SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES

Computer Science

F28FS2

Formal Specification Mock exam

Semester 2 201415

Sometime before 14 May 2015

Duration: As long as you like (actual exam: 2 hours)

ANSWER THREE QUESTIONS

Some words on using this mock paper

Every concept in this paper appears in the lecture notes and exercises. The difficulty level of a question is graduated from rather easy at the start, to rather hard towards the end.

The questions in this mock are harder, and often more open-ended, than what you will face in the exam. However, things seem more difficult during exams because of the stress.

If you understand and can do these questions, then you are certain to get a decent grade in the exam.

- You *must* attempt this entire paper *before* looking at the answers. Have you attempted the paper yet?

  ANSWER:
  If you see this text, you are looking at the version with model answers. Close this document and try to version without model answers, first.

- Then look up answers.
- Then do the paper again.
- Repeat until perfect.

Good luck.

1. (a) Explain in English the meanings of the following sets:

1. $\{x : \mathbb{N} \bullet x\}$. (1)

   ANSWER:
   Natural numbers

2. $\{x : \mathbb{N} \mid x \geq 1\}$. (1)

   ANSWER:
   Nonzero natural numbers.

3. $\{x : \mathbb{N} \bullet 2 * x\}$. (1)

   ANSWER:
   Even numbers

4. $\{X : \mathbb{P}\mathbb{N} \bullet \#X = 2\}$. (2)

   ANSWER:
   Set of unordered pairs of distinct natural numbers.

5. $\{x, y : \mathbb{N} \bullet \{x, y\}\}$. (2)

   ANSWER:
   Set of singletons or unordered pairs of natural numbers.

(b) Write the following sets in Z notation:

1. Numbers divisible by 3. (1)

   ANSWER:
   $\{x : \mathbb{N} \bullet 3 * x\}$.

2. Numbers that are the sum of two distinct primes. Here and henceforth you may assume a set *prime* : $\mathbb{P}\mathbb{N}$ of prime numbers. (2)

   ANSWER:
   $\{x, y : \mathit{prime} \mid x \neq y \bullet x + y\}$

3. Numbers that are equal to the sum of two distinct primes, and also to the product of two distinct primes. (2)

   ANSWER:
   $\{x, y, x', y' : \mathit{prime} \mid x + y = x' * y' \wedge x \neq y \wedge x' \neq y' \bullet x + y\}$

   Comment on an ambiguity in this question. (1)

   ANSWER:
   Is $(x, y) = (x', y')$ insisted on, prohibited, or do we not care?

(c) 1. Explain in English the meaning of the Z type $\mathit{iseq}\,\mathbb{N}$. (1)

   ANSWER:
   Injective sequences of natural numbers; that is: finite (possibly empty) lists of distinct natural numbers.

2. Give a precise description of the underlying sets implementation of $\mathit{iseq}\,\mathbb{N}$ in Z. Your answer need not necessarily be in mathematical notation, but you must demonstrate you understand how in full detail how this type is implemented. (2)

   ANSWER:
   An element $l : \mathit{iseq}\,\mathbb{N}$ is a finite set of pairs of natural numbers having the form $\{(1, l_1), (2, l_2), \ldots, (n, l_n)\}$ and such that $\#\{l_1, \ldots, l_n\} = n$ (that is, the $\{l_1, \ldots, l_n\}$ are all distinct). The empty set is allowed and corresponds to the empty list.

(d) Explain in English the meanings of the following sets:

1. $\{(x', x), (y', y) : \mathbb{N} \times \mathbb{N}_1) \mid x' * y = y' * x \bullet ((x', x), (y', y))\}$. (2)

ANSWER:
Set of pairs of pairs of numbers which represent the same fraction if we write them as $x'/x = y'/y$.
NOTE: You will get no marks if you just write something of the form "The set of $((x', x), (y', y))$ such that x' times y is equal to y' times x". That answers the question 'Translate into English', not the question 'Explain in English'.
Also, because of the content of this subquestion and its position towards the end of the question, this is clearly intended to be a 'sting in the tail' and would be marked accordingly.

2. $\bigcap\{X : \mathbb{P}\mathbb{N} \mid 0 \in X \land \forall\, x{:}X \bullet x{+}2 \in X\}$. (2)

ANSWER:
Even numbers!

