CS3203 #4

6/7/04

Janak J Parekh

Administrivia

- HW#1 due, HW#2 will be out right after class
- Still need to confirm Aaron’s office hour; I’ll update the website after I talk to him tonight

Proof strategies

- Forward reasoning: either start with hypotheses (direct) or negation of conclusion (indirect)
  - We've done this several times
- Backward reasoning: To prove a statement *q*, find a statement *p* that we can prove with the property that $p \rightarrow q$.
  - Example: Prove that the square of every odd integer has the form 8k+1, where k is an integer.

Example…

We begin by taking an odd integer n, which must have the form $n = 2i + 1$ for some integer i. Then $n^2 = (2i+1)^2 = 4i^2 + 4i + 1$. We need to show that this has the form $8k + 1$. Reasoning backwards, this would follow if we could show that $4i^2 + 4i$ can be written as $8k$.

But $4i^2 + 4i = 4i(i+1)$. Note that $i(i+1)$ is the product of two consecutive integers. Since every other integer is even, either i or $i+1$ is even. Therefore the product $i(i+1)$ is even, and hence can be written as $2k$ for some integer k. Therefore

$$4i^2 + 4i = 4i(i+1) = 4(2j) = 8j \quad \longleftarrow (\text{this is a multiple of } 8)$$

Since we can write $4i^2 + 4i = 8k$, it follows that

$$n^2 = 4i^2 + 4i + 1 = (4i^2 + 4i) + 1 = 8k + 1$$

Proofs continued

- A hint to use proof by cases is when there's "extra information" in the problem
  - Book example: show that there are no solutions in integers *x* and *y* of $x^2+3y^2=8$.
- Adapt existing proofs when you can.

Conjecture and proof

- How do you come up with a concept *worth* proving in the first case?
- *Conjectures* are formed, and are settled either by proof or counterexample.
  - Book gives an example of Mersenne primes, where 2^p-1; can we conclude anything about a^n-1 in general?
  - Book proves that the integer a^n-1 is composite when $a > 2$ or when $a = 2$ and n is composite.

Sometimes conjectures are a little tricky…

- Euler conjectured that for every integer *n*, n >= 3, the sum of n-1 *n*th powers of positive integers cannot be an *n*th power itself.
  - Relatively easy to find counterexample for n=5.
  - For n = 4, though, $95800^4+217519^4+414560^4 = 422481^4$.
  - No one has found a counterexample for n=6. It might be true!

And there are open problems…

- Fermat's last theorem (from 17^{th} century) was not solved until the mid-90s.
- Goldbach's conjecture: every even integer *n*, n > 2, is the sum of two primes. Never been disproven or proven; has been checked for all positive integers up to $4(10^{14})$.

Halting problem

- Very simple proof by contradiction
- Given a procedure called H(P, I) where P is program and I is input, and returns “halt” if the program halts and “loops forever” otherwise
- Construct a procedure K(P) which calls H(P,P). If H(P,P) “halt”s, K loops forever. If H(P,P) “loops forever”, K halts.
- Now, what happens if we call K(K)?
- Need more precise definition of input to make proof rigorous – definition of a “Turing machine” useful in this regard (beyond scope of this class)

Sequences and Summations

- We defined an *ordered tuple* two weeks ago; sequences are a more function-oriented concept
- A **sequence** is a function from the subset of the set of integers (usually either non-negative or positive) to a set *S*. We use the notation a_n to denote the image of the integer *n*. We call a_n a *term* of the sequence.
  - Discrete structure used to represent an ordered list.
- We use the notation $\{a_n\}$ to describe a sequence.
  - If $a_n = 5n$, then what's the result?

More definitions…

- A **geometric progression** is a sequence of the form a, ar, ar^2, …, ar^n with *initial term* a and *common ratio* r being real numbers.
  - $\{b_n\}$ with $b_n = (-1)^n$.
- An **arithmetic progression** is a sequence of the form a, a+d, a+2d, …, a+nd where *initial term* a and the *common difference* d are real numbers.

