Chapter 1

Preliminaries

1.1 Induction

Let $\mathbb{N}$ be the set of natural numbers, including 0. Suppose that S is a subset of $\mathbb{N}$ with the following two properties: first $0 \in S$, and second, whenever $n \in S$, then $n + 1 \in S$ as well. Then, invoking the *Induction Principle* (IP) we can conclude that $S = \mathbb{N}$. (See footnote[1].)

We shall use the IP with a more convenient notation; let P be a property of natural numbers, in other words, P is a unary relation such that $P(i)$ is either true or false. The relation P may be identified with a set S_P in the obvious way, i.e., $i \in S_P$ iff $P(i)$ is true. For example, if P is the property of being prime, then $P(2)$ and $P(3)$ are true, but $P(6)$ is false, and $S_P = \{2, 3, 5, 7, 11, \ldots\}$. Using this notation the IP may be stated as:

$$[P(0) \;\&\; \forall n(P(n) \rightarrow P(n+1))] \rightarrow \forall m P(m), \qquad (1.1)$$

for any (unary) relation P over $\mathbb{N}$. In practice, we use (1.1) as follows: first we prove that $P(0)$ holds (this is the *basis case*). Then we show that $\forall n(P(n) \rightarrow P(n+1))$ (this is the *induction step*). Finally, using (1.1) and *modus ponens*, we conclude that $\forall m P(m)$.

As an example, let P be the assertion "the sum of the first i odd numbers equals i^2." We follow the convention that the sum of an empty set of numbers is zero; thus $P(0)$ holds as the set of the first zero odd numbers is an empty

---

[1] In fact, $\mathbb{N}$ and IP are very tightly related, for the rigorous definition of $\mathbb{N}$ is as the *unique* set satisfying the following three properties: (i) it contains 0, (ii) if n is in it, then so is $n + 1$, and (iii) it has the IP.

set. $P(3)$ is also true as $1 + 3 + 5 = 9 = 3^2$. We want to show that in fact $\forall m P(m)$ (i.e., P is always true, and so $S_P = \mathbb{N}$).

We use induction. The basis case is $P(0)$ and we already showed that it holds. Suppose now that the assertion holds for n, i.e., the sum of the first n odd numbers is n^2, i.e., $1 + 3 + 5 + \cdots + (2n - 1) = n^2$ (this is our *inductive hypothesis* or *inductive assumption*). Consider the sum of the first $(n + 1)$ odd numbers,

$$\boxed{1 + 3 + 5 + \cdots + (2n - 1)} + (2n + 1) = \boxed{n^2} + (2n + 1) = (n + 1)^2,$$

and so we just proved the induction step, and by IP we have $\forall m P(m)$.

Exercise 1.1.1 Prove that $1 + \sum_{j=0}^{i} 2^j = 2^{i+1}$.

Sometimes it is convenient to start our induction higher than at 0. We have the following generalized induction principle:

$$[P(k) \ \& \ \forall n(P(n) \rightarrow P(n + 1))] \rightarrow (\forall m \geq k)P(m), \qquad (1.2)$$

for any predicate P and any number k. Note that (1.2) follows easily from (1.1) if we simply let $P'(i)$ be $P(i + k)$, and do the usual induction on $P'(i)$.

Exercise 1.1.2 For every $n \geq 1$, consider a square of size $2^n \times 2^n$ where one square is missing. Show that the resulting square can be filled with "L" shapes.

Exercise 1.1.3 In the generalized IP (1.2) we can replace the induction step $\forall n(P(n) \rightarrow P(n + 1))$ with $(\forall n \geq k)(P(n) \rightarrow P(n + 1))$. Explain why both versions effectively yield the same principle.

Exercise 1.1.4 The *Fibonacci* sequence is defined as follows: $f_0 = 0$ and $f_1 = 1$ and $f_{i+2} = f_{i+1} + f_i$, $i \geq 0$. Prove that for all $n \geq 1$ we have:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}.$$

Exercise∗ 1.1.5 Prove the following: if m divides n, then f_m divides f_n, i.e., $m|n$ implies $f_m|f_n$.

