8 Advanced Counting Techniques

Introduction

In this chapter, we will describe how to apply Maple to three important topics in counting: recurrence relations, generating functions, and inclusion–exclusion. We begin by describing how Maple can be used to solve recurrence relations, including the recurrence relations that describe the complexity of divide-and-conquer algorithms. After studying recurrence relations, we show how to use Maple to manipulate generating functions using the package **powseries** and how these capabilities can help solve counting problems. We conclude the chapter with a discussion of the principle of inclusion and exclusion.

8.1 Recurrence Relations

A recurrence relation describes a relationship between the members of a sequence and their predecessors. For example, the famous Fibonacci sequence $\{f_n\}$ satisfies the recurrence relation

$$f_n = f_{n-1} + f_{n-2}.$$

Together with the initial conditions $f_1 = 1$ and $f_2 = 1$, this relation is sufficient to define the entire sequence $\{f_n\}$.

To understand how we can work with recurrence relations in Maple, we have to remember that a sequence $\{a_n\}$ is a function whose domain is a subset of the integers (usually the positive integers or nonnegative integers, depending on the context) and whose codomain contains the terms of the sequence (which can be numbers, matrices, circles, functions, etc.). (See the definition of sequence given in Section 2.4 of the textbook.)

With this point of view, the sequence $\{a_n\}$ is a function a and the nth term of the sequence is the value of the function evaluated at the integer n, that is, $a_n = a(n)$. This is only a change in notation, but it makes it easier to see that a recurrence relation can be represented in Maple as a procedure taking integer arguments.

We can represent the Fibonacci sequence by the procedure below, which we use to compute the first 20 terms of the Fibonacci sequence. This procedure takes one argument, a positive integer, and returns the appropriate term in the Fibonacci sequence.

```
Fibonacci := proc(n::posint)
   option remember;
   if n = 1 or n = 2 then
      return 1;
   end if;
   Fibonacci(n-1) + Fibonacci(n-2);
end proc:
```

```
> seq(Fibonacci(n), n = 1..20)
     1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
        2584, 4181, 6765                                                    (8.1)
```

Note that the **remember** option given in the first line instructs Maple to “remember” the values of the procedure that have already been calculated. (To be precise, the **remember** option causes Maple to store the output of the procedure in a “remember table.” If it is called with that same input again, Maple simply looks up the result from the table rather than recomputing. You can force Maple to delete the remember table for a procedure with the **forget** command.)

Sometimes, a recursive implementation of an algorithm may be too costly (specifically, in time and memory) due to the nature of the algorithm, regardless of steps you may take to improve efficiency. A recursive implementation of a recurrence relation can be avoided if we can find an explicit formula for the general term of the sequence. The process of finding such a formula is referred to as “solving” the recurrence relation. In the next section, we will see how to use Maple to solve certain kinds of recurrence relations.

Tower of Hanoi Problem

In Example 2 of Section 8.1 of the textbook, the author describes the famous “Tower of Hanoi” puzzle and derives the recurrence relation

$$H_n = 2\,H_{n-1} + 1,\ H_1 = 1,$$

where H_n represents the number of moves required to solve the puzzle for n disks. As discussed in the text, this has the solution

$$H_n = 2^n - 1.$$

Later, we will see how to use Maple to derive this result.

Rather than just computing the values, we can illustrate the solution to the Tower of Hanoi puzzle by writing a Maple program to compute the moves needed and to describe them to us. We will write a small program consisting of three Maple procedures: the main procedure **Hanoi**, a utility routine **PrintMove**, and **TransferDisk**, which does most of the work.

The easiest part to write is the function **PrintMove**, which merely displays the move to make at a given step.

```
PrintMove := proc(src::string, dest::string)
   printf("Move disk from peg %s to peg %s\n",src,dest);
end proc:
```

In the above, we call the Maple command **printf**, which is used for formatted output. The first argument to **printf** is the “format string,” that is, a string that contains both regular characters, format specifications (in this case “**%s**”), and escape characters (such as “**\n**”). The format specification “**%s**” tells the **printf** command to put the next argument in that location in the output and format it as a string. (The first **%s** refers to the second argument, **src** and the second **%s** refers to the third argument, **dest**.) The escape character “**\n**” tells **printf** to insert a new line. The **printf** command is very flexible with many available options. You should refer to Maple’s help pages for more detailed information. (Note: Maple’s **printf** command is very similar to the command of the same name in C and other programming languages.)

