Recursion and Induction

CS 220 — Applied Discrete Mathematics

April {9, 14}, 2025

UMass Boston

Recursive Definitions

Definition (Recursive)

A definition is **recursive** if its right-hand side refers to the name or symbol being defined. We also say the definition is **self-referential**.

Recursive Definitions

Some things that look like recursive definitions may not be:

Examples

- $x = x^2$ — specification (two solutions)
- $x = x + 1$ — unsatisfiable/contradictory
- $x = x$ — underconstrained
- $x = \frac{1}{2}(x^2 + 1)$ — ?? (one solution)
- "This statement is false." — interestingly problematic

For functions, we usually play it safe: definition by cases, with

- one or more non-recursive **base cases**
- and **recursive cases** that recur on *strictly smaller* arguments

If a recursive function definition does not have that structure, it may not be well-defined.

Recursively-Defined Things

- recursively-defined functions
- recursively-defined sets (and `types`)
- recursively-defined relations

Don't say "recursive set".

The set itself is not recursive (eg, self-containing; see *Russell's paradox*).

Also, "**recursive set**" is jargon in *computability theory*, where it means the same thing as "computable" and "decidable".

Recursively-Defined Functions and Sequences

Definition of Factorial

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$

Example

$$0! = 1$$
$$1! = 1 \cdot 0! = 1 \cdot 1 = 1$$
$$2! = 2 \cdot 1! = 2 \cdot 1 \cdot 1 = 2$$
$$3! = 3 \cdot 2! = 3 \cdot 2 \cdot 1 \cdot 1 = 6$$
$$4! = 4 \cdot 3! = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 24$$

Definition of Fibonacci Numbers

The Fibonacci sequence $(0, 1, 1, 2, 3, 5, 8, 13, \dots)$ can be defined like this:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

or like this:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \quad \text{if } n > 1 \end{aligned}$$

or like this:

$$F_0 = 0 \qquad F_1 = 1 \qquad F_{n+2} = F_{n+1} + F_n$$

Structure matters more than *notation*: multiple cases, with at least one base case, and recursion on *smaller* arguments.

Another Recursive Sequence

$$s(0) = 0 \qquad\qquad s(n+1) = 2n + 1 + s(n)$$

Another Recursive Sequence

$$s(0) = 0 \qquad s(n+1) = 2n + 1 + s(n)$$

Examples

$$s(1) = 2 \cdot 0 + 1 + 0 = 1$$
$$s(2) = 2 \cdot 1 + 1 + 1 = 4$$
$$s(3) = 2 \cdot 2 + 1 + 4 = 9$$
$$s(4) = 2 \cdot 3 + 1 + 9 = 16$$

Another Recursive Sequence

$$s(0) = 0 \qquad\qquad s(n+1) = 2n + 1 + s(n)$$

Examples

$$s(1) = 2 \cdot 0 + 1 + 0 = 1$$
$$s(2) = 2 \cdot 1 + 1 + 1 = 4$$
$$s(3) = 2 \cdot 2 + 1 + 4 = 9$$
$$s(4) = 2 \cdot 3 + 1 + 9 = 16$$

Conjecture

The sequence s also seems to have a **closed form**: $s(n) = n^2$.
(How could we be sure?)

Definition of Exponentiation, etc

We can define exponentiation to powers in $\mathbb{N}$ recursively:

$$x^0 = 1$$
$$x^n = x \cdot x^{n-1} \quad \text{if } n > 0$$

Definition of Exponentiation, etc

We can define exponentiation to powers in $\mathbb{N}$ recursively:

$$x^0 = 1$$
$$x^n = x \cdot x^{n-1} \quad \text{if } n > 0$$

In fact, we can define multiplication in $\mathbb{N}$ recursively:

$$0 \cdot n = 0$$
$$m \cdot n = n + (m-1) \cdot n \quad \text{if } m > 0$$

Definition of Exponentiation, etc

We can define exponentiation to powers in $\mathbb{N}$ recursively:

$$x^0 = 1$$
$$x^n = x \cdot x^{n-1} \quad \text{if } n > 0$$

In fact, we can define multiplication in $\mathbb{N}$ recursively:

$$0 \cdot n = 0$$
$$m \cdot n = n + (m-1) \cdot n \quad \text{if } m > 0$$

In fact, we can define addition in $\mathbb{N}$ recursively in terms of a simpler *successor* ("1+") function:

$$0 + n = n$$
$$\mathit{successor}(m) + n = \mathit{successor}(m + n)$$

This view of the natural numbers is called **Peano arithmetic** (1889).

Lots of simple, familiar operations are "hiding" recursion and induction underneath.

