CSE 311 Lecture 18: Recursively Defined Functions and Sets

Emina Torlak and Sami Davies

Topics

Advice for Homework 6

Start early, start early, start early, …

Strong induction and recursive functions

A brief review of Lecture 17.

Recursively defined sets

Recursive definitions of sets.

Structural induction

A method for proving properties of recursive structures.

Advice for Homework 6

Start early, start early, start early, …

Homework 6 has fewer problems but …

You may find it to be more work than the previous assignments.

So please start early :)

Pay special attention to Problem 6.5.

Requires keeping careful track of
(1) what you know and
(2) what you need to prove.
You will have to be methodical and meticulous.
Details will make or break this proof.

Homework 6 has fewer problems but …

You may find it to be more work than the previous assignments.

So please start early :)

Pay special attention to Problem 6.5.

Requires keeping careful track of
(1) what you know and
(2) what you need to prove.
You will have to be methodical and meticulous.
Details will make or break this proof.

Before starting to work on this problem, we *strongly* advise you to write out the set T_t for $t \in \{1, 2, 3, 4\}$. Then, calculate $\mathcal{P}(T_t)$ and S_t for each T_t. From this, you will be able to guess the answer for part (a), which you can then prove. You will also convince yourself that the theorem you need to prove for (b) is true!

Strong induction and recursive functions

A brief review of Lecture 17.

Recall why (strong) induction works

$$\textbf{Induction}\ \frac{P(0);\ \forall k.\ P(k) \to P(k+1)}{\therefore\ \forall n.\ P(n)}$$

Domain: natural numbers ($\mathbb{N}$).

How do we get $P(3)$ from $P(0)$ and $\forall k.\ P(k) \to P(k+1)$?

1. First, we have $P(0)$.
2. Since $P(k) \to P(k+1)$ for all k, we have $P(0) \to P(1)$.
3. Applying Modus Ponens to 1 and 2, we get $P(1)$.
4. Since $P(k) \to P(k+1)$ for all k, we have $P(1) \to P(2)$.
5. Applying Modus Ponens to 3 and 4, we get $P(2)$.
6. Since $P(k) \to P(k+1)$ for all k, we have $P(2) \to P(3)$.
7. Applying Modus Ponens to 6 and 7, we get $P(3)$.

$P(0)$
$\Downarrow P(0) \to P(1)$
$P(1)$
$\Downarrow P(1) \to P(2)$
$P(2)$
$\Downarrow P(2) \to P(3)$
$P(3)$

Note that we have $P(0), \ldots, P(k)$ when proving $k+1$.
So we can safely assume $P(0) \wedge \ldots \wedge P(k)$, rather than just $P(k)$.

Strong inductive proofs for any base case $b \in \mathbb{Z}$

① **Let $P(n)$ be** *[definition of $P(n)$]*.
We will show that $P(n)$ is true for every integer $n \geq b$ by strong induction.

② **Base case ($n = b$):**
[Proof of $P(b)$.]

③ **Inductive hypothesis:**
Suppose that for some arbitrary integer $k \geq b$, $P(j)$ is true for every integer $b \leq j \leq k$.

④ **Inductive step:**
We want to prove that $P(k+1)$ is true.
*[Proof of $P(k+1)$. The proof **must** invoke the strong inductive hypothesis.]*

⑤ **The result follows for all $n \geq b$ by strong induction.**

$$\frac{P(b); \forall k.\,(P(b) \wedge P(b+1) \wedge \ldots \wedge P(k)) \rightarrow}{\therefore \forall n \geq b.\,P(n)}$$

Strong induction is particularly useful when …

We need to reason about procedures that given an input k invoke themselves recursively on an input different from $k - 1$.

Example:

Euclidean algorithm for computing $\mathrm{GCD}(a, b)$.

```
// Assumes a >= b >= 0.
public static int gcd(int a, int b) {
  if (b == 0)
    return a;              // GCD(a, 0) = a
  else
    return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)
}
```

We use strong induction to reason about this algorithm and other *functions with recursive definitions*.

Recursively defined functions with a single base case

To define a recursive function f over $\mathbb{N}$, give its output in two cases:

Base case: the value of $f(0)$.

Recursive case: the value of $f(n+1)$, given in terms of $f(n)$.

Examples:

$F(0) = 1, F(n+1) = F(n) + 1$ $\quad$ $n+1$ for $n \in \mathbb{N}$

$G(0) = 1, G(n+1) = 2 \cdot G(n)$ $\quad$ 2^n for $n \in \mathbb{N}$

$K(0) = 1, K(n+1) = (n+1) \cdot K(n)$ $\quad$ $n!$ for $n \in \mathbb{N}$

When the recursive case refers only to $f(n)$, as in these examples, we can prove properties of $f(n)$ easily using ordinary induction.

