CS 250 Final Review

mguiney

December 2018

Contents

1 Exam 1 Review

Note: we won't have questions from this material, but we may need to use it for questions from later sections, so I'm circling back to it once finished with more relevent sections.

1.1 Section 1.1: Proof Primer Logic

Content coming soon

1.2 Section 1.2: Sets and Set Operators

Content coming soon

1.3 Section 1.4: Graphs and Trees

Content coming soon

1.4 Section 2.1: Functions, Definitions, and Examples

Content coming soon

1.5 Section 2.2: Composition of Functions

Content coming soon

1.6 Section 2.3: Properties and Applications

Content coming soon

2 Section 2.4: Countability

2.1 Cardinality Notation

Given a set "A", A's cardinality is denoted as |A |

The expression |A |= |B |indicates that there is bijection between A and B.

The expression |A |<= |B |indicates that there is injection from A to B

2.2 General Concepts

Sets "A" and "B" have the same cardinality IFF there is a one-to-one correspondance (i.e. a Bijection) from A to B.

IF a set is finite or has the same cardinality as **N**, then it is called **countable**, otherwise it is **uncountable**

In mathematics, the **cardinality** of a set is a measure of the number of elements of the set. The cardinality of the natural numbers **N** is denoted aleph-null

If the cardinality of some set "S" is equal to that of the set of natural numbers. then the set S is called **countably infinite**.

2.3 Cardinality Proof Example

Prove: that the set of positive rational numbers is countable.

$$E = \{n \mid n \bmod 2 = 0\}$$

Let us define: f: **N** –> E as f:(n) = 2n.

- f maps N to E
- the defined function f is a one-to-one correspondence

Therefore: f is countable.

2.4 Cantor Diagonalization Proof Method

Prove: that the set of positive rational numbers is countable.

Q = { m/n — m, n $\in N$}

RETURNING TO THIS BIT LATER BECAUSE I HATE FORMATTING THIS PROOF

2.5 Cardinality of Integers Proof

The set of **Z** is countable, so it has the same cardinality as **N**.

Prove: there is a one-to-one correspondence from **N** to the Integers.

f(n) = n/2 if **N** is even and -(n-1)/2 if odd.

- f maps N to E
- the defined function f is a one-to-one correspondence

Therefore: the function f is countable, as f maps the set of integers.

2.6 Some Countability Results

1. Subsets and images of countable sets are also countable.
2. Set "S" is countable IFF |S |<= |**N** |

3. **N** x **N** is countable. This can be proved by using Cantor's Bijection to associate (x,y) with ((x+y)$^2 + 3x + y)/2$

2.7 Countability Fun Facts (wow so fun)

- The set **R** of real numbers is not countable.
- **Cantor's Result:** |A |< |power(A) |for any set A.
  - Power(**N**) is uncountable because |**N** |< |power(**N**)|.
  - The power set of **N** has the same cardinality as **R**.
- Most of the other sets that we have used thus far are countable:
  - **N** is a subset of **R**, but **R** is not a subset at **N**

- **Q** is a subset of **R**, but **R** is not a subset at **Q**
- **N**, **Z**, and **Q** are all the same, which is to say aleph-null
- Every finite set "A" of elements from N is a subset of N, so |A |< |N |

• The cardinality of the set of real numbers (**R**) js not the same as that of **N**

3 Section 3.1: Inductively-defined Sets

3.1 Inductive Definition Components

An inductively-defined set "S" has three primary components:

1. **Basis:** Specify one of more elements of S (having more than one of these is fine).

2. **Induction:** Specify one or more rules to construct elements of S from existing elements of S.

3. **Closure:** Specify that no other elements are in S (this step is always implicit)

The basis elements and the induction rules are called **constructors**.

3.1.1 Inductively-defined Set Example #1

Problem: Find an inductive definition for S = {3, 16, 29, 42...}
Solution:

1. **Basis:** 3 $\in$ S

2. **Induction:** If x $\in$ $3, then x + 13 \in$ 3. The constructors are 3, and the operation of adding 13.