2. Assume an abstract types of *people* $[PERSON]$ and *rooms* $[ROOM]$.

Henceforth you may assume standard Z operations on sets and functions, such as sets union and intersection, sets subtraction, domain, range, domain and range (anti)restriction, functional and relation application, image, and inverse, and so forth.

(a) Write a schema $State$ with precisely one schema variable $inRoom : PERSON \mathrel{+\!\!\!\!\!\to} ROOM$. The state predicate should reflect that every person is in at most one room. (2)

ANSWER:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;State\;}} \\
inRoom : PERSON \mathrel{+\!\!\!\!\!\to} ROOM \\
\hline
\\
\hline
\end{array}$$

The condition that every person be in at most one room is built in to the type $\mathrel{+\!\!\!\!\!\to}$.

(b) Write the schemas $\Delta State$ and $\Xi State$ in full. (2)

ANSWER:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;\Delta State\;}} \\
inRoom, inRoom' : PERSON \mathrel{+\!\!\!\!\!\to} ROOM \\
\hline
\\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;\Xi State\;}} \\
\Delta State \\
\hline
inRoom' = inRoom \\
\hline
\end{array}$$

(c) Write a schema *Move* which inputs $p? : PERSON$ and $r? : ROOM$ and moves $p?$ to be in $r?$, provided that $p?$ is not already in $r?$. (4)

ANSWER:
This is probably the slickest answer:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;Move\;}} \\
\Delta State \\
p? : PERSON \\
r? : ROOM \\
\hline
inRoom' = (\{p?\} ⩤ inRoom) \cup \{p \mapsto r?\} \\
p? \in dom(inRoom) \Rightarrow inRoom(p?) \neq r? \\
\hline
\end{array}$$

I'm being slightly pedantic checking that $p? \in dom(inRoom)$ and if I were writing a maths paper then my colleagues would probably expect this. You wouldn't lose marks for just writing $inRoom(p?) \neq r?$ directly.

(d) Assume a type $MESSAGE ::= alreadyInRoom \mid success$. Totalise the schema $Move$ with an appropriate error message. (4)

ANSWER:

$$\begin{array}{|l}
\hline \textit{Success} \\
m! : \mathit{MESSAGE} \\
\hline m! = \mathit{success} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline \textit{AlreadyInRoom} \\
\Xi \mathit{State} \\
p? : \mathit{PERSON} \\
r? : \mathit{ROOM} \\
m! : \mathit{MESSAGE} \\
\hline p? \in \mathrm{dom}(\mathit{inRoom}) \\
\mathit{inRoom}(p?) = r? \\
m! = \mathit{alreadyInRoom} \\
\hline
\end{array}$$

$\mathit{totalMove} \stackrel{def}{=} (\mathit{Move} \wedge \mathit{success}) \vee \mathit{AlreadyInRoom}$

(e) Write a schema $\mathit{Outdoors}$ with state variable including $\mathit{outdoors}! : \mathbb{P}\mathit{PERSON}$ that outputs the set of people who are not in any room; $\mathit{Outdoors}$ should not change the state. (2)

ANSWER:
Slick:

$$\begin{array}{|l}
\hline \textit{Outdoors} \\
\Xi \mathit{State} \\
\mathit{outdoors}! : \mathbb{P}\mathit{PERSON} \\
\hline \mathit{outdoors}! = \mathit{PERSON} \setminus \mathrm{dom}(\mathit{inRoom}) \\
\hline
\end{array}$$

Less slick but very functional:

$$\begin{array}{|l}
\hline \textit{Outdoors} \\
\Xi \mathit{State} \\
\mathit{outdoors}! : \mathbb{P}\mathit{PERSON} \\
\hline \mathit{outdoors}! = \{p : \mathit{PERSON} \mid \neg \exists\, r : \mathit{ROOM} \bullet \mathit{inRoom}(p) = r\} \\
\hline
\end{array}$$

(f) Write a schema $\mathit{FireAlert}$ that empties all the rooms. (2)

ANSWER:

$$\begin{array}{|l}
\hline \textit{FireAlert} \\
\Delta \mathit{State} \\
\hline \mathit{inRoom}' = \varnothing \\
\hline
\end{array}$$

(g) Write a schema Mingle which shuffles people so that people who are indoors stay indoors, and people outdoors stay outdoors, but nobody is in the same room after the schema as they were before. (2)

ANSWER:

$$\begin{array}{|l}
\textit{Mingle} \\
\Delta \textit{State} \\
\hline
\textit{dom}(\textit{inRoom}) = \textit{dom}(\textit{inRoom}') \\
\forall\, p : \textit{dom}(\textit{inRoom}) \bullet \textit{inRoom}(p) \neq \textit{inRoom}'(p) \\
\hline
\end{array}$$

(h) Assume a global constant $charlie : PERSON$. $charlie$ (family name Chaplin) is a fantastic conversationalist; people want to be around him.

