TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2006

MID-YEAR

COMP 304

Programming Languages

Time Allowed: 3 Hours

Instructions: There are five (5) questions, each worth thirty (30) marks. You should attempt all of the questions.

Question 1.

(a) Explain what it means to describe a function as a higher-order function. [6 marks]

A type of trees can be defined in Haskell as:

```
data Tree a = Leaf a
            | Node (Tree a) (Tree a)
```

(b) Give a suitable definition, including the type, for a Haskell function `leafcount` which counts the number of leaves in a tree. For example:

```
Main> leafcount (Node (Node (Leaf 'a') (Leaf 'b')) (Node (Leaf 'c') (Leaf 'd')))
4
```

[5 marks]

(c) The function `map :: (a -> b) -> [a] -> [b]` is defined in the Standard Prelude. It generates a list by applying a function to each item in a list. For example:

```
Main> map even [1,2,3,4]
[False,True,False,True]
Main> map (3+) [1,2,3,4]
[4,5,6,7]
```

Give a suitable definition, including the type, for a Haskell function `maptree`, which generates a tree by applying a function to each item at the leaf of a tree. For example:

```
Main> maptree even (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4)))
Node (Node (Leaf False) (Leaf True)) (Node (Leaf False) (Leaf True))
Main> maptree (3+) (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4)))
Node (Node (Leaf 4) (Leaf 5)) (Node (Leaf 6) (Leaf 7))
```

[6 marks]

(d) Give a suitable definition, including the type, for a Haskell function `treerec` which captures the pattern which `leafcount` and `maptree` follow. [6 marks]

(e) Use `treerec` to define `treeprod` and `treesum`, both having type `(Tree Integer) -> Integer`. The function `treeprod` should compute the product of the values in a tree of integers, `treesum` should compute the sum of the values in a tree of integers. For example:

```
Main> treesum (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4)))
10
Main> treeprod (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4)))
24
```

[7 marks]

Question 2.

(a) State what answer Prolog will give to the following queries:

(i) `f(X) = f(f).`

(ii) `f(X, a(Z, c)) = f(Z, a(b, X)).`

(iii) `[A, B | C] = [1, 2, B].` [6 marks]

(b) You have a friend who is learning Prolog. She is only beginning and needs your help to understand the language. Your friend is very interested in football, and she has written the following Prolog code to record the outcome of the matches in the knockout phase of the 2002 FIFA World Cup:

```
beat(brazil, england).
beat(brazil, turkey).
beat(brazil, germany).
beat(germany, south_korea).
beat(germany, united_states).
beat(south_korea, spain).
beat(turkey, senegal).
```

Brazil beat Turkey and Turkey beat Senegal, so your friend believes that Brazil are a better team than Senegal. She reasons that one team, A, is better than another team, B, if A beat B or if there is another team, C, which A is better than and which beat B. Hence she has written the following code:

```
better(Better, Worst) :-
       better(Better, Worse),
       beat(Worse, Worst).
better(Better, Worse) :- beat(Better, Worse).
```

Unfortunately, when she asks Prolog to solve the query `better(brazil, senegal)` this happens:

```
?- better(brazil, senegal).
ERROR: Out of local stack
```

Write an explanation for your friend of how Prolog searches for solutions to a query to explain to her why this happens. [10 marks]

(c) Write a version of `better/2` which works correctly, and show how Prolog will solve the query `better(brazil, senegal).` [8 marks]

(d) Your friend has discovered that it is possible to define `not` in Prolog like this:

```
not(A) :- A, !, fail.
not(_).
```

Since the winner of a knockout competition has not been beaten by anyone, she defines:

```
winner(Win) :- not(beat(Anyone, Win)).
```

Write an explanation for her of why she gets this behaviour:

```
?- winner(scotland).

Yes
?- winner(W).

No
```

[6 marks]

Question 3.

(a) Describe a model of imperative programming. [10 marks]

(b) For any imperative programming language with which you are familiar explain how the programming language differs from the model. [10 marks]

(c) Describe the advantages that lazy evaluation provides for the functional programmer. [10 marks]

Question 4.

(a) State three benefits of providing a formal semantics for a programming language. [6 marks]

(b) Briefly explain how we use operational semantics to describe the meaning of a program. [5 marks]

(c) Briefly explain how we use axiomatic semantics to describe the meaning of a program. [5 marks]

(d) Briefly describe the role of:

(i) the abstract syntax;

(ii) the semantic algebra;

(iii) the valuation function;

in denotational semantics. [9 marks]

(e) A unary number is a sequence of 1's. For example, 1, 11, 111, 1111 correspond to the decimal numbers 1, 2, 3, 4. Give a suitable valuation function for unary numbers U, using the natural numbers Nat as a semantic algebra, and using the following abstract syntax:

$$\begin{array}{lcl} D & \rightarrow & 1 \\ U & \rightarrow & D \\ & | & DU \end{array}$$

[5 marks]

Question 5.

(a) Indicate the redexes in the following λ-terms by underlining them:

(i) $(\lambda z.z)x$

(ii) $(\lambda z.z)((\lambda x.xy)(\lambda p.p))$

(iii) $\lambda z.((\lambda x.xx)\lambda x.xx)$

(iv) $(\lambda x.(\lambda y.(\lambda z.zxy)))pq$ [8 marks]

(b) State what it means to use applicative order evaluation. [4 marks]

(c) Reduce $(\lambda x.xxxx)((\lambda v.v)w)$ to normal form using applicative order evaluation. [4 marks]

(d) State what it means to use normal order evaluation. [4 marks]

(e) Reduce $(\lambda x.(\lambda y.x))z((\lambda x.xxxx)((\lambda v.v)w))$ to normal form using normal order evaluation. [4 marks]

(f) Describe how the λ-calculus can be used in the study of programming languages **apart from** its role as a model for functional programming. [6 marks]

****