COMPSCI 230: Discrete Mathematics for Computer Science

Recitation 5: Weak and Strong Induction

Spring 2020 *Created By: David Fischer*

Recall the boiler plate for weak induction: For a proof by weak (ordinary) induction in some domain S with an order, we start by stating we will proceed by weak induction. We first show $P(1)$ for some predicate $P(n) : S \to \{true, false\}$ (and some base case $1 \in S$). Then, we assume for some arbitrary $k \in S$, $P(k-1)$ holds. Using this, we show that $P(k)$ must hold. We conclude by stating that by induction we have shown for all $t \geq 1,\ P(t)$ holds.

1. We begin with a quick and easy example of weak induction. Calculate a closed form expression for the n^{th} *triangle number*, that is prove the following using weak induction.

   Theorem 1. $\forall n \in \mathbb{N}.\ \sum_{i=1}^{n}(i) = \frac{n(n+1)}{2}$

   Solution

   - Our base case is $P(1) = \frac{1(2)}{2} = 1$ as needed.
   - Assume for some arbitrary positive integer t, that $P(t-1)$ is true.
   - Note that $\sum_{i=1}^{t}(i) = \sum_{i=1}^{t-1}(i) + (t)$. By our inductive hypothesis we have, then, $\sum_{i=1}^{t-1}(i) + (t) = \frac{(t-1)t}{2} + t = \frac{(t-1)t+2t}{2} = \frac{(t((t-1)+2)}{2} = \frac{t(t+1)}{2}$, and so by weak induction theorem 1 is proven.

2. Dangers of Induction

   Theorem 2. *All horses are the same color.*

   Proof. We proceed by induction on the number of horses, $n = 1$. For $n = 1$, clearly one horse can only be one color (we define "color" to include patterns of colors), so our property holds. Now, suppose for some positive integer $k > 1$, we have $P(1) \wedge P(2) \wedge ... \wedge P(k-1)$. Now, in a set of k horses, call it H, consider two horses h_1 and h_2 in H, and the sets $A = H \setminus \{h_1\}$ and $B = H \setminus \{h_2\}$. Note that $A \cup B = H$, so that $|H| = |A \cup B| = k$ and that $|A| = k-1$ and $|B| = k-1$. Since the size of A and B is less than k, our inductive hypothesis states that our property holds true for all horses in A and B. Let h_3 be a horse in $A \cap B$. Since all horses in both A and B are the same color, we have that h_1 has the same color as all horses in B, which includes h_3, and h_2 has the same color as all horses in A, which also includes h_3. So, the color of h_1, h_2 and h_3 are all the same, and so the color of all horses in $K = A \cup B$ must be the same. So, by induction we have proven $P(n)$ for all positive integers n. □

   Discussion
   Note this proof is clearly invalid - we can prove the theorem false by giving a simple counter example of two horses that do not share a color. So, what went wrong? We must be very clear about how we move from one step to the next - here our inductive step relies on an *intersection* between A and B, and then used a base case of $n = 1$. However, it is not true that $P(1) \to P(2)$, as our inductive step does not hold if $A \cap B = \emptyset$. So, if we wanted this proof to work, we must use $n = 2$ as our base case (and in fact $P(2)$ is false).

3. We now give a relatively easy example of a proof by strong induction.
Recall the "boilerplate" for a proof by strong induction of a statement of the form $\forall n \in \mathbb{Z}_0^+.P(n)$ for some predicate P. (Importantly, when the domain of discourse is different, the steps might differ slightly; specifically, the so-called 'base case' might be different.) Here we give two boiler plates, note carefully the differences.

   (a) State the following proof is by strong induction
   (b) For some positive integer k, prove $P(0) \wedge P(1) \wedge ... \wedge P(k-1)$ directly.
   (c) Prove $\forall n \in \mathbb{Z}_0^+ \setminus \{0\}.(P(n-k) \wedge P(n-k+1) \wedge P(n-k+2) \wedge ... \wedge P(n-1)) \rightarrow P(n)$
   Review - How to prove this universally quantified implication:
      i. Assume $P(i)$ holds for every non-negative integer i such that $n-k \leq i < n$, i.e. assume $P(n-k) \wedge P(n-k+1) \wedge ... \wedge P(n-1)$
      ii. Show that $P(n)$ holds.
   (d) Conclude that $\forall n \in \mathbb{Z}.P(n)$ by strong induction (i.e. by the statements proven in steps 3 and 4 and the strong induction principle).

Alternatively, the following form is used:

   (a) State the following proof is by strong induction
   (b) Prove P(0)
   (c) Prove $\forall n \in \mathbb{Z}_0^+ \setminus \{0\}.(P(0) \wedge P(1) \wedge P(2) \wedge ... \wedge P(n-1)) \rightarrow P(n)$
   Review - How to prove this universally quantified implication:
      i. Let n be an arbitrary non-negative integer.
      ii. Assume $P(k)$ holds for every non-negative integer k such that $k < n$, i.e. assume $P(0) \wedge P(1) \wedge ... \wedge P(n-1)$
      iii. Show that $P(n)$ holds.
   (d) Conclude that $\forall n \in \mathbb{Z}.P(n)$ by strong induction (i.e. by the statements proven in steps 3 and 4 and the strong induction principle).