Recursively defined functions with multiple base cases

A recursive function can have more than one base case.

Base cases give the value of $f(0), \ldots, f(m)$ where $m \geq 0$.

Recursive case defines $f(n+1)$ in terms of $f(n-m), \ldots, f(n-1), f(n)$ for all $n \geq m+1$, or it defines $f(n)$ in terms of $f(n-1-m), \ldots, f(n-1)$.

Example: Fibonacci numbers

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2} \text{ for all } n \geq 2$$

When the recursive function has multiple base cases, we use strong induction to prove its properties. And we also extend the strong induction proof template to account for the additional base cases.

Strong inductive proofs with base cases $b, \ldots, b + m$

① **Let $P(n)$ be** *[definition of $P(n)$]*.

We will show that $P(n)$ is true for every integer $n \geq b$ by strong induction.

② **Base cases ($n = b, \ldots, n = b + m$):**

[Proof of $P(b), \ldots, P(b + m)$.]

③ **Inductive hypothesis:**

Suppose that for some arbitrary integer $k \geq b + m$, $P(j)$ is true for every integer $b \leq j \leq k$.

④ **Inductive step:**

We want to prove that $P(k + 1)$ is true.

*[Proof of $P(k + 1)$. The proof **must** invoke the strong inductive hypothesis.]*

⑤ **The result follows for all $n \geq b$ by strong induction.**

Recursively defined sets

Recursive definitions of sets.

Giving a recursive definition of a set

A recursive definition of a set S has the following parts:

Basis step specifies one or more initial members of S.

Recursive step specifies the rule(s) for constructing new elements of S from the existing elements.

Exclusion (or closure) rule states that every element in S follows from the basis step and a finite number of recursive steps.

Giving a recursive definition of a set

A recursive definition of a set S has the following parts:

Basis step specifies one or more initial members of S.

Recursive step specifies the rule(s) for constructing new elements of S from the existing elements.

Exclusion (or closure) rule states that every element in S follows from the basis step and a finite number of recursive steps.

The exclusion rule is assumed, so no need to state it explicitly.

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Recursive: if $x \in S$, then $x + 2 \in S$

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Recursive: if $x \in S$, then $x + 2 \in S$

Powers of 3

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Recursive: if $x \in S$, then $x + 2 \in S$

Powers of 3

Basis: $1 \in S$

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Recursive: if $x \in S$, then $x + 2 \in S$

Powers of 3

Basis: $1 \in S$

Recursive: if $x \in S$, then $3x \in S$

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Recursive: if $x \in S$, then $x + 2 \in S$

Powers of 3

Basis: $1 \in S$

Recursive: if $x \in S$, then $3x \in S$

Fibonacci numbers

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Recursive: if $x \in S$, then $x + 2 \in S$

Powers of 3

Basis: $1 \in S$

Recursive: if $x \in S$, then $3x \in S$

Fibonacci numbers

Basis: $(0, 0) \in S, (1, 1) \in S$

Examples of recursively defined sets

Natural numbers

Basis: $0 \in S$

Recursive: if $n \in S$, then $n + 1 \in S$

Even natural numbers

Basis: $0 \in S$

Recursive: if $x \in S$, then $x + 2 \in S$

Powers of 3

Basis: $1 \in S$

Recursive: if $x \in S$, then $3x \in S$

Fibonacci numbers

Basis: $(0, 0) \in S, (1, 1) \in S$

Recursive: if $(n - 1, x) \in S$ and $(n - 2, y) \in S$, then $(n, x + y) \in S$

More examples of recursively defined sets

Strings

An *alphabet* Σ is any finite set of characters.

The set Σ^* of *strings* over the alphabet Σ is defined as follows.

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

More examples of recursively defined sets

Strings

An *alphabet* Σ is any finite set of characters.

The set Σ^* of *strings* over the alphabet Σ is defined as follows.

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Palindromes (strings that are the same forwards and backwards)

More examples of recursively defined sets

Strings

An *alphabet* Σ is any finite set of characters.

The set Σ^* of *strings* over the alphabet Σ is defined as follows.

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Palindromes (strings that are the same forwards and backwards)

Basis: $\varepsilon \in S$ and $a \in S$ for every $a \in \Sigma$

More examples of recursively defined sets

Strings

An *alphabet* Σ is any finite set of characters.

The set Σ^* of *strings* over the alphabet Σ is defined as follows.

