UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1A

Monday 14 August 2006

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

Unless otherwise stated, you may use either list comprehension or recursion in your answers (or both).

In the answer to any part of any question, you may use any function specified in an earlier part of that 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.

Unless otherwise stated, you may use any operators and functions from the standard prelude and from the libraries Char, List, and Maybe. You need not write import declarations.

As an aid to memory, copies of Figures 5.1 and 5.2 (pp. 91 and 92) are included from Simon Thompson, *Haskell, The Craft of Functional Programming.* You will not need all of the functions listed.

1. A date is a triple of integers, representing day, month, and year.

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

A day of the week is represented by a number from 0 to 6, where 0 is Sunday, 1 is Monday, and so on.

```
type DayOfWeek = Int
```

In the following, you may use the function `dayOfWeek :: Date -> DayOfWeek` that given a date returns the corresponding day of the week. For example:

```
dayOfWeek (13,1,2006) == 5   -- 13 Jan 2006 is a Friday
dayOfWeek  (6,6,2006) == 2   --  6 Jun 2006 is a Tuesday
dayOfWeek (13,8,2006) == 0   -- 13 Aug 2006 is a Sunday
```

(a) Write a function `unlucky :: Date -> Bool` that returns true if the date is unlucky, and false otherwise. A day is unlucky if it is Friday the 13th in any month, or if it is the sixth day of the sixth month of any year ending in six. For example, the first two days listed above are unlucky but the third is not. [*5 marks*]

(b) Write a function `showDayOfWeek :: DayOfWeek -> String` that takes the number representing a day of the week and returns the corresponding string. [*5 marks*]

(c) Write a function `showDate :: Date -> String` that converts a date to a corresponding string, indicated as day, month, and year separated by slashes. For example, the three days above return `"13/1/2006"`, `"6/6/2006"`, and `"13/8/2006"`. [*5 marks*]

(d) Write a function `showDay :: Date -> String` that converts a date to a day and an indication of whether or not it is unlucky. For example:

```
dayOfWeek (13,1,2006) == "Friday(unlucky)"
dayOfWeek  (6,6,2006) == "Tuesday(unlucky)"
dayOfWeek (13,8,2006) == "Sunday"
```

[*5 marks*]

2. (a) Write a function `kind :: Char -> Char` that converts any lower case letter to `'a'`, any upper case letter to `'A'`, any digit to `'1'`, and any other character to `'.'`. Thus, applying this function to `'G'`, `'o'`, `'4'`, `'i'`, `'t'`, `'!'` respectively returns `'A'`, `'a'`, `'1'`, `'a'`, `'a'`, `'.'`. [*6 marks*]

   (b) Write a function `kinds :: String -> String` that applies the function `kind` to each letter in a string to yield a new string; characters in the initial string that are not letters or digits are omitted. Thus, applying this function to `"Go4it!"` returns `"Aa1aa"`. Your definition should use a *list comprehension*, not recursion. [*6 marks*]

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

3. (a) Write a function `ascending :: [Int] -> Bool` that takes a list of numbers and returns true if the list is in strictly ascending order (each number larger than the one before it), and false otherwise. For instance, it returns true for `[3,5,8,10]` but false for `[3,8,5,10]` and for `[3,5,5,10]`. It returns true for the empty list or a list with one element. Your definition should use a *list comprehension* and library functions, not recursion. [*6 marks*]

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

Part B COMPUTATION AND LOGIC

4. You are given the following proof rules:

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

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 $\mathcal{F}$; $A \to B$ means that A implies B; $A \text{ and } B$ means that A and B both are true; and $A \text{ or } B$ means that at least one of A or B is true.

Using the proof rules above, prove the following sequent:

$$[(a \text{ or } c) \to d,\ (f \text{ and } e) \to b,\ d \to f,\ d \to e] \vdash\ a \to b$$

Show precisely how the proof rules are applied. [*10 marks*]

5. The following expression is a tautology:

$$(not(b) \text{ and } (a \to b)) \to not(a)$$

(a) Demonstrate in detail, using a truth table, that the expression above is a tautology. [*10 marks*]

(b) Convert the expression above into an equivalent expression containing only '*or*' and '*not*' logical operators (and no '$\to$' operator) by applying the following rules of equivalence between expressions:

$(P \to Q)$	is equivalent to	$not(P) \text{ or } Q$
$not(P \text{ and } Q)$	is equivalent to	$not(P) \text{ or } not(Q)$

Show precisely how the equivalence rules are applied. [*5 marks*]

6. You are given the following description of the events that take place when someone, whom we shall call "Dave", answers a telephone:

   Dave waits for the phone to ring. If it rings he picks up the phone and asks the caller who he/she wants to contact. If the call is to a wrong number then Dave informs the caller of this and hangs up the phone. If the caller wants to contact someone who lives with Dave then he either brings that person to the phone and they talk to the caller or, if that person is not at home, Dave tells the caller to ring back later and hangs up the phone. If the call is for Dave himself then he talks to the caller. After some time talking to the caller, Dave (or the person he brings to the phone) will say goodbye and hang up the phone. After the phone has been hung up Dave waits for it to ring again.

   (a) Identify all the individual states in the telephone answering scenario above. [*3 marks*]

   (b) Define a finite state machine that models the behaviour of this telephone answering system. [*8 marks*]

   (c) Briefly explain whether or not your finite state machine is deterministic. [*2 marks*]

7. A palindrome is any string over some alphabet that reads the same either forwards or backwards. Assume your alphabet is the set of characters $\{a, b\}$. Examples of strings that are palindromes are "a", "b", "aba", "$ababa$", "$aabaa$". Examples that are not a plaindrome are "ab", "abb" and "$baba$".

   (a) Why is it not possible to write a finite state machine to accept palindromes? [*3 marks*]

   (b) How might a finite state machine be extended to allow it to accept palindromes? [*2 marks*]

8. Assuming that your alphabet is the set of characters $\{a, b\}$, write regular expressions for the following languages:

   (a) The set of strings that contain no more than two a's [*2 marks*]

   (b) The set of strings that both start and end with an a. [*2 marks*]

   (c) The set of strings with an even number of b's. [*3 marks*]