Other Recurrences

Recursive equations or **recurrences** also show up in the running-time analysis of algorithms. For example:

$$T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 \qquad \text{if } n > 1$$

describes the (simplified) running time of BINARY SEARCH, and

$$T(n) = 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n \qquad \text{if } n > 1$$

describes the (simplified) running time of MERGESORT.

Recursively-Defined Sets and Relations

Recursively-Defined Sets

Let $S \subseteq \mathbb{N}$ be defined recursively as follows:

$$S = \{7\} \cup \{a + b \mid a, b \in S\}$$

What is S?

Recursively-Defined Sets

Let $S \subseteq \mathbb{N}$ be defined recursively as follows:

$$S = \{7\} \cup \{a + b \mid a, b \in S\}$$

What is S?

I can actually think of three answers:

- S_1: the positive multiples of 7 $\{7, 14, 21, 28, \dots\}$
- S_2: the nonnegative multiples of 7 $\{0, 7, 14, 21, 28, \dots\}$
- $\mathbb{N}$: all natural numbers $\{0, 1, 2, 3, 4, \dots\}$

All three answers actually satisfy the equation above. (Check!)

Recursively-Defined Sets

Let $S \subseteq \mathbb{N}$ be defined recursively as follows:

$$S = \{7\} \cup \{a + b \mid a, b \in S\}$$

What is S?

I can actually think of three answers:

- S_1: the positive multiples of 7 $\{7, 14, 21, 28, \dots\}$
- S_2: the nonnegative multiples of 7 $\{0, 7, 14, 21, 28, \dots\}$
- $\mathbb{N}$: all natural numbers $\{0, 1, 2, 3, 4, \dots\}$

All three answers actually satisfy the equation above. (Check!)

In practice, we usually want the *smallest* answer, S_1:

$$S_1 \subset S_2 \qquad S_1 \subset \mathbb{N}$$

That is, we don't want anything that doesn't have to be there.

Recursively-Defined Sets

Thus, recursive set definitions are often written as follows:

> Let $S \subseteq \mathbb{N}$ be the least set such that $S = \{7\} \cup \{a + b \mid a, b \in S\}$.

or

> Let $S \subseteq \mathbb{N}$ be the smallest set such that
> - $7 \in S$, and
> - for every $a, b \in S$, $a + b \in S$.

Iterative Construction of Recursively-Defined Sets

Let $S \subseteq \mathbb{N}$ be the least set such that

$$S = \{7\} \cup \{a + b \mid a, b \in S\}$$

We can **iteratively** build **approximations** to S as follows:

- In the first round ("round 0"), the approximation set is just $\{7\}$.
- At each subsequent round, let a and b both take on every value from the previous round, and add $a + b$ to the new approximation set.

That is, we define a sequence of approximation sets S_n as follows:

$$S_0 = \{7\} \qquad\qquad S_{n+1} = S_n \cup \{a + b \mid a, b \in S_n\}$$

Then we take the limit of the approximations: $S = \bigcup_{n=0}^{\infty} S_n$

Recursion and Fixed Points

Let $S \subseteq \mathbb{N}$ be the least set such that

$$S = \{7\} \cup \{a + b \mid a, b \in S\}$$

We can define a function $CloserToS : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ that takes **any** approximation to S and makes a better one by one "round":

$$CloserToS(A) = \{7\} \cup \{a + b \mid a, b \in A\}$$

We can rewrite the original recursive equation for S as

$$S = CloserToS(S)$$

That is, a **solution** to the equation is a **fixed point** of the function.

- The **Knaster-Tarski Theorem** says that a **least fixed point** exists because $CloserToS$ is a monotonic function (with respect to $\subseteq$).
- The **Kleene Fixed-Point Theorem** says that the **least fixed point** is equal to $\lim_{n \to \infty} CloserToS^{(n)}(\emptyset)$ — that is, the limit of the approximations.

Example: Arithmetic Expressions

Let AE be the smallest set such that:

- $\mathbb{Z} \subseteq AE$, and
- for every $e_1, e_2 \in AE$, "$(e_1 + e_2)$" $\in AE$, and
- for every $e_1, e_2 \in AE$, "$(e_1 - e_2)$" $\in AE$, and
- for every $e_1, e_2 \in AE$, "$(e_1 * e_2)$" $\in AE$

Examples

- Step 0: $1, 2, -7, 50, \ldots$
- Step 1: $(1 + 2)$, $(2 * -7)$, $(50 - 1)$, ...
- Step 2: $(50 - (1 + 2))$, $(2 * (2 * -7))$, $((1 + 2) * (50 - 1))$, ...
- and so on ...

In fact, for a specific example, we can predict which step first discovers it. (How?)

