Section 8

CSE 311 BB/BD - Sp 2022

Allie Pfleger
Kasper Lindberg

Administrivia

Announcements and Reminders

- HW6
  - Due yesterday, Wednesday 5/18 @ 10pm
  - Late due date Saturday 5/21 @ 10pm
- HW7
  - Out now, due next Wednesday 5/25 @ 10pm
- Final Exam Info:
  - Planning for in-person exam!
  - Date and time should be on your MyUW
  - more info about our logistics coming soon in an Ed post
- If you have any questions or concerns about your grade:
  - Reach out to Robbie to schedule a quick grade chat!
  - Reach out to your TAs for extra help if you need it - we are here to help!

References

- How to LaTeX
  - https://courses.cs.washington.edu/courses/cse311/22sp/assignments/HowToLaTeX.pdf
- Logical Equivalences
  - https://courses.cs.washington.edu/courses/cse311/22sp/resources/reference-logical_equiv.pdf
- Inference Rules
  - https://courses.cs.washington.edu/courses/cse311/22sp/resources/InferenceRules.pdf
- Set Definitions
  - https://courses.cs.washington.edu/courses/cse311/22sp/resources/reference-sets.pdf
- Modular Arithmetic Definitions and Properties
  - https://courses.cs.washington.edu/courses/cse311/22sp/resources/reference-number-theory.pdf
- Induction Templates
  - https://courses.cs.washington.edu/courses/cse311/22sp/resources/induction-templates.pdf

Even More Induction

Problem 4 - Walk the Dawgs

Suppose a dog walker takes care of $n \geq 12$ dogs. The dog walker is not a strong person, and will walk dogs in groups of 3 or 7 at a time (every dog gets walked exactly once). Prove the dog walker can always split the n dogs into groups of 3 or 7.

Work on this proof with the people around you, and then we'll go over it together!

Problem 4 - Walk the Dawgs

Problem 4 - Walk the Dawgs

Let P(n) be “a group with n dogs can be split into groups of 3 or 7 dogs.”
We show P(n) holds for all $n \geq 12$ by strong induction on n.

Problem 4 - Walk the Dawgs

Let P(n) be “a group with n dogs can be split into groups of 3 or 7 dogs.”
We show P(n) holds for all $n \geq 12$ by strong induction on n.

Base Cases: n = 12, 13, 14: 12 = 3 + 3 + 3 + 3, 13 = 3 + 7 + 3, 14 = 7 + 7. So, P(12), P(13), and P(14) hold.

Problem 4 - Walk the Dawgs

Let P(n) be “a group with n dogs can be split into groups of 3 or 7 dogs.”
We show P(n) holds for all $n \geq 12$ by strong induction on n.

Base Cases: n = 12, 13, 14: 12 = 3 + 3 + 3 + 3, 13 = 3 + 7 + 3, 14 = 7 + 7. So, P(12), P(13), and P(14) hold.

Inductive Hypothesis: Suppose $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$ holds for an arbitrary $k \geq 14$

Problem 4 - Walk the Dawgs

Let P(n) be “a group with n dogs can be split into groups of 3 or 7 dogs.”
We show P(n) holds for all $n \geq 12$ by strong induction on n.

Base Cases: n = 12, 13, 14: 12 = 3 + 3 + 3 + 3, 13 = 3 + 7 + 3, 14 = 7 + 7. So, P(12), P(13), and P(14) hold.

Inductive Hypothesis: Suppose $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$ holds for an arbitrary $k \geq 14$

Inductive Step: Goal: Show P(k + 1): k + 1 dogs can be split into groups of size 3 or 7.

Problem 4 - Walk the Dawgs

Let P(n) be “a group with n dogs can be split into groups of 3 or 7 dogs.”
We show P(n) holds for all $n \geq 12$ by strong induction on n.

Base Cases: n = 12, 13, 14: 12 = 3 + 3 + 3 + 3, 13 = 3 + 7 + 3, 14 = 7 + 7. So, P(12), P(13), and P(14) hold.

Inductive Hypothesis: Suppose $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$ holds for an arbitrary $k \geq 14$

Inductive Step: Goal: Show P(k + 1): k + 1 dogs can be split into groups of size 3 or 7.

We first form one group of 3 dogs. Then we can divide the remaining $k-2$ dogs into groups of 3 or 7 by the assumption $P(k-2)$. (Note that $k \geq 14$ and so $k-2 \geq 12$; thus, $P(k-2)$ is among our assumptions $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$.)

Problem 4 - Walk the Dawgs

Let P(n) be “a group with n dogs can be split into groups of 3 or 7 dogs.”
We show P(n) holds for all $n \geq 12$ by strong induction on n.

