

## Quiz 1 – solutions

### Question 1. (16 marks) Context-Free Grammars

Consider the following context-free grammar.

Terminals: {a, b, ... z, 0, 1, ... 9, op1, op2, op3 }

Non-terminals: <s>, <op>, <terminal>

Production rules:

<s> ::= <s> <op> <s> | op3 <s> | <terminal>

<op> ::= op1 | op2

<terminal> ::= a | b | ... z | 0 | 1 | ... 9

a. (4 marks) Prove that the grammar is ambiguous.

Two distinct parse trees for a op1 b op2 c .

b. (4 marks) Rewrite the grammar, so that (a) it generates the same language, and (b) op1 has higher precedence than op2. (You don't have to copy the rule for <terminal>).

<s> ::= <s> op2 <s> | <op1>

<op1> ::= <op1> op1 <op1> | op3 <op1> | <terminal>

c. (4 marks) Rewrite the grammar in part b, so that (a) it generates the same language, and (b) op1 and op2 are right-associative. (You don't have to copy the rule for <terminal>).

<s> ::= <op1> op2 <s> | <op1>

<op1> ::= <terminal> op1 <op1> | op3 <op1> | <terminal>

d. (4 marks) Do there exist two context-free grammars  $G_1$  and  $G_2$ , such that  $L(G_1) = L(G_2)$  and the set of non-terminals of  $G_1$  is different from the set of non-terminals of  $G_2$ ? If your answer is yes, give an example of such two grammars. If your answer is no, explain why not.

$G_1$ :  $\Sigma_{G_1} = \{0\}$ ,  $V_{G_1} = \{\langle s \rangle, \langle t \rangle\}$ ,  $S_{G_1} = \langle s \rangle$ ,  $P_{G_1}$ :

$\langle s \rangle ::= \langle t \rangle$

$\langle t \rangle ::= 0$

$G_2$ :  $\Sigma_{G_2} = \{0\}$ ,  $V_{G_2} = \{\langle s \rangle\}$ ,  $S_{G_2} = \langle s \rangle$ ,  $P_{G_2}$ :

$\langle s \rangle ::= 0$

$V_{G_1} = \{\langle s \rangle, \langle t \rangle\} \neq \{\langle s \rangle\} = V_{G_2}$  and  $L(G_1) = L(G_2) = \{0\}$

Question 2. (15 marks) Scheme

a. (5 marks) Complete the following definition. Your solution should be **linear** in the number of elements in the input list.

```
;; (reverse lst) returns the reverse of lst
;; e.g. (reverse '(1 2 3 4 5)) => (5 4 3 2 1)
;;      (reverse '(1 (2 3) 4 5)) => (5 4 (2 3) 1)
;; Pre : lst is a list
(define reverse
  (lambda (lst)
    (letrec ((reverse-acc
              (lambda (lst rev)
                (if (null? lst) rev
                    (reverse-acc (cdr lst) (cons (car lst) rev))))))
      (reverse-acc lst '()))))
```

b. (5 marks) Complete the definition of **bar** below, so that for all valid inputs  $x$  and  $y$ :

$$(\text{foo } x \ y) = ((\text{bar } x) \ y)$$

Prove / argue that your solution is correct.

```
(define foo
  (lambda (f x) (f x x)))
```

```
(define bar
  (lambda (f)
    (lambda (x) (f x x))))
```

```
(foo a b) = (a b b) = ((lambda (x) (a x x)) b) = ((bar a) b)
```

c. (5 marks) Complete the following definition. Your solution should use **one** call to a Scheme higher-order procedure in your solution.

```
;; (cond-apply f g lst) applies f to every element of lst, of which g is true
;; and returns the resulting list. That is, for lst = (x1 x2 ... xn),
;; return (y1 y2 ... yn), where yi = f(xi), if g(xi)
;; and yi = xi, otherwise
;; Pre: f, g - procedures applicable to every element of lst, g is a predicate
;; lst - a list
;; Example:
;; (cond-apply (lambda (x) (+ x 1)) positive? '(-1 2 -3 4)) ==> ;Value: (-1 3 -3 5)
(define cond-apply
  (lambda (f g lst)
    (map (lambda (x) (if (g x) (f x) x)) lst)))
```