Example: Binary Trees

Let BT be the smallest set such that:

- NIL $\in BT$, and
- for every $n \in \mathbb{Z}$ and $t_l, t_r \in BT$,  $\in BT$

Example: Recursively-Defined Relations

Suppose R is a relation on A, and suppose T is defined as follows:

$$T = R \cup (T \circ T)$$

Example: Recursively-Defined Relations

Suppose R is a relation on A, and suppose T is defined as follows:

$$T = R \cup (T \circ T)$$

or equivalently:

Let T be the smallest relation such that

- $R \subseteq T$, *and*
- *if* $(a, b) \in T$ *and* $(b, c) \in T$, *then* $(a, c) \in T$

Example: Recursively-Defined Relations

Suppose R is a relation on A, and suppose T is defined as follows:

$$T = R \cup (T \circ T)$$

or equivalently:

Let T be the smallest relation such that

- $R \subseteq T$, *and*
- *if* $(a, b) \in T$ *and* $(b, c) \in T$, *then* $(a, c) \in T$

That's the transitive closure: T (aka R^+) is the transitive closure of R.

Example: Rotations in Binary Trees

Suppose R is a relation on BT for "tree rotations", defined by

$$\left\{ \left(\begin{array}{ccc} & x & \\ y & & t_3 \\ t_1 & t_2 & \end{array} \;,\; \begin{array}{ccc} & y & \\ t_1 & & x \\ & t_2 & t_3 \end{array} \right) \;\middle|\; x,y \in \mathbb{Z},\ t_1,t_2,t_3 \in BT \right\}$$

Let R_C be the smallest relation such that

- $R \subseteq R_C$, and
- if $x \in \mathbb{Z}$ and $t_2 \in BT$ and $(t_1, t_1') \in R_C$, then
$$\left(\begin{array}{ccc} & x & \\ t_1 & & t_2 \end{array} \;,\; \begin{array}{ccc} & x & \\ t_1' & & t_2 \end{array} \right) \in R_C$$
- if $x \in \mathbb{Z}$ and $t_1 \in BT$ and $(t_2, t_2') \in R_C$, then
$$\left(\begin{array}{ccc} & x & \\ t_1 & & t_2 \end{array} \;,\; \begin{array}{ccc} & x & \\ t_1 & & t_2' \end{array} \right) \in R_C$$

R_C means you can rotate at the root *or within any sub-tree*.

Exercise: Recursive Functions and Sets

1. Let f be defined as follows:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ \left\lceil \frac{n}{2} \right\rceil + f(n-1) & \text{if } n > 0 \end{cases}$$

   Calculate $f(1)$ up through $f(4)$.

2. Let X be a set of strings defined as the smallest set such that
   - {"a", "bb"} $\subseteq X$, and
   - if $x, y \in X$, then $\text{concatenate}(x, y) \in X$

   The "zeroth" approximation is $X_0 = \{\text{"a"}, \text{"bb"}\}$.
   Calculate approximations ("rounds") X_1 through X_3.

Mathematical Induction

Idea of Induction

Suppose you know both of the following:

$$P(0) \qquad\qquad \forall n \in \mathbb{N},\ P(n) \Rightarrow P(n+1)$$

Then

- $P(0)$ is true.
- Since $P(0)$ is true, $P(1)$ must be true.
- Since $P(1)$ is true, $P(2)$ must be true.
- Since $P(2)$ is true, $P(3)$ must be true.
- Since $P(3)$ is true, $P(4)$ must be true.
- And so on.

Mathematical induction recognizes that this pattern of reasoning eventually covers every natural number.

That is, $\forall n \in \mathbb{N},\ P(n)$.

Proof by Induction

Definition (Induction)

Induction is a proof technique for proving universal statements about the natural numbers (or other recursively-defined sets).

$$\frac{P(0) \qquad \forall n \in \mathbb{N},\, P(n) \Rightarrow P(n+1)}{\forall n \in \mathbb{N},\, P(n)}\ \textsc{Induction}$$

My Template for a Proof by Induction

Theorem

For every $n \in \mathbb{N}$, $P(n)$ is true.

Proof.

Proof by induction on n.

Base case: Prove $P(0)$

Goal: write $P(0)$ here: just plug in 0 for n (no simplification)

Now prove the proposition $P(0)$.

Inductive case: Prove $\forall n \in \mathbb{N},\ P(n) \Rightarrow P(n+1)$

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): write $P(n)$ here

Goal: write $P(n+1)$ here: just plug in $n+1$ for n (no simplification)

Now prove the proposition $P(n+1)$.
The proposition $P(n)$ is available as an assumption. Use it!