Base Cases: n = 12, 13, 14: 12 = 3 + 3 + 3 + 3, 13 = 3 + 7 + 3, 14 = 7 + 7. So, P(12), P(13), and P(14) hold.

Inductive Hypothesis: Suppose $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$ holds for an arbitrary $k \geq 14$

Inductive Step: Goal: Show P(k + 1): k + 1 dogs can be split into groups of size 3 or 7.

We first form one group of 3 dogs. Then we can divide the remaining $k-2$ dogs into groups of 3 or 7 by the assumption $P(k-2)$. (Note that $k \geq 14$ and so $k-2 \geq 12$; thus, $P(k-2)$ is among our assumptions $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$.)

This proves P(k + 1), a group with k + 1 dogs can be split into groups of 3 or 7 dogs.

Problem 4 - Walk the Dawgs

Let P(n) be “a group with n dogs can be split into groups of 3 or 7 dogs.”
We show P(n) holds for all $n \geq 12$ by strong induction on n.

Base Cases: n = 12, 13, 14: 12 = 3 + 3 + 3 + 3, 13 = 3 + 7 + 3, 14 = 7 + 7. So, P(12), P(13), and P(14) hold.

Inductive Hypothesis: Suppose $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$ holds for an arbitrary $k \geq 14$

Inductive Step: Goal: Show P(k + 1): k + 1 dogs can be split into groups of size 3 or 7.

We first form one group of 3 dogs. Then we can divide the remaining $k-2$ dogs into groups of 3 or 7 by the assumption $P(k-2)$. (Note that $k \geq 14$ and so $k-2 \geq 12$; thus, $P(k-2)$ is among our assumptions $P(12) \wedge P(13) \wedge \ldots \wedge P(k)$.)

This proves P(k + 1), a group with k + 1 dogs can be split into groups of 3 or 7 dogs.

Conclusion: Therefore, P(n) holds for all $n \geq 12$ by the principle of strong induction.

Problem 5 - For All

For this problem, we'll see an incorrect use of induction. For this problem, we'll think of all of the following as binary trees:

- A single node.
- A root node, with a left child that is the root of a binary tree (and no right child)
- A root node, with a right child that is the root of a binary tree (and no left child)
- A root node, with both left and right children that are roots of binary trees.

Let P(n) be "for all trees of height n, the tree has an odd number of nodes" Take a moment to realize this claim is false.

Problem 5 - For All

We'll prove P(n) for all $n \in N$ by induction on n.

Base Case (n = 0): There is only one tree of height 0, a single node. It has one node, and $1 = 2 \cdot 0 + 1$, which is an odd number of nodes.

Inductive Hypothesis: Suppose P(i) holds for i = 0, ..., k, for some arbitrary $k \geq 0$.

Inductive Step: Let T be an arbitrary tree of height k. All trees with nodes (and since $k \geq 0$, T has at least one node) have a leaf node. Add a left child and right child to a leaf (pick arbitrarily if there's more than one), This tree now has height k + 1 (since T was height k and we added children below). By IH, T had an odd number of nodes, call it 2j + 1 for some integer j. Now we have added two more, so our new tree has $2j + 1 + 2 = 2(j + 1) + 1$ nodes. Since j was an integer, so is j + 1, and our new tree has an odd number of nodes, as required, so P(k + 1) holds.

By the principle of induction, P(n) holds for all $n \in N$. Since every tree has an (integer) height of 0 or more, every tree is included in some P(), so the claim holds for all trees.

Problem 5 - For All

(a) What is the bug in the proof?

(b) What should the starting point and target of the IS be (you can't write a full proof, as the claim is false).

Work on this proof with the people around you, and then we'll go over it together!

Problem 5 - For All

(a) What is the bug in the proof?

(b) What should the starting point and target of the IS be (you can't write a full proof, as the claim is false).

Problem 5 - For All

(a) What is the bug in the proof?

The proof, in trying to show something about an arbitrary of height k+1, builds a particular tree of height k + 1, not an arbitrary one. While the tree built is indeed of height k + 1 and has an odd number of nodes, it is not an arbitrary tree of height k + 1.

(b) What should the starting point and target of the IS be (you can't write a full proof, as the claim is false).

Problem 5 - For All

(a) What is the bug in the proof?

The proof, in trying to show something about an arbitrary of height k+1, builds a particular tree of height k + 1, not an arbitrary one. While the tree built is indeed of height k + 1 and has an odd number of nodes, it is not an arbitrary tree of height k + 1.

(b) What should the starting point and target of the IS be (you can't write a full proof, as the claim is false).

