# Notes on induction proofs and recursive definitions

James Aspnes

December 13, 2010

## 1 Simple induction

Most of the proof techniques we've talked about so far are only really useful for proving a property of a single object (although we can sometimes use generalization to show that the same property is true of all objects in some set if we weren't too picky about which single object we started with). **Mathematical induction** (which mathematicians just call *induction*) is a powerful technique for showing that some property is true for many objects, where you can use the fact that it is true for small objects as part of the proof that it is true for large objects.

The basic framework for induction is as follows: given a sequence of statements $P(0), P(1), P(2)$, we'll prove that $P(0)$ is true (the **base case**), and then prove that for all $k$, $P(k) \Rightarrow P(k+1)$ (the **induction step**). We then conclude that $P(n)$ is in fact true for all $n$.

### 1.1 Why induction works

There are three ways to show that induction works, depending on where you got your natural numbers from.

**Peano axioms** If you start with the Peano axioms, induction is one of them. Nothing more needs to be said.

**Well-ordering of the naturals** A set is **well-ordered** if every subset has a smallest element. (An example of a set that is *not* well-ordered is the integers $\mathbb{Z}$.) If you build the natural numbers using $0 = \{\}$ and $x + 1 = x \cup \{x\}$, it is possible to prove that the resulting set is well-ordered. Because it is well-ordered, if $P(n)$ does *not* hold for all $n$, there is a smallest $n$ for which $P(n)$ is false. But then either this $n = 0$, contradicting the base case, or $P(n-1)$ is true (because otherwise $n$ would not be the smallest) and $P(n)$ is false, contradicting the induction step.

**Method of infinite descent** The original version, due to Fermat, goes like this: Suppose $P(n)$ is false for some $n > 0$. Since $P(n-1) \Rightarrow P(n)$

is logically equivalent to $\neg P(n) \Rightarrow \neg P(n-1)$, we can conclude (using the induction step) $\neg P(n-1)$. Repeat until you reach 0. The problem with this version is that the "repeat" step is in effect using an induction argument. The modern solution to this problem is to recast the argument to look like the well-ordering argument above, by assuming that $n$ is the smallest $n$ for which $P(n)$ is false and asserting a contradiction once you prove $\neg P(n-1)$. Historical note: Fermat may have used this technique to construct a plausible but invalid proof of his famous "Last Theorem" that $a^n + b^n = c^n$ has no non-trivial integer solutions for $n > 2$.

### 1.2 Examples

- The pigeonhole principle.
- The number of subsets of an $n$-element set is $2^n$.
- $1 + 3 + 5 + 7 + ... + (2n + 1) = (n + 1)^2$.
- $2^n > n^2$ for $n \geq 5$.

## 2 Strong induction

Sometimes when proving that the induction hypothesis holds for $n+1$, it helps to use the fact that it holds for all $n' < n + 1$, not just for $n$. This sort of argument is called **strong induction**. Formally, it's equivalent to simple induction: the only difference is that instead of proving $\forall k : P(k) \Rightarrow P(k+1)$, we prove $\forall k : (\forall m \leq k : Q(m)) \Rightarrow Q(k+1)$. But this is exactly the same thing if we let $P(k) \equiv \forall m \leq k : Q(m)$, since if $\forall m \leq k : Q(m)$ implies $Q(k+1)$, it also implies $\forall m \leq k+1 : Q(m)$, giving us the original induction formula $\forall k P(k) \Rightarrow P(k+1)$.

### 2.1 Examples

- Every $n > 1$ can be factored into a product of one or more prime numbers. Proof: By induction on $n$. The base case is $n = 2$, which factors as $2 = 2$ (one prime factor). For $n > 2$, either (a) $n$ is prime itself, in which case $n = n$ is a prime factorization; or (b) $n$ is not prime, in which case $n = ab$ for some $a$ and $b$, both greater than 1. Since $a$ and $b$ are both less than $n$, by the induction hypothesis we have $a = p_1 p_2 \ldots p_k$ for some sequence of one or more primes and similarly $b = p'_1 p'_2 \ldots p'_{k'}$. Then $n = p_1 p_2 \ldots p_k p'_1 p'_2 \ldots p'_{k'}$ is a prime factorization of n.
- Every deterministic bounded two-player perfect-information game that can't end in a draw has a winning strategy for one of the players. A perfect-information game is one in which both players know the entire state of the game at each decision point (like Chess or Go, but unlike Poker or Bridge); it is deterministic if there is no randomness that affects

the outcome (this excludes Backgammon and Monopoly, some variants of Poker, and multiple hands of Bridge), and it's bounded if the game is guaranteed to end in at most a fixed number of moves starting from any reachable position (this also excludes Backgammon and Monopoly). Proof: For each position $x$, let $b(x)$ be the bound on the number of moves made starting from $x$. Then if $y$ is some position reached from $x$ in one move, we have $b(y) < b(x)$ (because we just used up a move). Let $f(x) = 1$ if the first player wins starting from position $x$ and $f(x) = 0$ otherwise. We claim that $f$ is well-defined. Proof: If $b(x) = 0$, the game is over, and so $f(x)$ is either 0 or 1, depending on who just won. If $b(x) > 0$, then $f(x) = \max\{f(y)|y \text{ is a successor to } x\}$ if it's the first player's turn to move and $f(x) = \min\{f(y)|y \text{ is a successor to } x\}$ if it's the second player's turn to move. In either case each $f(y)$ is well-defined (by the induction hypothesis) and so $f(x)$ is also well-defined.

