1 Partial orders

A *partial order* of a set B is a binary relation of B that is

1. reflexive,
2. antisymmetric, and,
3. transitive.

Normally as a symbol for a partial order we use "$\leq$" and '$a \leq b$' is read as "a is less than or equal to b". If $\leq$ is a partial order on B, the pair $(B, \leq)$ is called a *partially ordered set.*

Suppose that $(B, \leq)$ is a partially ordered set and $a, b \in B$. If $a \leq b$ or $b \leq a$, we say that elements a and b are *comparable,* and otherwise they are said to be *incomparable,* which is denoted $a \parallel b$. If any two elements of B are comparable, $\leq$ is a *total order,* and the pair $(B, \leq)$ is called a *chain.*

Note that for any $a \in B$, $a \leq a$ because a partial order is reflexive. If $a \leq b$ and $a \neq b$, we say that a is *stictly smaller than* b, and denote $a < b$.

Example 1.1

- *The sets $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ each form a chain with respect to the usual $\leq$-relation on numbers.*
- *For any set B, the pair $(2^B, \subseteq)$ is a partially ordered set. Recall that 2^B is the power set of B. The pair $(2^B, \subseteq)$ is not a chain when $|B| > 1$. Why?*
- *Consider a set of natural numbers $A \subseteq \mathbb{N}$ and, for $a, b \in A$, define the divisibility relation $\mid$ by setting*

  $$a \mid b \text{ iff } (\exists k \in \mathbb{N})\ a \cdot k = b.$$

  The divisibility relation $\mid$ is a partial order of A.

The following notions are used for a partially ordered set $(A, \leq)$, $H \subseteq A$ and $c \in A$:

- c is a *maximal element* of H if

  $$c \in H \text{ and } (\forall a \in H)\ c \leq a \text{ implies } c = a.$$

- c is a *minimal element* of H if

$$c \in H \text{ and } (\forall a \in H)\ a \leq c \text{ implies } c = a.$$

- c is the *greatest element* of H if $c \in H$ and $(\forall a \in H)\ a \leq c$.
- c is the *least element* of H if $c \in H$ and $(\forall a \in H)\ c \leq a$.
- c is an *upper bound* for H if $(\forall a \in H)\ a \leq c$.
- c is a *lower bound* for H if $(\forall a \in H)\ c \leq a$.
- c is the least upper bound for H, or *supremum* of H, if it is a minimal element in the set of upper bounds for H.
- c is the greatest lower bound for H, or *infimum* of H, if it is a maximal element in the set of lower bounds for H.

When considering the usual order of integers $\leq$, 0 is the unique minimal element of $\mathbb{N}$ (that is, the least element of $\mathbb{N}$) and $\mathbb{N}$ does not have a maximal element. The set $\mathbb{Z}$ has neither a minimal nor a maximal element. A finite partially ordered set always has both minimal and maximal elements (see Prop. 55.3 in the text).

Maximal and minimal elements need not be unique. By definition, a supremum of H (or infimum of H) is unique, if it exists

Definition 1.1 *A partially ordered set* $(A, \leq)$ *is* well-ordered *if every non-empty subset of* A *has a least element.*

A well-ordered set is always a chain but $(\mathbb{Z}, \leq)$ is a chain that is not well-ordered.

The *well-ordering principle* (see Statement 21.6 in the text) states that the set of natural numbers is well-ordered and it is the basis for proofs by induction.

2 Induction

The induction principle is perhaps the most important general proof method of discrete mathematics. Formally the proofs by induction are justified by the well-ordering principle of natural number (Definition 1.1). The standard induction principle can be modified to various situations and to inductive constructions.

The standard induction principle works as follows. Consider a claim $C(n)$, involving a natural number n as a parameter. When we want to prove that $C(n)$ holds for all values $n = 0, 1, 2, \ldots$, it is sufficient to establish the following:

1. Show that $C(0)$ holds.
2. Assume that $C(k)$ holds for a value $k \geq 0$, and using this *induction assumption* show that also $C(k + 1)$ must hold.

Assuming we are successful to show 1. and 2. above, the induction principle guarantees that $C(n)$ holds for all values $n = 0, 1, 2, \ldots$.

