UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1A

Thursday 8 December 2005

14:30 to 16:30

Convener: M Jerrum
External Examiner: R Irving

INSTRUCTIONS TO CANDIDATES

1. **Candidates in the third or later year of study for the degrees of MA(General), BA(Relig Stud), BD, BCom, BSc(Social Science), BSc(Science) and BEng should put a cross (X) in the box on the front cover of the script book.**
2. **Answer Part A and Part B in SEPARATE SCRIPT BOOKS. Mark the question number clearly in the space provided on the front of the book.**
3. **Note that ALL QUESTIONS ARE COMPULSORY.**
4. **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.**

Write as legibly as possible.

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

Part A FUNCTIONAL PROGRAMMING

In the answer to any part of any question, you may use any function specified in an earlier part of that question or in an earlier question. You may do this whether or not you actually provided a definition for the earlier part; nor will you be penalized in a later part if your answer to an earlier part is incorrect.

As an aid to memory for the programming part of the exam, some library functions are listed overleaf.

```
div, mod :: Int -> Int -> Int
13 `div` 4 == 3
13 `mod` 4 == 1
```

```
(&&), (||) :: Bool -> Bool -> Bool
True && False == False
True || False == True
```

```
(:) :: a -> [a] -> [a]
'g' : "oodbye" == "goodbye"
```

```
(++) :: [a] -> [a] -> [a]
"good" ++ "bye" == "goodbye"
```

```
sum, product :: (Num a) => [a] -> a
sum [1.0,2.0,3.0] == 6.0
product [1,2,3,4] == 24
```

```
(&&), (||) :: Bool -> Bool -> Bool
True && False == False
True || False == True
```

```
fromIntegral :: Int -> Float
fromIntegral 3 == 3.0
```

```
intToChar: Int -> Char
intToChar 3 == '3'
```

```
charToInt: Char -> Int
charToInt '3' == 3
```

```
head :: [a] -> a
head "goodbye" == 'g'
```

```
tail :: [a] -> [a]
tail "goodbye" == "oodbye"
```

```
take :: Int -> [a] -> [a]
take 4 "goodbye" == "good"
```

```
drop :: Int -> [a] -> [a]
drop 4 "goodbye" == "bye"
```

```
elem :: (Eq a) => a -> [a] -> Bool
elem 'd' "goodbye" == True
```

```
replicate :: Int -> a -> [a]
replicate 5 '*' == "*****"
```

```
concat :: [[a]] -> [a]
concat ["con","cat","en","ate"] == "concatenate"
```

```
zip :: [a] -> [b] -> [(a,b)]
zip [1,2,3,4] [1,4,9] == [(1,1),(2,4),(3,9)]
```

1. A date is a pair, representing a day and month.

```
type Day = Int
type Month = String
type Date = (Day, Month)
```

A month is represented by the first three letters of its name. You may use the following list of month names.

```
months :: [Month]
months =  ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
           "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
```

(a) Write a function `index :: Month -> Int` that replaces a month name by the corresponding number. For example, `index "Mar"` returns 3. Your function should raise an error if passed an invalid month name. [*5 marks*]

(b) Write a function `sensible :: Date -> Bool` that returns true if given a date in which the day is between 1 and 31 and the month is valid, and false otherwise. For example, `sensible (1,"Feb")` and `sensible (31,"Feb")` return true, but `sensible (0,"Feb")` and `sensible (32,"Feb")` and `sensible (1,"Foo")` return false. [*5 marks*]

(c) Write a function `before :: Date -> Date -> Bool` that returns true if the first date precedes the second date. For example, `before (31,"Mar") (1,"Apr")` is true, and `before (31,"Mar") (1,"Mar")` is false. You may assume that this function is only passed sensible dates. [*5 marks*]

(d) Write a function `showDate :: Date -> String` that converts a date to a string. For example, `showDate (2,"Mar")` returns `"02/03"`; note the leading zeros. If the function is passed a date that is not sensible, it should return `**/**`. [*5 marks*]

2. The *greatest common divisor of six* and a positive integer is computed as follows.

   - If the given integer is *not* divisible by two and is *not* divisible by three, the answer is 1.
   - If the given integer *is* divisible by two and is *not* divisible by three, the answer is 2.
   - If the given integer is *not* divisible by two and *is* divisible by three, the answer is 3.
   - If the given integer *is* divisible by two and *is* divisible by three, the answer is 6.

