Recursion and Structural Induction

CS231
Dianna Xu

Fibonacci Sequence

• Definition
  – Non-recursive: \( F(n) = \left\lfloor \frac{1 + \sqrt{5}}{2} \right\rfloor - \left\lfloor \frac{1 - \sqrt{5}}{2} \right\rfloor \)
  – Recursive: \( F(n) = F(n-1) + F(n-2) \)

• Must always specify base case(s):
  – \( F(1) = 1, F(2) = 1 \)

Fibonacci Sequence in Java

```java
int F(int n) {
    if ((n == 1) || (n == 2))
        return 1;
    else
        return F(n-1) + F(n-2);
}
```

Bad Recursive Definitions

• Consider:
  – \( f(0) = 1 \)
  – \( f(n) = 1 + f(n-2) \)
  – What is \( f(1) \)?

• Consider:
  – \( f(0) = 1 \)
  – \( f(n) = 1 + f(-n) \)
  – What is \( f(1) \)?

Defining Sets via Recursion

• Three components:
  1. Base
  2. Recursion
  3. Restriction: nothing else belongs to the set other than those generated by 1 and 2

• Example: the set of positive integers
  – Base: \( 1 \in S \)
  – Recursion: if \( x \in S \), then \( x+1 \in S \)

Recursively Defined Sets

• The set of odd positive integers
  – \( 1 \in S \)
  – If \( x \in S \), then \( x+2 \in S \)

• The set of positive integer powers of 3
  – \( 3 \in S \)
  – If \( x \in S \), then \( 3^x \in S \)

• The set of polynomials with integer coefficients
  – \( 0 \in S \)
  – If \( p(x) \in S \), then \( p(x) + cx^n \in S \)
    • \( c \in \mathbb{Z}, n \in \mathbb{Z} \) and \( n \geq 0 \)
Recursive String Definition

• Terminology
  – \( \lambda \) is the empty string: “”
  – \( \Sigma \) is the alphabet, i.e. the set of all letters: \{ a, b, c, ..., z \}
• We define a set of strings \( \Sigma^* \) as follows
  – Base: \( \lambda \in \Sigma^* \)
  – If \( w \in \Sigma^* \) and \( x \in \Sigma \), then \( wx \in \Sigma^* \)
  – Thus, \( \Sigma^* \) is the set of all the possible strings that can be generated with the alphabet

Defining Strings via Recursion

• Let \( \Sigma = \{0, 1\} \)
• Thus, \( \Sigma^* \) is the set of all binary numbers
  – Or all binary strings
  – Or all possible machine executables

Length of a String

• How to define string length recursively?
  – Base: \( \text{len}(\lambda) = 0 \)
  – Recursion: \( \text{len}(wx) = \text{len}(w) + 1 \) if \( w \in \Sigma^* \) and \( x \in \Sigma \)
• Example: \( \text{len}(“aba”) \)
  – \( \text{len}(“aba”) = \text{len}(“ab”) + 1 \)
  – \( \text{len}(“ab”) = \text{len}(“a”) + 1 \)
  – \( \text{len}(“a”) = \text{len}(“”) + 1 \)
  – \( \text{len}(“”) = 0 \)
  – Output: 3

Palindromes

• Give a recursive definition for the set of strings that are palindromes
  – We will define set \( P \), which is the set of all palindromes
  • Base:
    – \( \lambda \in P \)
    – \( x \in P \) when \( x \in \Sigma \)
  • Recursion: \( xp \in P \) if \( p \in P \), \( x \in \Sigma \), \( p \in \Sigma^* \)

Recursion vs. Induction

• Consider the recursive definition for factorial:
  – \( f(0) = 1 \)
  – \( f(n) = n \cdot f(n-1) \)
• Consider the set of all positive integers that are multiples of 3
  – \( \{ 3, 6, 9, 12, 15, ... \} \)
  – \( \{ x \mid x = 3k \text{ and } k \in \mathbb{Z}^+ \} \)
  • Recursive definition:
    – Base: \( 3 \in S \)
    – Recursion: if \( x \in S \) and \( y \in S \), then \( x+y \in S \)
### Proof

- Prove that $S$ contains all positive integers divisible by 3
- Let $P(n) = 3n$, $n \geq 1$, show $3n \in S$
  - **Base case:** $P(1) = 3 \times 1 \in S$
    - By the base of the recursive definition
  - **Inductive hypothesis:** $P(k) = 3^k \in S$
  - **Recursive step:** $P(k+1) = 3^k + 1 \in S$
    - $3^k + 1 = 3^{k+1}$
    - $3^k \in S$ by the inductive hypothesis
    - $3 \in S$ by the base case
    - Thus, $3^{k+1} \in S$ by the recursive definition

### What did we just do?

- Notice what we did:
  - Showed the base case
  - Assumed the inductive hypothesis
  - For the recursive step, we:
    - Showed that each of the “parts” were in $S$
      - The parts being $3k$ and $3k+3$
    - Showed that since both parts were in $S$, by the recursive definition, the combination of those parts is in $S$
      - I.e., $3k+3 \in S$
- This is called structural induction