The induction principle can also be used to show that some claim $C(n)$ holds for all values $n \geq n_0$ where n_0 is any fixed natural number. To do this we verify that

1. $C(n_0)$ holds, and,
2. the implication $C(n) \Rightarrow C(n + 1)$ holds for all values $n \geq n_0$.

Note that the above can be viewed as a standard proof by induction for the claim $C'(n)$ that is defined as $C'(n) = C(n + n_0)$.

The so called *strong version of mathematical induction* replaces in the inductive step $C(n) \Rightarrow C(n+1)$ the premise $C(n)$ by the conjunction of all the conditions $C(0), C(1), \ldots, C(n)$, see Theorem 22.9 in the text.

Thus, to prove that $C(n)$ holds for all $n \in \mathbb{N}$ we verify that

1. $C(0)$ holds, and,
2. assuming that for some value $n \geq 0$, the claim $C(k)$ holds for all $k \leq n$, then also

$C(n+1)$ holds.

The principle of strong induction is illustrated in the following example by proving the *Fundamental Theorem of Arithmetic.*

Example 2.1 *We claim that every natural number $n \geq 2$ can be represented as a product of one or more primes.*

1. *The claim holds for $n = 2$ because 2 is a prime.*

2. *We assume that the claim holds for all values k, where $2 \leq k \leq n$ ($n \geq 2$).*

   We consider the number $n+1$. If $n+1$ is a prime, we are done. Otherwise there exist natural numbers b nad c such that $n+1 = b \cdot c$ and $2 \leq b, c \leq n$. By the (strong) inductive assumption the numbers b and c can be written as

$$b = p_1 \cdot \ldots \cdot p_r, \quad c = q_1 \cdot \ldots \cdot q_s,$$

   where $r, s \geq 1$ and the p_i's and q_j's are primes. Now we have the required representation for $n+1$:

$$n + 1 = p_1 \cdot \ldots \cdot p_r \cdot q_1 \cdot \ldots \cdot q_s.$$

2.1 Inductive definitions and structural induction

Inductive definitions are closely related to the general induction principle. An *inductive definition* of a set A defines A as a smallest set (included in a given universal set) that

(ID1) contains certain given elements, and,

(ID2) is closed under certain operations (or constructions).

When a set A is defined as above, claims on the elements of A can be naturally defined using *structural induction,* that is, a proof that follows the inductive definition of the set A. A proof by structural induction shows that

1. all elements listed in part (ID1) of the definition of A have the required property, and,

2. if an element $z \in A$ is obtained using some operation of (ID2) from elements $x_1, \ldots, x_m \in A$ that have the required property, then also z has the required property.

The principle of structural induction can be justified by the original induction principle (on natural numbers) by viewing it as an inductive proof on the number of steps needed to define $z \in A$ according to the inductive definition of A.

Often sets of terms or formulas used in logic have a natural inductive definition. The notions of inductive definition and proof by structural induction are illustrated in the following example.

Example 2.2 The *well formed formulas* of propositional logic can be defined as follows.

The alphabet X of propositional logic consists of

- a countably infinite[1] set of *propositional variables* p_1, p_2, ...
- the *connectives* $\neg$ (negation), $\vee$ (disjunction) and $\wedge$ (conjunction), and,
- left and right parentheses "(", ")".

The set of well formed formulas of propositional logic, or *propositions,* denoted PROP, is defined as the smallest set of words S over alphabet X that satisfies the following conditions:

1. $p_1, p_2, \ldots \in \mathcal{S}$,
2. if $P, Q \in \mathcal{S}$, then also $(\neg P)$, $(P \vee Q)$ and $(P \wedge Q)$ are in $\mathcal{S}$.

As a simple illustration of the principle of structural induction we show that every proposition in PROP contains equally many left and right parentheses. For $P \in$ PROP we denote the number of left (respectively, right) parentheses occurring in P by left(P) (respectively, right(P)).

We prove inductively that for any proposition P, left$(P) =$ right(P).

1. If P is a propositional variable p_i, then left$(P) =$ right$(P) = 0$.

[1]The term "countably infinite" means that the variables can be indexed by the natural numbers. An infinite set, like the set of real numbers, may be *uncountable* but this is a topic not discussed in CISC-203.

