Induction and Recursion

Lucia Moura

Winter 2010

Mathematical Induction

Principle (of Mathematical Induction)

Suppose you want to prove that a statement about an integer n is true for every positive integer n.

- *Define a propositional function $P(n)$ that describes the statement to be proven about n.*
- *To prove that $P(n)$ is true for all $n \geq 1$, do the following two steps:*
  - *Basis Step: Prove that $P(1)$ is true.*
  - *Inductive Step: Let $k \geq 1$. Assume $P(k)$ is true, and prove that $P(k+1)$ is true.*

Types of statements that can be proven by induction

1. **Summation formulas**
   Prove that $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$, for all integers $n \geq 0$.
2. **Inequalities**
   Prove that $2^n < n!$ for every positive integer n with $n \geq 4$.
3. **Divisibility results**
   Prove that $n^3 - n$ is divisible by 3 for every positive integer n.
4. **Results about sets**
   Prove that if S is a set with n elements where n is a nonnegative integer, then S has 2^n subsets.
5. **Creative use of mathematical induction**
   Show that for n a positive integer, every $2^n \times 2^n$ checkerboard with one square removed can be tiled using right triominoes (L shape).
6. **Results about algorithms**
   Prove that procedure $\text{FAC}(n)$ returns $n!$ for all nonnegative integers $n \geq 0$.

Prove that algorithm $\text{FAC}(n)$ returns $n!$ for all nonnegative integers $n \geq 0$.

procedure $\text{FAC}(n$: nonnegative integer)
    if $(n = 0)$ **then return** 1
        else return $n * \text{FAC}(n-1)$

Strong Induction

Principle (of Strong Induction)

Suppose you want to prove that a statement about an integer n is true for every positive integer n.

- *Define a propositional function $P(n)$ that describes the statement to be proven about n.*
- *To prove that $P(n)$ is true for all $n \geq 1$, do the following two steps:*
  - *Basis Step: Prove that $P(1)$ is true.*
  - *Inductive Step: Let $k \geq 1$. Assume $P(1), P(2), \ldots, P(k)$ are all true, and prove that $P(k+1)$ is true.*

Use strong induction to prove:

Theorem (The Fundamental Theorem of Arithmetic)

Every positive integer greater than 1 *can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of nondecreasing size.*

Proof:

Part 1: Every positive integer greater than 1 can be written as a prime or as the product of two or more primes.

Part 2: Show uniqueness, when the primes are written in nondecreasing order.

Proof of Part 1: Consider $P(n)$ the statement "n can be written as a prime or as the product of two or more primes.". We will use strong induction to show that $P(n)$ is true for every integer $n \geq 1$.

BASIS STEP: $P(2)$ is true, since 2 can be written as a prime, itself.

INDUCTION STEP: Let $k \geq 2$. Assume $P(1), P(2), \ldots, P(k)$ are true. We will prove that $P(k+1)$ is true, i.e. that $k+1$ can be written as a prime or the product of two or more primes.

Case 1: $k+1$ is prime.

If $k+1$ is prime, then the statement is true as $k+1$ can be written as itself, a prime.

Case 2: $k+1$ is composite.

By definition, there exist two positive integers a and b with $2 \leq a \leq b < k+1$, such that $k+1 = ab$. Since $a, b < k+1$, we know by induction hypothesis that a and b can each be written as a prime or the product of two or more primes. Thus, $k+1 = ab$ can be written as a product of two or more primes, namely those primes in the prime factorization of a and those in the prime factorization of b.

We want to prove Part 2. The following Lemma has been proven.

Lemma (A)

If a, b, and c are positive integers such that gcd$(a, b) = 1$ and $a|bc$, then $a|c$.

We prove the following lemma using induction.

Lemma (B)

If p is a prime and $p|a_1a_2 \cdots a_n$, where each a_i is an integer and $n \geq 1$, then $p|a_i$ for some i, $1 \leq i \leq n$.

