CSM205 The Construction of Formal Specifications of Programs

Answer any 3 questions

Important:

- Unless otherwise specified all specifications are to be formally expressed using Z.
- Do not forget to introduce any new types you will need in your specification.
- Give a brief informal description of any mathematics written.

1. A machine which allows processes to run concurrently with a single writer, multiple readers variable store is to be specified.

Variables and processes are uniquely identifiable:

\[ \text{PROCESS}\_\text{ID}, \text{ VARIABLE} \]

- At any time the machine will be controlling a subset of the variables and a running set of processes.
- Variables store a value of integer type, which is stored in the machine state.
- Each variable has a related process id which is the only process which can write to that variable. This will be referred to as the writer of the variable.
- Each process has an associated non-negative integer which represents its priority (the smaller the number, the higher the priority). No two processes may have the same priority.
- For a process to read a variable, the reader must have priority at least as high as that of the writer of the variable.
- Initially, only one process, the root process will be running, with priority 0.
(a) Use \( Z \) to specify the state of the machine.

(b) Specify the machine initialisation.

(c) Prove the initialisation theorem.

(d) Specify the operation by which, a particular process \( P \) may launch a new process \( Q \). The machine will automatically set the priority of \( Q \) to a value lower than that of \( P \).

(e) Specify the operation by which a process \( P \) demands to become the writer of a variable \( v \) which must not yet be in use.

(f) Specify the operation which, given a process \( P \) and variable \( v \), outputs the value of \( v \). Note that \( P \)'s priority must be at least as high as that of the writer of variable \( v \).

2. A rectangle on the xy-plane, with sides parallel to the axes can be defined by specifying four values: the \((x, y)\) coordinates of the lower left hand corner of the rectangle, and the length and width of the rectangle.

(a) Define the type \( \text{RECTANGLE} \), which contains all rectangles described in the above format.

(b) Define the relation \( \text{in} \) such that for any two rectangles \( r_1 \) and \( r_2 \), \( r_1 \text{ in } r_2 \) if and only if rectangle \( r_1 \) lies completely within rectangle \( r_2 \).

(c) Similarly, define the relation \( \text{overlap} \) such that for any two rectangles \( r_1 \) and \( r_2 \), \( r_1 \text{ overlap } r_2 \) if and only if rectangle \( r_1 \) has an area in common with rectangle \( r_2 \).

A simple designer’s drawing package allows the user to plan the layout of furniture in a house using rectangles. The package allows drawing a layout of rooms and objects such that:

- No two rooms may overlap.
- An object must lie completely within a room.
- No two objects may overlap.

(d) Specify the state of the system.

(e) Define the operation \( \text{AddObject} \) which, given an object and room, adds the object inside the room.

(f) State the simplified precondition of \( \text{AddObject} \).

\textbf{Note:} Do not open up the definition of \( \text{in} \) and \( \text{overlap} \) when stating the precondition.
3. Tic-tac-toe is played on a $3 \times 3$ grid which starts off empty. Player 1 plays by placing the symbol $\times$ in any empty square. Similarly player 2 places the symbol $\circ$ in any empty square. Players take turns placing their symbols until the board contains 3 similar symbols in a row, column or diagonal. If the aligned symbols are crosses, player 1 has won, otherwise player 2 has won. If the board is filled completely with neither side winning, the result is a draw.

(a) Define the type $BOARD$ which contains all possible board configurations (the layout of the symbols on the grid).

(b) State the type of the relation $wins$ between players and boards such that $p \ wins \ b$ if and only if player $p$ has three symbols of his/her type in a straight line in board position $b$. For the rest of the question you may assume that this relation has been defined.

(c) Define the state of the game, which must include the board configuration and whose turn it is to play.

(d) Define the operation which changes the state of a game by performing any move for the player whose turn it is to play. Note that the move is not given as input but is chosen non-deterministically by the operation.

