Lecture 16: Recursively Defined Sets & Structural Induction
Last time: Fibonacci Numbers

\[ f(0) = 0 \]
\[ f_1 = 1 \]
\[ f_n = f_{n-1} + f_{n-2} \quad \text{for all } n \geq 2 \]
1. Let $P(n)$ be “$f_n < 2^n$”. We prove that $P(n)$ is true for all integers $n \geq 0$ by strong induction.

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

3. Inductive Hypothesis: Assume that for some arbitrary integer $k \geq 0$, we have $f_j < 2^j$ for every integer $j$ from 0 to $k$.

4. Inductive Step: Goal: Show $P(k+1)$; that is, $f_{k+1} < 2^{k+1}$
   
   Case $k+1 = 1$: Then $f_1 = 1 < 2 = 2^1$ so $P(k+1)$ is true here.
   
   Case $k+1 \geq 2$: Then $f_{k+1} = f_k + f_{k-1}$ by definition
   
   $< 2^k + 2^{k-1}$ by the IH since $k-1 \geq 0$
   
   $< 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$
   
   so $P(k+1)$ is true in this case.

   These are the only cases so $P(k+1)$ follows.

5. Therefore by strong induction, $f_n < 2^n$ for all integers $n \geq 0$. 

\[ f_0 = 0 \quad f_1 = 1 \quad f_n = f_{n-1} + f_{n-2} \quad \text{for all } n \geq 2 \]
Inductive Proofs with Multiple Base Cases

1. “Let \( P(n) \) be... . We will show that \( P(n) \) is true for all integers \( n \geq b \) by induction.”

2. “Base Cases:” Prove \( P(b), P(b + 1), \ldots, P(c) \)

3. “Inductive Hypothesis:
   Assume \( P(k) \) is true for an arbitrary integer \( k \geq c \)”

4. “Inductive Step:” Prove that \( P(k + 1) \) is true:
   Use the goal to figure out what you need.
   Make sure you are using I.H. and point out where you are using it. (Don’t assume \( P(k + 1) \) !!)

5. “Conclusion: \( P(n) \) is true for all integers \( n \geq b \)”
(Strong) Inductive Proofs with Multiple Base Cases

1. “Let $P(n)$ be... . We will show that $P(n)$ is true for all integers $n \geq b$ by **strong** induction.”

2. “Base Cases:” Prove $P(b), P(b + 1), \ldots, P(c)$

3. “Inductive Hypothesis:”
   
   Assume that for some arbitrary integer $k \geq c$,
   
   $P(j)$ is true for every integer $j$ from $b$ to $k$

4. “Inductive Step:” Prove that $P(k + 1)$ is true:
   
   *Use the goal to figure out what you need.*

   *Make sure you are using I.H. (that $P(b), \ldots, P(k)$ are true)*

   *and point out where you are using it.*

   *(Don’t assume $P(k + 1)$ !!)*

5. “Conclusion: $P(n)$ is true for all integers $n \geq b$”
Bounding Fibonacci I: $f_n < 2^n$ for all $n \geq 0$

1. Let $P(n)$ be “$f_n < 2^n$”. We prove that $P(n)$ is true for all integers $n \geq 0$ by strong induction.

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

3. Inductive Hypothesis: Assume that for some arbitrary integer $k \geq 0$, we have $f_j < 2^j$ for every integer $j$ from 0 to $k$.

4. Inductive Step: Goal: Show $P(k+1)$; that is, $f_{k+1} < 2^{k+1}$

   Case $k+1 = 1$: Then $f_1 = 1 < 2 = 2^1$ so $P(k+1)$ is true here.

   Case $k+1 \geq 2$: Then $f_{k+1} = f_k + f_{k-1}$ by definition

   $< 2^k + 2^{k-1}$ by the IH since $k-1 \geq 0$

   $< 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$

   so $P(k+1)$ is true in this case.

   These are the only cases so $P(k+1)$ follows.

5. Therefore by strong induction, $f_n < 2^n$ for all integers $n \geq 0$. 

   $f_0 = 0 \quad f_1 = 1$

   $f_n = f_{n-1} + f_{n-2}$ for all $n \geq 2$
Bounding Fibonacci Inequality: \( f_n < 2^n \) for all \( n \geq 0 \)

1. Let \( P(n) \) be “\( f_n < 2^n \)”. We prove that \( P(n) \) is true for all integers \( n \geq 0 \) by strong induction.