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Palindromes (strings that are the same forwards and backwards)

Basis: $\varepsilon \in S$ and $a \in S$ for every $a \in \Sigma$

Recursive: if $p \in S$, then $apa \in S$ for every $a \in \Sigma$

More examples of recursively defined sets

Strings

An *alphabet* Σ is any finite set of characters.

The set Σ^* of *strings* over the alphabet Σ is defined as follows.

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Palindromes (strings that are the same forwards and backwards)

Basis: $\varepsilon \in S$ and $a \in S$ for every $a \in \Sigma$

Recursive: if $p \in S$, then $apa \in S$ for every $a \in \Sigma$

All binary strings with no 1's before 0's

More examples of recursively defined sets

Strings

An *alphabet* Σ is any finite set of characters.

The set Σ^* of *strings* over the alphabet Σ is defined as follows.

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Palindromes (strings that are the same forwards and backwards)

Basis: $\varepsilon \in S$ and $a \in S$ for every $a \in \Sigma$

Recursive: if $p \in S$, then $apa \in S$ for every $a \in \Sigma$

All binary strings with no 1's before 0's

Basis: $\varepsilon \in S$

More examples of recursively defined sets

Strings

An *alphabet* Σ is any finite set of characters.

The set Σ^* of *strings* over the alphabet Σ is defined as follows.

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Palindromes (strings that are the same forwards and backwards)

Basis: $\varepsilon \in S$ and $a \in S$ for every $a \in \Sigma$

Recursive: if $p \in S$, then $apa \in S$ for every $a \in \Sigma$

All binary strings with no 1's before 0's

Basis: $\varepsilon \in S$

Recursive: if $x \in S$, then $0x \in S$ and $x1 \in S$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ for $w \in \Sigma^*, a \in \Sigma$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ for $w \in \Sigma^*, a \in \Sigma$

Concatenation

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ for $w \in \Sigma^*, a \in \Sigma$

Concatenation

$x \bullet \varepsilon = x$ for $x \in \Sigma^*$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ for $w \in \Sigma^*, a \in \Sigma$

Concatenation

$x \bullet \varepsilon = x$ for $x \in \Sigma^*$

$x \bullet (wa) = (x \bullet w)a$ for $x, w \in \Sigma^*, a \in \Sigma$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ for $w \in \Sigma^*, a \in \Sigma$

Concatenation

$x \bullet \varepsilon = x$ for $x \in \Sigma^*$

$x \bullet (wa) = (x \bullet w)a$ for $x, w \in \Sigma^*, a \in \Sigma$

Number of c's in a string

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^{\text{R}} = \varepsilon$

$(wa)^{\text{R}} = aw^{\text{R}}$ for $w \in \Sigma^*, a \in \Sigma$

Concatenation

$x \bullet \varepsilon = x$ for $x \in \Sigma^*$

$x \bullet (wa) = (x \bullet w)a$ for $x, w \in \Sigma^*, a \in \Sigma$

Number of c's in a string

$\#_c(\varepsilon) = 0$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ for $w \in \Sigma^*, a \in \Sigma$

Concatenation

$x \bullet \varepsilon = x$ for $x \in \Sigma^*$

$x \bullet (wa) = (x \bullet w)a$ for $x, w \in \Sigma^*, a \in \Sigma$

Number of c's in a string

$\#_c(\varepsilon) = 0$

$\#_c(wc) = \#_c(w) + 1$ for $w \in \Sigma^*$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Functions on recursively defined sets

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$ for $w \in \Sigma^*, a \in \Sigma$

Reversal

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ for $w \in \Sigma^*, a \in \Sigma$

Concatenation

$x \bullet \varepsilon = x$ for $x \in \Sigma^*$

$x \bullet (wa) = (x \bullet w)a$ for $x, w \in \Sigma^*, a \in \Sigma$

Number of c's in a string

$\#_c(\varepsilon) = 0$

$\#_c(wc) = \#_c(w) + 1$ for $w \in \Sigma^*$

$\#_c(wa) = \#_c(w)$ for $w \in \Sigma^*, a \in \Sigma, a \neq c$

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$, where ε is the empty string.