Proof: Let $P(n)$ be the statement "If $a_1, a_2, \ldots, a_n$ are integers and p is a prime number such that $p|a_1a_2 \cdots a_n$, then $p|a_i$ for some i, $1 \leq i \leq n$". We will prove $P(n)$ is true for all $n \geq 1$.

Let $P(n)$ be the statement "If $a_1, a_2, \ldots, a_n$ are integers and p is a prime number such that $p|a_1 a_2 \cdots a_n$, then $p|a_i$ for some i, $1 \leq i \leq n$".
We will prove $P(n)$ is true for all $n \geq 1$.

Basis: Prove that $P(1)$ is true.
The statement is trivially true, since for $n = 1$, $p|a_1$ already gives that $p|a_i$ for some i, $1 \leq i \leq 1$.

Induction step: Let $n \geq 2$. Assume $P(n-1)$ is true. Prove $P(n)$ is true.
Let p be a prime such that $p|a_1 a_2 \cdots a_n$. In the case that $p|a_n$, we are done. So, consider the case $p \not| a_n$. Since p is prime, we have $\gcd(p, a_n) = 1$, thus, by Lemma A, $p|a_1 \ldots a_{n-1}$. By induction hypothesis, we have that $p|a_i$ for some i, $1 \leq i \leq n-1$. Combining both cases, we get that $p|a_i$ for some i, $1 \leq i \leq n$.
□

Proof of Part 2: (uniqueness of the prime factorization of a positive integer).

Suppose by contradiction that n can be written as a product of primes in two different ways, say $n = p_1 p_2 \dots p_s$ and $n = q_1 q_2 \dots q_t$, where each p_i and q_j are primes such that $p_1 \leq p_2 \leq \cdots \leq p_s$ and $q_1 \leq q_2 \leq \cdots \leq q_t$.
When we remove all common primes from the two factorizations, we have: $p_{i_1} p_{i_2} \cdots p_{i_u} = q_{j_1} q_{j_2} \cdots q_{j_v}$, where no primes occur on both sides of this equations and u and v are positive integers.
By Lemma B, p_{i_1} must divide q_{j_k} for some k, $1 \leq k \leq v$. Since p_{i_1} and q_{j_k} are primes we must have $p_{i_1} = q_{j_k}$, which contradicts the fact that no primes appear on both sides of the given equation.
□

Examples of statements that can be proven by strong induction

1. Consider a game with 2 players that take turns removing any positive number of matches they want from one of two piles of matches. The player that removes the last match wins the game. Prove that if both piles contains the same number of matches initially, the second player can always guarantee a win.
2. Prove that every amount of 12 cents or more can be formed with 4-cent and 5-cent stamps. (also try to prove it in a different way using mathematical induction)
3. Prove that algorithm $\text{GCD}(a, b)$ (given in page 313 of the textbook) returns the gcd(a, b) for all integers a, b, $a < b$.

Prove that procedure $\text{GCD}(a, b)$ returns the gcd(a, b) for all integers a, b, $a < b$.

procedure $\text{GCD}(a, b$: nonnegative integers with $a < b)$
    if $(a = 0)$ **then return** b
        else return $\text{GCD}(b \bmod a, a)$

Recursive Definitions

We can use recursion to define:

- functions,
- sequences,
- sets.

Mathematical induction and strong induction can be used to prove results about recursively defined sequences and functions.

Structural induction is used to prove results about recursively defined sets.

Recursively Defined Functions

Examples:

- Defining the factorial function recursively:

$$\begin{aligned} F(0) &= 1, \\ F(n) &= n \times F(n-1), \text{ for } n \geq 1. \end{aligned}$$

- Defining the maximum number of comparisons for the Mergesort algorithm (given in page 318):

$$\begin{aligned} T(1) &= 0, \\ T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1, \text{ for } n \geq 2. \end{aligned}$$

- Number of moves needed to solve the Hanoi tower problem:

$$\begin{aligned} H(1) &= 1, \\ H(n) &= 2H(n-1) + 1, \text{ for } n \geq 2. \end{aligned}$$

Recursively Defined Sequences

Consider the Fibonacci numbers, recursively defined by:

$$\begin{aligned} f_0 &= 0, \\ f_1 &= 1, \\ f_n &= f_{n-1} + f_{n-2}, \text{ for } n \geq 2. \end{aligned}$$

Prove that whenever $n \geq 3$, $f_n > \alpha^{n-2}$ where $\alpha = (1 + \sqrt{5})/2$.