2. **Base Cases:** \( f_0 = 0 < 1 = 2^0 \) so \( P(0) \) is true. \( f_1 = 1 < 2 = 2^1 \) so \( P(1) \) is true.

3. **Inductive Hypothesis:** Assume that for some arbitrary integer \( k \geq 1 \), we have \( f_j < 2^j \) for every integer \( j \) from 0 to \( k \).

4. **Inductive Step:** **Goal:** Show \( P(k+1) \); that is, \( f_{k+1} < 2^{k+1} \).

   We have \( f_{k+1} = f_k + f_{k-1} \) by definition since \( k+1 \geq 2 \).
   
   \[
   < 2^k + 2^{k-1} \quad \text{by the IH since } k-1 \geq 0
   \]
   \[
   < 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}
   \]

   so \( P(k+1) \) is true.

5. Therefore, by strong induction, \( f_n < 2^n \) for all integers \( n \geq 0 \).
Bounding Fibonacci II: \( f_n \geq 2^{n/2} - 1 \) for all \( n \geq 2 \)

1. Let \( P(n) \) be “\( f_n \geq 2^{n/2} - 1 \)”. We prove that \( P(n) \) is true for all integers \( n \geq 2 \) by strong induction.

Two base cases, and two previous values used

\[
\begin{align*}
  f_0 &= 0 \\
  f_1 &= 1 \\
  f_n &= f_{n-1} + f_{n-2} & \text{for all } n \geq 2
\end{align*}
\]
Bounding Fibonacci II: \( f_n \geq 2^{n/2} - 1 \) for all \( n \geq 2 \)

1. Let \( P(n) \) be “\( f_n \geq 2^{n/2} - 1 \)”. We prove that \( P(n) \) is true for all integers \( n \geq 2 \) by strong induction.

2. Base Cases: \( f_2 = f_1 + f_0 = 1 \) and \( 2^{2/2} - 1 = 2^0 = 1 \) so \( P(2) \) holds

\[
f_3 = f_2 + f_1 = 2 \geq 2^{1/2} = 2^{3/2-1} \text{ so } P(3) \text{ holds}
\]

\[
f_0 = 0 \quad f_1 = 1 \\
f_n = f_{n-1} + f_{n-2} \text{ for all } n \geq 2
\]
Bounding Fibonacci II: \( f_n \geq 2^{n/2} - 1 \) for all \( n \geq 2 \)

1. Let \( P(n) \) be “\( f_n \geq 2^{n/2} - 1 \)”. We prove that \( P(n) \) is true for all integers \( n \geq 2 \) by strong induction.

2. Base Cases: \( f_2 = f_1 + f_0 = 1 \) and \( 2^{2/2} - 1 = 2^1 - 1 = 1 \) so \( P(2) \) holds

   \[ f_3 = f_2 + f_1 = 2 \geq 2^{1/2} = 2^{3/2 - 1} \] so \( P(3) \) holds

3. Inductive Hypothesis: Assume that for some arbitrary integer \( k \geq 3 \), \( P(j) \) is true for every integer \( j \) from 2 to \( k \).

\[ P(k+1) \quad f_{k+1} \geq 2^{(k+1)/2} - 1 \]

\( f_0 = 0 \quad f_1 = 1 \)

\( f_n = f_{n-1} + f_{n-2} \) for all \( n \geq 2 \)
Bounding Fibonacci II:  \( f_n \geq 2^{n/2} - 1 \) for all \( n \geq 2 \)

1. Let \( P(n) \) be “\( f_n \geq 2^{n/2} - 1 \)”. We prove that \( P(n) \) is true for all integers \( n \geq 2 \) by strong induction.

2. Base Cases: \( f_2 = f_1 + f_0 = 1 \) and \( 2^{2/2} - 1 = 2^0 - 1 = 0 \) so \( P(2) \) holds
   \( f_3 = f_2 + f_1 = 2 \geq 2^{1/2} = 2^{3/2-1} \) so \( P(3) \) holds

3. Inductive Hypothesis: Assume that for some arbitrary integer \( k \geq 3 \), \( P(j) \) is true for every integer \( j \) from 2 to \( k \).

