CSE 311 Lecture 18: Recursively Defined Functions and Sets

Emina Torlak and Kevin Zatloukal

Topics

Midterm overview

Topics and format.

Strong induction

A brief review of Lecture 17.

Recursively defined functions

Recursive function definitions and example proofs.

Recursively defined sets

Recursive definitions of sets.

Midterm overview

Topics and format.

What will be on the midterm?

Format

In-class exam: **5 problems** in **50 minutes**.

Closed book, closed notes, closed neighbors :)

No calculators, no cellphones.

What will be on the midterm?

Format

In-class exam: **5 problems** in **50 minutes**.
Closed book, closed notes, closed neighbors :)
No calculators, no cellphones.

Questions and topics

(1) Translate sentences from English to predicate logic and vice versa.
(2) Boolean circuits, algebra, and normal forms.
(3) Solving modular equations.
(4) A proof by (ordinary) induction.
(5) A proof about set theory.

What will be on the midterm?

Format

In-class exam: **5 problems** in **50 minutes**.
Closed book, closed notes, closed neighbors :)
No calculators, no cellphones.

Questions and topics

(1) Translate sentences from English to predicate logic and vice versa.
(2) Boolean circuits, algebra, and normal forms.
(3) Solving modular equations.
(4) A proof by (ordinary) induction.
(5) A proof about set theory.

You've solved similar problems on homeworks and in sections.

Do the easy parts of all the problems first. Those are worth most points. So, if you don't get to the hard parts, e.g., establishing $P(k+1)$ after invoking the hypothesis, you'll still get a lot of credit!

Strong induction

A brief review of Lecture 17.

Recall how induction works

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

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

Recall how induction works

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

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

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 $\mathbf{b} \in \mathbb{Z}$

① Let $\mathbf{P(n)}$ 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 ($\mathbf{n = b}$):**

[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 $\mathbf{n \geq b}$ by strong induction.**

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

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

Recursive function definitions and example proofs.

Giving a recursive definition for a function

To define a recursive function $\mathbf{f}$ 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 $f(n)$.

Giving a recursive definition for a function

To define a recursive function $\mathbf{f}$ 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:

$F(0) = 1, F(n+1) = F(n) + 1$

$G(0) = 1, G(n+1) = 2 \cdot G(n)$

$K(0) = 1, K(n+1) = (n+1) \cdot K(n)$

Giving a recursive definition for a function

To define a recursive function $\mathbf{f}$ 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:

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

$G(0) = 1, G(n+1) = 2 \cdot G(n)$

$K(0) = 1, K(n+1) = (n+1) \cdot K(n)$

Giving a recursive definition for a function

To define a recursive function $\mathbf{f}$ 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 $n \in \mathbb{N}$

$K(0) = 1, K(n+1) = (n+1) \cdot K(n)$

Giving a recursive definition for a function

To define a recursive function $\mathbf{f}$ 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 $n \in \mathbb{N}$

Giving a recursive definition for a function

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.

Example: prove $n! \leq n^n$ for all $n \geq 1$

Example: prove $\mathbf{n! \leq n^n}$ for all $\mathbf{n \geq 1}$

① **Let $\mathbf{P(n)}$ be $\mathbf{n! \leq n^n}$.**

We will show that $P(n)$ is true for every integer $n \geq 1$ by induction.

Example: prove $\mathbf{n! \leq n^n}$ for all $\mathbf{n \geq 1}$

① **Let $\mathbf{P(n)}$ be $\mathbf{n! \leq n^n}$.**

We will show that $P(n)$ is true for every integer $n \geq 1$ by induction.

② **Base case ($\mathbf{n = 1}$):**

$1! = 1 \cdot 0! = 1 \cdot 1 = 1 = 1^1$ so $P(1)$ is true.

Example: prove $\mathbf{n! \leq n^n}$ for all $\mathbf{n \geq 1}$

① **Let $\mathbf{P(n)}$ be $\mathbf{n! \leq n^n}$.**

We will show that $P(n)$ is true for every integer $n \geq 1$ by induction.