2. Now inductively assume that the claim holds for propositions Q and R. According to the inductive definition of PROP we have three subcases:
   (a) If $P = (\neg Q)$, then $\text{left}(P) = \text{left}(Q) + 1 =^{(*)} \text{right}(Q) + 1 = \text{right}(P)$.
   (b) if $P = (Q \vee R)$ then $\text{left}(P) = \text{left}(Q) + \text{left}(R) + 1 =^{(*)} \text{right}(Q) + \text{right}(R) + 1 = \text{right}(P)$.
   (c) if $P = (Q \wedge R)$ then $\text{left}(P) = \text{left}(Q) + \text{left}(R) + 1 =^{(*)} \text{right}(Q) + \text{right}(R) + 1 = \text{right}(P)$.

   Above (*) indicates equalities that rely on the inductive assumption that the claim holds for propositions Q and R.

2.2 Inductive proofs of algorithm correctness

Lastly we consider the use of induction for proving the correctness of algorithms. By correctness we mean that on all inputs, the algorithm terminates and produces the correct output.

A technique for such proofs is the method of *inductive claims* with an idea as follows.[2] We associate to different parts of the algorithm claims on the parameters of the algorithm that are selected in a way that assuming the claims hold during execution of the algorithm at corresponding positions, then at the end the algorithm outputs the correct value. The correctness of the claims is shown *inductively* by establishing

1. at the start of the execution the claims hold based on assumptions on the inputs, and,
2. showing, for each claim C, that assuming the claims for previous stages of the execution are valid, also the current claim C holds.

The above proof method is based on the induction principle: it is induction on the number of steps used by the algorithm.

[2]Here we present only a rough idea and an example to illustrate the connection of the induction principle to program verification. You will learn a more systematic approach for verifying algorithms in the winter term course on Software Specifications CISC-223.

As an example we consider the well known Euclid's algorithm that is given as input natural numbers a abd b and that outputs their greatest common divisor, $\gcd(a, b)$. Thus the inputs and the output are specified as:

Input: $a, b \in \mathbb{N}$ where $a \neq 0$ or $b \neq 0$.

Output: $\gcd(a, b)$

and Euclid's algorithm in a C-like pseudo-code can be written as

```
int n = a; int m = b; int r;
/*(1)*/ while( m > 0 ) {
           r = n % m;
/*(2)*/    n = m; m = r;
          } //end while
/*(3)*/  return n;
```

For the inductive proof of correctness we associate to positions (1), (2) and (3), respectively, the following claims:

(C1) $\gcd(n, m) = \gcd(a, b)$ and $(n, m) \neq (0, 0)$;

(C2) $\gcd(m, r) = \gcd(a, b)$ and $(m, r) \neq (0, 0)$;

(C3) $n = \gcd(a, b)$.

Inductively we prove that the claim (Ci) always holds at position (i), $i = 1, 2, 3$.

1. As the base case we observe that when the while-loop is entered the first time (position (1)), n has value a and m has value b and, consequently, $\gcd(n, m) = \gcd(a, b)$. Additionally, because of the assumptions on the inputs n and m cannot both be 0.

2. Next we verify that each of the claims (Ci) holds in the execution of the code at the corresponding position (i), $i = 1, 2, 3$, assuming that the claims at the previous positions during the execution were valid.

   (a) Consider when the execution enters position (1) for the 2nd (or later) time. Previously the code executed the assignments `n = m` and `m = r`. Because claim (C2) was valid before the assignments we have

$$\gcd(n, m) = \gcd(m', r) = \gcd(n', m') = \gcd(a, b),$$

where n' and m' are the previous values of the variables n and m. Additionally, $(n, m) \neq (0, 0)$ because n has value m' which is non-zero. This means that (C1) holds when entering (1) for the second or later times.

(b) When the execution reaches (2), $m \neq 0$ because the while-test evaluated to true. Using the fact that (C1) holds at (1) we have

$$\gcd(m, r) = \gcd(n, m) = \gcd(a, b).$$

This shows that (C2) holds.

(c) When the execution reaches (3), the while-test has evaluated to false and $m = 0$. By claim (C1) we know that $n \neq 0$ and consequently $n = \gcd(n, 0) = \gcd(a, b)$ and claim (C3) holds.

The output of the program is correct because (C3) holds when exiting the while-loop. The program terminates because each execution of the while-loop reduces the value of m and m remains non-negative. This means that the while-loop is executed at most b times.