4. Inductive Step: **Goal:** Show \( P(k+1) \); that is, \( f_{k+1} \geq 2^{(k+1)/2} - 1 \)

   \[
   \frac{(k+1)/2 - 1}{2} = \frac{(k+1)/2 - 2/2}{2} = \frac{(k+1)/2 - 2}{2} \cdot \frac{k-1}{2}
   \]

   \( f_{k+1} \geq 2^\left(\frac{k-1}{2}\right) \)

\[
\begin{align*}
& f_0 = 0 \quad f_1 = 1 \\
& f_n = f_{n-1} + f_{n-2} \quad \text{for all } n \geq 2
\end{align*}
\]
Bounding Fibonacci II: \( f_n \geq 2^{n/2 - 1} \) for all \( n \geq 2 \)

1. Let \( P(n) \) be “\( f_n \geq 2^{n/2 - 1} \)”. We prove that \( P(n) \) is true for all integers \( n \geq 2 \) by strong induction.

2. Base Cases: \( f_2 = f_1 + f_0 = 1 \) and \( 2^{2/2 - 1} = 2^0 = 1 \) so \( P(2) \) holds
   \( f_3 = f_2 + f_1 = 2 \geq 2^{1/2} = 2^{3/2 - 1} \) so \( P(3) \) holds

3. Inductive Hypothesis: Assume that for some arbitrary integer \( k \geq 3 \), \( P(j) \) is true for every integer \( j \) from 2 to \( k \).

4. Inductive Step: \textbf{Goal: Show} \( P(k+1) \); that is, \( f_{k+1} \geq 2^{(k+1)/2 - 1} \)
   
   We have \( f_{k+1} = f_k + f_{k-1} \) by definition since \( k+1 \geq 2 \)
   \[ \geq 2^{k/2 - 1} + 2^{(k-1)/2 - 1} \] by the IH since \( k-1 \geq 2 \)
   \[ \geq 2^{(k-1)/2 - 1} + 2^{(k-1)/2 - 1} = 2^{(k-1)/2} = 2^{(k+1)/2 - 1} \]
   so \( P(k+1) \) is true.

5. Therefore by strong induction, \( f_n \geq 2^{n/2 - 1} \) for all integers \( n \geq 2 \).

\[
\begin{align*}
f_0 &= 0 & f_1 &= 1 \\
f_n &= f_{n-1} + f_{n-2} & \text{for all } n \geq 2
\end{align*}
\]
Running time of Euclid’s algorithm

**Theorem:** Suppose that Euclid’s Algorithm takes \( n \) steps for \( \gcd(a, b) \) with \( a \geq b > 0 \). Then, \( a \geq f_{n+1} \).

\[
\hat{n}_{n+1} = \hat{n} + \hat{n}_{n-1}
\]

\[
\hat{n} \geq \hat{n}_{n-1} + \hat{n}_{n-2}
\]
Running time of Euclid’s algorithm

**Theorem:** Suppose that Euclid’s Algorithm takes $n$ steps for $\gcd(a, b)$ with $a \geq b > 0$. Then, $a \geq f_{n+1}$.

Why does this help us bound the running time of Euclid’s Algorithm?

We already proved that $f_n \geq 2^{n/2} - 1$ so $f_{n+1} \geq 2^{(n-1)/2}$

Therefore: if Euclid’s Algorithm takes $n$ steps for $\gcd(a, b)$ with $a \geq b > 0$
then $a \geq 2^{(n-1)/2}$
so $(n - 1)/2 \leq \log_2 a$ or $n \leq 1 + 2 \log_2 a$
i.e., # of steps $\leq 1 +$ twice the # of bits in $a$. 
Running time of Euclid’s algorithm

**Theorem:** Suppose that Euclid’s Algorithm takes \( n \) steps for \( \gcd(a, b) \) with \( a \geq b > 0 \). Then, \( a \geq f_{n+1} \).