"Let T be an arbitrary tree of height k + 1" should be the first sentence. "T has an odd number of nodes" is the target.

Regular Expressions

Regular Expressions - recursively defined set of strings

Basis:

- ε is a regular expression. The empty string itself matches the pattern (and nothing else does).
- $\emptyset$ is a regular expression. No strings match this pattern.
- a is a regular expression, for any $a \in \Sigma$ (i.e. any character). The character itself matching this pattern.

Recursive:

- If A, B are regular expressions then $(A \cup B)$ is a regular expression. matched by any string that matches A or that matches B [or both]).
- If A, B are regular expressions then AB is a regular expression. matched by any string x such that $x = yz$, y matches A and z matches B.
- If A is a regular expression, then A^* is a regular expression. matched by any string that can be divided into 0 or more strings that match A.

Regular Expressions

- A regular expression is a recursively defined set of strings that form a language.

- A regular expression will generate all strings in a language, and won't generate any strings that ARE NOT in the language
  - Come up with a few examples of strings that ARE and ARE NOT in your language
  - Then, after you write your regex, check to make sure that it CAN generate all of your examples that are in the language, and it CAN'T generate those that are not

- Some examples:
  - 0 (0 ∪ 1) 1
  - 0*
  - (0 ∪ 1)*

Regular Expressions

- A regular expression is a recursively defined set of strings that form a language.

- A regular expression will generate all strings in a language, and won't generate any strings that ARE NOT in the language
  - Come up with a few examples of strings that ARE and ARE NOT in your language
  - Then, after you write your regex, check to make sure that it CAN generate all of your examples that are in the language, and it CAN'T generate those that are not

- Some examples:
  - 0 (0 ∪ 1) 1 {001, 011}
  - 0*
  - (0 ∪ 1)*

Regular Expressions

- A regular expression is a recursively defined set of strings that form a language.

- A regular expression will generate all strings in a language, and won't generate any strings that ARE NOT in the language
  - Come up with a few examples of strings that ARE and ARE NOT in your language
  - Then, after you write your regex, check to make sure that it CAN generate all of your examples that are in the language, and it CAN'T generate those that are not

- Some examples:
  - 0 (0 ∪ 1) 1 $\{001, 011\}$
  - 0* $\{\varepsilon, 0, 00, 000, 0000, \ldots\}$
  - (0 ∪ 1)*

Regular Expressions

- A regular expression is a recursively defined set of strings that form a language.

- A regular expression will generate all strings in a language, and won't generate any strings that ARE NOT in the language
  - Come up with a few examples of strings that ARE and ARE NOT in your language
  - Then, after you write your regex, check to make sure that it CAN generate all of your examples that are in the language, and it CAN'T generate those that are not

- Some examples:
  - 0 (0 ∪ 1) 1 {001, 011}
  - 0* {ε, 0, 00, 000, 0000, … }
  - (0 ∪ 1)* The set of all binary strings

Problem 1 - Regular Expressions

a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

b) Write a regular expression that matches all base-3 numbers that are divisible by 3.

c) Write a regular expression that matches all binary strings that contain the substring “111”, but not the substring “000”.

Work on this problem with the people around you, and then we’ll go over it together!

Problem 1 - Regular Expressions

a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

b) Write a regular expression that matches all base-3 numbers that are divisible by 3.

c) Write a regular expression that matches all binary strings that contain the substring “111”, but not the substring “000”.

Problem 1 - Regular Expressions

a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

0 ∪ ((1 ∪ 2 ∪ 3 ∪ 4 ∪ 5 ∪ 6 ∪ 7 ∪ 8 ∪ 9) (0 ∪ 1 ∪ 2 ∪ 3 ∪ 4 ∪ 5 ∪ 6 ∪ 7 ∪ 8 ∪ 9)*)

b) Write a regular expression that matches all base-3 numbers that are divisible by 3.

c) Write a regular expression that matches all binary strings that contain the substring “111”, but not the substring “000”.

Problem 1 - Regular Expressions

a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

0 ∪ ((1 ∪ 2 ∪ 3 ∪ 4 ∪ 5 ∪ 6 ∪ 7 ∪ 8 ∪ 9) (0 ∪ 1 ∪ 2 ∪ 3 ∪ 4 ∪ 5 ∪ 6 ∪ 7 ∪ 8 ∪ 9)*)

b) Write a regular expression that matches all base-3 numbers that are divisible by 3.

0 ∪ ((1 ∪ 2) (0 ∪ 1 ∪ 2)* 0)

c) Write a regular expression that matches all binary strings that contain the substring “111”, but not the substring “000”.