Next, we write the recursive procedure **TransferDisk**, which does most of the work. This function models the idea of transferring a stack of **ndisks** disks from the source peg, which is given as the argument **src**, to the destination peg, **dest**, via the intermediate peg, **via**. As described in the text, in order to move a stack of n disks, you first move the top $n-1$ pegs to the intermediate peg (using the destination as the intermediary), then move the bottom disk to the destination, and then move the smaller stack from the intermediate peg to the destination. Unless, of course, there is only 1 disk, in which you just move that disk to the destination. This is coded as follows.

```
TransferDisk := proc(src::string, via:: string, dest::string,
   ndisks::posint)
   if ndisks = 1 then
     PrintMove(src, dest);
   else
     TransferDisk(src, dest, via, ndisks - 1);
     PrintMove(src, dest);
     TransferDisk(via, src, dest, ndisks - 1);
   end if;
end proc:
```

Finally, we package the recursive procedure in a top-level procedure, **Hanoi**, providing an interface to the recursive engine.

```
Hanoi := proc(ndisks::posint)
  TransferDisk("A", "B", "C", ndisks);
end proc:
```

Our **Hanoi** program can exhibit a specific solution to the Tower of Hanoi puzzle for any number of disks:

```
> Hanoi(2)
    Move disk from peg A to peg B
    Move disk from peg A to peg C
    Move disk from peg B to peg C
```

```
> Hanoi(3)
    Move disk from peg A to peg C
    Move disk from peg A to peg B
    Move disk from peg C to peg B
    Move disk from peg A to peg C
    Move disk from peg B to peg A
    Move disk from peg B to peg C
    Move disk from peg A to peg C
```

Try experimenting with different values of **ndisk** to get a feel for how large the problem becomes for even moderately large numbers of disks.

Dynamic Programming

We conclude this section with an implementation of Algorithm 1 from Section 8.1 of the text. Recall that the goal of this algorithm is to find the maximum number of attendees that can be achieved by a schedule of talks.

We will represent each talk as a list of three elements, with the start time being the first element, the end time in the second position, and the weight, or attendance, will be last. Time of day will be represented by a single number with whole part equal to the hour in the 24-hour system and with fractional part equal to the part of an hour that corresponds to the number of minutes. For instance, 2:30 P.M. would be represented as 14.5.

As an example, consider the following eight talks.

Start Time	End Time	Attendance
9:00 AM	11:00 AM	17
9:00 AM	10:30 AM	15
10:00 AM	11:30 AM	22
10:30 AM	12:00 PM	11
11:30 AM	1:30 PM	18
12:00 PM	1:00 PM	12
1:30 PM	3:00 PM	21
2:00 PM	4:00 PM	17

We create the following list of lists to represent the talks.

```
> talks := [[9, 11, 17], [9, 10.5, 15], [10, 11.5, 22], [10.5, 12, 11], [11.5, 13.5, 18],
    [12, 13, 12], [13.5, 15, 21], [14, 16, 17]]
    talks := [[9, 11, 17], [9, 10.5, 15], [10, 11.5, 22], [10.5, 12, 11],
      [11.5, 13.5, 18], [12, 13, 12], [13.5, 15, 21], [14, 16, 17]]                (8.2)
```

Recall the description of Algorithm 1 from the text. We summarize the general outline of the algorithm below.

1. Sort the talks in order of increasing end time.
2. For each index j, compute $p(j)$—the maximum index i less than j such that talk i is compatible with talk j.
3. For each index j, compute $T(j)$, which is computed by the recurrence relation $T(j) = \max\left(w_j + T(p(j)), T(j-1)\right)$ and with initial condition $T(0) = 0$.
4. The maximum total number of attendees is $T(n)$, where n is the number of talks.