- We now consider the *fundamental theorem of arithmetic.*

  Theorem 3. *Every non-prime positive integer greater than one can be written as the product of prime numbers.*

  Proof. We proceed by strong induction. Note that we say a product with only one term is still a product for simplicity's sake. $P(n) \leftrightarrow n$ can be written as the product of prime numbers. For a base case, note that 2 can be written as $2 = 2$, so $P(2)$ holds. Suppose for some integer $k > 1$, we have $P(2) \wedge ... \wedge P(k-1)$. Now, we have two cases for k. If k is prime, then we are done and theorem 3 holds (it's a product with one term). If k is not prime, then there are two integers n, m such that $n * m = k$. Then note that $n < k$ and $m < k$, so that if either m or n is not itself prime, it can be written as the product of prime numbers (precisely, say $\exists S$ such that $((\prod_{s \in S} = m) \wedge (\forall s \in S,\ s$ is prime), and similarly $\exists T$ such that $((\prod_{t \in T} = n) \wedge (\forall t \in T,\ t$ is prime)). Then, k can be written as the product of prime numbers ($\prod_{s \in S} s * \prod_{t \in T} t = k$), and so theorem 3 is proven by strong induction. □

- A more complicated example of strong induction (from Stanford's lectures on induction)
  Recall the definition of a *continued fraction*: a number is a continued fraction if it is either some integer n or $n + \frac{1}{F}$, where F is a continued fraction. (Some examples: $1 + \frac{1}{1+\frac{1}{2}} = \frac{5}{3}$, $\pi = 3 + \frac{1}{7+\frac{1}{15+\frac{1}{1+\frac{1}{1+\frac{1}{292+\frac{1}{\dots}}}}}}$. We will give three theorems related to rational numbers, and will prove part of one of them using strong induction. For reference, recall the *division theorem* from grade school: $\forall n, m \in \mathbb{Z},\ \exists x, y \in \mathbb{Z}.(n > m) \to n = x * m + y \land y < n \land x < n$ (x is the *quotient*, and y is the *remainder*).

  Theorem 4. *Any rational number (including negative numbers) can be represented as a finite continued fraction.*

  Theorem 5. *Any irrational number can be represented as an infinite continued fraction.*

  Theorem 6. *If we progressively truncate a continued fraction representation of an irrational number, we can achieve progressively better approximates of the irrational number.*

  We will prove Theorem 4 using strong induction.

  Theorem 7. *All rational numbers have a continued fraction representation.*

  Proof. We proceed by strong induction on the denominator of any rational number. Let $P(n) \leftrightarrow$ any rational number with denominator n can be written as a continued fraction. Note that $P(1) = \frac{q}{1}$ for some integer q, which is a continued fraction as desired. Now, suppose for the sake of strong induction that for some $k \in \mathbb{N}$ we have $P(1), P(2), ..., P(k-1)$. Now consider some rational number with denominator k, call it $\frac{j}{k}$ for $j \in \mathbb{Z}$. Using the division theorem, note that $\exists a, b \in \mathbb{Z}.\ j = ak + b$. We now do cases on b. Case 1: $b = 0$. In this case $\frac{j}{k} = a$, so a is a continuing fraction for the rational number $\frac{j}{k}$. Case 2: $b \neq 0$. In this case, consider the following application of the division theorem:

  $$\begin{aligned} j &= ak + b \\ \frac{j}{k} &= a + \frac{b}{k} \\ &= a + \frac{1}{k/b} \end{aligned}$$

  Now, remember by the division theorem, $b < k$, and so by our inductive hypothesis, there is a continuing fraction representation of $\frac{k}{b}$, call it F. Finally, $\frac{j}{k} = a + \frac{1}{F}$, and the right hand side is a continuing fraction, and so we have a continuing fraction representation for $\frac{j}{k}$ as desired. Note that negative rational numbers are encompassed by this proof, as $\frac{p}{-q} = \frac{-p}{q}$, and we have shown it holds for all positive denominators. □

4. We now move to some more applicable theorems that can be useful in your computer science career. Recall Binary Search:

Algorithm 1: Binary Search(A,a,b,x)

```
Result: Determine whether x is in a sorted array A[1...n], and return the index of x.
if a > b then
  | return False;
end
else
  | mid ← floor((a+b)/2);
end
if x = A[mid] then
  | return (true,mid);
end
if x < A[mid] then
  | Binary Search(A,a+1,mid,x);
end
else
  | Binary Search(A,mid,b-1,x);
end
```

If we didn't know this worked ahead of time (because our professor said so), how could we be sure it will always work?