Problem 1 - Regular Expressions

a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

$0 \cup ((1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)\,(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)^*\,)$

b) Write a regular expression that matches all base-3 numbers that are divisible by 3.

$0 \cup ((1 \cup 2)\,(0 \cup 1 \cup 2)^*\,0)$

c) Write a regular expression that matches all binary strings that contain the substring “111”, but not the substring “000”.

$(01 \cup 001 \cup 1^*\,)^*\,(0 \cup 00 \cup \varepsilon)\,111\,(01 \cup 001 \cup 1^*\,)^*\,(0 \cup 00 \cup \varepsilon)$

Context-Free Grammars

Context-Free Grammars

A context free grammar (CFG) is a finite set of production rules over:
An alphabet Σ of “terminal symbols”
A finite set V of “nonterminal symbols”
A start symbol (one of the elements of V) usually denoted S.

A production rule for a nonterminal $A \in V$ takes the form
$A \rightarrow w_1 \mid w_2 \mid \ldots \mid w_k$
Where each $w_i \in V \cup \Sigma^*$ is a string of nonterminals and terminals.

CFGs

- This is another way we can describe a language using recursive rules.
- We can think of CFGs as *generating* strings:
    1. Start from the start symbol S.
    2. Choose a nonterminal in the string, and a production rule $A \rightarrow w_1 \mid w_2 \mid \ldots \mid w_k$ replace that copy of the nonterminal with w_i
    3. If no nonterminals remain, you're done! Otherwise, goto step 2.
- A string is in the language of the CFG iff it can be generated starting from S.
- All regular expressions can be written as context-free grammars, but not all context-free grammars can be written as regular expressions!

CFGs - examples

$\mathbf{S} \to 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

$\mathbf{S} \to 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

$\mathbf{S} \to (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

CFGs - examples

$\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

Set of all binary palindromes (strings the same forwards as backwards)

$\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

$\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

CFGs - examples

$\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

Set of all binary palindromes (strings the same forwards as backwards)

$\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

Set of all binary strings with 0s at the front and 1s at the end.

$\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

CFGs - examples

$\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

Set of all binary palindromes (strings the same forwards as backwards)

$\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

Set of all binary strings with 0s at the front and 1s at the end.

$\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

Set of all strings with matched parentheses.

Problem 2 - CFGs

a) All binary strings that end in 00.

b) All binary strings that contain at least three 1's.

c) All binary strings with an equal number of 1's and 0's.

Work on this problem with the people around you, and then we'll go over it together!

Problem 2 - CFGs

a) All binary strings that end in 00.

b) All binary strings that contain at least three 1’s.

c) All binary strings with an equal number of 1’s and 0’s.

Problem 2 - CFGs

a) All binary strings that end in 00.

$\mathbf{S} \rightarrow 0\mathbf{S} \mid 1\mathbf{S} \mid 00$

b) All binary strings that contain at least three 1's.

c) All binary strings with an equal number of 1's and 0's.

Problem 2 - CFGs

a) All binary strings that end in 00.

$\mathbf{S} \rightarrow 0\mathbf{S} \mid 1\mathbf{S} \mid 00$

b) All binary strings that contain at least three 1's.

$\mathbf{S} \rightarrow \mathbf{TTT}$
$\mathbf{T} \rightarrow 0\mathbf{T} \mid \mathbf{T}0 \mid 1\mathbf{T} \mid 1$

c) All binary strings with an equal number of 1's and 0's.

Problem 2 - CFGs

a) All binary strings that end in 00.

$\mathbf{S} \rightarrow 0\mathbf{S} \mid 1\mathbf{S} \mid 00$

b) All binary strings that contain at least three 1's.

$\mathbf{S} \rightarrow \mathbf{TTT}$
$\mathbf{T} \rightarrow 0\mathbf{T} \mid \mathbf{T}0 \mid 1\mathbf{T} \mid 1$

c) All binary strings with an equal number of 1's and 0's.

$\mathbf{S} \rightarrow 0\mathbf{S}1\mathbf{S} \mid 1\mathbf{S}0\mathbf{S} \mid \varepsilon$
or
$\mathbf{S} \rightarrow \mathbf{SS} \mid 0\mathbf{S}1 \mid 1\mathbf{S}0 \mid \varepsilon$

Bonus:
More Induction!