Strings and integers

- A **string** is a sequence of the form $a_1a_2\ldots a_n$. **Length** of the string S is the number of terms in the string. **Empty string** is denoted by λ and has length zero.
- Lots of special integer sequences.
- Often, the goal is to find a formula or general rule for constructing the terms of a sequence.
  - Look for patterns.

Examples

- Find formulas for:
  - 1, 2, 1, 2, 1, 2, …
    - Either $a_n = 1.5+0.5(-1)^n$ or (a **mod** 2) + 1
  - 0, 2, 6, 12, 20, 30, 42, …
    - $a_n = n^2 - n$
- Not always trivial to produce a formula
- Some useful common sequences on page 228

Summations

- Goal: add a_m, a_{m+1}, … , a_n together.
- Both compact and regular notation
- *j* is **index** of summation, **lower limit** *n*, **upper limit** n.

Summations cont'd

- Sums of terms of geometric progressions are called **geometric series**. If a and r are real numbers and r not 0, then

$$\sum_{j=0}^{n} ar^{j} = \begin{cases} \dfrac{ar^{n+1} - a}{r - 1} \text{ if } r \neq 1 \\ (n+1)a \text{ if } r = 1 \end{cases}$$

- Double summations are processed inside-out (expand the inner summation first).
- Set notation is also possible; just state s $\in$ S at the bottom of the summation, and nothing at the top.
- Useful summation formulas on page 232; what are they?

Cardinality

- Generalize for non-finite sets.
- Sets *A* and *B* have the same *cardinality* iff there is a 1-to-1 correspondence from A to B.
- A set that is either finite or has the same cardinality as the set of positive integers is considered *countable*. Those that aren't are called *uncountable*.
  - Show that the set of odd positive integers is a countable set.
  - Infinite sets are only countable if it is possible to list the elements of the set in a sequence – needed for 1-to-1 correspondence.

Cardinality (II)

- The set of positive rational numbers are actually countable – “diagonalization”.
- The set of real numbers is *not*countable.
  - Basic idea: show there’s always a real number that’s not “in the list”.

Induction

- When you have sequences or summations, induction is a useful proof technique.
  - For example, prove that the sum of the first *n* positive odd integers is n^2.
- Actually quite straightforward to do. Note it's not a tool for discovering formulas or theorems.
- Principle: if the concept being proven can be generically captured as a single step in a long series, and if it already happened for the first n steps, showing it holds for n+1 shows it to be sufficient for all n.
  - Dominoes?

Mechanics

- Used to show propositions of the form ∀nP(n) where the universe of discourse is the set of positive integers.
- Basis: show proposition P to be true for P(1).
- Inductive step: Show that the implication P(k) → P(k+1) to be true.
  - Therefore, the **inductive hypothesis** P(k) is assumed to be true.
- In other words,

$$\left[P(1) \wedge \forall k\left(P(k) \rightarrow P(k+1)\right)\right] \rightarrow \forall n P(n)$$

Notes

- Need to show that P(k+1) cannot be false when P(k) is true.
- Does not assume that P(k) is true for all integers! "If it is assumed P(k) is true, then P(k+1) is also true." Therefore, induction isn't circular reasoning.
- What we're going to do is to take the equation for the *k*th term, add the *k+1*th term, substitute the answer for the *k*th term for the previous elements, and show the entire result is the answer for the *k+1*th term.
- Let's do our first example now.

More examples, inequalities

- Use induction to prove that

$$1+5+5^2+5^3+\ldots+5^n=\frac{5^{n+1}-1}{4}, n\geq 0$$

  - Just change basis step!
- Use induction to prove that $n < 2^n$ for all positive integers *n*.
  - Yes, inequalities work too – a bit simpler, sometimes.
- Lots more examples in the book.

