Induction on Flat Lists, Strong Induction, and Introduction to Combinatorics

Previously:
- Numerical induction
- Introduce list-based induction

Today:
- Finish list-based induction
- Strong induction
- Introduce combinatorics

Then:
- Assignment 5 out later today

Example: Recursive definition of exponentiation

Exponentiation on the natural numbers can be defined recursively, similar to multiplication.

We defined multiplication as:

\[ x \cdot 0 = 0 \]
\[ x \cdot S(y) = (x \cdot y) + x \]

We can define exponentiation as:

\[ x^0 = 1 \]
\[ x^{S(k)} = x^k \cdot x \]
Example: Recursive definition of exponentiation

We can define exponentiation as:
\[ x^0 = 1 \]
\[ x^{S(k)} = x^k \cdot x \]
This lets you generate any natural power of \( x \):
\[ x^1 = x^0 \cdot x = x \]
\[ x^2 = x^1 \cdot x = x \cdot x \]
\[ x^3 = x^2 \cdot x = x \cdot x \cdot x \]

Note: This definition only works for non-zero real numbers \( x \); \( 0^0 \) is undefined.

Example: Inductive proof about exponentiation

Claim: For real numbers \( a \neq 0 \) and natural numbers \( m \) and \( n \), \( a^m \cdot a^n = a^{m+n} \)

You should know this is true from high-school math, but let’s see how we can use our recursive definition of exponentiation to prove it inductively.

\((1)\) \( x^0 = 1 \)
\((2)\) \( x^{S(k)} = x^k \cdot x \)

Claim: For real numbers \( a \neq 0 \) and natural numbers \( m \) and \( n \), \( a^m \cdot a^n = a^{m+n} \)

Proof. By induction.

Base case: \( n = 0 \)
\[ a^m \cdot a^0 = a^m \cdot 1 \]
\[ = a^m \]
\[ = a^{m+0} \]
by mult. prop. of 1
by additive identity

Inductive case: ...

(1) \( x^0 = 1 \)
(2) \( x^{S(k)} = x^k \cdot x \)

Claim: For real numbers \( a \neq 0 \) and natural numbers \( m \) and \( n \), \( a^m \cdot a^n = a^{m+n} \)

Proof. By induction.

Base case: ...

Inductive case:
Suppose as our inductive hypothesis \( a^m \cdot a^k = a^{m+k} \).
Then \( a^m \cdot a^{S(k)} = a^m \cdot (a^k \cdot a) \) by (2)
\[ = (a^m \cdot a^k) \cdot a \] by algebra
\[ = a^{m+k} \cdot a \] substitution
\[ = a^{S(m+k)} \] by (2)
\[ = a^{m+S(k)} \] by associativity of \( S \)

Conclusion: Therefore, \( a^m \cdot a^n = a^{m+n} \) for all natural numbers \( n \).
Structural induction

**Structural induction** is used to prove claims about all items constructed by a recursive definition.

To prove property $P$ holds for all elements of a recursively defined set:

- **Base case(s):** Show that $P$ holds for every element in the basis for the recursive definition.
- **Inductive case(s):** Show that every constructor in the definition preserves property $P$.

What’s a constructor?

In the Peano axioms:

- Every natural number $n$ has a natural number *successor*, denoted by $S(n)$.

It stipulates how additional natural numbers are constructed from existing numbers.

Structural induction example

The transitive closure of a set $R$ is defined as:

- $R_0 = R$
- $R_{n+1} = R_n \cup \{ (a, c) \mid \exists x \text{ s.t. } (a, x) \in R_n \text{ and } (x, c) \in R \}$

**Claim:** The original set $R$ is a subset of $R_i$ for all $i$. 

(1) \( R_0 = R \)
(2) \( R_{n+1} = R_n \cup \{(a, c) \mid \exists x \text{ s.t. } (a, x) \in R_n \text{ and } (x, c) \in R\} \)

\( P(i) \): \( R \subseteq R_i \)

Claim: \( P(i) \) holds for all natural numbers \( i \).

Proof: By structural induction on the definition of transitive closure.