(e) Define the similar operation which plays slightly better by preferring moves which lead to an immediate win. If no such moves are available, no restrictions are to be placed on the move to be taken (as in the previous question).

(f) Formally state (not prove) the conditions which must be satisfied for the operation given in (e) to be a refinement of that in (d). Informally discuss whether you believe it to be a refinement or not.

4. A simple processor has three registers $A$, $B$ and $C$.

$$REG ::= A \mid B \mid C$$

Each register can store one byte. There is a function which translates bytes to and from natural numbers:

$$[BYTE]$$

$$\begin{align*}
byte2num : & \ \ BYTE \rightarrow N \\
num2byte : & \ \ N \rightarrow BYTE \\
byte2num \circ num2byte : & \ \ N \rightarrow N
\end{align*}$$

$$byte2num \circ num2byte = Id_{BYTE}$$

The machine can execute a number of simple instructions:
\( \text{INST} ::= \text{NOP} \\
| \text{MOV}_{RV}\langle\langle \text{REG} \times \text{BYTE} \rangle \rangle \\
| \text{MOV}_{RR}\langle\langle \text{REG} \times \text{REG} \rangle \rangle \\
| \text{ADD}_{RV}\langle\langle \text{REG} \times \text{BYTE} \rangle \rangle \\
| \text{ADD}_{RR}\langle\langle \text{REG} \times \text{REG} \rangle \rangle \\
| \text{SWAP}\langle\langle \text{REG} \times \text{REG} \rangle \rangle \\
\)

- \text{NOP} does not change the state of the registers.
- \text{MOV}_{RV} moves the value (given as the second parameter) into the register (the first parameter). \text{MOV}_{RR} takes two registers and moves the value of the second register into the first.
- \text{ADD}_{RV} takes a register and a value and leaves the sum of the initial value of the register and the value in the specified register. \text{ADD}_{RR} takes two registers as parameters and stores the value of the sum of the registers in the first register.
- \text{SWAP} swaps the values of two registers.

Addition assumes that the result is a valid byte. Note that apart from the registers, the machine has no memory whatsoever.

(a) Specify the state of the machine.

(b) Formally specify operations \text{ExecNOP}, \text{ExecMOV}_{RR}, \text{ExecADD}_{RV} and \text{ExecSWAP} which specify the behaviour of the machine upon executing the instruction given in the operation name (the full instruction is also given as input to the operation).

(c) A program is a sequence of instructions. Specify the operation which is given a program as input and executes the instructions sequentially.

(d) A program is said to be idempotent if it does not modify the state of the machine. Using set comprehension and the previous schema, define the set of programs \text{Idemprograms} which are idempotent.

(e) A program \( P \) is said to be equivalent to another program \( Q \) if, whatever the initial state of the machine, executing \( P \) leaves the machine in the same state as it would be in after executing \( Q \).

(f) Formally express the conjecture that a program is idempotent if and only if it is equivalent to the program \text{NOP}.

5. (a) An edit field stores a string of (printable) characters which the user may edit by typing in symbols (in overwrite or insert mode), pressing delete and moving around in the field using the left and right cursors. No limit is placed on the length of the string which can be stored in an edit field.
Using Z, formalise the concept of an edit field. In particular, specify the initialisation of the edit field, changing mode, pressing delete and typing a (printable) character.

(b) A form is a collection of edit fields. Each field is identified by a field name and has an associated screen position at which it is displayed. At any point, only one field is in focus and editing commands apply only to that field. The edit mode (insert or overwrite) is common to all fields, but the cursor position is stored for each field. Thus imagine, for example, a user editing field 1 with the cursor at position 5 and in insert mode. The user then changes the focus to field 2, where she moves the cursor to position 10 and changes the mode to overwrite. If she then changes focus back to field 1 the cursor position is reset to 5 but the mode is left unchanged (overwrite).

Use promotion to specify forms. In particular specify what happens when the form is initialised, delete is pressed, mode is changed, focus is moved to the next field and when a (printable) character is typed.