Write a schema $Charlie$ which assumes that $charlie$ is in some room, and after the schema, at least one person has moved from every other nonempty room to be in the same room as $charlie?$. No other people move. (2)

ANSWER:
$\textit{inRoom}^{\text{-}1}(r)$ is functional inverse image (often called preimage *in the literature); it can also be written using relational inverse and relational image as $\textit{inRoom}^{\text{-}1}(\!|\{r\}|\!)$. Intuitively: this expresses "the people in room r".*

$$\begin{array}{|l}
\textit{Charlie} \\
\Delta \textit{State} \\
\hline
\textit{charlie} \in \textit{dom}(\textit{inRoom}) \\
\forall\, r : \textit{ROOM} \bullet r \neq \textit{inRoom}^{\text{-}1}(\textit{charlie}) \Rightarrow \\
\qquad (\textit{inRoom}'^{\text{-}1}(r) \subseteq \textit{inRoom}(r) \wedge \\
\qquad\; (\textit{inRoom}^{\text{-}1}(r) \neq \varnothing \Rightarrow \textit{inRoom}'^{\text{-}1}(r) \neq \textit{inRoom}^{\text{-}1}(r))) \\
\textit{dom}(\textit{inRoom}) = \textit{dom}(\textit{inRoom}') \\
\hline
\end{array}$$

3. (a) Your boy or girlfriend wants you to be assertive, yet sensitive; spontaneous, yet deep; popular, yet devoted only to him or her.

Translate this requirement into Z and comment on the possibilities of implementing this specification. (2)

ANSWER:
$\perp$. Unsatisfiable.
Sorry dear.

(b) For each of the the following jobs, give a concrete example of where formal or semi-formal specification is important in that job, and one aspect of the job where formal specification is likely to be less useful:

1. Working the telephone in a Virgin Broadband call centre.
2. Teaching computer science at undergraduate level.
3. Parachute jumping instructor.
4. Chip designer. (8)

ANSWER:

1. *They operate off a flowchart which specifies exactly what they should say in every situation. I encountered this one day when I had to phone up and was asked "What version of Windows are you using Sir?". "I'm not; I'm using Linux". "What version of Windows is that, Sir?". The operator needed a response that matched her flowchart. This is a formal specification; not in formal logic, but very very specific nonetheless, as the operator's response demonstrated.*
   Aspects that are harder to formally specify: When to bounce the customer up to a superviser. How to calm down an irate customer. Etc.
2. *I work off a module descriptor which is a reasonably formal document. Dealing with students however is hard to formalise.*
3. *I expect the safety checklist for a parachute instructor is as detailed and carefully designed as anything I might write in first-order logic.*
   Handling a customer at 12,500ft is unlikely to be formally specified.
4. *Chips nowadays are often fully specified in languages very much like Z.*
   Chip designers must negotiate with one another for chip real estate on which to print their designs. This used to be specified at top level but my understanding is that nowadays this is actually managed using a bidding/auction system using a virtual currency, so that chip space gets allocated to the designers that really need it and are willing to pay.

(c) Consider the following specification for hacking any account: "Open login screen. Type in the right password."

Using this example, explain the difference between *specification* and *implementation*. (2)

ANSWER:
Though written in English, it would not be hard to translate this to Z. Specification is saying what you want; implementation is saying how to do it. In the example above we have a clear specification of what we want, but no idea how to do it.

(d) Consider the following refined specification for hacking any account: "Open login screen. Type in every possible password until you find the right one."

Using this example, elaborate further on the difference between *specification* and *implementation*. (2)

ANSWER:
This specification is so precise as to be almost an implementation; you should easily be able to script a program to do just this and so save you the trouble of typing. However, it is computationally intractable; you'll be there forever, or nearly so.

