COMPUTER SCIENCE TRIPOS Part II – 2014 – Paper 9

9 Optimising Compilers (AM)

(a) Explain the idea of an effect system (also known as a type and effect system), including a typical judgement form $\Gamma \vdash e : t, F$ explaining how this differs from a traditional ML-like type system. Use the words 'immediate' and 'latent' in your answer. [3 marks]

(b) Give (i) the set of effects and (ii) the inference rules for an effect system for a language based on lambda-calculus but including constants, if-then-else, let-in, and reads and writes to channels ξ. The language types should consist of integers and functions. At this point you should only give effect rules corresponding to standard typing rules for your language (i.e. one per syntactic form). [5 marks]

(c) Now give an expression e which is well-typed without effects, but which fails to have an effect judgement of the form $\Gamma \vdash e : t, F$. Explain how an additional rule, similar to subtyping, enables this program to be given an effect judgement. [4 marks]

(d) Suppose now your language contains an infix constant '`+`' representing curried addition with left-to-right evaluation of its operands. We would like to be able to optimise so as to execute e_1 and e_2 in parallel *when this is safe* before adding their results. Give a (safe but not unreasonably restrictive) criterion on the effects of e_1 and e_2 for this optimisation. Discuss any issues that arise in the case that e_1 and e_2 share a common channel. [4 marks]

(e) Finally consider, *using your rules from* Parts (b) *and* (c), whether your criterion from Part (d) can successfully (i) ascribe a type and effect to, and (ii) parallelise, the two calls to `apply` in the program:

```
let apply = λf. λx. f x
in let add1 = λx. x+1
in let add2 = λx. (print!42. x+2)
in apply add1 10 + apply add2 20
```

where `print!`e_1`.`e_2 evaluates e_1 printing its value to standard output, then evaluates e_2 yielding its value as result.

Either justify your parallelisation or suggest a means of avoiding any issue inhibiting parallelisation. [4 marks]