

COMPUTER SCIENCE TRIPOS Part IA – 2014 – Paper 1

1 Foundations of Computer Science (LCP)

types,
polymorphism

- (a) Write brief notes on polymorphism in ML, using lists and standard list functions such as `@` (append) and `map`. [4 marks]

Answer: Key points are that polymorphism assigns a type to every expression — at compile time — while at the same time allowing natural genericity. For instance, the elements of a list must have the same type, but it can be any type. The type of `append`, `'a list * 'a list -> 'a list`, indicates that it combines two lists of the same type, returning another list of that type. The type of `map`, `('a -> 'b) -> 'a list -> 'b list`, indicates that it transforms a list of one type to another, as indicated by the type `('a -> 'b)` of the function.

datatypes,
functions

- (b) Explain the meaning of the following declaration and describe the corresponding data structure, including the role of polymorphism.

```
datatype 'a se = Void | Unit of 'a | Join of 'a se * 'a se;
```

[4 marks]

Answer: This declares a type containing three constructors: `Void`, `Unit` and `Join`. The latter two constructors are functions to build elements of the new type, while `Void` is a constant. This is a tree-like data structure with unlabelled binary branching (`Join`), labelled leaves (`Unit`) and unlabelled leaves (`Void`). Type `'a se` is polymorphic, as indicated by the type variable `'a`, which shows that `'a` is the type of the labels. Functions involving the new type can be declared using pattern matching.

datatypes,
functions,
recursion

- (c) Show that ML lists can be represented using this datatype by writing the functions `encode_list` of type `'a list -> 'a se` and `decode_list` of type `'a se -> 'a list`, such that `decode_list (encode_list xs) = xs` for every list `xs`. [3 marks]

Answer:

```
fun encode_list [] = Void
  | encode_list (x::xs) = Join (Unit x, encode_list xs);

fun decode_list Void = []
  | decode_list (Join(Unit x, v)) = x :: decode_list v;
```

functions as
values

- (d) Consider the following function declaration:

```
fun cute p Void = false
  | cute p (Unit x) = p x
  | cute p (Join(u,v)) = cute p u orelse cute p v;
```

What does this function do, and what is its type?

[4 marks]

Answer: The function `cute` has type `('a -> bool) -> ('a se -> bool)`, and `cute p s` returns true if and only if `s` contains an element of the form `Unit x`, where `p x` is true. It is analogous to the function `exists`, for lists.

functions as
values

(e) Consider the following expression:

```
fn p => cute (cute p)
```

What does it mean, and what is its type? Justify your answer carefully.

[5 marks]

Answer: This is a function of type `('a -> bool) -> 'a se se -> bool`. Through the `fn` binder, it takes an argument `p`, which has type `'a -> bool`. Now `cute p` has type `'a se -> bool`, and because `cute` is polymorphic, `cute (cute p)` is well-defined and has type `'a se se -> bool`.

Now if `fn p => cute (cute p)` is applied to some specific `p` and then to a term `s`, it returns true if and only if `s` contains an element of the form `Unit x`, where `cute p x` is true. Thus the expression is like `cute` but for type `'a se se -> bool`, that is, for the data structure nested in itself.