② **Base case ($\mathbf{n = 1}$):**

$1! = 1 \cdot 0! = 1 \cdot 1 = 1 = 1^1$ so $P(1)$ is true.

③ **Inductive hypothesis:**

Suppose that $P(k)$ is true for an arbitrary integer $k \geq 1$.

Example: prove $\mathbf{n! \leq n^n}$ for all $\mathbf{n \geq 1}$

① Let $\mathbf{P(n)}$ be $\mathbf{n! \leq n^n}$.

We will show that $P(n)$ is true for every integer $n \geq 1$ by induction.

② Base case ($\mathbf{n = 1}$):

$1! = 1 \cdot 0! = 1 \cdot 1 = 1 = 1^1$ so $P(1)$ is true.

③ Inductive hypothesis:

Suppose that $P(k)$ is true for an arbitrary integer $k \geq 1$.

④ Inductive step:

We want to prove that $P(k+1)$ is true, i.e., $(k+1)! \leq (k+1)^{(k+1)}$.

$$
\begin{aligned}
(k+1)! &= (k+1) \cdot k! && \text{by definition of } ! \\
&\leq (k+1) \cdot k^k && \text{by the inductive hypothesis} \\
&\leq (k+1) \cdot (k+1)^k && \text{since } k \geq 0 \\
&= (k+1)^{(k+1)} && \text{which is exactly } P(k+1).
\end{aligned}
$$

Example: prove $\mathbf{n! \leq n^n}$ for all $\mathbf{n \geq 1}$

① Let $\mathbf{P(n)}$ be $\mathbf{n! \leq n^n}$.

We will show that $P(n)$ is true for every integer $n \geq 1$ by induction.

② Base case ($\mathbf{n = 1}$):

$1! = 1 \cdot 0! = 1 \cdot 1 = 1 = 1^1$ so $P(1)$ is true.

③ Inductive hypothesis:

Suppose that $P(k)$ is true for an arbitrary integer $k \geq 1$.

④ Inductive step:

We want to prove that $P(k+1)$ is true, i.e., $(k+1)! \leq (k+1)^{(k+1)}$.

$$
\begin{aligned}
(k+1)! &= (k+1) \cdot k! && \text{by definition of } ! \\
&\leq (k+1) \cdot k^k && \text{by the inductive hypothesis} \\
&\leq (k+1) \cdot (k+1)^k && \text{since } k \geq 0 \\
&= (k+1)^{(k+1)} && \text{which is exactly } P(k+1).
\end{aligned}
$$

⑤ The result follows for all $\mathbf{n \geq 1}$ by induction.

Fun: can we verify $n! \leq n^n$ for all natural numbers?

Prove $n! \leq n^n$ for $n \in \mathbb{N}$ with Dafny:

```
// x^y where 0^0 = 1
function expt(x : nat, y: nat) : nat {
   if y == 0 then 1 else x * expt(x, y-1)
}

// n!
function fact(n : nat) : nat {
  if n == 0 then 1 else n * fact(n-1)
}

// n! <= n^n for all natural numbers
lemma factLemma(n : nat)
  ensures fact(n) <= expt(n, n)
{ }
```

Fun: can we verify $n! \leq n^n$ for all natural numbers?

Prove $n! \leq n^n$ for $n \in \mathbb{N}$ with Dafny:

```
// x^y where 0^0 = 1
function expt(x : nat, y: nat) : nat {
   if y == 0 then 1 else x * expt(x, y-1)
}

// n!
function fact(n : nat) : nat {
  if n == 0 then 1 else n * fact(n-1)
}

// n! <= n^n for all natural numbers
lemma factLemma(n : nat)
  ensures fact(n) <= expt(n, n)
{ }
```

Dafny can't prove this theorem because the proof involves several steps that are too difficult for Dafny to discover on its own.

Fun: can we verify $n! \leq n^n$ for all natural numbers?