The *Complete Induction Principle* (CIP) is just like IP except that in the induction step we show that if $P(i)$ holds for all $i \leq n$, then $P(n+1)$ also holds, i.e., the induction step is now $\forall n((\forall i \leq n P(k)) \rightarrow P(n+1))$.

Exercise 1.1.6 Use the CIP to prove that every number (in $\mathbb{N}$) greater than 1 may be written as a product of one or more prime numbers[2].

Exercise 1.1.7 Suppose that we have a (Swiss) chocolate bar consisting of a number of squares arranged in a rectangular pattern. Our task is to split the bar into small squares (always breaking along the lines between the squares) with a minimum number of breaks. How many breaks will it take? Make an educated guess, and prove it by induction.

The *Least Number Principle* (LNP) says that every non-empty subset of the natural numbers must have a least element. A direct consequence of the LNP is that every decreasing non-negative sequence of integers must terminate; that is, if $R = \{r_1, r_2, r_3, \ldots\} \subseteq \mathbb{N}$ where $r_i > r_{i+1}$ for all i, then R is a *finite* subset of $\mathbb{N}$. We are going to be using the LNP to show termination of algorithms.

Exercise 1.1.8 Show that IP, CIP, and LNP are equivalent principles.

There are three standard ways to list the nodes of a binary tree. We present them below, together with a recursive procedure that lists the nodes according to each scheme.

Infix: left sub-tree, root, right sub-tree.
Prefix: root, left sub-tree, right sub-tree.
Postfix: left sub-tree, right sub-tree, root.

For example, the tree given by figure 1.1 has the following representations: infix: 2164735, prefix: 1234675, and posftix: 2674531.

Figure 1.1: Binary tree.

Note that some authors use a different name for infix, prefix, and postfix; they call it inorder, preorder, and postorder, respectively[3].
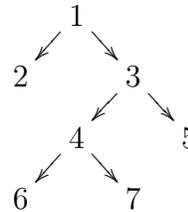
---

[2]Note that this is almost the *Fundamental Theorem of Arithmetic*; what is missing is the fact that up to reordering of primes this representation is unique; to show uniqueness we need to prove first that if $p|(a_1 a_2 \ldots a_n)$, and p is prime, then there exists an a_i such that $p|a_i$.

[3]See [5, §2.3.1, pg. 318] for more background on tree traversals.

Exercise 1.1.9 Show that given any two representations we can obtain from them the third one, or, put another way, from any two representations we can reconstruct the tree. Show, using induction, that your reconstruction is correct. Then show that having just one representation is not enough.

1.2   Invariance

The *Invariance Technique* (IT) is a technique for proving assertions about the outcomes of procedures. The IT identifies some property that remains true throughout the execution of a procedure. Then, once the procedure terminates, we use this property to prove assertions about the output.

As an example, consider an 8×8 board from which two squares from opposing corners have been removed (see Figure 1.2). The area of the board is 62 squares. Now suppose that we have 31 dominos of size 1×2. We want to show that the board cannot be covered by them[4].
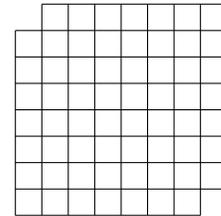
Verifying this by "brute force" is a laborious job. However, using IT we argue as follows: color the squares as a chess board. Each domino, covering two adjacent squares, covers 1 white and 1 black square, and, hence, each placement covers as many white squares as it covers black squares. Note that the number of white squares and the number of black squares differ by 2—opposite corners lying on the same diagonal have the same color—and, hence, no placement of dominos yields a cover.

Figure 1.2: An 8×8 board.

More precisely, we place the dominos one by one on the board, any way we want. The invariant is that after placing each new domino, the number of covered white squares is the same as the number of covered black squares. We prove that this *is* an invariant by induction on the number of placed dominos. The basis case is when zero dominos have been placed (so zero black and zero white squares are covered). In the induction step, we add one more domino which, no matter how we place it, covers one white and one black square, thus maintaining the property. At the end, when we are done placing dominos, we would have to have as many white squares as black

---

[4]This example comes from [3].

squares covered, which is not possible due to the nature of the coloring of the board (i.e., the number of black and whites squares is not the same).