3 Recurrence

Let $k \in \mathbb{N}$ be fixed. If we want to define for each $n \geq k$ some entity $P(n)$ ($P(n)$ could be a number, a set etc.), based on the induction principle this can be done as follows:

1. Define $P(k)$.
2. Give a rule that tells us how we get $P(n+1)$ when $P(k)$, $P(k+1)$, $\ldots$, $P(n)$ are known.

A definition of the above type is called a *recurrence relation* (or *recurrence,* for short). In the general form given in 2., $P(n+1)$ can depend on all the previous elements $P(k)$, $P(k+1)$,

$\ldots$, $P(n)$. In typical use of recurrence relations the term $P(n+1)$ depends only on a constant number of previous terms.

When $P(n)$ is defined using a recurrence relation, claims concerning $P(n)$ can be typically proven using induction that follows the recurrence relation.

Definition 3.1 (Recurrence relation) *A recurrence relation is a sequence $P(n)$ where each term of the sequence is either given as an initial term or produced from one or more previous terms using the rule 2.*

Thinking in the other direction, we say that a sequence is a **solution** of a recurrence relation if the terms of the sequence satisfy the recurrence relation. A recurrence relation together with a set of initial terms uniquely defines a sequence, so there exists only one sequence that satisfies a given recurrence relation.

Example 3.1 (Fibonacci sequence) *A well known example of a sequence of numbers defined by a recurrence is the* Fibonacci sequence *(Leonardo Bigollo Pisana, ca. 1170–1240) where the initial numbers*[3] *are*

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

The recurrence defining the Fibonacci sequence is

1. $F_0 = 0$ *and* $F_1 = 1$*;*
2. $F_{n+1} = F_n + F_{n=1}$, $(n \geq 1)$.

The definition of the Fibonacci numbers gives two base values (F_0 and F_1) and, in the recurrence relation, F_{n+1} depends on the two previous values. Because of this, in inductive proofs of properties of Fibonacci numbers in the base case we need to verify that the property holds for the first two values of n.

Fibonacci numbers satisfy many interesting identities. The below lemma gives two examples. The identities can be proved using induction, and we may go through one of the proofs in class.

[3]The textbook omits the initial zero from the sequence but this is not important.

Lemma 3.1 *1.* $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$ *(*$n \geq 1$*);*

2. $F_{m+n} = F_m F_{n+1} + F_{m-1}F_n$ *(*$n \geq 0,\ m \geq 1$*).*

A second example is the binomial coefficients that we have already encountered previously.

Example 3.2 *Binomial coefficients* $B_{n,k} = \binom{n}{k}$ *can be defined by the following recurrence:*

1. $\binom{n}{0} = 1$ *for all* $n \in \mathbb{N}$*;*
2. $\binom{n}{n} = 1$ *for all* $n \in \mathbb{N}$*; and*
3. $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ *for all* $1 \leq k \leq n$*.*

By Pascal's identity, the sequence $B_{n,k}$ *is generated by the recurrence* $B_{n,k} = B_{n-1,k} + B_{n-1,k-1}$ *with initial terms* $B_{n,0} = 1$ *and* $B_{n,n} = 1$*.*

In both of the above examples, we obtain new terms of the sequence recursively by adding together two previous terms of the sequence. We are also given two initial terms for each sequence, which we use to obtain the first recursive term of that sequence. However, it is possible to define recurrence relations that use more than two previous terms. In general, if a recurrence relation produces new terms from k previous terms, we say that the **degree** of the recurrence relation is k. We may also refer to such a recurrence relation as a ***k*th-order** recurrence relation.

Example 3.3 *We can define common functions using recurrence relations. Consider the exponential function* 2^n*. Each value of* 2^n *is given by the first-order recurrence relation* $a_n = 2 \cdot a_{n-1}$*, and the initial term of the recurrence relation is* $a_1 = 1$ *(where* a_1 *corresponds to* $2^0 = 1$*).*

Example 3.4 *Each value of the factorial function* $n!$ *is given by the first-order recurrence relation* $b_n = n \cdot b_{n-1}$*, and the initial term of the recurrence relation is* $b_1 = 1$*.*

Observe that, in Examples 3.3 and 3.4, the recurrence a_n had a coefficient of 2 for all terms, while the recurrence b_n had a coefficient of n for the nth term. We say that a_n is a recurrence relation with **constant coefficients**; that is, coefficients that do not depend on n. The recurrence relation b_n, however, contains the non-constant coefficient n.