   For example, the greatest common divisor of six and the numbers 1, 2, 3, 4, 5, and 6 is respectively 1, 2, 3, 2, 1, and 6.

   (a) Write a function `six :: Int -> Int` that returns the greatest common divisor of six and a given non-negative integer. For example, `six 1` returns 1, `six 2` returns 2, `six 3` returns 3, `six 4` returns 2, `six 5` returns 1, and `six 6` returns 6. You should raise an error if the argument is negative or zero. [*6 marks*]

   (b) Write a function `sixes :: [Int] -> [Int]` that applies the function `six` to each integer in a list to yield a new list; integers that are negative or zero are omitted. Thus, `sixes [-1,0,1,2,3,4,5,6]` returns `[1,2,3,2,1,6]`. Your definition should use a *list comprehension*, not recursion. [*6 marks*]

   (c) Write a second definition of `sixes`, this time using *recursion*, and not a list comprehension. [*6 marks*]

3. This question uses the type `Date` from Question 1. In answering this question you may use the function `before` from that question.

   (a) Write a function `befores :: [Date] -> Bool` that given a list of dates returns true if each date in the list is before the following date. For example, `befores [(1,"Jan"),(10,"Jan"),(5,"Feb"),(1,"Apr")]` returns true, and `befores [(1,"Jan"),(5,"Feb"),(10,"Jan")]` returns false. Your function should return true if given a list with fewer than two dates. Your definition should use a *list comprehension* and the library functions above, and not recursion. [*6 marks*]

   (b) Write a second definition of `befores`, this time using *recursion*, and no list comprehension or library functions. [*6 marks*]

Part B COMPUTATION AND LOGIC

4. For each of the following logical expressions draw a truth table (including in the table columns for all intermediate sub-expressions, as described in the lecture notes) and state whether the expression is a tautology, inconsistent or contingent.

   (a) $(not(a)\ or\ b) \leftrightarrow (b \rightarrow a)$

   (b) $(a \rightarrow b) \rightarrow (not(b) \rightarrow not(a))$

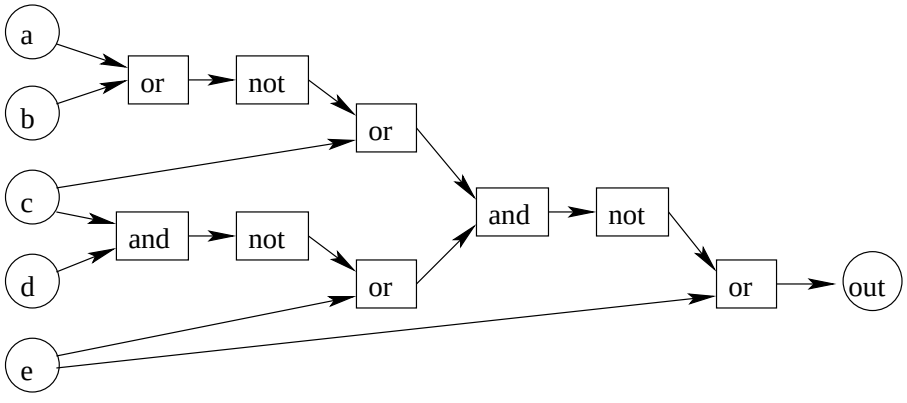
   (c) $not(a \rightarrow b)\ and\ (not(a)\ or\ b)$

   [*6 marks*]

5. The diagram below gives a form of logic circuit constructed from the following components:

   - An "and" connector (drawn in the diagram as a box labelled "and") sends as output the signal "true" only if both its inputs are signalling "true", otherwise it signals "false".
   - An "or" connector (drawn in the diagram as a box labelled "or") sends as output the signal "false" only if both its inputs are signalling "false", otherwise it signals "true".
   - A "not" connector (drawn in the diagram as a box labelled "not") sends as output the signal "true" if its input signals "false", otherwise it signals "false" if its input signals "true".