Note that this argument extends easily to the $n \times n$ board.

Exercise 1.2.1 Let n be an odd number. Write the numbers $\{1, 2, \ldots, 2n\}$ on a blackboard. Then pick any two numbers a, b, erase them, and write $|a - b|$ instead. Continue repeating this until just one number remains on the blackboard; show that this remaining number is odd.

Exercise 1.2.2 In the Parliament of some country, each member has at most three enemies (and being an enemy is a *reflexive relation*: if a is an enemy of b, then b is an enemy of a). Show that the house can be separated into two houses, so that each member has at most one enemy in his own house.

Exercise 1.2.3 $2n$ ambassadors are invited to a banquet. Every ambassador has at most $(n - 1)$ enemies. Prove that the ambassadors can be seated at a round table so that nobody sits next to an enemy.

Exercise 1.2.4 Handshakes are exchanged at a big international congress. We call a person an *odd person* if he has exchanged an odd number of handshakes. Show that, at any moment, there is an even number of odd persons.

1.3   Correctness of algorithms

How can we prove that an algorithm is correct? We make two assertions, called the *pre-condition* and the *post-condition*; by correctness we mean that whenever the pre-condition holds *before* the algorithm executes, the post-condition will hold *after* it executes[5]. By *termination* we mean that whenever the pre-condition holds, the algorithm will stop running after finitely many steps. Correctness without termination is called *partial correctness*, and *correctness* per se is partial correctness *with* termination.

Consider the algorithm for integer division given below, with the pre and post-conditions given in the double curly-braces before and after the code.

In this section we assume that number inputs are in $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$; this assumption is implicitly part of the pre-conditions. Also note the indentation in lines 4 and 5 of algorithm 1.3.1; this indentation means that these

---

[5]For the history of the concept of pre and post-condition see [5, pg. 17].

two lines are part of the body of the while-loop, and so we get to line 6 only when the condition of the while-loop, $y \leq r$, is no longer true.

Algorithm 1.3.1 (Division)
$\{\!\{x \geq 0 \ \& \ y > 0\}\!\}$
1. $q \leftarrow 0$
2. $r \leftarrow x$
3. while $(y \leq r)$
4. $r \leftarrow r - y$
5. $q \leftarrow q + 1$
6. return q, r
$\{\!\{x = (q \cdot y) + r \ \& \ 0 \leq r < y\}\!\}$

The q and r returned by the division algorithm are usually denoted as $\mathrm{div}(x, y)$ and $\mathrm{rem}(x, y)$, respectively.

A *loop invariant* is some assertion that we make that stays true after each execution of a "while" (or "for") loop. This assertion is then used for proving partial correctness of the algorithm. In the case of the division algorithm the following is an appropriate loop invariant:

$$x = (q \cdot y) + r \ \& \ r \geq 0. \tag{1.3}$$

Note that in general many different loop invariants (and for that matter pre and post-conditions) may yield a desirable proof of correctness; the art of the analysis of algorithms consists in selecting them judiciously. We usually need induction to prove that a chosen loop invariant holds after each iteration of a loop, and we also usually need the pre-condition as an assumption in this proof.

We turn to (1.3) and show that it holds after each iteration of the loop. Basis case (i.e., zero iterations of the loop): $q = 0, r = x$, so $x = (q \cdot y) + r$ and since $x \geq 0$ and $r = x$, $r \geq 0$.

Induction step: suppose $x = (q \cdot y) + r \ \& \ r \geq 0$ and we go once more through the loop, and let q', r' be the new values of q, r, respectively. Since we went through the loop one more time it follows that $y \leq r$, and since $r' = r - y$, we have that $r' \geq 0$. Since

$$x = (q \cdot y) + r = ((q + 1) \cdot y) + (r - y) = (q' \cdot y) + r',$$

we have that the q', r', still satisfy the loop invariant (1.3).

Now we use the loop invariant to show that the post-condition of the division algorithm holds, *if* the pre-condition holds. This is very easy in this case since the loop ends when it is no longer true that $y \leq r$, i.e., when it is true that $r < y$, while (1.3) holds after each iteration, and in particular the last iteration. Putting together (1.3) and $r < y$ we get our post-condition, and hence partial correctness.