□

Example: $s(n) = n^2$

Theorem

Let $s : \mathbb{N} \to \mathbb{N}$ be defined by $s(0) = 0$ and $s(n+1) = 2n + 1 + s(n)$. Then $s(n) = n^2$.

Proof.

By induction on n.

Example: $s(n) = n^2$

Theorem

Let $s : \mathbb{N} \to \mathbb{N}$ be defined by $s(0) = 0$ and $s(n+1) = 2n + 1 + s(n)$. Then $s(n) = n^2$.

Proof.

By induction on n.

Base case:

Goal: $s(0) = 0^2$

By definition $s(0) = 0 = 0^2$.

Example: $s(n) = n^2$

Theorem

Let $s : \mathbb{N} \to \mathbb{N}$ be defined by $s(0) = 0$ and $s(n+1) = 2n + 1 + s(n)$. Then $s(n) = n^2$.

Proof.

By induction on n.

Base case:

Goal: $s(0) = 0^2$

By definition $s(0) = 0 = 0^2$.

Inductive case:

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $s(n) = n^2$.
Goal: $s(n+1) = (n+1)^2$.

Example: $s(n) = n^2$

Theorem

Let $s : \mathbb{N} \to \mathbb{N}$ be defined by $s(0) = 0$ and $s(n+1) = 2n + 1 + s(n)$. Then $s(n) = n^2$.

Proof.

By induction on n.

Base case:

Goal: $s(0) = 0^2$

By definition $s(0) = 0 = 0^2$.

Inductive case:

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $s(n) = n^2$.
Goal: $s(n+1) = (n+1)^2$.

Calculate starting with $s(n+1)$:

$$\begin{aligned} s(n+1) &= 2n + 1 + s(n) && \text{by definition of } s \\ &= 2n + 1 + n^2 && \textbf{by IH} \\ &= n^2 + 2n + 1 = (n+1)^2 && \text{algebra} \end{aligned}$$

□

Induction and Summations

The recursively-defined function $s(n)$

$$s(0) = 0 \qquad\qquad s(n+1) = 2n + 1 + s(n)$$

could also be written as follows:

$$s(k) = \begin{cases} 0 & \text{if } k = 0 \\ 2k - 1 + s(k-1) & \text{if } k > 0 \end{cases}$$

(Note the difference in formula!)

Induction and Summations

The recursively-defined function $s(n)$

$$s(0) = 0 \qquad\qquad s(n+1) = 2n + 1 + s(n)$$

could also be written as follows:

$$s(k) = \begin{cases} 0 & \text{if } k = 0 \\ 2k - 1 + s(k-1) & \text{if } k > 0 \end{cases}$$

(Note the difference in formula!)

That is, each new $s(k)$ step just adds $2k - 1$ to the previous step.
So $s(n)$ could also be expressed as the following summation:

$$s(n) = \sum_{k=1}^{n} (2k - 1)$$

Example: Summation $\sum_{k=0}^{n} k$

Theorem (Gauss)

Let $n \in \mathbb{N}$. Then $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Proof.

By induction on n.

Example: Summation $\sum_{k=0}^{n} k$

Theorem (Gauss)

Let $n \in \mathbb{N}$. Then $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Proof.

By induction on n.

Base case:

Goal: $\sum_{k=1}^{0} k = \frac{0 \cdot (0+1)}{2}$

Calculate left-hand side: $\sum_{k=1}^{0} k = 0$

Calculate right-hand side: $\frac{0 \cdot (0+1)}{2} = \frac{0}{2} = 0$

Both sides of the equation are equal to 0, so the goal is satisfied.

Example: Summation $\sum_{k=0}^{n} k$

Theorem (Gauss)

Let $n \in \mathbb{N}$. Then $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Proof.

By induction on n.

Base case: ...

Inductive case:
Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.
Goal: $\sum_{k=1}^{n+1} k = \frac{(n+1)((n+1)+1)}{2}$.

Example: Summation $\sum_{k=0}^{n} k$

Theorem (Gauss)

Let $n \in \mathbb{N}$. Then $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Proof.

By induction on n.

Base case: ...

Inductive case:

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Goal: $\sum_{k=1}^{n+1} k = \frac{(n+1)((n+1)+1)}{2}$.

Let's calculate starting with the left-hand side of the goal:

$$\sum_{k=1}^{n+1} k \qquad = (n+1) + \sum_{k=1}^{n} k$$

Example: Summation $\sum_{k=0}^{n} k$

Theorem (Gauss)

Let $n \in \mathbb{N}$. Then $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Proof.

By induction on n.

Base case: ...

Inductive case:

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Goal: $\sum_{k=1}^{n+1} k = \frac{(n+1)((n+1)+1)}{2}$.