For step 1, we will make use of the **sort** command with a custom ordering procedure. Recall that **sort** can accept an optional argument in the form of a Boolean procedure of two arguments. This procedure should return true if the first argument precedes the second and false otherwise. Since we must sort the talks in increasing order of end time (which is stored in position 2 in the lists representing the talks), we use the following procedure.

```
sortEnd := proc(a,b)
   return a[2] < b[2];
end proc:
```

For step 2, we must compute $p(j)$. For this, we create a procedure that accepts the sorted list of talks and returns a table that represents the function p. Recall that the value of $p(j)$ is the largest index among talks compatible with the talk with index j, so we call this procedure **compatible**.

After declaring local variables and initializing **p** to the empty table, we will loop through all the indices, **j**, from 1 to the number of talks in the list. We use a local variable, **jstart**, to store the start time of the current talk being analyzed and we set the value of **p** for **j** to 0. We then consider all the talks earlier in the list beginning with the talk with index **j-1** and working backward to talk **1**. For each talk, we check to see if it ends before talk **j** starts. When we find such a talk, we set its index to the value of **p[j]** (since we are working backward, the first one found is the talk with the largest index) and terminate the loop. If no compatible talk is found, then **p[j]** was already set to **0**. Here is the procedure.

```
compatible := proc(talkList)
  local p, j, jstart, i;
  p := table();
  for j from 1 to nops(talkList) do
    jstart := talkList[j][1];
    p[j] := 0;
    for i from j-1 to 1 by -1 do
      if talkList[i][2] <= jstart then
        p[j] := i;
        break;
      end if;
    end do;
  end do;
  return p;
end proc:
```

For step 3, we must compute $T(j)$. To do this, we create a procedure that accepts as input the sorted list of talks and the table representing the function p. Initialize **T** to the empty table and set its value at 0 to 0. Then, consider each integer j from 1 to the number of talks and apply the formula: $T(j) = \max\left(w_j + T(p(j)), T(j-1)\right)$.

```
totalAttendance := proc(talkList,p)
  local j, T;
  T := table();
  T[0] := 0;
  for j from 1 to nops(talkList) do
    T[j] := max(talkList[j][3] + T[p[j]],T[j-1]);
  end do;
  return T;
end proc:
```

We can now put the pieces together as outlined at the start of this subsection.

```
maximumAttendees := proc(talkList)
  local L, p, T;
  L := sort(talkList,sortEnd);
```

```
    p := compatible(L);
    T := totalAttendance(L,p);
    return T[nops(L)];
end proc:
```

And thus, the maximum attendance for the talks described above is:

```
> maximumAttendees(talks)
    61                                                    (8.3)
```

8.2 Solving Linear Recurrence Relations

Maple has a very powerful recurrence solver, **rsolve**. Its use, however, can obscure some of the important ideas that are involved. Therefore, we will first use some of Maple's more fundamental facilities to solve certain kinds of recurrence relations one step at a time.

Given a recursively defined sequence $\{a_n\}$, we would like to find a formula, involving only the index n (and, perhaps, other fixed constants and known functions) which does *not* depend on knowing the value of any prior elements of the sequence.

Linear Homogeneous Recurrence Relations with Constant Coefficients

We will begin by considering recurrence relations that are *linear*, *homogeneous*, and which have *constant coefficients*; that is, they have the form

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

where $c_1, c_2, \ldots, c_k$ are real constants and c_k is nonzero. Recall that the integer k is called the *degree* of this recurrence relation. To have a unique solution, at least k initial conditions must be specified.

The general method for solving such a recurrence relation involves finding the roots of its characteristic polynomial

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

When this polynomial has distinct roots, all solutions are linear combinations of the nth powers of these roots. When there are repeated roots, the situation is a little more complicated, as we will see.

A First Example

Consider the linear homogeneous recurrence relation with constant coefficients of degree two

$$a_n = 2\,a_{n-1} + 3\,a_{n-2},$$

subject to the initial conditions

$$a_1 = 4 \text{ and } a_2 = 2.$$

Its characteristic equation is

$$r^2 - 2r - 3 = 0.$$

To solve the recurrence relation, we must solve for the roots of this equation. Using Maple makes this very easy; we use the **solve** function.

```
> solve(r^2 - 2 r - 3 = 0, r)
    3, -1                                                  (8.4)
```

The **solve** command computes the values of the variable r, given as the second argument, that satisfy the equation in the first argument. Note that you may omit **=0** from the command above and Maple will interpret it the same.

```
> solve(r^2 - 2 r - 3, r)
    3, -1                                                  (8.5)
```

Now that Maple has determined that the solutions are $r = 3$ and $r = -1$, we can write down the form of the solution to the recurrence as

$$a_n = \alpha\, 3^n + \beta\, (-1)^n,$$

where α and β are constants that we have yet to determine.