### Structural Induction

- A more convenient form of induction for recursively defined “things”
- Used in conjunction with recursive definitions
- Three parts:
  - **Base step:** Show the result holds for the elements in the base of the recursive definition
  - **Inductive hypothesis:** Assume $P(s-t) = P(s) + P(t)$
  - **Recursive step:** Show that the recursive definition allows the creation of a new element using the existing elements

### Structural Induction on Strings

- Part (a): Give the definition for $ones(s)$, which counts the number of ones in a bit string $s$
  - Let $\Sigma = \{0, 1\}$
  - **Base:** $ones(\lambda) = 0$
  - **Recursion:** $ones(wx) = ones(w) + x$
    - Where $x \in \Sigma$ and $w \in \Sigma^*$
    - Note that $x$ is a bit: either 0 or 1

### String Structural Induction Example

- Part (b): Use structural induction to prove that $ones(st) = ones(s) + ones(t)$
  - **Base case:** $t = \lambda$
    - $ones(s) = ones(s) + 0 = ones(s) + ones(\lambda)$
  - **Inductive hypothesis:** Assume $ones(s-t) = ones(s) + ones(t)$
  - **Recursive step:** Want to show that $ones(s \cdot t \cdot x) = ones(s) + ones(t \cdot x)$
    - Where $s, t \in \Sigma$ and $x \in \Sigma$
    - New element is $ones(s \cdot t \cdot x)$
    - $ones(s \cdot t \cdot x) = ones(s \cdot (t \cdot x))$ by associativity of concatenation
    - $= x \cdot ones(s \cdot t)$ by recursive definition
    - $= x + ones(s) + ones(t)$ by inductive hypothesis
    - $= ones(s) + (x + ones(t))$ by commutativity and assoc. of +
    - $= ones(s) + ones(t \cdot x)$ by recursive definition

### Induction Methods Compared

<table>
<thead>
<tr>
<th>Weak Mathematical</th>
<th>Strong Mathematical</th>
<th>Structural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used for</td>
<td>Usually formulate</td>
<td>Usually formulate not easily provable via mathematical induction</td>
</tr>
<tr>
<td>Assumption</td>
<td>Assume $P(k)$</td>
<td>Assume $P(1), P(2), \ldots, P(k)$</td>
</tr>
<tr>
<td>Assumptions</td>
<td></td>
<td>Assume statement is true for some “old” elements</td>
</tr>
<tr>
<td>What to prove</td>
<td>True for $P(k+1)$</td>
<td>True for $P(k+1)$</td>
</tr>
<tr>
<td>Statement</td>
<td></td>
<td>Statement is true for some “new” elements created with “old” elements</td>
</tr>
<tr>
<td>Step 1 called</td>
<td>Base case</td>
<td>Base case</td>
</tr>
<tr>
<td>Step 2 called</td>
<td>Inductive step</td>
<td>Inductive step</td>
</tr>
<tr>
<td>Step 3 called</td>
<td>Recursive step</td>
<td>Recursive step</td>
</tr>
</tbody>
</table>
Proof by Inductions

• Show that $F(n) < 2^n$
  – Where $F(n)$ is the $n^{th}$ Fibonacci number

• Fibonacci definition:
  – Base: $F(1) = 1$ and $F(2) = 1$
  – Recursion: $F(n) = F(n-1) + F(n-2)$

• Base case: Show true for $F(1)$ and $F(2)$
  – $F(1) = 1 < 2^1 = 2$
  – $F(2) = 1 < 2^2 = 4$

Via weak mathematical induction

• Inductive hypothesis: Assume $F(k) < 2^k$
• Inductive step: Prove $F(k+1) < 2^{k+1}$
  – $F(k+1) = F(k) + F(k-1)$
  – We know $F(k) < 2^k$ by the inductive hypothesis
  – Each term is less than the next, therefore:
    $F(k-1) < F(k)$
    • Thus, $F(k-1) < F(k) < 2^k$
    – Therefore, $F(k+1) = F(k) + F(k-1) < 2^k + 2^k = 2^{k+1}$ □

Via strong mathematical induction

• Inductive hypothesis: Assume $F(1) < 2^1$, $F(2) < 2^2$, …, $F(k-1) < 2^{k-1}$, $F(k) < 2^k$
• Inductive step: Prove $F(k+1) < 2^{k+1}$
  – $F(k+1) = F(k) + F(k-1)$
  – We know $F(k) < 2^k$ by the inductive hypothesis
  – We know $F(k-1) < 2^{k-1}$ by the inductive hypothesis
  – Therefore, $F(k) + F(k-1) < 2^k + 2^{k-1} < 2^{k+1}$ □

Via structural induction

• Inductive hypothesis: Assume $F(k) < 2^k$
• Recursive step:
  – Show true for “new element”: $F(k+1)$
  – $F(k+1) = F(k) + F(k-1)$
  – $F(k) < 2^k$ by the inductive hypothesis
  – $F(k-1) < F(k) < 2^k$
  – Therefore, $F(k) + F(k-1) < 2^k + 2^k = 2^{k+1}$
  – $F(k+1) < 2^{k+1}$ □