Let's calculate starting with the left-hand side of the goal:

$$\sum_{k=1}^{n+1} k \quad = (n+1) + \sum_{k=1}^{n} k \quad = (n+1) + \frac{n(n+1)}{2} \qquad \text{by IH}$$

Example: Summation $\sum_{k=0}^{n} k$

Theorem (Gauss)

Let $n \in \mathbb{N}$. Then $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Proof.

By induction on n.

Base case: ...

Inductive case:

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

Goal: $\sum_{k=1}^{n+1} k = \frac{(n+1)((n+1)+1)}{2}$.

Let's calculate starting with the left-hand side of the goal:

$$\sum_{k=1}^{n+1} k = (n+1) + \sum_{k=1}^{n} k = (n+1) + \frac{n(n+1)}{2} \quad \text{by IH}$$

$$= \frac{2n+2+n^2+n}{2} = \frac{(n+1)(n+2)}{2} = \frac{(n+1)((n+1)+1)}{2}$$

□

Example: $2^n > n$

Theorem

For all $n \in \mathbb{N}$, $2^n > n$.

Proof.

By induction on n.

Example: $2^n > n$

Theorem

For all $n \in \mathbb{N}$, $2^n > n$.

Proof.

By induction on n.

Base case:
Goal: $2^0 > 0$

Example: $2^n > n$

Theorem

For all $n \in \mathbb{N}$, $2^n > n$.

Proof.

By induction on n.

Base case:
Goal: $2^0 > 0$

Inductive case:
Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $2^n > n$.
Goal: $2^{n+1} > n + 1$.

Example: $2^n > n$

Theorem

For all $n \in \mathbb{N}$, $2^n > n$.

Proof.

By induction on n.

Base case:
Goal: $2^0 > 0$

Inductive case:
Let $n \in \mathbb{N}$. Assume (inductive hypothesis): $2^n > n$.
Goal: $2^{n+1} > n + 1$.

Let's calculate starting with the left-hand side of the goal:

$$2^{n+1} = 2 \cdot 2^n = 2^n + 2^n \underbrace{> n + 2^n}_{\text{by IH}} \geq n + 1$$

By the chain of inequalities, we get $2^{n+1} > n + 1$. □

Induction with Non-Zero Base Case

Sometimes the property in question does not hold for 0.

If a property holds for all numbers starting with b, then

- prove $P(b)$ as the base case, and
- (if needed) also assume $n \geq b$ in the inductive case.

To summarize:

$$\frac{b \in \mathbb{N} \qquad P(b) \qquad \forall n \in \mathbb{N},\ n \geq b \Rightarrow P(n) \Rightarrow P(n+1)}{\forall n \in \mathbb{N},\ n \geq b \Rightarrow P(n)}$$

This rule does not really add any new proof power.
You can show the same thing using ordinary induction on $P'(n) = (n \geq b \Rightarrow P(n))$.

Returning to the Infinitude of Primes Lemma

Lemma

Let $d, n \in \mathbb{N}$, and suppose $1 < d \leq n$. Then $d \mid (n!)$.

That is: *Let $d \in \mathbb{N}$, and suppose $d > 1$. Then for every $n \in \mathbb{N}$, if $n \geq d$, then $d \mid (n!)$.*

Proof.

Let $d \in \mathbb{N}$ with $d > 1$. We must show that for every $n \in \mathbb{N}$, if $n \geq d$ then $d \mid (n!)$.
We will show that by induction on n starting at d. $P(n)$ is $d \mid (n!)$

Base case: Prove $P(d)$
Goal: $d \mid (d!)$.
Since $d > 1$, we have $d! = d \cdot (d-1)!$, and $d \mid d \cdot (d-1)!$ by definition, so $d \mid (d!)$.

Inductive case: Prove $\forall n \in \mathbb{N},\ n \geq d \Rightarrow P(n) \Rightarrow P(n+1)$
Let $n \in \mathbb{N}$. Assume $n \geq d$, and assume (inductive hypothesis): $d \mid (n!)$
Goal: $d \mid ((n+1)!)$.

Since $d \mid (n!)$, we know d also divides any integer multiple of $n!$. So we multiply the right-hand side by $n+1$ to get $d \mid (n+1)(n!)$. Then $d \mid ((n+1)!)$ by the definition of factorial. □

Strong Induction

Sometimes each step does not rely on the number *immediately* before it.

Strong induction uses a "stronger" inductive hypothesis that knows that *every previous number* has the property.