   Signals are propagated through the circuit in the direction indicated by the arrows connecting components. There are five inputs to the circuit; these are the circles labelled "a", "b", "c", "d" and "e" on the diagram. There is a single output; this is the circle labelled "out" on the diagram.



   (a) Represent the circuit as a logical expression appropriate for analysis using a truth table. Your expression should be constructed using conjunction ("and"), disjunction ("or") and negation ("not") operators only. Draw a

truth table (including the table columns for all intermediate sub-expressions, as described in the lecture notes) to calculate the truth values for this expression when the propositions corresponding to inputs "a", and "d" are true (considering all combinations of truth values for "b", "c" and "e"). Explain what this tells us about the circuit. [*10 marks*]

(b) Prove the proposition e from the following set of axioms

$$[b, \quad (a \; or \; b) \rightarrow c, \quad c \rightarrow e]$$

using the set of proof rules given below:

Rule name	Sequent	Supporting proofs
immediate	$\mathcal{F} \vdash A$	$A \in \mathcal{F}$
and_intro	$\mathcal{F} \vdash A \; and \; B$	$\mathcal{F} \vdash A, \; \mathcal{F} \vdash B$
or_intro_left	$\mathcal{F} \vdash A \; or \; B$	$\mathcal{F} \vdash A$
or_intro_right	$\mathcal{F} \vdash A \; or \; B$	$\mathcal{F} \vdash B$
or_elim	$\mathcal{F} \vdash C$	$(A \; or \; B) \in \mathcal{F}, \; [A\|\mathcal{F}] \vdash C, \; [B\|\mathcal{F}] \vdash C$
imp_elim	$\mathcal{F} \vdash B$	$A \rightarrow B \in \mathcal{F}, \; \mathcal{F} \vdash A$
imp_intro	$\mathcal{F} \vdash A \rightarrow B$	$[A\|\mathcal{F}] \vdash B$

where $\mathcal{F} \vdash A$ means that expression A can be proved from set of axioms $\mathcal{F}$; $A \in \mathcal{F}$ means that A is an element of set $\mathcal{F}$; $[A|\mathcal{F}]$ is the set constructed by adding A to set $\mathcal{F}$; $A \rightarrow B$ means that A implies B; $A \; and \; B$ means that A and B both are true; and $A \; or \; B$ means that at least one of A or B is true. [*9 marks*]

6. Suppose that you are asked to describe the behaviour of a simplified cash dispensing machine using a transducer-style Finite State Machine. The following is a description in English of the machine:

"The machine allows a card to be inserted and then accepts a request for money or, alternatively, a request for a statement of funds. If the request is for money then it either dispenses money and then ejects the card, or it refuses to dispense money and then waits for more requests. If the request is for a statement of funds then it dispenses a statement and then waits for more requests."

(a) Draw a Finite State Machine for this system. [*6 marks*]

(b) Construct a table describing the transition relation (containing all necessary state-action combinations) for the Finite State Machine you have drawn. [*2 marks*]

(c) Is your Finite State Machine deterministic or non-deterministic? Explain why. [*2 marks*]

7. Assume we are building Finite State Machines with the input alphabet $\{0, 1\}$. For each of the following two acceptance tasks either draw the acceptor Finite State Machine or explain why it is impossible to construct a Finite State Machine. You may give either a deterministic or non-deterministic Finite State Machine.

   (a) Accept exactly those strings that include at least two occurrences of the pattern 010, where the two occurrences could overlap (so for example 010010, 01010 and 011101010101 are accepted). [*6 marks*]

   (b) Accept all strings for which the number of occurrences of the character 1 is the square of the number of occurrences of the character 0. [*3 marks*]

8. Write regular expressions that describe each of the two following types of strings, assuming the alphabet $\{0, 1\}$.

   (a) All strings in which there are never two consecutive occurrences of the character 1. [*3 marks*]

   (b) All strings that begin and end with the same character. [*3 marks*]