Prove $n! \leq n^n$ for $n \in \mathbb{N}$ with Dafny:

```
// x^y where 0^0 = 1
function expt(x : nat, y: nat) : nat {
   if y == 0 then 1 else x * expt(x, y-1)
}

// n!
function fact(n : nat) : nat {
  if n == 0 then 1 else n * fact(n-1)
}

// n! <= n^n for all natural numbers
lemma factLemma(n : nat)
  ensures fact(n) <= expt(n, n)
{ }
```

Dafny can't prove this theorem because the proof involves several steps that are too difficult for Dafny to discover on its own.

Really prove $n! \leq n^n$ for $n \in \mathbb{N}$ with Dafny:

```
// x^y where 0^0 = 1
function expt(x : nat, y: nat) : nat {
   if y == 0 then 1 else x * expt(x, y-1)
}

// n!
function fact(n : nat) : nat {
  if n == 0 then 1 else n * fact(n-1)
}

// n! <= n^n for all natural numbers
lemma factLemma(n : nat)
  ensures fact(n) <= expt(n, n)
{
  if (n == 0) {           // Base case
    assert fact(0) <= expt(0, 0);
  } else {                // Inductive step
    factLemma(n-1);       // Inductive hypothesis
    exptLemma(n-1, n-1); // (n-1)^(n-1) <= n^(n-1)
    assert fact(n) == n * fact(n-1);                // by fact defn
    assert n * fact(n-1) <= n * expt(n-1, n-1);     // by IH
    assert n * expt(n-1, n-1) <= n * expt(n, n-1); // by exptLemma
    assert fact(n) <= expt(n, n);                   // qed.
  }
}

// x^y <= (x+1)^y for all natural numbers.
lemma exptLemma(x: nat, y: nat)
  ensures expt(x, y) <= expt(x + 1, y)
{}
```

Defining a recursive function 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 $n \geq m+1$.

Defining a recursive function 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 $f(n-1-m), \ldots, f(n-1)$.

Defining a recursive function 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

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

Defining a recursive function 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 $\mathbf{b, \ldots, b + m}$

① **Let $\mathbf{P(n)}$ 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 ($\mathbf{n = b, \ldots, n = b + m}$):**

[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 $\mathbf{n \geq b}$ by strong induction.**

Example: prove $f_n < 2^n$ for all $n \geq 0$

$$
\begin{aligned}
f_0 &= 0 \\
f_1 &= 1 \\
f_n &= f_{n-1} + f_{n-2} \\
&\qquad \text{for all } n \geq 2
\end{aligned}
$$

Example: prove $f_n < 2^n$ for all $n \geq 0$

① **Let $P(n)$ be $f_n < 2^n$ where f is the Fibonacci function.**

We will show that $P(n)$ is true for every integer $n \geq 0$ by strong induction.

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

Example: prove $\mathbf{f_n < 2^n}$ for all $\mathbf{n \geq 0}$

① Let $\mathbf{P(n)}$ be $\mathbf{f_n < 2^n}$ where $\mathbf{f}$ is the Fibonacci function.

We will show that $P(n)$ is true for every integer $n \geq 0$ by strong induction.

② Base cases ($\mathbf{n = 0, n = 1}$):

$f_0 = 0 < 1 = 2^0$ so $P(0)$ is true.

$f_1 = 1 < 2 = 2^1$ so $P(1)$ is true.

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

Example: prove $\mathbf{f_n < 2^n}$ for all $\mathbf{n \geq 0}$

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

① Let $\mathbf{P(n)}$ be $\mathbf{f_n < 2^n}$ where $\mathbf{f}$ is the Fibonacci function.

We will show that $P(n)$ is true for every integer $n \geq 0$ by strong induction.

② Base cases ($\mathbf{n = 0, n = 1}$):

$f_0 = 0 < 1 = 2^0$ so $P(0)$ is true.

$f_1 = 1 < 2 = 2^1$ so $P(1)$ is true.

③ Inductive hypothesis:

Suppose that for some arbitrary integer $k \geq 1$, $P(j)$ is true for every integer $0 \leq j \leq k$.

Example: prove $f_n < 2^n$ for all $n \geq 0$

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

① Let $P(n)$ be $f_n < 2^n$ where f is the Fibonacci function.

We will show that $P(n)$ is true for every integer $n \geq 0$ by strong induction.