STRONG INDUCTION

$$\frac{P(0) \qquad \forall n \in \mathbb{N},\ (\forall k \in \mathbb{N},\ k \leq n \Rightarrow P(n)) \Rightarrow P(n+1)}{\forall n \in \mathbb{N},\ P(n)}$$

This rule also does not really add any new proof power.
You can show the same thing using ordinary induction on
$P'(n) = (\forall k \in \mathbb{N},\ k \leq n \Rightarrow P(n))$ and then instantiating k with n.

Example: Almost the Fundamental Theorem of Arithmetic

Theorem

Every $n \in \mathbb{N}$ greater than 1 can be written as a product of (one or more) primes.

Proof.

By strong induction on n starting at 2.

Example: Almost the Fundamental Theorem of Arithmetic

Theorem

Every $n \in \mathbb{N}$ greater than 1 can be written as a product of (one or more) primes.

Proof.

By strong induction on n starting at 2.

Base case:
Goal: 2 is a product of primes.

Example: Almost the Fundamental Theorem of Arithmetic

Theorem

Every $n \in \mathbb{N}$ greater than 1 can be written as a product of (one or more) primes.

Proof.

By strong induction on n starting at 2.

Base case:

Goal: 2 is a product of primes.

2 is prime, so the product of primes is just 2.

Example: Almost the Fundamental Theorem of Arithmetic

Theorem

Every $n \in \mathbb{N}$ greater than 1 can be written as a product of (one or more) primes.

Proof.

By strong induction on n starting at 2.

Base case:
Goal: 2 is a product of primes.

2 is prime, so the product of primes is just 2.

Inductive case:

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): for every $k \leq n$, k is a product of primes
Goal: $n + 1$ is a product of primes

Example: Almost the Fundamental Theorem of Arithmetic

Theorem

Every $n \in \mathbb{N}$ greater than 1 can be written as a product of (one or more) primes.

Proof.

By strong induction on n starting at 2.

Base case:
Goal: 2 is a product of primes.

2 is prime, so the product of primes is just 2.

Inductive case:

Let $n \in \mathbb{N}$. Assume (inductive hypothesis): for every $k \leq n$, k is a product of primes
Goal: $n+1$ is a product of primes

By case analysis on whether $n+1$ is prime or composite:

- If $n+1$ is prime, then the product of primes is just $n+1$.
- If $n+1$ is composite, then there are $a, b \in \mathbb{N}$ such that $n+1 = ab$ and $1 < a < n+1$ and $1 < b < n+1$. From these inequalities we can get $a \leq n$ and $b \leq n$, so we can apply the induction hypothesis to get a product of primes for a and a product of primes for b. Multiplying them together gives a product of primes for $n+1$, as required. □

Recursive Types, Recursive Functions

Recursive Type: Binary Trees

Let's revisit binary trees with a more text-friendly notation:

Let BT be the smallest set such that

- NIL $\in BT$, and
- NODE$(n, t_l, t_r) \in BT$ for every $n \in \mathbb{Z}$ and $t_l, t_r \in BT$

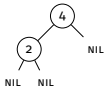
The NODE function is called a **constructor**.

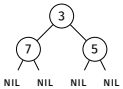
A constructor function is not defined by a computation rule.
Instead, it acts like a data structure.

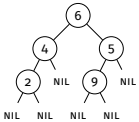
Functions on Recursive Types

Let $count : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the number of nodes it contains. (Note: NIL is not a node.)

NIL

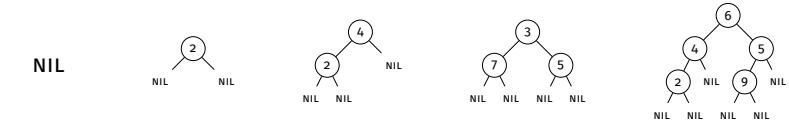






Functions on Recursive Types

Let $count : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the number of nodes it contains. (Note: NIL is not a node.)

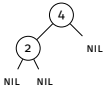


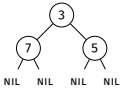
How do we define $count$?

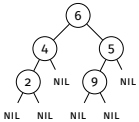
Functions on Recursive Types

Let $count : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the number of nodes it contains. (Note: NIL is not a node.)

NIL







0 nodes | 1 node | 2 nodes | 3 nodes | 5 nodes

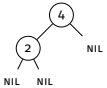
How do we define $count$?

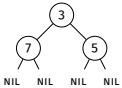
$$count(\text{NIL}) =$$
$$count(\text{NODE}(n, t_l, t_r)) =$$

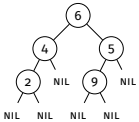
Functions on Recursive Types

Let $count : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the number of nodes it contains. (Note: NIL is not a node.)

