CS 145 – Foundations of Computer Science

Professor Eric Aaron

Lecture – T Th 9:00am
Lab – F 10:30am

Lecture Meeting Location: SP 309
Lab Meeting Location: SP 309

Business

• HW5 out—extended deadline: Nov. 16 / Nov. 17 (extended from Nov. 11 / Nov. 12, on the HW sheet)
• HW6 out today, due Nov. 23 / Nov. 24 (see HW sheet)

• Lab due by end of the day today

• Reading: Makinson, Ch.4.1-4.6
  – Our coverage of the material will be different from that in the textbook, but it’s good to see the textbook’s presentation, as well
• Also Reading: Makinson, Ch. 5
• Document on structural induction now available from course website (follow the Additional Notes link)
Exercise: Might As Well Be Postage Stamps

• Claim: Every positive integer \( n \geq 14 \) can be written as a sum of 3s and 8s
  – i.e., there exist some \( x, y \) s.t. \( n = 3x + 8y \)

• Proof: By induction!
  – (What’s the proof?)

Proving Statements about Recursively Defined Sets

• A related kind of induction is called structural induction, which can be used to prove claims about all items constructed by a recursive definition.

• To prove property \( P \) holds for all elements of a recursively defined set:
  – Base case(s): Show that \( P \) holds for every element in the basis for the recursive definition.
  – Inductive case(s): Show that every constructor in the definition preserves property \( P \).

• Recall the definition of transitive closure:
  – \( R_0 = R \);
  – \( R_{n+1} = R_n \cup \{ (a,c) \mid \exists x \text{ s.t. } (a,x) \in R_n \text{ and } (x,c) \in R \} \);

• Claim: In the above definition, \( R \) is a subset of \( R_i \) for all \( i \). Prove by structural induction.
Proving Statements about Recursively Defined Sets

• A related kind of induction is called *structural induction*
  – Base case(s): Show that P holds for every element in the basis for the recursive definition.
  – Inductive case(s): Show that every *constructor* in the definition preserves property P.

• Recall our recursive definition of propositional logic expressions
  – Base: Given an initial set A of propositional letters (e.g., p, q, r, ...), all elements of A are propositional logic expressions
  – Induction: If P, Q are propositional logic expressions, then the following are also propositional logic expressions (note that the parentheses are part of the expressions)
    * (¬P); (P ^ Q); (P v Q); (P → Q); (P ↔ Q)  *(Note: 5 constructors)*

• Claim: All propositional logic expressions contain an even number of parentheses. (We consider 0 to be an even number.) Prove by structural induction.
Counting

• Counting things is an important part of Computer Science problem solving
  – The number of elements in a set, or subsets of a set
  – The number of routes through a map that visit every city
  – The number of words of length $L$ formed from alphabet $A$

  – Can be important for: *brute-force* solutions (i.e., exhaustively enumerating every possibility under some circumstances), *probability*, etc.

Count On It!

• More principles for counting elements of sets
• Subtraction principle for (finite set) difference
  – $|A - B| = …$?
• Addition principle for finite sets
  – $|A \cup B| = …$?
Count On It!

- More principles for counting elements of sets
- Subtraction principle for (finite set) difference
  - $|A - B| = |A| - |A \cap B|$
- Addition principle for finite sets
  - $|A \cup B| = |A| + |B| - |A \cap B|$
  - If $A$, $B$ are disjoint, then $|A \cup B| = |A| + |B|$ (do you see why?)

- Example application: (calling it that sounds better than calling it "Practice Counting")
  - A logic class has 20 students who also take calculus, 9 who also take philosophy, 11 who take neither, and 2 who take both calculus and philosophy. How many students are in the class?

- (The next slides are for Friday’s mini-lecture)
A Note On Scheme:
(apply append (…))

• In Scheme, the apply function takes a function F and a list L = (L1 .. Lk), and it applies function F with arguments L1 ... Lk.
  – That is, the elements of the list become the arguments of F
• This can be useful when F is the append function—then, (apply append (…)) can “flatten” a list
• Examples:
  – (apply append '(a b c) (d e f) (g h) (i) (j k l)))

A Note On Scheme:
(apply append (…))

• In Scheme, the apply function takes a function F and a list L = (L1 .. Lk), and it applies function F with arguments L1 ... Lk.
  – That is, the elements of the list become the arguments of F
• This can be especially useful when working with the result of the map function, which returns a list
• Example:
  – (apply append (map (lambda (x) (list x (* 10 x))) '(1 2 3 4 5)))
A Note On Scheme:
(apply append (…))

• In Scheme, the `apply` function takes a function F and a list L = (L1 .. Lk), and it applies function F with arguments L1 ... Lk.
  – That is, the elements of the list become the arguments of F

• This can be especially useful when working with the result of the map function, which returns a list

• Example:
  – `(apply append (map (lambda (listy)
                            (map (lambda (elt)
                                  (list elt elt))
                             listy))
                 '((1 2 3) (4 5) (6 7 8))))`