Example 3.5 *Consider the recurrence relation $d_n = d_{n-1} + 1$ with initial term $d_1 = 1$. This first-order, constant-coefficient recurrence relation simply produces the sequence of all positive integers.*

Example 3.6 *By modifying the Fibonacci recurrence relation to multiply previous terms together, we get a slightly more interesting recurrence relation. Call this recurrence relation $e_n = e_{n-1} \cdot e_{n-2}$, and define the initial terms to be $e_1 = 1$ and $e_2 = 2$. This second-order recurrence relation produces the sequence* $1, 2, 2, 4, 8, 32, 256, 8192, \ldots$.

In the recurrence relation e_n, we get new terms by multiplying previous terms instead of adding. If a recurrence relation defines the nth value by a linear function of some previous value, then we say that the recurrence is **linear** (and, thus, the recurrence relation e_n is non-linear, since the nth term is the product of e_{n-1} and e_{n-2}).

3.1 Solving Recurrences

Now that we are familiar with defining recurrence relations, we turn to the converse question: how do we solve recurrence relations? Here, we use the word "solve" in the sense of finding a closed-form equation that is equivalent to the recursively-defined recurrence relation. There exist a number of approaches we can take when solving a recurrence relation, and these approaches range from the simple to more complex ones. Our choice of approach ultimately depends on the recurrence relation we are trying to solve.

To illustrate the simplest method for solving let us consider the so called *Towers of Hanoi problem.*

Example 3.7 (Towers of Hanoi) *We are given n discs of different size placed on three pegs. A disc can slide into any peg. Initially all the discs are placed in the first peg in order of size (thus making a conical shape).*

The goal is to move the n discs to the third peg following the rules:

1. *only one disc can be moved at a time;*
2. *at any moment a larger disc cannot be placed on top of a smaller disc.*

One can verify that for small values of n the smallest number of moves required will be, respectively,

$$1, 3, 7, 15, 31, 63, 127, \ldots$$

Notice that each term of the sequence $H_n = (1, 3, 7, 15, 31, 63, 127, \ldots)$ seems to be twice the previous term plus one. This suggests that the sequence H_n can be generated by a recurrence $H_n = 2H_{n-1} + 1$. Since we have three pegs, we can model the problem recursively by moving all but the largest disk from the first peg to the second peg, then moving the largest disk to the third peg before placing all of the other disks on the third peg as well.

Lemma 3.2 *Given n disks, $H_n = 2H_{n-1} + 1$ moves are sufficient to transfer all disks from peg 1 to peg 3, where $H_1 = 1$.*

Proof We begin with all n disks on peg 1. Move the top $n - 1$ disks from peg 1 to peg 2. This step requires H_{n-1} moves. Then, move the nth disk (that is, the largest disk) to peg 3. Finish by moving the top $n - 1$ disks from peg 2 to peg 3. Altogether, this process uses a total of $2H_{n-1} + 1$ moves. □

You might question whether this bound is also the minimum and, indeed, this bound is the best possible. Roughly, this can be verified as follows: given n disks, the largest disk must be moved at some point. To move the largest disk, the $n - 1$ smaller disks must be moved out of the way first. Then, to complete the puzzle, the $n - 1$ smaller disks must be moved back on top of the largest disk. Altogether, this process requires at least H_n moves.

3.1.1 Substitution Method

The simplest method of solving recurrence relations, the **substitution method** (also called the “guess and check” method) exploits the fact that recurrence relations are recursively defined and, roughly speaking , we could find a solution to the recurrence relation by induction: take the initial terms as the base case, and take the recursive term to be the inductive case.

With the substitution method, we guess a solution to the recurrence relation and verify it's correctness by induction. Though this method is simple, it is not the most straightforward and intuitive method; there is no general heuristic to help in making a right guess. Making a wrong guess is possible, and a wrong guess means we have to start from scratch. However, with practice, the substitution method can become a quick way to check a potential solution without having to put in as much work as other methods require.

Example 3.8 *By Lemma 3.2, we know that $H_n = 2H_{n-1}+1$ is the recurrence for the Towers of Hanoi problem, and $H_1 = 1$. Now the question is: what is a closed form for H_n?*

Let's make a guess… looking at the sequence H_n, it appears that the nth term is equal to $2^n - 1$. Is our guess correct? Check using a proof by induction. Let $S(n)$ be the statement "$H_n = 2^n - 1$".