Let $P(n)$ be the statement "$f_n > \alpha^{n-2}$". We will show that $P(n)$ is true for $n \geq 3$ using strong induction.

BASIS: We show that $P(3)$ and $P(4)$ are true:
$\alpha = (1+\sqrt{5})/2 < (1+3)/2 = 2 = f_3$.
$\alpha^2 = ((1+\sqrt{5})/2)^2 = (1^2+2\sqrt{5}+5)/4 = (3+\sqrt{5})/2 < (3+3)/2 = 3 = f_4$.

INDUCTIVE STEP: Let $k \geq 4$. Assume $P(j)$ is true for all integers j with $3 \leq j \leq k$. Prove that $P(k+1)$ is true.
We have:

$$\begin{aligned} f_{k+1} &= f_k + f_{k-1}, \quad \text{(by the definition of the Fibonacci sequence)} \\ &> \alpha^{k-2} + \alpha^{k-3}, \quad \text{(by induction hypothesis)} \\ &= \alpha^{k-3}(\alpha+1) = \alpha^{k-3}((1+\sqrt{5})/2+1) = \alpha^{k-3}((3+\sqrt{5})/2) \\ &= \alpha^{k-3}\alpha^2 = \alpha^{k-1}. \end{aligned}$$

Recursively Defined Sets and Structures

Definition (Set of strings over an alphabet)

The set Σ^* of strings over the alphabet Σ can be defined recursively by:
BASIS STEP: $\lambda \in \Sigma^*$ (where λ is the empty string)
RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Example: If $\Sigma = \{0, 1\}$, then
$\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}$.

Definition (Well-formed formulas of Operators and Operands)

BASIS STEP: x is a well-formed formula if x is a numeral or variable.
RECURSIVE STEP: If F and G are well-formed formulas, then $(F + G)$, $(F - G)$, $(F * G)$, (F/G) and $(F \uparrow G)$ are well-formed formulas.

Example: The following are well-formed formulas:
$(x * 3)$, $(3/0)$, $((x + 2) * y)$, $((2 + 3) - (x/y))$, etc.

Structural Induction

Structural induction is used to show results about recursively defined sets.

Principle (of Structural Induction)

To show that a statement holds for all elements of a recursively defined set, use the following steps:

- BASIS STEP: *Prove that the statement holds for all elements specified in the basis step of the set definition.*
- RECURSIVE STEP: *Prove that if the statement is true for each of the elements used to construct elements in the recursive step of the set definition, then the result holds for these new elements.*

The validity of this principle comes from the validity of mathematical induction, as we can transform the above argument on an induction on n where n is the number of applications of the recursive step of the set definition needed to obtain the element we are analysing.

Example of Structural Induction I

Prove that every well-formed formula of Operators and Operands contains an equal number of left and right parentheses.

Proof by structural induction:

BASIS STEP: A numeral or a variable, each contains 0 parentheses, so clearly they contain an equal number of right and left parentheses.

RECURSIVE STEP: Assume F and G are well-formed formulas each containing an equal number of left and right parentheses. That is, if l_F and l_G are the number of left parentheses in F and G, respectively, and r_F and r_G are the number of right parentheses in F and G, respectively, then $l_F = r_F$ and $l_G = r_G$. We need to show that $(F + G)$, $(F - G)$, $(F * G)$, (F/G) and $(F \uparrow G)$ also contain an equal number of left and right parenthesis. For each of these well-formed formulas, the number of left parentheses is $L = l_F + l_G + 1$ and the number of right parentheses is $R = r_F + r_G + 1$. Since $l_F = r_F$ and $l_G = r_G$, it follows that $L = l_F + l_G + 1 = r_F + r_G + 1 = R$. This concludes the inductive proof. □

Example of Structural Induction II

Recall the definition of a set of strings.

