero.

The ALU works on a sequence of instructions specified as follows:

```
spec Instruction =
  Nat
then
  sort Instruction
  ops add, mult, jmpZ: Instruction

  sort InstructionSequence
  op eval: InstructionSequence * Nat -> Instruction
end
```

Specify in CASL

(a) the `State` of the ALU,

(b) a `next` operation that, given a `State` and an `Instruction`, returns the new state of the ALU, and

(c) an operation `alu` that, given a `State`, a clock signal taking values in the natural numbers, and an `InstructionSequence`, returns the next state of the ALU. Use the operation `next` within the axioms.

[10 Marks]

Useful specifications from the CASL standard library

```
spec Nat =
  free type Nat ::= 0 | suc(pre:? Nat)
  preds   __ <= __, __ >= __,
          __ <  __, __ > __:   Nat * Nat;
          even, odd:           Nat
  ops   __! :  Nat -> Nat;
        __ + __, __ * __, __ ^ __,
        min, max, __ -!__ :         Nat * Nat ->  Nat;
        __ -?__, __ /? __,
        __ div __, __ mod __:       Nat * Nat ->? Nat;
  ...
end

spec Boolean =
  free type Boolean ::= True | False
  ops   Not__ :           Boolean -> Boolean;
        __And__, __Or__ : Boolean * Boolean -> Boolean
  forall x,y:Boolean
  . Not(False) = True                %(Not_False)%
  . Not(True) = False                %(Not_True)%
  . False And False = False          %(And_def1)%
  . False And True = False           %(And_def2)%
  . True  And False = False          %(And_def3)%
  . True  And True = True            %(And_def4)%
  . x Or y = Not(Not(x) And Not(y))  %(Or_def)%
end

spec GenerateList [sort Elem] =
  free type List[Elem] ::=  [] |  __ :: __ (first:? Elem; rest:? List[Elem])
end
spec List [sort Elem] given Nat = GenerateList[sort Elem] then
     pred  isEmpty: List[Elem];
           __eps__: Elem * List[Elem]
     ops   __ + __ : List[Elem] * Elem -> List[Elem];
           first, last: List[Elem] ->? Elem;
           front, rest: List[Elem] ->? List[Elem];
           #__: List[Elem] -> Nat;
           __ ++ __ : List[Elem] * List[Elem] -> List[Elem];
           reverse:     List[Elem] -> List[Elem];
           __ ! __:     List[Elem] * Nat ->? Elem;
           take,drop:   Nat * List[Elem] ->? List[Elem];
           freq: List[Elem] * Elem -> Nat
     ...
end
```