3.1.2 Inductively-defined Set Example #2

Problem: Find an inductive definition for S = {3, 16, 29, 42, ...}
Solution: To simplify, we might try the method of Ďivide and Conquer; by writing S as the union of more familiar sets, like this:

S = {3, 4, 5, 8, 9, 12, 16, 17, 20, 24, 33, ...} $\cup\{4, 8, 12, 16, 20, 24, ...\}$

1. **Basis:** 3, 4 $\in$ S

2. **Induction:**

```
If x $\in$ S then:
 if x is odd:
     2x-1 $\in$ S
 else:
     x + 4 $\in$
```

3.1.3 Example of Inductively Defined Sets of Strings

Problem: Find an Inductive definition for S = { λ, ac, aacc,aaaccc, ...} = / a$^n c^n | n \in$ **N**

Solution:

1. **Basis:** $\lambda \in$ S.
2. **Induction:** if x $\in$ S then axc $\in$ S.

NOTE: For strings, we start with a *middle* element, and inductive build outwards.

3.1.4 Example of Inductively Defined Sets of Lists

Problem: Describe the set S defined by:

1. **Basis:** $\langle 0 \rangle \in$ S
2. **Induction:** x $\in$ S implies cons(1, x) $\in$ S.

Solution: S = $\{\langle 0 \rangle, \langle 1, 0 \rangle, \langle 1, 1, 0 \rangle, ...\}$

3.2 Infix Notation for Lists and Cons

Notation: cons(h,t) = h :: t. Associate to the right. This means that "x :: y :: z = x :: (y :: z)" is equivalent to cons(x, const(y, z)).

3.2.1 Infix notation Example #1:

Problem: Find an inductive definition for S = { $\langle \rangle, \langle a, b \rangle, \langle a, b, a, b \rangle, ...\}$
Solution:
Basis: $\langle \rangle \in$ S.
Induction: x $\in$ S implies a :: b :: x $\in$ S (or cons(a, cons(b, x))), if you prefer.

3.2.2 Infix Notation Example #2 (A more confusing case)

Problem: Find and inductive definition for S = { $\langle \rangle, \langle \langle \rangle \rangle, \langle \langle \langle \rangle \rangle \rangle, ...\}$
Solution:
Basis: $\langle \ \rangle \in$ S.
Induction: x $\in$ S implies x :: $\langle \rangle \in$ S.

3.3 Notation for Binary Trees

Let:

- t(L, x, R) be the tree with root x, left subtree L, and right subtree R
- $\langle\ \rangle$ denote the empty binary tree.

If T = t(L, x, R), then root(T) = x, left(T) = L, and right(T) = R.

3.3.1 Binary Tree Notation Example

Problem: Describe the set "S" defined inductively as follows:

1. **Basis:** $t(\langle\ \rangle, \cdot, \langle\ \rangle) \in S$.
2. **Induction:** $T \in S$ implies $t(T, \cdot, t(\langle\ \rangle, \cdot, \langle\ \rangle)) \in S$.

4 Section 3.2: Recursively Defined Functions and Procedures

4.1 What I need to cover

- Graph traversals
  - binary trees
  - binary search trees
  - depth-first search
- Search type definitions for:
  - preorder traversal
  - inorder traversal
  - postorder traversal

4.2 Recursive definitions and Procedures

In a **recursive definition**, an object is defined in terms of itself. Sequences, sets, and functions can all be defined recursively.

A function "f" is **recursively defined** if at least one value f(x) is defined in terms of another value f(y), where x != y. **Recursive procedures** are much the same, but define things in terms of P rather than f.

4.3 Recursively defining functions and procedures:

There are two steps to the technique used in the creation of recursive definitions:

1. Specify a value f(x) or action P(x), for each basis element x of S.

2. Specify rules that, for each inductively-defined element x in S, define f(x) or P(x) in terms of previously defined values of f or P.

4.3.1 Recursive definition of functions example

Problem: Find a recursive definition of for the function f:$\mathbf{N} \rightarrow \mathbf{N}$, as defined by:

$$f(n) = 0 + 3 + 6 + ... + 3n$$

Solution: **N** is an inducively defined set, so we need to give f(0) a value in **N**, and we need to define f(n+1) in terms of f(n). Let's iterate through a few of our terms and see what we find.