Base case: \( i = 0 \).
\[ R \subseteq R_0 \]
\[ R \subseteq R \quad \text{by (1)} \]

Inductive case:
Assume as our inductive hypothesis \( P(k) : R \subseteq R_k \)
Show \( P(k+1) \).
\[ R_k \cup \{(a, c) \mid \exists x \text{ s.t. } (a, x) \in R_k \text{ and } (x, c) \in R\} \]

Structural induction and lists

Just as the Peano axioms defined the set of real numbers, we can give axioms to define a set of lists.

We define our set of lists over a domain \( D \), e.g., for lists of integers, \( D \) would be \( \mathbb{Z} \).

Definition of a set of lists

1. The empty list, denoted by \( \square \), is an element of \( L_D \).
2. For each item \( d \in D \) and each list \( l \in L_D \), there is a constructed list \( (d : l) \) in \( L_D \), where \( d \) is the first element and \( l \) is the rest.
3. No constructed list is equal to the empty list.
4. For any items, \( d_1 \) and \( d_2 \in D \) and any lists \( l_1 \) and \( l_2 \in L_D \), if \( d_1 \neq d_2 \) or \( l_1 \neq l_2 \), \( (d_1 : l_1) \neq (d_2 : l_2) \).
5. Let \( P \) be any property for which the following conditions hold:
   - \( P(\square) \) holds.
   - For any \( d \in D \) and any \( l \in L_D \), if \( P(l) \) holds, then \( P(d : l) \) also holds.

Then \( P(l) \) holds for every \( l \in L_D \).
Constructing lists

Axiom 2 is again our constructor:

- For each item \( d \in D \) and each list \( l \in L_D \), there is a constructed list \((d : l)\) in \( L_D \), where \( d \) is the first element and \( l \) is the rest.

We prepend (\texttt{cons}) an item \( d \) onto a list \( l \).

Proofs over flat lists

Axiom 5 gives the basis for proof by induction over flat lists.

What do we do proofs about?
Let’s define functions over flat lists.

Defining functions over flat lists

Just as we can prove that every real number is either 0 or the successor of a number, we can prove that every list in \( L_D \) is either \( \texttt{\_} \) or \((d : l)\) for some item \( d \) and list \( l \).

So, to define a function \( f \) over flat lists:

Define \( f(\texttt{\_}) \)
Define \( f(d : l) \) in terms of \( d \) and \( f(l) \)

Like you’ve been doing in Racket since CMPU 101!

Inductive proofs for functions over flat lists

[See example in section 4 of the handout.]
Now we can prove properties about the code we write rather than just run tests.
Recursive definitions for strings

The theory of computation, which you’ll learn about if you take CMPU 240, is based on languages — sets of strings.

These sets of strings are defined very much like the sets of lists we’ve defined.

The set $\Sigma^*$ of all strings over an alphabet of symbols $\Sigma$ is defined recursively:

1. $\varepsilon$, the empty string, is a string in $\Sigma^*$.
2. If $x$ is a string and $a \in \Sigma$, then the concatenation $ax$ is a string in $\Sigma^*$.

Limits of standard induction

Standard induction:

Inductive hypothesis: $P(k)$.

Assuming the inductive hypothesis, show $P(k + 1)$.

Sometimes the inductive hypothesis isn’t strong enough to prove what we need.
Let’s try an example.

Induction example

Claim: Every natural number $n \geq 2$ is the product of one or more prime numbers.

Recall that a natural number $n \geq 2$ is prime iff there is no pair of strictly smaller natural numbers $a$ and $b$ such that $a \cdot b = n$. 
**Induction example**

**Claim:** Every natural number \( n \geq 2 \) is the product of one or more prime numbers.

**Proof:** By induction.

- **Base case:** \( n = 2 \). We know \( 2 = 1 \cdot 2 \), which are prime.
- **Inductive case:** Assume arbitrarily chosen \( k \) is either prime or the product of primes, then prove \( k + 1 \) is also.
  - **Problem:** \( k + 1 \) isn’t built from \( k \) in a way that is helpful for this proof.
  - This isn’t a useful inductive hypothesis; we need something stronger!

**Intuition for strong induction**

If the two cases \( P(k) \) and \( P(k+1) \) aren’t nicely related, the deduction may not go through.

But \( P(k + 1) \) may still depend on earlier cases of \( P(n) \).

These \( P(n) \) can be considered to be already established by the time you’re dealing with \( P(k + 1) \).

