CS270 - Final Study Guide

Students are responsible for all material from class, from the notes, the assigned reading, and from the homeworks. The exam is cumulative, in that the material after the midterm relies on knowledge from before the midterm; however, the focus of the exam will be on material after the midterm.

The exam will be held on Wednesday of finals week during your scheduled final exam. Students in the evening section have their exam at the regular class time during finals week. Students must take the exam at their scheduled time.

The exam must be taken individualy and is closed book, without computers. No crib sheets, cell phones, shoe phones, etc. Code needed for the exam will be provided so that it is not necessary to memorize code.

Topics

Students should be able to:

- Use natural deduction to prove that an argument is valid Natural Deduction and Proof by Contradiction). Proof Tactics, Strategies and Derived Rules).
- Provide specifications for Racket functions See lectures Specifications, Testing and Formal Verification and Elementary Metamathematics.
- Write an inductive proof for a property about a recursively defined data structure. See lectures Induction and Recusion and Induction.
- Use equational reasoning and induction to prove properties of Racket functions. See lectures Proving Properties of Recursive Lists Functions and Proving Properties of Recursive Functions and Data Structures.
- Prove termination of a Racket function. See lecture Proving Properties of Recursive Functions and Data Strucgures.

Practice Problems

The problems involve truth tables, equational reasoning, evaluation of Racket expressions, recursive functions and induction. These problems illustrate what we would expect you to be able to do. Solutions will be provided and we are happy to go over any of them and similar problems during the review session.

- Use a truth table to prove $(A \wedge B) \rightarrow C$ is equivalent to $A \rightarrow (B \rightarrow C)$.
- Use natural deduction to prove $(A \wedge B) \rightarrow C$ is equivalent to $A \rightarrow (B \rightarrow C)$.
- Use natural deduction to prove the validity of the cut rule, i.e. If $(A \vee B)$ and $(\neg A \vee C)$ then $(B \vee C)$
- The remainder of the questions use the following definitions.

```
BinTree := null | (value BinTree BinTree)
```

Define the following access functions to access the left and right children and value of a node.

```
(define (left T) (second T))
(define (right T) (third T))
(define (value T) (first T))
```

The number of nodes in a binary tree can be computed recursively

```
(define (N T) (if (null? T)
                  0
                  (+ 1 (N (left T)) (N (right T))))))
```

The height of a binary tree is the longest path from the root to a leaf node. It can be computed recursively as

```
(define (H T) (if (null? T)
                   -1
                  (+1 (max (H (left T))) (H (right T))))))
```

A full binary tree is a binary tree where every node has either 0 or 2 children. A complete binary tree of height h is a a binary tree of height h where every node has 2 children. A full binary tree can be defined recursively as

```
FullBinTree := (value null null) |
               (value FullBinTree FullBinTree)
```

Similarly a complete binary tree can be defined recursively as

```
CompleteBinTree := null|
                   (value CompleteBinTree CompleteBinTree)
```

where in this case the left and right trees must be the same.

- Write a Racket function to construct a complete binary tree of a given height.
- Use induction to prove that a complete binary tree of height h has 2^(h+1)-1 nodes. Hint explain how to generate a complete binary tree of height h from two complete binary trees of height h-1, and derive a recurrence relation for the number of nodes. Use the recurrence relation in your proof.
- Write Racket functions to compute the number of leaf nodes in a binary tree and the number of internal nodes in a binary tree.
- Let L = the number of leaf nodes in a full binary binary tree Let I = the number of internal nodes in a full binary tree Use induction to prove that L = I + 1. Hint derive a recurrence relation for the number of leave nodes and internal nodes in a complete binary tree in terms of the left and right children.

    - Show that the height of a binary tree with n nodes is between floor(log[2](n)) and n-1 for n=1..8. Then prove this using induction.
    - The number of binary trees can be computed from the following recurrence relation. Let T(n) be the number of binary trees with n nodes. T(n) = sum_{i=0}^{n-1} T(i)*T(n-1-i), n > 0, T(0) = 1. Use induction to prove that T(n) ≤ n^n.
    - Write a Racket function to compute the number of binary trees with a given number of nodes. Hint: use range to generate the indices in the sum and then use map and reduce. This would be an extra credit problem.
- Use induction and the following definitions to prove (LTE x L) ∧(0 < i ≤ (length L)) → (≤ x (nth i L))

```
(define (LTE x L)
  (if (null? L)
      #t
      (and (<= x (first L))
           (LTE x (rest L)))))


(define (nth n L)
  (cond
    [ (null? L) null ]
    [ (= n 1) (first L) ]
    [else (nth (- n 1) (rest L)) ]
))
```