1. f(0) = 0
2. f(n+1) = (0 + 3 + 6 + ... + 3n) + 3(n+1)
3. f(n+1) = (0 + 3 + 6 + ... + 3n) + 3(n+1) + 3(n+2)

Using this, we can piece together the recursive definition for f:
f(0) = 0
f(n+1) = f(n) + 3(n+1)

4.4 Strings and Languages Review, String Recursion

- An **alphabet** "A" is a given finite set of symbols.
- A **string** "w' over alphabet A is a finite-length sequence of elements of A.
  - The concatenation of strings "x" and "y" is denoted "xy"
  - Concatenation of x n-many times is x^n
- The length of a given string "w" is |w |.
- The empty string is Λ, and |Λ |= 0.
- The set of all strings over A is A∗ (includes Λ)
- a **language** "L" is a subset of A∗, which is to say it is a set of strings.
- The **lexicographic ordering** of strings is the same as the dictionary ordering. In **short-lex**, the shorter strings precede the longer ones.

Given a set "A" and a string "a":

- A^n != a^n
  - A^2 = A × A
  - a^2 = aa
- $\emptyset$, Λ, and { Λ } are all different
- $|\emptyset$ $|= 0$
- $|\emptyset *$ $|=$ $|\Lambda$ $|= 1$
- Concatenation rules:
  - (xy)z = x(yz)
  - xΛ = Λx = x
  - $a^n a^n = a^m + n$

4.4.1 Recusively-defined Strings Example

Problem: Find a recursive definition for cat: $A* \times A* \rightarrow A*$.
Solution:

1. $A*$ is inductively defined: $\Lambda \in A*$; a $\in$ A and x $\in$ $A*$ imply that ax $\in$ $A*$, where ax denotes the string version of cons.

2. Define **cat** recursively using the first arg. The definition is:
   - cat(Λ, t) = Λt = t
   - recursive component: cat(ax, t) = axt = a(xt) = a(cat(x,t))

This gives us our final recursive string definitions:

- cat(Λ, t) = t
- cat(ax, t) = a(cat(x, t))

4.5 Lists Review, ists recursion

- Operations on lists:
  - head($\langle$a, b, a, c $\rangle$) = a
  - tail($\langle$a, b, a, c $\rangle$) = $\langle$ b, a, c $\rangle$
  - cons($\langle$a, $\langle$ b, a, c $\rangle$ $\rangle$) = $\langle$a, b, a, c$\rangle$
- The set of lists whose elements are in A is denoted by lists(A)

4.6 Graph Traversals

- A **graph traversal** starts at some vertex "v" and visits all yet-unvisited vertices on the paths that start at v.
- vertices are not revisted
- Any traversal of a connected graph will visit all of its vertices

Graph-traversal Methods

- Breadth-first search
  - Search across levels.
  - Breadth-first algo
    1. Start at root
    2. Explore all neighboring nodes until all nodes have been visited.
- Depth-first Search
  - Search to each leaf (unvisited graph node) as fast as possible
  - Depth-first algo
    1. Start with some arbitrary node as your root.
    2. traverse graph until you hit all of the nodes with no children
    3. backtrack along the path from the original root node.

4.7 Traversing Binary Trees

There are 3 types of Binary tree traversal that are often used:

- Preorder Traversal
  - preorder(T): if T != $\langle\ \rangle$, then visit root(T); preorder(left(T)); preorder(right(t)); fi
  - process node data, traverse left, traverse right
  - List each vertex the first time it is encountered.
- Inorder Traversal
  - inorder(T): if T != $\langle\ \rangle$ then inorder(left(T)); visit root(T); inorder(right(t)); fi
  - traverse left, process node data, traverse right
  - List each leaf when initially encountered, and all other nodes the second time it is passed (i.e. when we process its data)

- Postorder Traversal
  - postorder(T): if t != $\langle\ \rangle$, then postorder(left(T)); postorder(right(T)); visit root(T); fi
  - traverse left, traverse right, process node data.
  - List each vertex the last time it is encountered.

4.8 Infinite Sequences

We can construct recursive definitions for infinite sequences by defining a value f(x) in terms of x and f(y) for some value y in the sequence.

NOTE: I'll circle back and add an example here if i have time, but it is pg 81 in the slides