Definition (Set of strings over an alphabet)

The set Σ^* of strings over the alphabet Σ can be defined recursively by:
BASIS STEP: $\lambda \in \Sigma^*$ (where λ is the empty string)
RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

We now give a definition of concatenation of two strings.
Note how this definition is built on the definition of string.

Definition (Concatenation of two strings)

BASIS STEP: If $w \in \Sigma^*$, then $w \cdot \lambda = w$.
RECURSIVE STEP: If $w_1 \in \Sigma^*$, $w_2 \in \Sigma^*$ and $x \in \Sigma$, then
$w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

We now give a recursive definition of the reversal of a string.

Definition (Reversal of a string)

BASIS STEP: $\lambda^R = \lambda$
RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $(wx)^R = x \cdot (w)^R$.

Exercise: Use structural induction to prove that if w_1 and w_2 are strings, then $(w_1 \cdot w_2)^R = w_2^R \cdot w_1^R$.

Note that this proof needs to use the 3 definitions given above.

Proving the correctness of recursive programs

Mathematical induction (and strong induction) can be used to prove that a recursive algorithm is correct:

to prove that the algorithm produces the desired output for all possible input values.

We will see some examples next.

Recursive algorithm for computing a^n

procedure power(a: nonzero real number, n: nonnegative integer)
  if $(n = 0)$ **then** return 1
    else return $a \times$power$(a, n - 1)$

We will prove by mathematical induction on n that the algorithm above is correct.
We will show $P(n)$ is true for all $n \geq 0$, for
$P(n)$: For all nonzero real numbers a, power(a, n) correctly computes a^n.

Proving $power(a, n)$ is correct

Basis: If $n = 0$, the first step of the algorithm tells us that power(a,0)=1. This is correct because $a^0 = 1$ for every nonzero real number a, so $P(0)$ is true.

Inductive step:
Let $k \geq 0$.
Inductive hypothesis: power$(a, k) = a^k$, for all $a \neq 0$.
We must show next that power$(a, k + 1) = a^{k+1}$.
Since $k + 1 > 0$ the algorithm sets power$(a, k + 1) = a \times$power(a, k).
By inductive hypotheses $power(a, k) = a^k$, so
power$(a, k + 1) = a \times$power$(a, k) = a \times a^k = a^{k+1}$.

Recursive algorithm for computing $b^n \mod m$

procedure mpower(b, n, m: integers with $m \geq 2, n \geq 0$)
  if $n = 0$ then return 1;
  else if n is even then return mpower$(b, n/2, m)^2 \mod m$
  else return ((mpower$(b, \lfloor n/2 \rfloor, m)^2 \mod m) * (b \mod m)) \mod m$

Examples:

$$
\begin{aligned}
& power(2,5,6) = \\
= \; & ((power(2,2,6)^2 \mod 6) * (2 \mod 6)) \mod 6 \\
= \; & (((power(2,1,6)^2 \mod 6)^2 \mod 6) * (2)) \mod 6 \\
= \; & ((((power(2,0,6)^2 \mod 6) * (2 \mod 6)) \mod 6)^2 \mod 6)^2 \mod 6 \\
& * 2) \mod 6 \\
= \; & ((((1^2 \mod 6) * 2) \mod 6)^2 \mod 6)^2 \mod 6) * 2) \mod 6 \\
= \; & 2
\end{aligned}
$$

Proving $mpower(a, n, m)$ is correct, using induction on n

Basis: Let b and m be integers with $m \geq 2$, and $n = 0$. In this case, the algorithm returns 1. This is correct because $b^0 \mod m = 1$.