Strong induction

- This time, we assume that P(j) is true for j = 1, …, k and show that P(k+1) must also be true.
  - That is, we can assume j = 1 is true, and j = 2 is true, and so on, and use intermediate results.
  - Actually, the two are equivalent (i.e., each can be shown to be a valid proof technique assuming the other.)
- Example 14 in book: if *n* is an integer greater than 1, then *n* can be written as the product of primes.
  - Basis: 2 is prime.
  - Inductive step: Assume P(j), and show P(k+1) is true.
  - If k+1 is prime, done.
  - If k+1 composite, it's the product of two integers, a and b, which are both less than k+1. Since a and b are assumed true under strong induction, they must themselves be the product of primes (i.e., factorization of a and b).

Why does this work?

- Well-ordering property: Every nonempty set of nonnegative integers has a least element.
- If so, and we know P(1) is true, let's do a proof by contradiction.
  - Let's say there exists at least one positive integer for which P(n) is false.
  - Then, the set S of positive integers for which P(n) is false is nonempty.
  - Therefore, S must have a least element, which we'll call m, for which P(m) is false.
  - We know that m cannot be 1.
  - Sinceu m is > 1, m-1 is a positive integer not in S.
  - But if P(m-1) → P(m) has been shown as part of the proof, then P(m) is true.
  - Contradiction!

Recursive definitions and structural induction

- Fundamental idea of **recursion**: define something in terms of itself (often a smaller version of itself).
- We can use this mechanism to define sequences, functions, and sets.
  - For example, define the sequence of powers of two as $a_0 = 1$ and $a_n = 2a_n$.
  - As it suggests, a version of induction (called structural induction) can be used to prove results.

Recursively defined functions

- For a function whose domain is the set of nonnegative integers:
  - Basis: specify the value of the function at zero
  - Recursive step: Give a rule for finding its value at an integer from values at smaller integers.
  - *Recursive* or *inductive* definition
- Examples
  - Give an inductive definition of F(n) = n!.
  - Recursive definition of a^n?

Common recursive sequences, functions

- Fibonacci numbers
  - $f_0 = 0$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$
  - For n = 2, 3, 4…
- Euclidean algorithm, which we saw before.
  - Book now has proof showing that Euclidean algorithm has a running time that's logarithmic in # of digits.

Recursively defined sets and structures

- Again, a basis and recursive step
- Occasionally, an **exclusion rule**: nothing else in the set other than elements specified in basis or recursions
  - Generally assumed
- Examples
  - 1.
    - Basis: $3 \in S$.
    - Recursive step: If $x \in S$ and $y \in S$, then $x+y \in S$.
  - 2. Give a recursive definition for S = {4, 7, 10, 13, 16, 19, …}

String operations

- The set Σ^* of strings over the alphabet Σ can be defined recursively by
  - Basis step: $\lambda \in \Sigma^*$ (λ is the empty string containing no symbols)
  - Recursive step: If $w \in \Sigma^*$ and $x \in \Sigma$ then $wx \in \Sigma^*$.
  - Much like power set.
  - Example: the set where $\Sigma = \{0,1\}$
- Concatenation
  - Let Σ be a set of symbols and Σ^* be the set of strings formed from symbols in Σ. We define the concatenation of two strings, denoted by a $\cdot$, recursively as follows.
  - Basis: If $w \in \Sigma^*$, then $w \cdot \lambda = w$
  - Recursive step: if $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.
- Length
  - Basis: $l(\lambda)=0$;
  - Recursive step: $l(wx) = l(w)+1$ if $w \in \Sigma^*$ and $x \in \Sigma$.
- A number of other examples in the book.

Trees

- Will be studied more extensively later, but useful in context of recursion.
- A tree is a special type of graph with *no cycles* (a graph is a data representation with vertices and edges).
- The set of *rooted* trees is defined recursively by having a basis being a vertex *r*, the root, and the recursive step is having any existing tree and creating a new root for it.