② Base cases ($n = 0, n = 1$):

$f_0 = 0 < 1 = 2^0$ so $P(0)$ is true.

$f_1 = 1 < 2 = 2^1$ so $P(1)$ is true.

③ Inductive hypothesis:

Suppose that for some arbitrary integer $k \geq 1$, $P(j)$ is true for every integer $0 \leq j \leq k$.

④ Inductive step:

We want to prove that $P(k+1)$ is true, i.e., $f_{k+1} < 2^{k+1}$ for $k + 1 \geq 2$.

$$\begin{aligned} f_{k+1} &= f_k + f_{k-1} && \text{by definition of } f \\ &< 2^k + 2^{k-1} && \text{by the inductive hypothesis} \\ &< 2^k + 2^k = 2 \cdot 2^k = 2^{k+1} && \text{which is exactly } P(k+1). \end{aligned}$$

Example: prove $f_n < 2^n$ for all $n \geq 0$

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

① Let $P(n)$ be $f_n < 2^n$ where f is the Fibonacci function.

We will show that $P(n)$ is true for every integer $n \geq 0$ by strong induction.

② Base cases ($n = 0, n = 1$):

$f_0 = 0 < 1 = 2^0$ so $P(0)$ is true.

$f_1 = 1 < 2 = 2^1$ so $P(1)$ is true.

③ Inductive hypothesis:

Suppose that for some arbitrary integer $k \geq 1$, $P(j)$ is true for every integer $0 \leq j \leq k$.

④ Inductive step:

We want to prove that $P(k+1)$ is true, i.e., $f_{k+1} < 2^{k+1}$ for $k + 1 \geq 2$.

$$\begin{aligned} f_{k+1} &= f_k + f_{k-1} && \text{by definition of } f \\ &< 2^k + 2^{k-1} && \text{by the inductive hypothesis} \\ &< 2^k + 2^k = 2 \cdot 2^k = 2^{k+1} && \text{which is exactly } P(k+1). \end{aligned}$$

⑤ The result follows for all $n \geq 0$ by induction.

Fun: can we verify $f_n < 2^n$ for all $n \geq 0$?

Prove $f_n < 2^n$ for $n \geq 0$ with Dafny:

```
// 2^n
function pow2(n : nat) : nat {
   if n == 0 then 1 else 2 * pow2(n-1)
}

// Fibonacci function f_n
function fib(n: nat): nat {
  if n == 0 then 0
  else if n == 1 then 1
  else fib(n-2) + fib(n-1)
}

// f_n < 2^n
lemma fibLemma(n : nat)
  ensures fib(n) < pow2(n)
{ }
```

Fun: can we verify $f_n < 2^n$ for all $n \geq 0$?

Prove $f_n < 2^n$ for $n \geq 0$ with Dafny:

```
// 2^n
function pow2(n : nat) : nat {
   if n == 0 then 1 else 2 * pow2(n-1)
}

// Fibonacci function f_n
function fib(n: nat): nat {
  if n == 0 then 0
  else if n == 1 then 1
  else fib(n-2) + fib(n-1)
}

// f_n < 2^n
lemma fibLemma(n : nat)
  ensures fib(n) < pow2(n)
{ }
```

Yes, Dafny can prove this theorem automatically!

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

$\varepsilon^R = \varepsilon$

$(wa)^R = aw^R$ 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

$\#_c(\varepsilon) = 0$

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.

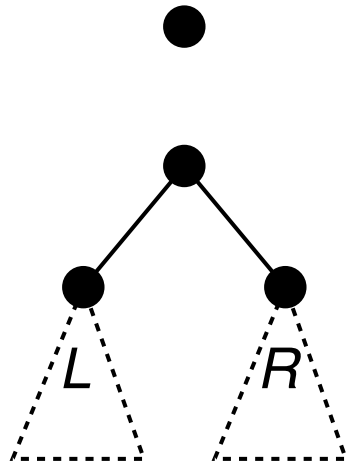
Recursive: if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