Base case: When $n = 1$, we have $2^1 - 1 = 1$. Since $H_1 = 1$, $S(1)$ is true.

Inductive hypothesis: Assume that $S(k)$ is true for some $k \in \mathbb{N}$ where $k \geq 1$. That is, assume that $H_k = 2^k - 1$.

Inductive case: We now show that $S(k+1)$ is true. By the inductive hypothesis, we have $H_k = 2^k - 1$. By our recurrence relation, we have

$$\begin{aligned} H_{k+1} &= 2H_k + 1 \\ &= 2(2^k - 1) + 1 \\ &= 2^{k+1} - 2 + 1 \\ &= 2^{k+1} - 1. \end{aligned}$$

Therefore, $S(k+1)$ is true. By the principle of mathematical induction, $S(n)$ is true for all $n \in \mathbb{N}$.

Note that, while the substitution method is good for proving easy solutions to recurrence relations, it can be a struggle to prove even a moderately-difficult solution using substitution. In such cases, you might try to show that the recurrence relation is bounded from above by a looser but nicer-looking expression. However, this approach comes with its own set of potential traps.

3.1.2 Iteration Method

As opposed to the substitution method, which uses a proof by induction to find the closed form solution for a recurrence relation, the **iteration method** uses the recurrence relation itself to find its corresponding closed form. This method works by iteratively replacing occurrences of smaller terms in the recurrence relation with the corresponding equation for that term, then simplifying the expression. Once the initial terms are reached, the entire expression simplifies to the closed-form equation.

Example 3.9 *Consider again the recurrence for the Towers of Hanoi problem:* $H_n = 2H_{n-1} + 1$ *and* $H_1 = 1$*. What is a closed form for* H_n*?*

Using the iteration method, we proceed as follows:

$$\begin{aligned}
H_n &= 2H_{n-1} + 1 \\
&= 2(2H_{n-2} + 1) + 1 = 2^2 H_{n-2} + 2 + 1 \\
&= 2^2(2H_{n-3} + 1) + 2 + 1 = 2^3 H_{n-3} + 2^2 + 2 + 1 \\
&\vdots \\
&= 2^{n-2}(2H_1 + 1) + 2^{n-3} + \cdots + 2 + 1 \\
&= 2^{n-1} H_1 + 2^{n-2} + \cdots + 2 + 1 \\
&= 2^{n-1} + 2^{n-2} + \cdots + 2 + 1 \\
&= 2^n - 1.
\end{aligned}$$

The second-last line of the previous derivation is the sum of a geometric series: $\sum_{i=0}^{n-1} 2^i = \frac{2^{(n-1)+1}-1}{2-1} = 2^n - 1.$

3.1.3 First-Order Recurrence Relations

First-order recurrence relations are some of the easiest recurrence relations to deal with. Remember that we say a recurrence relation is "first-order" or "degree 1" if it produces new terms from only the previous term. In mathematical terms, a first-order recurrence relation is of the form

$$a_n = ca_{n-1} + x,$$

where the coefficient c and the additive term x are constants.

We can use the iteration method to obtain a general closed form for first-order recurrence relations. Observe that

$$\begin{aligned}
a_n &= ca_{n-1} + x \\
&= c(ca_{n-2} + x) + x \\
&= c^2(ca_{n-3} + x) + cx + x \\
&\vdots \\
&= c^{n-1}(ca_0 + x) + c^{n-2}x + c^{n-3}x + \cdots + cx + x \\
&= c^n a_0 + c^{n-1}x + c^{n-2}x + c^{n-3}x + \cdots + cx + x \\
&= c^n a_0 + (c^{n-1} + c^{n-2} + \cdots + c + 1)x.
\end{aligned}$$

Just like we saw in Example 3.9, the last line of the previous derivation includes a sum of a geometric series: $\sum_{i=0}^{n-1} c^i = \frac{c^n-1}{c-1}$. Altogether, we have

$$a_n = c^n a_0 + \left(\frac{c^n - 1}{c - 1}\right) x,$$

and, by collecting like terms and rearranging, we get the general closed form.