To show termination we use the least number principle (LNP). We need to relate some non-negative monotone decreasing sequence to the algorithm; just consider $r_0, r_1, r_2, \ldots$, where $r_0 = x$, and r_i is the value of r after the i-th iteration. Note that $r_{i+1} = r_i - y$. First, $r_i \geq 0$, because the algorithm enters the while loop only if $y \leq r$, and second, $r_{i+1} < r_i$, since $y > 0$. By LNP such a sequence "cannot go on for ever," (in the sense that the set $\{r_i | i = 0, 1, 2, \ldots\}$ is a subset of the natural numbers, and so it has a least element), and so the algorithm must terminate.

Thus we have full correctness of the division algorithm.

One of the oldest known algorithm is Euclid's algorithm[6], a process for finding the greatest common divisor of two numbers. It appeares in Euclid's *Elements* (Book 7, Propositions 1 and 2) around 300 BC. Given two positive integers a and b, the *gratest common divisor*, denoted as $\gcd(a, b)$ is the largest positive integer that divides them both.

Algorithm 1.3.2 (Euclid)
$\{\!\{a > 0 \ \& \ b > 0\}\!\}$
1. $m \leftarrow a$
2. $n \leftarrow b$
3. $r \leftarrow \mathrm{rem}(m, n)$
4. while $(r > 0)$
5. $m \leftarrow n$
6. $n \leftarrow r$
7. $r \leftarrow \mathrm{rem}(m, n)$
8. return n
$\{\!\{n = \gcd(a, b)\}\!\}$

Note that to compute $\mathrm{rem}(n, m)$ in lines 1 and 5 of Euclid's algorithm we need to use algorithm 1.3.1 (the division algorithm) as a subroutine; this is a typical "composition" of algorithms.

---

[6]For an extensive study of Euclid's algorithm see [5, §1.1].

To prove the correctness of Euclid's algorithm we are going to show that after each iteration of the while loop the following assertion holds:

$$\gcd(a, b) = \gcd(m, n), \tag{1.4}$$

that is, (1.4) is our loop invariant. We prove this by induction on the number of iterations. Basis Case: after zero iterations (i.e., just before the while loop starts—so after executing lines 1, 2, and 3, and before executing line 4) we have that $m = a$ and $n = b$, so (1.4) holds trivially.

For the Induction Step, suppose that $\gcd(a, b) = \gcd(m, n)$, and we go through the loop one more time, yielding m', n'. We want to show that $\gcd(m, n) = \gcd(m', n')$. Note that from lines 5, 6, and 7 of the algorithm we see that $m' = n, n' = r = \mathrm{rem}(m, n)$. In other words, it is enough to prove that in general $\gcd(m, n) = \gcd(n, \mathrm{rem}(m, n))$.

Exercise 1.3.3 Show that for all $m, n > 0$, $\gcd(m, n) = \gcd(n, \mathrm{rem}(m, n))$.

Now the correctness of Euclid's algorithm follows from (1.4), since the algorithm stops when $r = \mathrm{rem}(m, n) = 0$, so $m = q \cdot n$, and so $\gcd(m, n) = n$.

Exercise∗ 1.3.4 Show that Euclid runs in time $O(\log(\min\{a, b\}))$. Try to give explicit constants instead of the big-Oh notation. Do you have any ideas to speed-up this algorithm?

Exercise∗ 1.3.5 Given integers n, m, and $d = \gcd(n, m)$, it is possible to compute integers a, b such that $an + bm = d$. The algorithm for computing a, b (besides also computing the $\gcd(n, m)$) is called the *Extended* Euclid's algorithm. Design this algorithm and prove its correctness.

The following algorithm tests strings for *palindromes*, which are strings that read the same backwards as forwards, for example, `madamimadam`[7] or `racecar`.