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$(\bullet, L, R) \in S$



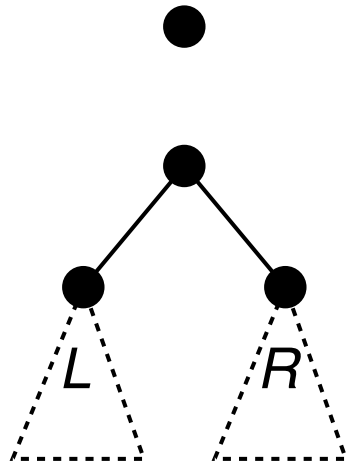
Rooted binary trees and functions on them

Rooted binary trees

Basis: $\bullet \in S$

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

Size of a rooted binary tree



Rooted binary trees and functions on them

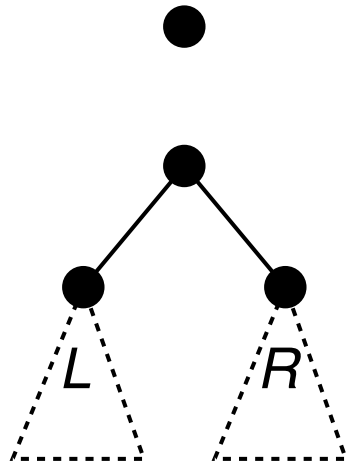
Rooted binary trees

Basis: $\bullet \in S$

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

Size of a rooted binary tree

$|\bullet| = 1$



Rooted binary trees and functions on them

Rooted binary trees

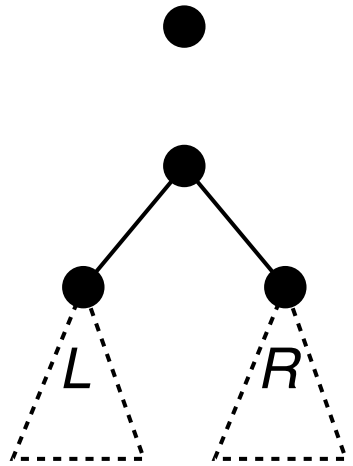
Basis: $\bullet \in S$

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: $\bullet \in S$

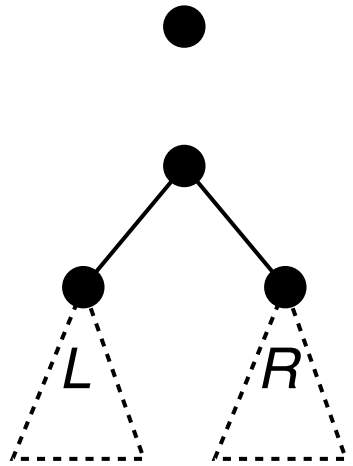
Recursive: if $L \in S$ and $R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$

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 $\mathsf{Tree}(\bullet, L, R) \in S$

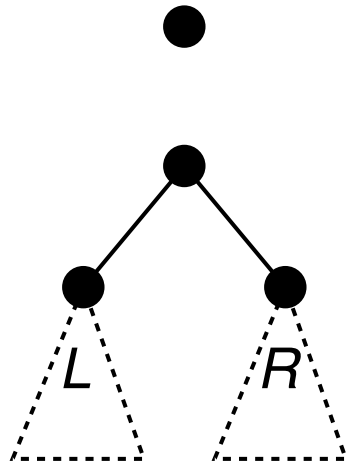
Size of a rooted binary tree

$|\bullet| = 1$

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

Height of a rooted binary tree

$\lceil \bullet \rceil = 0$

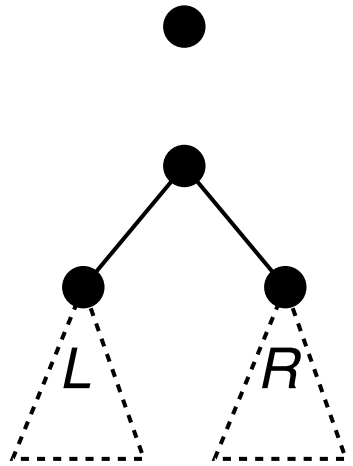


Rooted binary trees and functions on them

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

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

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

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.