3 Foundations of Functional Programming

2000
p6q 10
ACN

(a) Give as simple a set of rules as you can for transforming lambda calculus to a form where there are no bound variables mentioned, but where there are many instances of the three standard combinator constants **S**, **K** and **I**;
(6 marks)
(b) Describe tree-rewrites suitable for reducing expressions written in terms of combinators;
(6 marks)
(c) Explain how you might deal with the issue of keeping track of the values of bound variables if you were to interpret lambda calculus directly.
(8 marks)

3.1 Marking notes

```
[x]x = x
[x]y = Ky
[x]fg = S([x]f)([x]g)
```

```
Ix -> x    Kxy -> x   Sfgx -> (fx)(gx)
```


but with these three drawn out as trees, plus the observation that to get a good order of evaluation you might like to do leftmost outermost reductions first.

Description of an "environment" as in [(name1,var1), ...] in the fully traditional style. A full answer explains that when you evaluation in the function position you should capture the environment to get a closure.