Inductive step:

Induction hypothesis: Let $k \geq 1$. Assume power$(b, j, m) = b^j \mod m$ for all integers j with $0 \leq j \leq k-1$, whenever b is a positive integer and m is an integer with $m \geq 2$.

We must show next that power$(b, k, m) = b^k \mod m$. There are two cases to consider.

- Case 1: k is even. In this case, the algorithm returns mpower$(b, k/2, m)^2 \mod m = (i.h.)(b^{k/2} \mod m)^2 \mod m = b^k \mod m$.
- Case 2: k is odd. In this case, the algorithm returns $((\text{mpower}(b, \lfloor k/2 \rfloor, m)^2 \mod m) * (b \mod m)) \mod m$
  $= (i.h.)(b^{\lfloor k/2 \rfloor} \mod m)^2 \mod m) * (b \mod m)) \mod m$
  $= (b^{2\lfloor k/2 \rfloor + 1} \mod m) = b^k \mod m$.

Program verification

We want to be able to prove that a given program meets the intended specifications.
This can often be done manually, or even by automated program verification tools. One example is PVS (People's Verification System).

A program is **correct** if it produces the correct output for every possible input.
A program has **partial correctness** if it produces the correct output for every input for which the program eventually halts.
Therefore, a program is correct if and only if it has partial correctnes and terminates.

Hoare's triple notation

- A program's I/O specification can be given using initial and final assertions.
  - The **initial assertion** p is the condition that the program's input (its initial state) is guaranteed (by its user) to satisfy.
  - The **final assertion** q is the condition that the output produced by the program (its final state) is required to satisfy.
- Hoare triple notation:
  - The notation $p\{S\}q$ means that, for all inputs I such that $p(I)$ is true, if program S (given input I) halts and produces output $O = S(I)$, then $q(O)$ is true.
  - That is, S is partially correct with respect to specification p, q.

A simple example

- Let S be the program fragment
  `y:= 2; z:= x+y`
- Let p be the initial assertion `x=1`.
  The variable `x` will hold 1 in all initial states.
- Let q be the final assertion `z= 3`.
  The variable `z` must hold 3 in all final states.
- Prove $p\{S\}q$.
  Proof: If `x`=1 in the program's input state, then after running `y`:=2 and `z`:=`x`+`y`, then `z` will be $1 + 2 = 3$.

Rules of inference for Hoare triples

- The composition rule:

$$\frac{p\{S_1\}q \quad q\{S_2\}r}{\therefore p\{S_1; S_2\}r}$$

- It says: If program S_1 given condition p produces condition q, and S_2 given q produces r,
  then the program "S_1 followed by S_2", if given p, yields r.

Inference rule for **if-then** statements

$$\frac{(p \wedge cond)\{S\}q \quad (p \wedge \neg cond) \rightarrow q}{\therefore p\{\textbf{if } cond \textbf{ then } S\}q}$$

Example: Show that: $T\{\textbf{if } x > y \textbf{ then } y := x\}(y \geq x)$.

Proof:
When initially T is true, if $x > y$, then the **if**-body is executed, setting $y = x$, and so afterwards $y \geq x$ is true. Otherwise, $x \leq y$ and so $y \geq x$. In either case, the final assertion $y \geq x$ is true. So the rule applies, and so the fragment meets the specification.

Inference rule for **if-then-else** statements

$$\frac{(p \wedge cond)\{S_1\}q \quad (p \wedge \neg cond)\{S_2\}q}{\therefore p\{\textbf{if } cond \textbf{ then } S_1 \textbf{ else } S_2\}q}$$

Example: Prove that
$T\{\textbf{if } x < 0 \textbf{ then } abs := -x \textbf{ else } abs := x\}(abs = |x|)$

Proof:
If the initial assertion is true and $x < 0$ then after the **if**-body, abs will be $-x = |x|$.
If the initial assertion is true, but $\neg(x < 0)$ is true, i.e., $x \geq 0$, then after the **else**-body, $abs = x$, which is $|x|$.
So using the above rule, we get that this segment is true with respect to the final assertion.