- The **division algorithm**: For each $n, m \in \mathbb{N}$ with $m \neq 0$, there is a unique pair $q, r \in \mathbb{N}$ such that $n = qm + r$ and $0 \leq r < m$. Proof: Fix $m$ then proceed by induction on $n$. If $n < m$, then if $q > 0$ we have $n = qm + r \geq 1 \cdot m \geq m$, a contradiction. So in this case $q = 0$ is the only solution, and since $n = qm + r = r$ we have a unique choice of $r = n$. If $n \geq m$, by the induction hypothesis there is a unique $q'$ and $r'$ such that $n - m = q'm + r'$ where $0 \leq r' < m$. But then $q = q' + 1$ and $r = r'$ satisfies $qm + r = (q' - 1 + 1)m + r = (q'm + r') + m = (n - m) + m = n$. To show that this solution is unique, if there is some other $q''$ and $r''$ such that $q''m + r'' = n$, then $(q'' - 1)m + r'' = n - m = q'm + r'$, and by the uniqueness of $q'$ and $r'$ (ind. hyp. again), we have $q'' - 1 = q' = q - 1$ and $r'' = r' = r$, giving that $q'' = q$ and $r'' = r$. So $q$ and $r$ are unique.

# 3 Recursion

A definition with the structure of an inductive proof (give a base case and a rule for building bigger structures from smaller ones) Structures defined in this way are **recursively-defined**.

Examples of recursively-defined structures:

**Finite Von Neumann ordinals** A finite von Neumann ordinal is either (a) the empty set $\emptyset$, or (b) $x \cup \{x\}$, where $x$ is a finite von Neumann ordinal.

**Complete binary trees** A complete binary tree consists of either (a) a *leaf node*, or (b) an *internal node* (the *root*) with two complete binary trees as children (or *subtrees*).

**Boolean formulas** A boolean formula consists of either (a) a variable, (b) the negation operator applied to a Boolean formula, (c) the AND of two Boolean formulas, or (d) the OR of two Boolean formulas. A monotone Boolean formula is defined similarly, except that negations are forbidden.

**Finite sequences, recursive version** Before we defined a finite sequence as a function from some natural number (in its set form: $n = \{0, 1, 2, ..., n-1\}$) to some set $S$. We could also define a finite sequence over $S$ recursively, by the rule: $\langle\rangle$ (the empty sequence) is a finite sequence, and if $a$ is a finite sequence and $x \in S$, then $(x, a)$ is a finite sequence. (Fans of LISP will recognize this method immediately.)

The key point is that in each case the definition of an object is **recursive**—the object itself may appear as part of a larger object. Usually we assume that this recursion eventually bottoms out: there are some base cases (e.g. leaves of complete binary trees or variables in Boolean formulas) that do not lead to further recursion. If a definition doesn't bottom out in this way, the class of structures it describes might not be well-defined (i.e., we can't tell if some structure is an element of the class or not).

## 3.1 Recursively-defined functions

We can also define functions on recursive structures recursively:

**The depth of a binary tree** For a leaf, 0. For a tree consisting of a root with two subtrees, $1 + \max(d_1, d_2)$, where $d_1$ and $d_2$ are the depths of the two subtrees.

**The value of a Boolean formula given a particular variable assignment** For a variable, the value (true or false) assigned to that variable. For a negation, the negation of the value of its argument. For an AND or OR, the AND or OR of the values of its arguments. (This definition is not quite as trivial as it looks, but it's still pretty trivial.)

Or we can define ordinary functions recursively:

**The Fibonacci series** Let $F(0) = F(1) = 1$. For $n > 1$, let $F(n) = F(n-1) + F(n-2)$.

**Factorial** Let $0! = 1$. For $n > 0$, let $n! = n \cdot ((n-1)!)$.

## 3.2 Recursive definitions and induction

Recursive definitions have the same form as an induction proof. There are one or more base cases, and one or more recursion steps that correspond to the induction step in an induction proof. The connection is not surprising if you think of a definition of some class of objects as a predicate that identifies members of the class: a recursive definition is just a formula for writing induction proofs that say that certain objects are members.

Recursively-defined objects and functions also lend themselves easily to induction proofs about their properties; on general structures, such induction arguments go by the name of *structural induction*.

# 4 Structural induction

For finite structures, we can do induction over the structure. Formally we can think of this as doing induction on the size of the structure or part of the structure we are looking at.

Examples:

**Every complete binary tree with $n$ leaves has $n-1$ internal nodes** Base case is a tree consisting of just a leaf; here $n = 1$ and there are $n - 1 = 0$ internal nodes. The induction step considers a tree consisting of a root and two subtrees. Let $n_1$ and $n_2$ be the number of leaves in the two subtrees; we have $n_1 + n_2 = n$; and the number of internal nodes, counting the nodes in the two subtrees plus one more for the root, is $(n_1 - 1) + (n_2 - 1) + 1 = n_1 + n_2 - 1 = n - 1$.

**Monotone Boolean formulas generate monotone functions** What this means is that changing a variable from false to true can never change the value of the formula from true to false. Proof is by induction on the structure of the formula: for a naked variable, it's immediate. For an AND or OR, observe that changing a variable from false to true can only leave the values of the arguments unchanged, or change one or both from false to true (induction hypothesis); the rest follows by staring carefully at the truth table for AND or OR.

**Bounding the size of a binary tree with depth $d$** We'll show that it has at most $2^{d+1} - 1$ nodes. Base case: the tree consists of one leaf, $d = 0$, and there are $2^{0+1} - 1 = 2 - 1 = 1$ nodes. Induction step: Given a tree of depth $d > 1$, it consists of a root (1 node), plus two subtrees of depth at most $d - 1$. The two subtrees each have at most $2^{d-1+1} - 1 = 2^d - 1$ nodes (induction hypothesis), so the total number of nodes is at most $2(2^d - 1) + 1 = 2^{d+1} + 2 - 1 = 2^{d+1} - 1$.