CS 360 Online Test 1

CS 360 Online Exam Honesty Statement
I am fully aware that once I access exam problems I am allowed to work individually only 120 minutes on them, that I may use the textbook, lecture materials, and all other resources available via course website, but neither of the following is permitted: other books or materials, personal notes, web search tools, calculators, contacting other individuals. By making a submission of my answers to the instructor I acknowledge that I followed the statement of online honesty.

You have 120 minutes for working out exam problems, and still 15 more minutes for packaging your answers into pdf format and submitting them to the instructor via e-mail.

Choose five out of the following eight problems. Each problem counts for three points. Indicate what problems you select for grading in case you submit solutions to more than five problems.

1. Design a deterministic finite automaton, with a minimal number of states, recognizing the language consisting of all binary strings, which start with 10, contain 00, and end with 01. Justify correctness of your construction. Explain why your automaton has the minimal number of states.

2. Design a non-ambiguous grammar generating the language consisting of all binary strings, which contain an odd number of 0's and an odd number of 1's. Justify correctness of your construction. Explain why your grammar is non-ambiguous.
Hint: Construct your grammar out of the DFA (i) of figure 11 of weeks 1-2 file.

3. A Huffman tree constructed via the Scheme code presented in slides 8-15 of Week 3 Part 2 file is given. Implement in Scheme algorithms performing traversals of the Huffman tree, (i) in pre-order, (ii) in in-order, (iii) in post-order (check FCS index for definitions, if needed), and constructing lists representing encountered weights. Justify correctness of your implementations.
Hint: Modify the code of SICP Exercise 2.63.

4. Explain the role of the condition *if (leaf? next-branch)* from the point of view of the execution of the decoding algorithm. What is the depth of the recursive calls to *decode-1* during the execution of the decoding algorithm?

The following procedure implements the decoding algorithm. It takes as arguments a list of zeros and ones, together with a Huffman tree.

```
(define (decode bits tree)
  (define (decode-1 bits current-branch)
    (if (null? bits)
        '()
        (let ((next-branch
               (choose-branch (car bits) current-branch)))
          (if (leaf? next-branch)
              (cons (symbol-leaf next-branch)
                    (decode-1 (cdr bits) tree))
              (decode-1 (cdr bits) next-branch)))))
  (decode-1 bits tree))
```

```
(define (choose-branch bit branch)
  (cond ((= bit 0) (left-branch branch))
        ((= bit 1) (right-branch branch))
        (else (error "bad bit -- CHOOSE-BRANCH" bit))))
```

5. Explain the algorithm, as well as data structures and language mechanisms used, of the DFA simulation Scheme code of PLP Section 11.3.6.

```
(define simulate
  (lambda (dfa input)
    (letrec ((helper  ; note that helper is tail recursive,
               ; but builds the list of moves in reverse order
               (lambda (moves d2 i)
                 (let ((c (current-state d2)))
                   (if (null? i) (cons c moves)
                       (helper (cons c moves) (move d2 (car i)) (cdr i))))))))
      (let ((moves (helper '() dfa input)))
        (reverse (cons (if (is-final? (car moves) dfa)
                           'accept 'reject) moves))))))

;; access functions for machine description:
(define current-state car)
(define transition-function cadr)
(define final-states caddr)
(define is-final? (lambda (s dfa) (memq s (final-states dfa))))

(define move
  (lambda (dfa symbol)
    (let ((cs (current-state dfa)) (trans (transition-function dfa)))
      (list
       (if (eq? cs 'error)
           'error
           (let ((pair (assoc (list cs symbol) trans)))
             (if pair (cadr pair) 'error))) ; new start state
       trans                                ; same transition function
       (final-states dfa))))))              ; same final states
```

6. Define in Scheme an infinite stream consisting of all pairs of twin prime numbers, i.e. all pairs of prime numbers differing by 2, listed in increasing order according to the first number of a pair, in every pair the first item being smaller than the second item.
Hint: Apply the sieve of Eratosthenes in order to construct a predicate *is-prime?*.

7. Trace execution of the parsing algorithm of grammar (iii) of figure 23 of weeks 1-2 file with the parsing table (iv) of figure 24 of weeks 1-2 file applied to the string $(id + id) * id$ and draw the resulting parse tree. Follow the approach of figure 24 of weeks 1-2 file. Explain why grammar (ii) needed to be converted to grammar (iii) before constructing the parsing table (iv).

8. Explain the roles of states and the symbol • of the CFSM of PLP Figure 2.26 (Figure 2.25 in 3rd edition). What is the role of the parse stack of PLP Figure 2.30 (Figure 2.29 in 3rd edition)? How are states used in the process described in Figure 2.30 (Figure 2.29 in 3rd edition)? Why additional symbols are placed inside the input during the process? Your explanations should refer to shift and reduce actions.