Mathematical Induction

Up to this point in the course, we've been using only structural induction. Another kind of induction, called *mathematical induction*, proves a property for all natural numbers. This really is a special case of structural induction, since the natural numbers are inductively defined, but mathematical induction doesn't explicitly use the structure of natural numbers.

A proof by mathematical induction has the following structure:

> $P(n)$: State the predicate you want to prove about each natural number n.
>
> Proof by mathematical induction:
>
> **Base case:** Prove that $P(0)$ is true.
>
> **Induction step:** Assume that $P(k)$ is true, for some $k \geq 0$. (This is the induction hypothesis.) Using this assumption, prove that $P(k+1)$ is true.

This variant of mathematical induction is called *weak induction* because in the induction hypothesis you assume only one $P(k)$ is true. In *strong induction*, the induction hypothesis becomes: "Assume that $P(0), P(1), \ldots, P(k)$ is true, for some $k \geq 0$." Weak and strong induction actually are equivalent, so you can choose whichever induction hypothesis is most useful to you.

If necessary, you also can change what you choose to be the base elements in your proof. We used 0 above, but we can choose other base elements in our mathematical induction. This shows why we introduced structural induction first in this course: with structural induction, the base elements usually are obvious, but with mathematical induction, it's quite easy to choose base elements that don't generate all the natural numbers we're trying to prove $P(n)$ for.

Examples

EXERCISE: Prove that $2 + 4 + \cdots + 2k = k(k+1)$ for $k \geq 0$.

Our predicate $P(k)$ is $2+4+\cdots+2k = k(k+1)$. To prove it by mathematical induction for all $k \geq 0$, our base element is 0: $P(0)$ is trivially true. For the induction step, assume that $P(k)$ is true for some $k \geq 0$. Starting from this induction hypothesis, we have

$$\begin{aligned} 2 + 4 + \cdots + 2k &= k(k+1) \\ 2 + 4 + \cdots + 2k + 2(k+1) &= k(k+1) + 2(k+1) \\ &= (k+1)(k+2) \end{aligned}$$

Thus $P(k+1)$ is true, and we have completed our proof.

EXERCISE: Prove that any amount above 3 cents can be formed using 2-cent and 5-cent coins.

Our predicate $P(n)$, stated informally, says that n cents can be formed using 2-cent and 5-cent coins. To prove it by mathematical induction for all $n > 3$, our base element is 4: $P(4)$ is true because we can form 4 cents using two 2-cent coins. For the induction step, assume that $P(n)$ is true for some $n > 3$. This induction hypothesis says that we can form n cents using 2-cent and 5-cent coins. Now we need to show that we can form $n + 1$ cents. If our current formation has a 5-cent coin, we can replace it with three 2-cent coins to form $n + 1$ cents. Otherwise, our

current formation has only 2-cent coins, but since $n \geq 4$, it has *at least two* 2-cent coins; we can replace those two coins by a 5-cent coin. Thus we have formed $n+1$ cents using only 2-cent and 5-cent coins, so $P(n+1)$ is true.

EXERCISE: Prove that

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}.$$

Proving Program Correctness

Just as we used structural induction earlier in the course to prove the correctness of recursive methods over inductively defined data, we can use mathematical induction to prove the correctness of recursive methods over integers. For example, here's the `power` function we wrote earlier:

```
int power(int x, int y) {
  if (y == 0) return 1;
  else return x * power(x, y-1);
}
```

To prove the correctness of this function, the predicate $P(n)$ is that `power(x,n)` returns x^n for all `x`.

We can prove the correctness of nonrecursive functions, too. Here's our factorial function:

```
int fact(int n) {
  int r = 1, i = 1;
  // invariant: r = (i-1)!
  while (i <= n) {
    r *= i;
    i++;
  }
  return r;
}
```

To prove the correctness of this function, the predicate $P(i)$ is precisely the loop invariant!

Quiz

Problems

1. Identify the loop invariant in the following method:

```
int max(int a[]) {
  int i = 0, curmax = Integer.MIN_VALUE;
  // invariant: ???
  while (i < a.length) {
    if (a[i] > curmax)
      curmax = a[i];
    i++;
  }
  return curmax;
}
```

Answer: Unfortunately the invariant here isn't completely obvious: `curmax` is the smallest integer greater than or equal to all the elements of $a[0 \ldots i-1]$. Give 1 point for having something approximately correct.

2. Identify the loop invariant in the following method:

```
int_list insertion_sort(int a[]) {
  int i = 0;
  int_list r = new int_list();
  // invariant: ???
  while (i < a.length) {
    r.insert(a[i]);
    i++;
  }
  return r;
}
```

Answer: `r` is a sorted list containing the elements of $a[0 \ldots i-1]$. Give 1 point for having this.

3. Prove the following equation by mathematical induction:

$$\sum_{i=1}^{n} 2i - 1 = n^2.$$

Answer: The property $P(n)$ is the equation itself. The base case is $P(1)$ since the summation starts at 1: $P(1)$ is true because $2i - 1 = n^2$ for $i = n = 1$. The induction hypothesis is the assumption that $P(n)$ is true for some $n \geq 1$. Starting from the induction hypothesis, we prove $P(n+1)$ as follows:

$$\sum_{i=1}^{n} 2i - 1 = n^2$$

$$2(n+1) - 1 + \sum_{i=1}^{n} 2i - 1 = n^2 + 2(n+1) - 1$$

$$\sum_{i=1}^{n+1} 2i - 1 = (n+1)^2$$

1 point each for stating $P(n)$, showing the base case $P(1)$, stating the induction hypothesis, and identifying the use of the induction hypothesis.

4. Vote for what bonus topic you'd like us to cover during the last two days of class. Possibilities include GUIs and multithreaded programming.