NIL







0 nodes | 1 node | 2 nodes | 3 nodes | 5 nodes

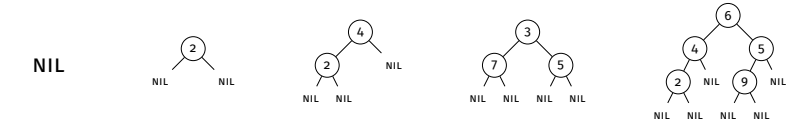
How do we define $count$?

$$count(\text{NIL}) = 0$$

$$count(\text{NODE}(n, t_l, t_r)) =$$

Functions on Recursive Types

Let $count : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the number of nodes it contains. (Note: NIL is not a node.)



How do we define $count$?

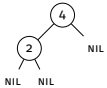
$$count(\text{NIL}) = 0$$

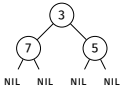
$$count(\text{NODE}(n, t_l, t_r)) = 1 + count(t_l) + count(t_r)$$

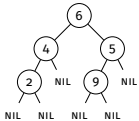
Functions on Recursive Types

Let $height : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the maximum number of edges from the root down to a NIL.

NIL

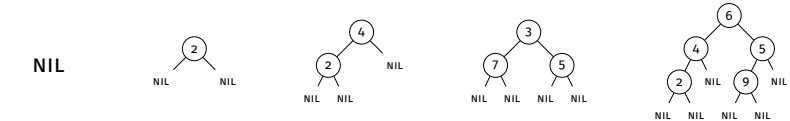






Functions on Recursive Types

Let $height : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the maximum number of edges from the root down to a NIL.

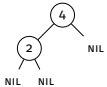


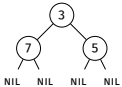
How do we define $height$?

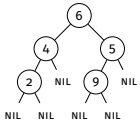
Functions on Recursive Types

Let $height : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the maximum number of edges from the root down to a NIL.

NIL







0 height | 1 height | 2 height | 2 height | 3 height

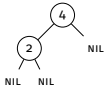
How do we define $height$?

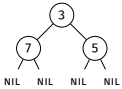
$$height(\text{NIL}) =$$
$$height(\text{NODE}(n, t_l, t_r)) =$$

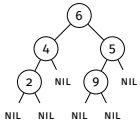
Functions on Recursive Types

Let $height : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the maximum number of edges from the root down to a NIL.

NIL







0 height | 1 height | 2 height | 2 height | 3 height

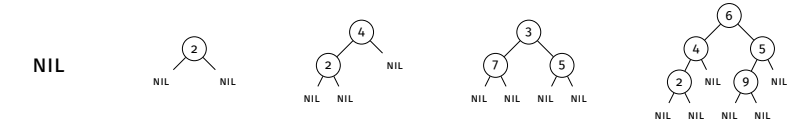
How do we define $height$?

$$height(\text{NIL}) = 0$$

$$height(\text{NODE}(n, t_l, t_r)) =$$

Functions on Recursive Types

Let $height : BT \to \mathbb{N}$ be the function that takes a binary tree and returns the maximum number of edges from the root down to a NIL.

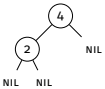


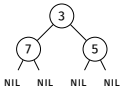
How do we define $height$?

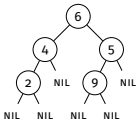
$$height(\text{NIL}) = 0$$
$$height(\text{NODE}(n, t_l, t_r)) = 1 + \max(height(t_l), height(t_r))$$

Observations about $count(t)$ and $height(t)$

NIL







0 nodes	1 node	2 nodes	3 nodes	5 nodes
0 height	1 height	2 height	2 height	3 height

Conjecture

For every $t \in BT$, $count(t) \geq height(t)$.

Conjecture

For every $t \in BT$, $count(t) \leq 2^{height(t)} - 1$.

Structural Induction

Induction Proofs for Recursively-Defined Sets

Suppose you want to prove $\forall x \in S,\ P(x)$,
where S is a recursively-defined set.

There are two main options:

- Use strong induction on the "size" of the objects in the set.

  You actually prove $\forall n \in \mathbb{N},\ \forall x \in S,\ (size(x) = n) \Rightarrow P(x)$.

  A good candidate for "size" is the number of the "round"
  in which the object is added when iteratively building the set.
  (For example, for BT, that is the same as a tree's $height$.)

- Use **structural induction** based on the set definition.

  The proof has a base case for each base case in the definition.
  The proof has an inductive case for each recursive case in the definition.

  In the inductive cases, you have an induction hypothesis for each "ingredient"
  from S used by the recursive rule. (For example, the left and right sub-trees.)