(e) Microsoft used to be famous for shipping buggy code (less so now). Explain why this made economic sense in the 90s and early 2000s, and discuss why it is less acceptable now. (4)

ANSWER:
Back when the technology was evolving very rapidly and during Microsoft's golden years when it had virtually no competition, it was better to release software with bugs sooner. Nowadays many things have changed:

1. *Technological change has slowed, at least in the PC segment that is Microsoft's heartland.*
2. *Competition has hotted up: not only OS X but also Linux are realistic alternatives to the Microsoft OS. Bugs are more expensive because they might lose you customers.*
3. *The user base has broadened; a seasoned programmer might shrug and carry on if their computer crashes and they lose a day's data, because that's their job. If your grandma loses the form she's filling out online, she'll likely be furious and stay that way for days.*

Where technology is changing rapidly—mobile phones, cloud and online services—the same pattern of 'release early, release buggy' is repeating itself. The modern version is more serious though; a bug nowadays can mean hackers emptying your bank account or stealing your identity. So the consequences are more severe and I think companies often try much harder to trap bugs. But not always. . .

(f) Give one example of a proper, full-fat, fully-formal specification that you encountered on your way in to the University today. (2)

ANSWER:
Bus timetable.

4. (a) 1. Write the ML type of `mysucc`.

```
fun mysucc (x:int) = x+1;
```
(2)

ANSWER:

`int -> int`

2. State the value and type computed by

```
mysucc ~1;
```
(2)

ANSWER:

$0 : int$

(b) Consider the following Z specification:

Factorial

$$fact : \mathbb{N} \rightarrow \mathbb{N}$$

$$fact\, 0 = 1$$
$$\forall\, n : \mathbb{N}_1 \bullet fact\, n = n * (fact(n-1))$$

1. Implement `fact` in ML. (2)

ANSWER:

```
fun fact 0 = 1 | fact n = n*(fact(n-1))
```

Don't worry about whether there are brackets; $fact(n)$ and $fact\ n$ are equivalent.

2. State the type of `fact` and explain why it is different from the type specified in Z. (2)

ANSWER:

`int -> int`

It is not `nat -> nat` *because ML does not have a primitive type of natural numbers (for the purposes of this course; there may exist an ML library somewhere that you could load!).*

(c) 1. Consider the ML function `real` and recall that `real (1:int)` evaluates to `1.0:real`. State the type of `real` and explain what it does. (2)

ANSWER:

`real:int->real`. *It* casts *an integer to a real.*

2. Explain why `real` is necessary. (1)

ANSWER:

ML is strictly typed.

3. The *binomial* is specified in Z as follows, where $\mathbb{R}$ is the type of real numbers:

Binomial

$$bin : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{R}$$

$$\forall\, n, k{:}\mathbb{N} \bullet k {\leq} n \Rightarrow bin\ n\ k = (fact\ n)/((fact\ k)) * (fact\ (n{-}k))$$

a) State the type of an implementation of `bin` in ML. (2)

ANSWER:

`int -> int -> real`

b) Implement `bin` in ML. You do not need to include error-handling code for out-of-bound inputs. (4)

ANSWER:
Not the prettiest, but it'll do:

```
fun bin n k =
(real (fact n))/((real (fact k))*(real (fact (n-k))));
```

4. Write an ML function `B` which inputs a number `n` and outputs the list $[\binom{n}{0},\ldots,\binom{n}{n}]$. You do not need to include error-handling code. You are free to define any helper functions you find useful. (3)
State the ML type of `B`. (1)

ANSWER:

```
fun binhelper n 0 = [bin n 0]
|   binhelper n k = (bin n k)::(binhelper n (k-1));
fun B n = binhelper n n;
val B = fn :  int -> real list;
```

5. Using `B` and a function `L : real list -> real`, which would normally be called `listsum` but we give it here a one-character name, write a function using only 3 non-whitespace characters that given $n:\mathbb{N}$ will calculate $\Sigma_{0\leq k\leq n}\binom{n}{k} = \binom{n}{0}+\cdots+\binom{n}{n}$. (1)

ANSWER:
Just for completeness: `fun L [] = 0.0 | L (hd::tl) = hd+(L tl);`

```
B o L;
```

END OF PAPER