Therefore, you ought to be able to use any of them to help demonstrate \( P(k+1) \), not just \( P(k) \).

**Strong induction**

An alternative to standard induction, following from the same principles; think of dominoes falling.

If the base case is some number \( b \), i.e.,

- **Base case:** Show \( P(b) \) is true.
Then in strong induction, the inductive hypothesis is:
- Assume \( P(n) \) for all \( b \leq n \leq k \).
And the inductive case is:
- Show \( P(k+1) \).

**Strong induction: simpler formulation**

Since we’re no longer relating a result to its immediate predecessor, there’s no good reason to use \( k + 1 \) as the focus of the induction step; we can use \( k \) instead.

- **Base case:** Show \( P(b) \) is true.
- **Inductive hypothesis:** Assume \( P(n) \) for all \( b \leq n < k \).
- **Inductive case:** Show \( P(k) \).
**Strong induction example**

**Claim:** Every natural number \( n \geq 2 \) is the product of one or more prime numbers.

**Proof:** By **strong** induction.

Let \( P(n) \) be “\( n \) is either prime or the product of primes.”

**Base case:** \( n = 2 \), and 2 is prime.

**Inductive case:**

As our inductive hypothesis, we assume \( P(n) \) for all \( 2 \leq n < k \) (for arbitrarily chosen \( k \)).

Show \( P(k) \)...

Let \( P(n) \) be “\( n \) is either prime or the product of primes.”

**Base case:** \( n = 2 \), and 2 is prime.

**Inductive case:**

As our inductive hypothesis, we assume \( P(n) \) for all \( 2 \leq n < k \) (for arbitrarily chosen \( k \)).

Show \( P(k) \). The number \( k \) must be either prime or not prime. We'll consider each case in turn:

**Case 1:** If \( k \) is a prime, then we’re done.

**Case 2:** If \( k \) is not prime, then there is an integer \( p \) with \( 2 \leq p \leq k \) that divides \( k \). That is,

\[
 k = pq, \text{ where } 2 \leq p \leq k \text{ and } 2 \leq q \leq k
\]

By the inductive hypothesis, \( p \) is a prime or a product of primes, as is \( q \). Thus, \( k \) is a product of primes.

**Strong induction can be considered a variation of normal induction:**

Replace the property

\[ P(n) \]

by the property

\[ Q(n) = P(b) \text{ and } P(b+1) \text{ and ... and } P(n) \]
Counting

Counting things is an important part of problem solving in computer science:

- The number of elements in a set or subsets of a set
- The number of routes through a map that visit every city
- The number of words of length $L$ formed from alphabet $A$

Counting can be important for brute-force solutions, i.e., exhaustively enumerating every possibility under some circumstances, probability, etc.

Principles for counting elements of sets

- **Subtraction principle for the difference of finite sets:**
  \[ |A - B| = \ldots? \]
- **Addition principle for finite sets:**
  \[ |A \cup B| = \ldots? \]

Principles for counting elements of sets

- **Subtraction principle for the difference of finite sets:**
  \[ |A - B| = |A| - |A \cap B| \]
- **Addition principle for finite sets:**
  \[ |A \cup B| = |A| + |B| - |A \cap B| \]
  If $A$ and $B$ are disjoint, then $|A \cup B| = |A| + |B|$

Counting example

A logic class has 20 students who also take calculus, 9 who also take philosophy, 11 who take neither, and 2 who take both calculus and philosophy.

How many students are in the class?
Counting example
How many distinct license plates are there consisting of two letters followed by four digits?

This is easier than counting the number of plates consisting of two letters and four digits without the restriction that letters come first.

Selection: Order and repetition
Sometimes it’s useful to consider the number of ways to select (or choose) \( k \) items out of \( n \) items.

Two factors to consider:
- **Order**: Does the order matter when choosing the elements, e.g., is selecting \( a, b \) different than selecting \( b, a \)?
- **Repetition**: Can the same element be selected multiple times?

Selection example
How many ways can we choose two letters from \{a, b, c, d, e\}? 

- If order matters and repetition is permitted?
- If order matters and repetition is not permitted?
- If order doesn’t matter and repetition is permitted?
- If order doesn’t matter and repetition is not permitted?