Problem 6 - Reversing a Binary Tree

Definition of (binary) Tree:
Basis Step: Nil is a Tree.
Recursive Step: If L is a Tree and R is a Tree, and x is an integer, then Tree(x, L, R) is a Tree

Definition of sum():
sum(Nil) = 0
sum(Tree(x, L, R)) = x + sum(L) + sum(R)

Definition of reverse():
reverse(Nil) = Nil
reverse(Tree(x, L, R)) = Tree(x, reverse(R), reverse(L))

Prove, for all Trees T, that sum(T) = sum(reverse(T))

Work on this proof with the people around you, and then we'll go over it together!

Problem 6 - Reversing a Binary Tree

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

sum(reverse(Tree(x, L, R))) = sum(Tree(x, reverse(R), reverse(L)))

def of reverse

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

sum(reverse(Tree(x, L, R))) = sum(Tree(x, reverse(R), reverse(L)))

def of reverse

= x + sum(reverse(R)) + sum(reverse(L))

def of sum

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

sum(reverse(Tree(x, L, R))) = sum(Tree(x, reverse(R), reverse(L)))

def of reverse

= x + sum(reverse(R)) + sum(reverse(L))

def of sum

= x + sum(R) + sum(L)

by IH

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

sum(reverse(Tree(x, L, R))) = sum(Tree(x, reverse(R), reverse(L)))

def of reverse

= x + sum(reverse(R)) + sum(reverse(L))

def of sum

= x + sum(R) + sum(L)

by IH

= x + sum(L) + sum(R)

commutativity

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

sum(reverse(Tree(x, L, R))) = sum(Tree(x, reverse(R), reverse(L)))

def of reverse

= x + sum(reverse(R)) + sum(reverse(L))

def of sum

= x + sum(R) + sum(L)

by IH

= x + sum(L) + sum(R)

commutativity

= sum(Tree(x, L, R))

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

sum(reverse(Tree(x, L, R))) = sum(Tree(x, reverse(R), reverse(L)))

def of reverse

= x + sum(reverse(R)) + sum(reverse(L))

def of sum

= x + sum(R) + sum(L)

by IH

= x + sum(L) + sum(R)

commutativity

= sum(Tree(x, L, R))

Problem 6 - Reversing a Binary Tree

For a Tree T, let P(T) be “sum(T) = sum(reverse(T))”.
We prove P(T) for all Trees T by structural induction on T.

Base Case: (T = Nil): By definition, reverse(Nil) = Nil. Applying sum to both sides, we get sum(Nil) = sum(reverse(Nil), which is exactly P(Nil), so the base case holds.

Inductive Hypothesis: Suppose P(L) and P(R) hold for some arbitrary Trees L and R.

Inductive Step: Let x be an arbitrary integer. Goal: Show P(Tree(x, L, R)) holds.

sum(reverse(Tree(x, L, R))) = sum(Tree(x, reverse(R), reverse(L)))

def of reverse

= x + sum(reverse(R)) + sum(reverse(L))

def of sum

= x + sum(R) + sum(L)

by IH

= x + sum(L) + sum(R)

commutativity

= sum(Tree(x, L, R))

That’s All, Folks!

Any questions?

Weak Induction Template

Let $P(n)$ be "(whatever you're trying to prove)".
We show $P(n)$ holds for (some range of) n by induction on n.

Base Case: Show $P(b)$ is true

Inductive Hypothesis: Suppose $P(k)$ holds for an arbitrary $k \geq b$

Inductive Step: Show $P(k + 1)$ (i.e. get $P(k) \rightarrow P(k + 1)$)

Conclusion: Therefore, $P(n)$ holds for all n by the principle of induction.

Strong Induction Template

Let $P(n)$ be "(whatever you're trying to prove)"
We show $P(n)$ holds for all $n \geq b_{min}$ by strong induction on n.

Base Cases: Show $P(b_{min})$, $P(b_{min+1})$, … , $P(b_{max})$ are all true

Inductive Hypothesis: Suppose $P(b_{min}) \wedge P(b_{min+1}) \wedge \ldots \wedge P(k)$ holds for an arbitrary $k \geq b_{max}$

Inductive Step: Show $P(k + 1)$ (i.e. get $[P(b_{min}) \wedge P(k)] \rightarrow P(k + 1)$)

Conclusion: Therefore, $P(n)$ holds for all $n \geq b_{min}$ by the principle of strong induction.

Structural Induction Template

For an $x \in S$, let *P(x)* be "(whatever you're trying to prove)"
We show *P(x)* holds for all $x \in S$ by structural induction on x.

Base Case: Show $P(x)$ for all base cases x in S

Inductive Hypothesis: Suppose $P(x)$ for all x listed as in S in the recursive rules.

Inductive Step: Show $P(?)$ holds for the "new element" given.

Conclusion: Therefore $P\ (x)$ holds for all $x \in S$ by the principle of induction.