

— Assignment 3 —

**Lambda Calculus, Grammars, PDAs and Turing Machines**

**Due: 11am Tuesday 11th October 2011**

---

The submission of your assignment must be done via the assignment boxes in the student foyer. Failure to submit by the due date will result in late penalties of ten percent per weekday. Other arrangements that are required because of truly exceptional circumstances need to be negotiated before your deadline.

Your submission must be well-presented on clean A4 paper with a fully completed standard cover page, including your *tutor's name* and your *tutorial group*. Failure to follow this instruction will result in a penalty. The COMP2600 Assignments page has a link to an appropriate cover.

## 1 Regular Languages and Grammars [4 marks]

Consider this right-linear grammar which generates a language  $L$ .

$$\begin{array}{ll} S \rightarrow a B & D \rightarrow e \\ B \rightarrow b C & C \rightarrow e \\ C \rightarrow c D & \end{array}$$

(It is understood that upper case letters are non-terminals, lower-case letters are terminals, and the start symbol is  $S$ ).

- (1) [2 marks] Give a NFA which accepts the language  $L$ .
- (2) [2 marks] Give a left-linear grammar which generates the same language  $L$ .

## 2 Context-free Languages and Pushdown Automata [7 marks]

Consider the following grammar which generates a language of simple expressions.

$$\begin{array}{l} E \rightarrow E E \\ E \rightarrow \langle E \rangle \\ E \rightarrow \mathbf{id} \end{array}$$

Here  $E$  is the start symbol, and you treat  $\mathbf{id}$  (which denotes an identifier) as a terminal symbol, and as a token of the language. So there are three terminal symbols,  $\mathbf{id}$ ,  $\langle$  and  $\rangle$ . (Think of  $\langle$  and  $\rangle$  as parentheses — we just use these symbols to make writing PDA transitions like  $\delta(\dots)\dots$  less confusing).

- (1) [1 mark] This grammar is ambiguous. Give an example of a string which can be generated two different ways. Which one corresponds to the interpretation of expressions in Haskell, or in the  $\lambda$ -calculus? (That is, if the angle brackets were in fact parentheses).
- (2) [2 marks] Give the (non-deterministic) pushdown automaton which corresponds naturally to this grammar, and give a trace show how it runs on the string “**id** **<id id>**”
- (3) [2 marks] Modify the grammar so that it accepts the same language, and is not ambiguous; make sure that it leads to parsing expressions as in Haskell, or in the  $\lambda$ -calculus.
- (4) [2 marks] The original grammar is left-recursive. Modify it so as to give a grammar which generates  $L$  but is not left-recursive, and where also the corresponding pushdown automaton requires only one lookahead symbol to choose the correct transition. Note: lookahead can detect end-of-input, use the symbol ‘ $\dashv$ ’ for end-of-input. **Added 20 Sept** For each non-terminal which is the left-hand side of more than one production, state what are the appropriate lookahead symbols for each of the productions.

### 3 Lambda Calculus [4 marks]

Given the following definitions

$$S = \lambda x y z. x z (y z) \quad K = \lambda x y. x \quad I = \lambda x. x$$

$$B = \lambda f g x. f (g x) \quad C = \lambda f x y. f x y$$

evaluate the following (by substituting for  $S, K, I, B, C$  and performing all possible  $\beta$ -reductions)

- (1)  $C S I$
- (2)  $\lambda f x. S f (K x)$
- (3)  $B (S B) K$

Hint: don’t substitute for  $S, K, I, C, B$  until necessary: your first step in each question would be to substitute for the first combinator only

Hint: you might like to check your answer (and your intermediate steps) using the Haskell type-inferencing: your answer should have the same type as the original (subject to the fact that Haskell may choose different names for the type variables). See the file `comb.hs` for examples of how this works.

## 4 Turing Machines [5 marks]

### 4.1 Deleting a symbol, and closing the gap

Construct a Turing Machine that deletes the symbol at the current head position.

In the initial state the tape of the TM will have a string made up of '0's and '1's with no embedded blanks and with the read head somewhere over the string. Beyond the ends of this string the tape contains blanks.

Your machine should delete the symbol where the read head currently points, and all the symbols to the right of this position should be moved one tape square to the left.

Finally the TM should halt with the head pointing at the same tape position as when it started (so, pointing to the symbol which was initially one position to the right). (You may invent a new symbol, say 'M', to mark this spot).

Example tape initially and on halt shown below

1	0	1	0	0	0	1	1	0	0	1		1	0	1	0	0	1	1	0	0	1
					↑												↑				

Give

- A description in plain English that simply describes how your machines works. **THIS IS IMPORTANT:** your tutor is not expected to figure out a TM diagram without a reasonable explanation! I suggest you give a brief overview and then a description of the role of each state: see the tutorial solutions for an example.
- A diagram for your Turing Machine, in the notation used in lectures.