Loop Invariants

For a **while**-loop "**while** cond S", we say that p is a **loop invariant** of this loop if $(p \wedge cond)\{S\}p$.

If p (and the continuation condition $cond$) is true before executing the body, then p remains true afterwards.

And so p stays true through all subsequent iterations.

This leads to the inference rule:

$$\frac{(p \wedge cond)\{S\}p}{\therefore p\{\textbf{while } cond\ S\}(\neg cond \wedge p)}$$

Example1: loop invariant

Prove that the following Hoare triple holds:
$T\{i := 1; fact := 1; \textbf{while } i < n\{i++; fact = fact * i\}\}(fact = n!)$

Proof:
Let p be the assertion "'$fact = i! \wedge i \leq n$". We will show tht p is a loop invariant.
Assume that at the beginning of the **while**-loop p is true and the condition of the **while**-loop holds, in other words, assume that $fact = i!$ and $i < n$.
The new values i_{new} and $fact_{new}$ of i and $fact$ are
$i_{new} = i + 1$ and
$fact_{new} = fact \times (i + 1) = (i!) \times (i + 1) = (i + 1)! = i_{new}!$.
Since $i < n$, we also have $i_{new} = i + 1 \leq n$.
Thus p is true at the end of the execution of the loop. This shows p is a loop invariant.

Final example: combining all rules

procedure multiply$(m, n : integers)$

$p :=$ "$(m, n \in Z)$"

if $n < 0$ **then** $a := -n$ (segment S_1)

else $a := n$

$q :=$ "$(p \wedge (a = |n|))$"

$k := 0; x := 0$ (segment S_2)

$r :=$ "$q \wedge (k = 0) \wedge (x = 0)$"

$(x = mk \wedge k \leq a)$

while $k < a$ { (segment S_3)

$x = x + m; \quad k = k + 1;$

}Maintains loop invariant: $(x = mk \wedge k \leq a)$

$(x = mk \wedge k = a) \therefore s :=$ "$(x = ma) \wedge a = |n|$"

$s \Rightarrow (n < 0 \wedge x = -mn) \vee (n \leq 0 \wedge x = mn)$

if $n < 0$ **then** $prod := -x$ (segment S_4)

else $prod := x$

$t =$ "$(prod = mn)$"

Correctness of multiply(m, n)

The proof is structured as follows, by using propositions p, q, r, s, t as defined in the previous page.

- Prove $p\{S_1\}q$ by using **if-then-else** inference rule.
- Prove $q\{S_2\}r$ by examining this trivial segment.
- Prove $r\{S_3\}s$ by using **while-loop** inference rule.
- Prove $s\{S_4\}t$ by using **if-then-else** inference rule.
- Use the rule of composition to show that $p\{S_1; S_2; S_3; S_4\}t$; recall that $p :=$ "$(m, n \in Z)$" and $t =$ "$(prod = mn)$", which is what we wanted to show for the partial correctness.

To complete the proof of correctness, given the partial correctness, we must verify that each segment terminates.
Termination is trivial for segments S_1, S_2 and S_4; for the **while-loop** (S_4) it is easy to see that it runs for a iterations.
(See general rule for proving termination of loops in the next page)
We leave the details of each step above as an exercise.

Proving termination of a loop

- Associate with each iteration i a natural number k_i, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.

Proving termination of a loop

- Associate with each iteration i a natural number k_i, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.
- Using the well-ordering principle, every decreasing sequence of natural numbers is finite.

Proving termination of a loop

- Associate with each iteration i a natural number k_i, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.
- Using the well-ordering principle, every decreasing sequence of natural numbers is finite.
- Find a decreasing sequence of natural numbers for the while-loop in the previous example:

  Define $k_i = a - k$
  $< k_0, k_1, k_2, \ldots >$ is decreasing as a is constant and k increases by 1 at each iteration.