University of Glasgow

Tuesday, 28 May 2009
2.00 pm – 3.30 pm
(Duration: 1 hour 30 minutes)

DEGREES OF MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

COMPUTING SCIENCE 3Z: PROGRAMMING LANGUAGES 3

Answer all 4 questions.

This examination paper is worth a total of 80 marks.

You must not leave the examination room within the first half-hour or the last fifteen minutes of the examination.

1. **(a)** Pure functional languages (such as Haskell) are highly expressive, despite lacking the assignments and loops of imperative (and object-oriented) languages. Identify and briefly explain *three* features of functional languages that account for their expressive power.

[3]

(b) A *dictionary* (of the kind used by a spell-checker) is a set of words.

(i) Define a Haskell type suitable for implementing a dictionary.

[1]

(ii) Write a Haskell function, `lookup w d`, that yields true if and only if dictionary `d` contains word `w`.

[4]

(iii) Write a Haskell function, `add w d`, that yields the dictionary obtained by adding word `w` to dictionary `d` (or yields `d` if it already contains `w`).

[3]

Explicitly declare the type of each function in (ii) and (iii).

(*Note:* A simple implementation using linear search is acceptable.)

(c) Consider the following Haskell type definition:

```
data Tree a = NULL | NODE a (Tree a) (Tree a)
-- A value of type Tree a is a binary tree whose nodes contain elements of type a.
```

(i) Write a Haskell function, `depth t`, that yields the depth of tree `t`. (*Note*: a tree with a single node has depth 0; an empty tree has depth –1.)

[3]

(ii) Write a Haskell function, `postorder t`, that yields a list of all elements of tree `t`, using post-order traversal.

[3]

(iii) Write a Haskell function, `mirror t`, that yields the mirror-image of tree `t` (i.e., the tree obtained from `t` by swapping every pair of subtrees).

[3]

Explicitly declare the type of each function in (ii) and (iii).

2. The following is part of the BNF grammar of a hypothetical programming language:

```
exp  ::= pexp
       | exp + exp
       | exp - exp

pexp ::= var
       | ( exp )

var  ::= a | b | c | d | …
```

(Here *exp* is an expression; *pexp* is a primary expression; *var* is a variable.)

(a) Show that the expression "`a-b+c`" is *ambiguous* by drawing its two syntax trees. [5]

(b) Show how to modify the grammar to eliminate the ambiguity, in such a way that expressions associate to the left. For example, "`a-b+c`" should be interpreted like "`(a-b)+c`". Illustrate your answer by drawing the unique syntax tree of "`a-b+c`". [5]

3. **(a)** What is an *interpretive compiler*? Why are interpretive compilers useful?

[4]

The remainder of this question is about the Tiny interpretive compiler presented and used in the *Programming Languages 3* course.

(b) Draw tombstone diagrams representing the components of the Tiny interpretive compiler (expressed in Haskell).

[2]

(c) Draw tombstone diagrams showing how to install the Tiny interpretive compiler on machine *M*. (Assume that a Haskell → *M* compiler is available.)

Also show how the interpretive compiler would be used to compile and run a Tiny program *P*.

[4]

(d) Suppose now that you are required to build a compiler that will translate Tiny to *M* machine code. Using tombstone diagrams, show how you would build your compiler using the Tiny interpretive compiler components, adding just one new component.

Also show how your compiler would be used to compile and run a Tiny program *P*.

[10]

4. **(a)** Define *Cartesian products* (×), *disjoint unions* (+), and *mappings* (→). Briefly explain how each concept is relevant to the understanding of programming languages.

[6]

(b) Using the concepts of part (a), write equations defining the set of values of each of the following Haskell types:

```
data Piece         = PAWN | KING
data Colour        = WHITE | BLACK
type ColouredPiece = (Colour, Piece)
data Square        = EMPTY | CONT ColouredPiece
type Board         = Int -> Square
```

[5]

(c) Using the concepts of part (a), write an equation defining the set of objects in a Java program that includes the following classes:

```
class Event {
   private Date date;
   private String description;
   … // methods
}

class Appointment extends Event {
   private Time time;
   … // methods
}

class Meeting extends Appointment {
   private int location;
   private String[] participants;
   … // methods
}
```

(Assume that `Date` and `Time` are library classes.)

[5]

(d) Explain the difference between *statically-typed* and *dynamically-typed* programming languages.

[4]

(e) [continued on next page]

(e) What are the advantages and disadvantages of static and dynamic typing? Illustrate your answer using the following Java method definition and method call:

```
static int max (int x, int y) {
   if (x > y)
   then return x;
   else return y;
}

int m = …;
int n = …;
int p = max(m, n);
```

and the following Python function definition and function call:

```
def max (x, y) :
   if x > y :
      return x
   else :
      return y

m = …
n = …
p = max(m, n);
```

(Java is statically-typed, whilst Python is dynamically-typed.)

[10]