An informal way to get the idea: Consider an \( n \) step gcd calculation starting with \( r_{n+1}=a \) and \( r_n=b \):

\[
\begin{align*}
    r_{n+1} &= q_n r_n + r_{n-1} \\
    r_n &= q_{n-1} r_{n-1} + r_{n-2} \\
    &\quad \ldots \\
    r_3 &= q_2 r_2 + r_1 \\
    r_2 &= q_1 r_1 + 0
\end{align*}
\]

For all \( k \geq 2 \), \( r_{k-1} = r_{k+1} \mod r_k \)

“Euclid’s algorithm is slowest on Fibonacci numbers and it takes only \( n \) steps for \( \gcd(f_{n+1}, f_n) \)”

Now \( r_1 \geq 1 \) and each \( q_k \) must be \( \geq 1 \). If we replace all the \( q_k \)'s by 1 and replace \( r_1 \) by 1, we can only reduce the \( r_k \)'s. After that reduction, \( r_k = f_k \) for every \( k \).
Running time of Euclid’s algorithm

Theorem: Suppose that Euclid’s Algorithm takes \( n \) steps for \( \gcd(a, b) \) with \( a \geq b > 0 \). Then, \( a \geq f_{n+1} \).

We go by strong induction on \( n \).

Let \( P(n) \) be “gcd\((a,b)\) with \( a \geq b>0 \) takes \( n \) steps → \( a \geq f_{n+1} \)” for all \( n \geq 1 \).

Base Case: \( n=1 \) Suppose Euclid’s Algorithm with \( a \geq b > 0 \) takes 1 step. By assumption, \( a \geq b \geq 1 = f_2 \) so \( P(1) \) holds.

\( n=2 \) Suppose Euclid’s Algorithm with \( a \geq b > 0 \) takes 2 steps. Then \( a = qb + r \) \( b = q'r + 0 \) for \( r \geq 1 \).

Since \( a \geq b > 0 \), we must have \( q \geq 1 \) and \( b \geq 1 \) so \( a = qb + r \geq b + r \geq 1+1 = 2 = f_3 \) and \( P(2) \) holds.

Induction Hypothesis: Suppose that for some integer \( k \geq 2 \), \( P(j) \) is true for all integers \( j \) s.t. \( 1 \leq j \leq k \)
Running time of Euclid’s algorithm

**Induction Hypothesis:** Suppose that for some integer \( k \geq 2 \), \( P(j) \) is true for all integers \( j \) s.t. \( 1 \leq j \leq k \).

**Inductive Step:** Goal: if gcd\((a,b)\) with \( a \geq b > 0 \) takes \( k+1 \) steps, then \( a \geq f_{k+2} \).

Since \( k \geq 2 \), if gcd\((a,b)\) with \( a \geq b > 0 \) takes \( k+1 \geq 3 \) steps, the first 3 steps of Euclid’s algorithm on \( a \) and \( b \) give us

\[
\begin{align*}
a &= q \cdot b + r \\
b &= q' \cdot r + r' \\
r &= q'' \cdot r' + r''
\end{align*}
\]

and there are \( k-2 \) more steps after this. Note that this means that the gcd\((b, r)\) takes \( k \) steps and gcd\((r, r')\) takes \( k-1 \) steps.

So since \( k, k-1 \geq 1 \), by the IH we have \( b \geq f_{k+1} \) and \( r \geq f_k \).
Running time of Euclid’s algorithm

**Induction Hypothesis:** Suppose that for some integer $k \geq 2$, $P(j)$ is true for all integers $j$ s.t. $1 \leq j \leq k$.

**Inductive Step:**

**Goal:** if $\gcd(a, b)$ with $a \geq b > 0$ takes $k+1$ steps, then $a \geq f_{k+2}$.

Since $k \geq 2$, if $\gcd(a, b)$ with $a \geq b > 0$ takes $k+1 \geq 3$ steps, the first 3 steps of Euclid’s algorithm on $a$ and $b$ give us

- $a = q \cdot b + r$
- $b = q' \cdot r + r'$
- $r = q'' \cdot r' + r''$

and there are $k-2$ more steps after this. Note that this means that the $\gcd(b, r)$ takes $k$ steps and $\gcd(r, r')$ takes $k-1$ steps.

So since $k, k-1 \geq 1$, by the IH we have $b \geq f_{k+1}$ and $r \geq f_k$.

Also, since $a \geq b$, we must have $q \geq 1$.

So $a = q \cdot b + r \geq b + r \geq f_{k+1} + f_k = f_{k+2}$ as required. ■
Last time: Recursive definitions of functions

• $0! = 1; (n + 1)! = (n + 1) \cdot n!$ for all $n \geq 0$.