Algorithm 1.3.6 (Palindromes)
$\{\!\{n \geq 1 \ \& \ A[1 \dots n] \text{ is a character array}\}\!\}$
1. $i \leftarrow 1$
2. while$(i \leq \lfloor \frac{n}{2} \rfloor)$
3.   if $(A[i] \neq A[n - i + 1])$

---

[7] This example comes from Joyce's *Ulysses*.

4.   return FALSE
5. $i \leftarrow i + 1$
6. return TRUE
$\{\!\{$ return True iff A is a palindrome $\}\!\}$

Let the loop invariant be: after the k-th iteration, $i = k + 1$ & for all j such that $1 \leq j \leq k$, $A[j] = A[n - j + 1]$. We prove that the loop invariant holds by induction on k. Basis Case: before any iterations take place (i.e., after zero iterations), there are no j's such that $1 \leq j \leq 0$, so the second statement is (vacuously) true. The first statement holds since i is initially set to 1.

Induction Step: we know that after k iterations, $A[j] = A[n-j+1]$ for all $1 \leq j \leq k$; after one more iteration we know that $A[k+1] = A[n-(k+1)+1]$, so the statement follows for all $1 \leq j \leq k + 1$.

Exercise 1.3.7 Show that the algorithm for palindromes always terminates. (Hint: consider $d_i = \lfloor \frac{n}{2} \rfloor - i$.)

Exercise 1.3.8 What does the following algorithm compute? Prove your claim.

Algorithm 1.3.9
1. $x \leftarrow m$; $y \leftarrow n$; $z \leftarrow 0$
2. while $(x \neq 0)$
3.   if $(\text{rem}(x, 2) = 1)$
4. $z \leftarrow z + y$
5. $x \leftarrow \text{div}(x, 2)$
6. $y \leftarrow y \cdot 2$
7. return z

Exercise 1.3.10 We have positive integers a, b, and the following algorithm.

Algorithm 1.3.11
1. while $(a > 0)$
2.   if $(a < b)$
3. $(a, b) \leftarrow (2a, b - a)$
4.   else
5. $(a, b) \leftarrow (a - b, 2b)$

For which starting a, b does this algorithm terminate? In how many steps does it terminate, if it does terminate?

1.4   Stable marriage

The *Stable Marriage Problem*[8] was introduced by Gale and Shapley in 1962 and is related to the problem of college admissions. They gave an algorithm for solving the finite problem, which was later discovered to have been used in the matching of graduate medical students with hospitals since 1952. Other variants of the problem have been studied in computer science, economics, game theory and operations research.

An instance of the stable marriage problem of size n consists of two disjoint finite sets of equal size $B = \{b_1, b_2, \ldots, b_n\}$ (the set of *boys*) and $G = \{g_1, g_2, \ldots, g_n\}$ (the set of *girls*). In addition, each boy b_i has a ranking or a linear ordering $<_i$ of G which reflects his preference for the girls that he wants to marry. That is, if $g_j <_i g_k$, then b_i would prefer to marry g_j over g_k. Similarly each girl g_j has a ranking or linear ordering $<^j$ of B which reflects her preference for the boys she would like to marry.

A *matching* (or *marriage*) M is a 1-1 correspondence between B and G. We say that b and g are *partners* in M if they are matched in M and write $p_M(b) = g$ and also $p_M(g) = b$. A matching M is *unstable* if there is a pair (b, g) from $B \times G$ such that b and g are not partners in M but b prefers g to $p_M(b)$ and g prefers b to $p_M(g)$. Such a pair (b, g) is said to *block* the matching M and is called a *blocking pair* for M

Figure 1.3: Blocking pair.
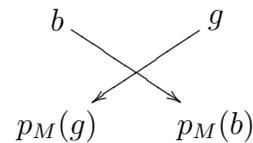
(see figure 1.3). A matching M is stable if there is no blocking pair for M.

The result of Gale and Shapley is that any marriage problem has a solution, i.e., there always exists a stable matching, no matter what the lists of preferences. In fact, they give an algorithm which produces a solution in n stages and takes $\leq O(n^3)$ steps.

---

[8]This section is based on §2 in in the article *Proof-Theoretic Strength of the Stable Marriage Theorem and Other Problems*, by Douglas Cenzer and Jeffrey B. Remmel.