Binary trees

- For a binary tree:
  - Basis step: The empty set is an extended binary tree.
  - Recursion: If T_1 and T_2 are extended binary trees, denoted by $T_1 \cdot T_2$, consisting of a root *r* together with edges connecting to the root to each of the roots of the subtrees when those trees are nonempty.
- The set of *full* binary trees can be defined recursively by these steps
  - Basis: There is a full binary tree consisting only of a single vertex r.
  - Recursion: If T_1 and T_2 are full binary trees, $T_1 \cdot T_2$ is the tree with a new root r with edges leading to each subtree T_1, T_2.

Structural induction

- Instead of using mathematical induction directly to prove results about recursively defined sets, we use a more convenient form.
  - Basis step: show that the result holds for all elements specified in the basis step of the recursive definition to be in the set.
  - Recursive step: show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

Example of structural induction

- Prove that l(xy), where l is the length of the string.
  - P(y) is l(xy) = l(x)+l(y) where x and y belong to Σ^*.
  - Basis step: show that P(λ) is true. Since l(xλ)=l(x)=l(x)+0=l(x)+l(λ) for every string x, P(λ) is true.
  - Recursive step: Assume P(y) is true, show P(ya) whenever a$\in\Sigma$. By the definition of l(w), l(xya) = l(xy)+1 and l(ya) = l(y)+1. Since l(xy) = l(x)+l(y) by the hypothesis, l(xya) = l(x)+l(y)+1 = l(x)+l(ya).
- Generalized induction: use well-ordered property on other sets than integers – we'll look at this later

Recursive algorithms

- An algorithm is called *recursive* if it solves a problem by reducing it to an instance of the same problem with smaller input.
- We implicitly did these in the previous section
- How about a^n?

  procedure power(a: nonzero real number, n: nonnegative integer)

  if n = 0 **then** *power(a,n)* := 1

  else *power(a,n)* := $a \cdot$*power(a, n-1)*

And more

- Linear search recursively?

  procedure search(i,j,x)

  if a_i = x **then**

  location := i

  else if i = j **then**

  location := 0

  else

  search(i+1,j,x)
- Recursive algorithm to find the sum of the first *n* positive integers?

Recursion vs. iteration

- I’ve hinted at this before.
- Recursion defines the problem in terms of itself and “works down”, whereas iteration “works up” to the answer from 1.
- Generally, iteration requires less computation than the recursive equivalent
- Best illustrated by recursive vs. iterative fibonacci (next page)
  - In fact, recursive fibonacci is exponential in complexity
  - There are ways of making recursion faster

Fibonacci algorithms

```
procedure recursivefib(n: nonnegative
    integer)
if n = 0 then fibonacci(0) := 0
else if n = 1 then fibonacci(1) := 1
else fibonacci(n) := fibonacci(n-1) +
    fibonacci(n-2)

procedure iterativefib(n: nonnegative
    integer)
if n = 0 then y := 0
else begin
     x := 0
     y := 1
     for i := 1 to n-1; begin
          z := x+y
          x := y
          y := z
     end
end
```

Mergesort

procedure mergesort(L = a_1, …, a_n)
if n > 1 **then**
    m := floor(n/2)
    L_1 := a_1, …, a_m
    L_2 := a_{m+1}, a_{m+2}, …, a_n
    L := merge(mergesort(L_1, L_2))
{L is now sorted into elements in nondecreasing order}

procedure merge(L_1, L_2: lists)
L := empty list
while L_1 and L_2 are nonempty
begin
    remove smaller of first of L_1 and L_2 and put it at the left end of L
    if this makes one list empty copy all elements of other list and append them to L
end {L is the merged list with elements in increasing order}

Cost of mergesort

- Merging two lists with m and n elements, respectively, takes m+n-1 comparisons.
- Since we split the lists, we're going to execute a logarithmic number of merge calls.
- As a result, the number of comparisons needed to merge sort a list with *n* elements is O(*n* log *n*).
- More precise proof in the book

Next time

- Counting