• $F(0) = 0; F(n + 1) = F(n) + 1$ for all $n \geq 0$.

• $G(0) = 1; G(n + 1) = 2 \cdot G(n)$ for all $n \geq 0$.

• $H(0) = 1; H(n + 1) = 2^{H(n)}$ for all $n \geq 0$. 
Recursive functions allow general computation
   - saw examples not expressible with simple expressions

So far, we have considered only simple data
   - inputs and outputs were just integers

We need general data as well...
   - these will also be described recursively
   - will allow us to describe data of real programs
     e.g., strings, lists, trees, expressions, propositions, ...

We’ll start simple: sets of numbers
Recursive Definitions of Sets (Data)

Natural numbers
   Basis: \( 0 \in S \)
   Recursive: If \( x \in S \), then \( x+1 \in S \)

Even numbers (\( \geq 0 \))
   Basis: \( 0 \in S \)
   Recursive: If \( x \in S \), then \( x+2 \in S \)
Recursive Definition of Sets

Recursive definition of set $S$

- **Basis Step:** $0 \in S$
- **Recursive Step:** If $x \in S$, then $x + 2 \in S$
- **Exclusion Rule:** Every element in $S$ follows from the basis step and a finite number of recursive steps.

We need the exclusion rule because otherwise $S = \mathbb{N}$ would satisfy the other two parts. However, we won’t always write it down on these slides.
Recursive Definitions of Sets

Natural numbers
Basis: \( 0 \in S \)
Recursive: If \( x \in S \), then \( x+1 \in S \).

Even numbers (\( \geq 0 \))
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 \( 3x \in S \).

Basis: \((0, 0) \in S , (1, 1) \in S \)
Recursive: If \((n-1, x) \in S \) and \((n, y) \in S \), then \((n+1, x + y) \in S \).
Recursive Definitions of Sets

Natural numbers
Basis: \(0 \in S\)
Recursive: If \(x \in S\), then \(x + 1 \in S\)

Even numbers (\(\geq 0\))
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 \(3x \in S\).

Basis: \((0, 0) \in S, (1, 1) \in S\)
Recursive: If \((n-1, x) \in S\) and \((n, y) \in S\), then \((n+1, x + y) \in S\).

“Indexed” Fibonacci numbers
\(\{(n, f_n): n \in \mathbb{N}\}\)
Last time: Recursive definitions of functions

• Before, we considered only simple data
  – inputs and outputs were just integers

• Proved facts about those functions with induction
  
  \[ n! \leq n^n \]
  
  \[ f_n < 2^n \text{ and } f_n \geq 2^{n/2-1} \]

• How do we prove facts about functions that work with more complex (recursively defined) data?
  – we need a more sophisticated form of induction
Structural Induction

How to prove $\forall x \in S, P(x)$ is true:

**Base Case:** Show that $P(u)$ is true for all specific elements $u$ of $S$ mentioned in the *Basis step*.

**Inductive Hypothesis:** Assume that $P$ is true for some arbitrary values of each of the *existing named elements* mentioned in the *Recursive step*.

**Inductive Step:** Prove that $P(w)$ holds for each of the *new elements* $w$ constructed in the *Recursive step* using the named elements mentioned in the *Inductive Hypothesis*.

**Conclude** that $\forall x \in S, P(x)$.
Structural Induction

How to prove $\forall x \in S, P(x)$ is true:

**Base Case:** Show that $P(u)$ is true for all specific elements $u$ of $S$ mentioned in the Basis step.

**Inductive Hypothesis:** Assume that $P$ is true for some arbitrary values of each of the existing named elements mentioned in the Recursive step.

**Inductive Step:** Prove that $P(w)$ holds for each of the new elements $w$ constructed in the Recursive step using the named elements mentioned in the Inductive Hypothesis.

Conclude that $\forall x \in S, P(x)$.
Structural Induction vs. Ordinary Induction

Structural induction follows from ordinary induction:

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

Ordinary induction is a special case of structural induction:

Recursive definition of $\mathbb{N}$

**Basis:** $0 \in \mathbb{N}$

**Recursive step:** If $k \in \mathbb{N}$ then $k + 1 \in \mathbb{N}$