Structural Induction for Binary Trees

Recall BT is the smallest set such that

- $\text{NIL} \in BT$, and
- $\text{NODE}(n, t_l, t_r) \in BT$ for every $n \in \mathbb{Z}$ and $t_l, t_r \in BT$

  That is, $\forall n \in \mathbb{Z},\ \forall t_l, t_r \in BT,\ \text{NODE}(n, t_l, t_r) \in BT$.

The structural induction rule for BT is

$$\frac{P(\text{NIL}) \qquad \forall n \in \mathbb{Z},\ \forall t_l, t_r \in BT,\ (P(t_l) \wedge P(t_r)) \Rightarrow P(\text{NODE}(n, t_l, t_r))}{\forall t \in BT,\ P(t)}$$

Every recursive set definition generates its own structural induction rule.
The structure of the definition determines the premises of the rule.

Example: $count(t) \geq height(t)$ by Structural Induction

Theorem

For every $t \in BT$, $count(t) \geq height(t)$.

Proof.

By structural induction on t.

Base case: $P(\text{NIL})$

Goal: $count(\text{NIL}) \geq height(\text{NIL})$.

Both sides evaluate to 0.

Inductive case: $\forall x \in \mathbb{Z},\ \forall t_l, t_r \in BT,\ (P(t_l) \wedge P(t_r)) \Rightarrow P(\text{NODE}(x, t_l, t_r))$

Let $x \in \mathbb{Z}$, and let $t_l, t_r \in BT$.

Assume (**inductive hypotheses**): $count(t_l) \geq height(t_l)$ and $count(t_r) \geq height(t_r)$.

Goal: $count(\text{NODE}(x, t_l, t_r)) \geq height(\text{NODE}(x, t_l, t_r))$.

Calculate starting with the left-hand side:

$$\begin{aligned} count(\text{NODE}(x, t_l, t_r)) &= 1 + count(t_l) + count(t_r) & &\geq 1 + height(t_l) + height(t_r) \\ &\geq 1 + \max(height(t_l), height(t_r)) & &= height(\text{NODE}(x, t_l, t_r)) \end{aligned}$$

So the desired inequality is shown. □

Example: More Structural Induction (1)

Let $E \subseteq \mathbb{Z}$ be the smallest set such that

- $2 \in E$
- if $m, n \in E$, then $m - n \in E$

$$E = \{2\} \cup \{0\} \cup \{-2\} \cup \{-4, 4\} \cup \{-8, -6, 6, 8\} \cup \ldots$$

Theorem

Every $n \in E$ is even.

Proof.

By structural induction on n.

Base case: $P(2)$

Goal: 2 is even
By definition of even, since $2 = 2 \cdot 1$.

Inductive case: $\forall m, n \in E,\ (P(m) \wedge P(n)) \Rightarrow P(m - n)$

Let $m, n \in E$.
Assume (inductive hypotheses): m is even and n is even
Goal: $m - n$ is even

Since m is even, $m = 2a$ for some $a \in \mathbb{Z}$.
Since n is even, $n = 2b$ for some $b \in \mathbb{Z}$.
Then $m - n = 2a - 2b = 2(a - b)$, so $m - n$ is even.

In "ordinary" induction (on $\mathbb{N}$), the inductive case relies on having already shown the property for *numerically smaller* elements of $\mathbb{N}$.

That won't work here, because E is constructed "out of order".

In structural induction, the inductive cases rely on having shown the property for elements added to the set in *previous "rounds"*.

□

Example: More Structural Induction (2)

Let $X \subseteq \mathbb{R}$ be the smallest set such that

- $1 \in X$
- if $x, y \in X$, then $x + y \in X$
- if $x \in X$ and $x \neq 0$, then $1/x \in X$

$$X = \{1\} \cup \{2\} \cup \left\{\frac{1}{2}, 3, 4\right\} \cup \left\{\frac{1}{3}, \frac{1}{4}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}, \frac{9}{2}, 5, 6, 7, 8\right\} \cup \ldots$$

Theorem

Every $z \in X$ is positive.

Proof.

By structural induction on z.

The definition of X has two recursive cases, so the proof has two separate inductive cases.

Base case: $P(1)$
Goal: 1 is positive
Yes, 1 is positive.

Inductive case 1: $\forall x, y \in X,\ (P(X) \wedge P(y)) \Rightarrow P(x + y)$
Let $x, y \in X$. Assume (inductive hypotheses): x is positive, y is positive.
Goal: $x + y$ is positive.
If x and y are both positive, then $x + y$ is also positive.

Inductive case 2: $\forall x \in X,\ x \neq 0 \Rightarrow P(X) \Rightarrow P(1/x)$
Let $x \in X$. Assume (inductive hypothesis): x is positive
Goal: $1/x$ is positive
If x is positive, then $1/x$ is also positive.

□