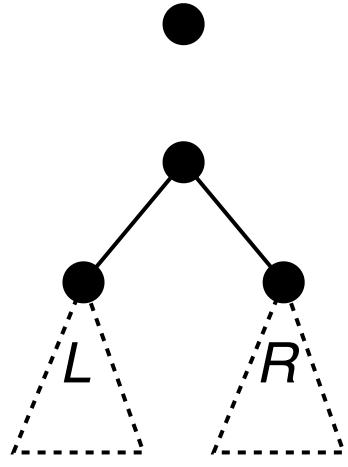
Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Rooted binary trees and functions on them

Rooted binary trees

Basis: $\bullet \in S$

Recursive: if $L \in S$ and $R \in S$, then Tree$(\bullet, L, R) \in S$



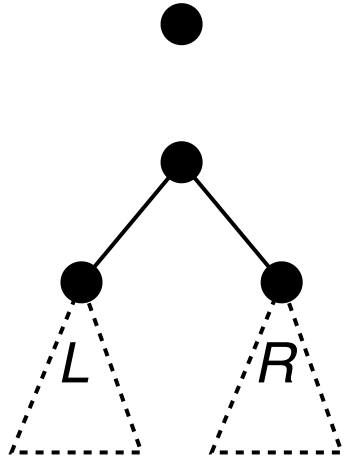
Rooted binary trees and functions on them

Rooted binary trees

Basis: $\bullet \in S$

Recursive: if $L \in S$ and $R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$

Size of a rooted binary tree



Rooted binary trees and functions on them

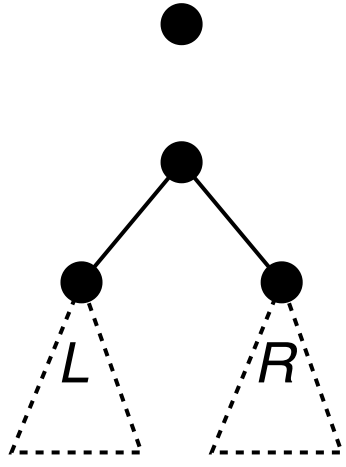
Rooted binary trees

Basis: $\bullet \in S$

Recursive: if $L \in S$ and $R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$

Size of a rooted binary tree

$|\bullet| = 1$



Rooted binary trees and functions on them

Rooted binary trees

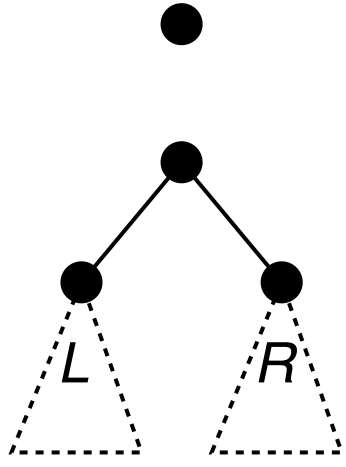
Basis: $\bullet \in S$

Recursive: if $L \in S$ and $R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$

Size of a rooted binary tree

$|\bullet| = 1$

$|\mathsf{Tree}(\bullet, L, R)| = 1 + |L| + |R|$



Rooted binary trees and functions on them

Rooted binary trees

Basis: $\bullet \in S$

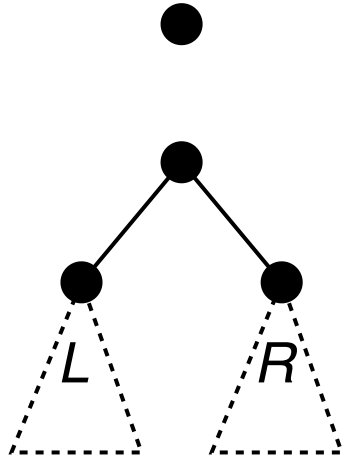
Recursive: if $L \in S$ and $R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$

Size of a rooted binary tree

$|\bullet| = 1$

$|\mathsf{Tree}(\bullet, L, R)| = 1 + |L| + |R|$

Height of a rooted binary tree



Rooted binary trees and functions on them

Rooted binary trees

Basis: $\bullet \in S$

Recursive: if $L \in S$ and $R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$

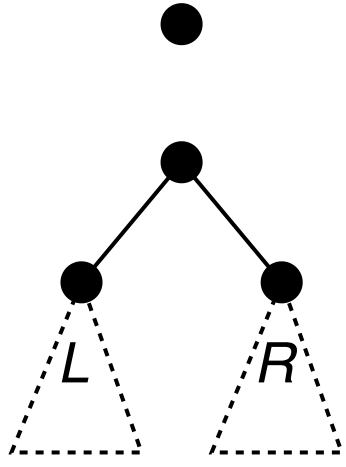
Size of a rooted binary tree

$|\bullet| = 1$

$|\mathsf{Tree}(\bullet, L, R)| = 1 + |L| + |R|$

Height of a rooted binary tree

$\lceil \bullet \rceil = 0$



Rooted binary trees and functions on them

Rooted binary trees

Basis: $\bullet \in S$

Recursive: if $L \in S$ and $R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$