Theorem 3.1 *Let $a_n = ca_{n-1} + x$ be a recurrence relation where $c \neq 1$. Then the sequence $A_n = (a_1, a_2, \ldots, a_n, \ldots)$ is a solution of the recurrence relation if and only if*

$$a_n = \left(a_0 + \frac{x}{c-1}\right) c^n - \frac{x}{c-1}$$

for all $n \in \mathbb{N}$.

Proof. By the iteration method. □

As Theorem 3.1 states, we cannot use this closed form if $c = 1$ because this would result in division by zero. Fortunately, another application of the iteration method to the similar recurrence relation $a_n = a_{n-1} + x$ gives us a result that works when $c = 1$.

Corollary 3.1 *Let $a_n = a_{n-1} + x$ be a recurrence relation. Then the sequence $A_n = (a_1, a_2, \ldots, a_n, \ldots)$ is a solution of the recurrence relation if and only if*

$$a_n = a_0 + nx$$

for all $n \in \mathbb{N}$.

3.1.4 Characteristic Root Method

So far, the methods we have seen for solving recurrence relations have been very general and very brute-force. With the substitution method, we resort to guessing and throwing induction at the problem. With the iteration method, we replace terms and simplify until something falls out of the expression that looks good. Although these methods work for many recurrence relations we need to solve, it would be nice to have a more refined method for solving recurrence relations.

Next we look at a method for solving a specific type of recurrence relation. In the general form, the **characteristic root method** is designed to solve linear homogeneous recurrence relations of degree k with constant coefficients. (Remember all of the terminology from earlier?) In mathematical terms, the characteristic root method solves recurrence relations of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

where each of the coefficients $c_1, c_2, \ldots, c_k$ are real numbers and $c_k \neq 0$.

We've already seen a few examples of recurrence relations of this form; consider the Fibonacci seuqence F_n or a_n from Example 3.3. Of course, not all recurrence relations are of this form; for instance, the recurrence relation for the pegs-and-disks problem, H_n, is not homogeneous because of its additive constant term. (This means we can't use H_n as our go-to example for the characteristic root method - but we already know how to solve the recurrence H_n.)

Where does the name "characteristic root method" come from? Characteristic roots are the values we use to solve the recurrence relation. From centuries of studying recurrence relations, mathematicians know that linear recurrence relations have exponential solutions;

that is, solutions of the form $a_n = r^n$ for some constant r. We won't launch into any discussion about how mathematicians know this fact, since it's outside the scope of these notes. However, we observe that r^n is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ if and only if

$$r^n = c_1 r^{n-1} + c_2 r^{n-2} + \cdots + c_k r^{n-k},$$

where we simply substitute all occurrences of a_i in the recurrence relation with r^i for $(n-k) \leq i \leq n$. If we divide both sides of this expression by r^{n-k} to get rid of the highest-order term on the right-hand side, then move all of the terms to one side, we end up with the equation

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_{k-1} r - c_k = 0. \tag{1}$$

We call such an expression the **characteristic equation** of the recurrence relation a_n, and we call the solutions of this equation the **characteristic roots** of a_n.

The roots of the equation (1) give a method for solving general k-the order recurrence relations. In the following we present the details only for 2nd order recurrence relations ($k = 2$) which is the case most commonly used. (Note that finding roots of 3rd or 4th order polynomials can be more complicated and the roots of 5th (or higher) order polynomials may not be possible to express in closed form.)

3.1.5 Second-Order Recurrence Relations with Two Characteristic Roots

The recurrence relations we deal with most frequently are second-order recurrence relations. The characteristic root method for recurrence relations of degree 2, in the case when the characteristic equation has two distinct roots, is based on the following result.

Proposition 3.1 *Let c_1 and c_2 be real numbers where $c_2 \neq 0$. Suppose that the recurrence relation*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} \tag{2}$$

has a corresponding characteristic equation

$$r^2 - c_1 r - c_2 = 0$$

with two distinct roots r_1 and r_2. Then every solution of the recurrence (2) is of the form

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n,$$

$n \in \mathbb{N}$, where α_1 and α_2 are constants.

Proof. Omitted $\square$

The constants α_1 and α_2 are typically determined by looking at values of a_n with small n.

Let's consider an example of the method in action with our favourite second-order recurrence relation defining the Fibonacci sequence F_n.