Theorem 8. *Binary search is correct.*

First, we have to figure out what this statement even means. *get ideas from students* What we mean is the following two statements:

Theorem 9. *If a key exists in a collection, binary search finds that key.*

Proof. Suppose the list A contains the key x. We proceed by induction on $n = b - a$. Note that we use 0-based indexing. Let $P(n)$ be the statement, for a list which contains the key, binary search correctly returns the key if $b - a = n$. $P(1)$ is true, since the algorithm correctly sets $mid = \text{floor}\frac{0+1}{2} = 0$, and then returns (true,0). Assume that $\forall n > k \geq 1$, binary search correctly returns the key and index for that key. After the assignment of mid, there are three cases. Either $x > A[mid]$, $x < A[mid]$ or $x = A[mid]$.

- Clearly, if $x = A[mid]$, then the algorithm correctly returns true
- If $x > A[mid]$, then since the array is sorted, we know that $x \in A[mid + 1 : b]$. Now, if the recursive call is correct, then the algorithm is correct. If $a + b$ is odd, then $mid = \frac{a+b-1}{2}$, so the value of n for the recursive call is $b - 1 - \frac{a+b-1}{2} = \frac{b-a}{2} - \frac{1}{2} < b - a$ surely. If $a + b$ is even, then $mid = \frac{a+b}{2}$, so our value of n for the recursive call is $b - 1 - \frac{a+b}{2} = \frac{b-a}{2} - 1 < b - a$ surely. So, if $x > A[mid]$, by the inductive hypothesis the algorithm is correct, and so the theorem holds.

- The case where $x < A[mid]$ is similar to the previous case, but for completeness is shown here. If $x < A[mid]$, then surely $x \in A[a : mid - 1]$. We have two cases on the value of $a + b$. If $a + b$ is even, then $mid = \frac{a+b}{2}$, so that the value of n for our recursive call is $\frac{a+b}{2} - a - 1 = \frac{b-a}{2} - 1 < b - a$ surely, and so by the inductive hypothesis, our theorem is shown. If $a + b$ is odd, then $mid = \frac{a+b-1}{2}$, and so the value of n for our recursive call is $\frac{a+b-1}{2} - a - 1 = \frac{b-a-3}{2} < b - a$, and so by our inductive hypothesis the theorem holds.

So, in all cases, our algorithm reduces to a case covered by our inductive hypothesis, and so we have shown theorem 9. □

Theorem 10. *If a key does not exists in a collection, binary search reports that there is no match.*

This is a three line proof by contradiction; say it to yourself before reading ahead.

Proof. Suppose for the sake of contradiction the key does not exist in a collection, but binary search reports a match. Since binary search reported a match, at some point it must have found $A[mid] = x$. This contradicts the assumption that $x \notin A$. □

5. Structural Induction
Recall the definition of a coloring and BFS from last recitation and class. Consider the following algorithm to define a coloring on a graph $G = (V, E)$, given that the size of the range is two (i.e. to define a two-coloring $f : V \to C$, where $C = \{c_1, c_2\}$). Pick an arbitrary starting vertex v. Perform $BFS(v)$, with the following modification to the $EXPLORE(v, explored)$ subroutine:

Algorithm 2: $2COLOR\_BFS(v, queue, explored, f)$

```
Result: Coloring function f
for (u ∈ V | (v, u) ∈ E) do
    if u ∉ explored then
        explored := explored ∪ {u};
        if f(v) == c1 then
            f(u) := c2;
        else
            f(u) := c1;
        end
    end
end
if !queue.empty() then
    2COLOR_BFS(queue.dequeue(), queue, explored, f);
end
return f
```

Theorem 11. *This algorithm is correct on a graph G if and only if G is bipartite.*

Lemma 1. $\leftarrow$. *If G is bipartite, the algorithm is correct.*

Proof. To start, note that the algorithm sets the descendent, b, of any node, a, such that $f(a) \neq f(b)$. As well, note that the algorithm is correct if no two nodes in the same partition, as defined by bipartite, have the same color. Suppose an arbitrary graph G is bipartite. We will show without loss of generality that $\forall v \in S$, as defined in the definition of bipartite, $f(v) = c_1$, and $\forall u \in S', \ f(u) = c_2$. We proceed by induction on the number of recursions performed. Note that on the first step (not mentioned in the algorithm, and again without loss of generality) for some arbitrary node s in S we set $f(s) = c_1$, and call $2COLOR\_BFS(s, newqueue, s, f)$. Suppose for some recursive step k that our algorithm is correct. Suppose at the k^{th} recursive step we are considering node $t \in S$. Let C be the set of children of node t. Note that by definition of bipartite, nodes $h \in C$ are in S' since t is in S. The algorithm then sets $f(h) = c_2 \forall h \in C$, as desired. Therefore, at step $k+1$ our algorithm is correct. □

Lemma 2. $\rightarrow$ *If the algorithm works, then the graph is bipartite.*

Proof. Suppose the algorithm works on some graph $G = (v, E)$. Then, define a set of nodes $A = \{n \in V | f(n) = c_1\}$, and $B = \{n \in V | f(n) = c_2\}$. Suppose for the sake of contradiction there were an edge between two nodes v_1 and v_2 in A (w.l.o.g.). Then, the algorithm would have set $f(v_1) \neq f(v_2)$, which contradicts our definition of A. □