Size of a rooted binary tree

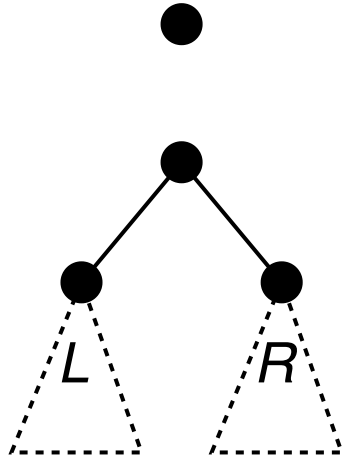
$$|\bullet| = 1$$

$$|\mathsf{Tree}(\bullet, L, R)| = 1 + |L| + |R|$$

Height of a rooted binary tree

$$\lceil \bullet \rceil = 0$$

$$\lceil \mathsf{Tree}(\bullet, L, R) \rceil = 1 + \max(\lceil L \rceil, \lceil R \rceil)$$



Structural induction

A method for proving properties of recursive structures.

How can we prove properties of recursive structures?

Suppose that S is a recursively defined set.

And we want to prove that every element of S satisfies a predicate P.

Can we use ordinary induction to prove $\forall x \in S.\, P(x)$?

How can we prove properties of recursive structures?

Suppose that S is a recursively defined set.

And we want to prove that every element of S satisfies a predicate P.

Can we use ordinary induction to prove $\forall x \in S.\, P(x)$?

Yes! Define $Q(n)$ to be "for all $x \in S$ that can be constructed in at most n recursive steps, $P(x)$ is true."

How can we prove properties of recursive structures?

Suppose that S is a recursively defined set.

And we want to prove that every element of S satisfies a predicate P.

Can we use ordinary induction to prove $\forall x \in S.\, P(x)$?

Yes! Define $Q(n)$ to be "for all $x \in S$ that can be constructed in at most n recursive steps, $P(x)$ is true."

But this proof would be long and cumbersome to do!
So we use **structural induction** instead.

- Follows from ordinary induction (on Q), while providing a more convenient proof template for reasoning about recursive structures.
- As powerful as ordinary induction, which is just structural induction applied to the recursively defined set of natural numbers.

Proving $\forall x \in S.\, P(x)$ by structural induction

① **Let $P(x)$ be** *[definition of $P(x)$]*.

We will show that $P(x)$ is true for every $x \in S$ by structural induction.

② **Base cases:**

[Proof of $P(s_0), \ldots, P(s_m)$.]

③ **Inductive hypothesis:**

Assume that $P(y_0), \ldots, P(y_k)$ are true for some arbitrary $y_0, \ldots, y_k \in S$.

④ **Inductive step:**

We want to prove that $P(y)$ is true.

*[Proof of $P(y)$. The proof **must** invoke the structural inductive hypothesis.]*

⑤ **The result follows for all $x \in S$ by structural induction.**

Recursive definition of S

Basis step:

$s_0 \in S, \ldots, s_m \in S.$

Recursive step:

if $y_0, \ldots, y_k \in S$, then $y \in S$.

Proving $\forall x \in S.\, P(x)$ by structural induction

① **Let $P(x)$ be** *[definition of $P(x)$]*.
We will show that $P(x)$ is true for every $x \in S$ by structural induction.

② **Base cases:**
[Proof of $P(s_0), \ldots, P(s_m)$.]

③ **Inductive hypothesis:**
Assume that $P(y_0), \ldots, P(y_k)$ are true for some arbitrary $y_0, \ldots, y_k \in S$.

④ **Inductive step:**
We want to prove that $P(y)$ is true.
*[Proof of $P(y)$. The proof **must** invoke the structural inductive hypothesis.]*

⑤ **The result follows for all $x \in S$ by structural induction.**

Recursive definition of S

Basis step:
$s_0 \in S, \ldots, s_m \in S.$

Recursive step:
if $y_0, \ldots, y_k \in S$, then $y \in S$.

If the **recursive step** of S includes multiple rules for constructing new elements from existing elements, then ③ **assume** P for the existing elements in every rule, and ④ **prove** P for the new element in every rule.

Summary

To define a function recursively, specify its base case(s) and recursive case.

Use (strong) induction to prove theorems about recursive functions.

To define a set recursively, specify its basis and recursive step.

Recursive set definitions assume the *exclusion rule*.

We use recursive functions to operate on elements of recursive sets.

Use structural induction to prove properties of recursive structures.

Structural induction follows from ordinary induction but is easier to use.