Example 3.10 *Recall that $F_n = F_{n-1} + F_{n-2}$, where $F_1 = F_2 = 1$. What is a closed form for F_n?*

From the statement of the recurrence relation for F_n, we see that $c_1 = 1$ and $c_2 = 1$, so the characteristic equation for F_n is

$$r^2 - r - 1 = 0.$$

The two distinct roots of this equation are $r_1 = (1+\sqrt{5})/2$ and $r_2 = (1-\sqrt{5})/2$. Therefore, the recurrence relation for F_n has a solution of the form

$$a_n = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

for all $n \in \mathbb{N}$, where α_1 and α_2 are constant.

To find the values of α_1 and α_2, we can use the initial terms of F_n. We have that

$$F_1 = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right)^1 + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)^1 = 1 \text{ and}$$
$$F_2 = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right)^2 + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)^2 = 1,$$

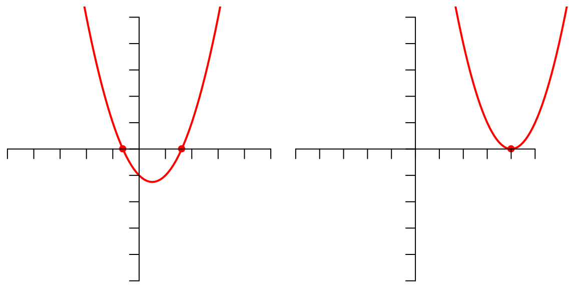
and solving these two expressions gives the values $\alpha_1 = 1/\sqrt{5}$ and $\alpha_2 = -1/\sqrt{5}$.

Therefore, the closed form for F_n is

$$F_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n.$$

3.1.6 Second-Order Recurrence Relations with One Characteristic Root

Proposition 3.1 assumes that the characteristic equation has two distinct roots. However, we don't have a guarantee that a quadratic equation will always have two distinct roots. We may instead have two non-distinct roots, say, when the curve corresponding to the characteristic equation intersects the x-axis at exactly one point. Consider, for example, the following plots:



The plot on the left corresponds to the equation $r^2 - r - 1$, which we saw in Example 3.10. The plot on the right corresponds to an equation that intersects the x-axis only once; namely, the equation $r^2 - 8r + 16$. In this case, we are still able to use the characteristic root method, but we must make one small modification to its formulation. The proof of the following proposition is again omitted.

Proposition 3.2 *Let c_1 and c_2 be real numbers where $c_2 \neq 0$. Suppose that the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ has a corresponding characteristic equation*

$$r^2 - c_1 r - c_2 = 0$$

with exactly one root r_1. Then the recurrense has a solution of the form

$$a_n = \alpha_1 \, r_1^n + \alpha_2 \, n \, r_1^n$$

for $n \in \mathbb{N}$, where α_1 and α_2 are constants.

To make the characteristic root method work for characteristic equations with one root, we multiply the second term (that is, the term including α_2) by a factor of n.

Example 3.11 *Consider the recurrence relation $g_n = 8g_{n-1} - 16g_{n-2}$ with initial terms $g_1 = 1$ and $g_2 = 6$. What is a closed form for g_n?*

From the statement of the recurrence relation for g_n, we see that $c_1 = 8$ and $c_2 = -16$, so the characteristic equation for g_n is

$$r^2 - 8r + 16 = 0.$$

The only root of this equation is $r_1 = 4$. Therefore, the recurrence relation for g_n has a solution of the form

$$a_n = \alpha_1\, 4^n + \alpha_2\, n\, 4^n$$

for all $n \in \mathbb{N}$, where α_1 and α_2 are constants.

To find the values of α_1 and α_2, we can use the initial terms of g_n. We have that

$$g_1 = \alpha_1\, 4^1 + \alpha_2\,(1)\,(4^1) = 4\,\alpha_1 + 4\,\alpha_2 = 1, \text{ and}$$
$$g_2 = \alpha_1\, 4^2 + \alpha_2\,(2)\,(4^2) = 16\,\alpha_1 + 32\,\alpha_2 = 6,$$

and solving these two equations gives the values $\alpha_1 = 1/8$ and $\alpha_2 = 1/8$.

Therefore, the closed form for g_n is

$$g_n = \frac{1}{8}\, 4^n + \frac{1}{8}\, n\, 4^n.$$