Since the initial conditions are $a_1 = 4$ and $a_2 = 2$, we know that our recurrence relation must satisfy the following pair of equations.

$$\begin{cases} 3\alpha - \beta = 4 \\ 3^2\alpha + \beta = 2 \end{cases}$$

To find the solution to this system of linear equations, we again use Maple's **solve** command:

```
> solve({3 α - β = 4, 9 α + β = 2}, {α, β})
    {α = 1/2, β = -5/2}                                    (8.6)
```

This time, we are telling Maple to solve the set of equations. Likewise, the variables to be solved for form a set.

Now that we have the values for α and β, we see that the complete solution to the recurrence relation is

$$a_n = \frac{1}{2} \cdot 3^n - \frac{5}{2} \cdot (-1)^n.$$

This formula allows us to write a Maple function for finding the terms of the sequence $\{a_n\}$, which can be more efficient than a recursive procedure.

> $a := n :: posint \rightarrow \frac{3^n}{2} - \frac{5(-1)^2}{2}$:

> $seq\left(a(n), n = 1..10\right)$

$$4, 2, 16, 38, 124, 362, 1096, 3278, 9844, 29\,522 \qquad \textbf{(8.7)}$$

A Second Example

Let us try another example. We will solve the recurrence relation

$$a_n = -\frac{5}{3}\,a_{n-1} + \frac{2}{3}\,a_{n-2}$$

with initial conditions

$$a_1 = \frac{1}{2} \text{ and } a_2 = 4.$$

To do this, we ask Maple to solve the characteristic equation of the recurrence relation, and then solve the system of linear equations obtained from the roots of the characteristic equation and the initial conditions. Note that this method works because this recurrence relation is linear, homogeneous, and has constant coefficients.

> $CharEqnRoots := solve\left(r^2 + \frac{5}{3}\,r - \frac{2}{3}, r\right)$

$$CharEqnRoots := \frac{1}{3}, -2 \qquad \textbf{(8.8)}$$

> $solve\left(\left\{\text{alpha} \cdot CharEqnRoots[1] + \text{beta} \cdot CharEqnRoots[2] = \frac{1}{2},\right.\right.$
$\left.\text{alpha} \cdot CharEqnRoots[1]^2 + \text{beta} \cdot CharEqnRoots[2]^2 = 4\right\},$
$\left.\{\text{alpha}, \text{beta}\}\right)$

$$\left\{\alpha = \frac{45}{7}, \beta = \frac{23}{28}\right\} \qquad \textbf{(8.9)}$$

Thus, we see that the solution to the recurrence relation is

$$a_n = \frac{45}{7}\left(\frac{1}{3}\right)^n + \frac{23}{28}\,(-2)^n.$$

The Fibonacci Sequence

We can derive an explicit formula for the Fibonacci sequence this way as well. The characteristic polynomial for the Fibonacci sequence is

$$r^2 - r - 1.$$

We find the roots of the characteristic equation.

> $CEqnRoots := solve\left(r^2 - r - 1, r\right)$

$$CEqnRoots := \frac{\sqrt{5}}{2} + \frac{1}{2}, -\frac{\sqrt{5}}{2} + \frac{1}{2} \tag{8.10}$$

Therefore, the formula for the *n*th Fibonacci number is of the form

> $Fn := \mathrm{alpha} \cdot CEqnRoots[1]^n + \mathrm{beta} \cdot CEqnRoots[2]^n$

$$Fn := \alpha \left(\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n + \beta \left(-\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n \tag{8.11}$$

We find the coefficients α and β in the formula by using the initial conditions.

> $alphas := solve\Big(\Big\{\mathrm{alpha} \cdot CEqnRoots[1] + \mathrm{beta} \cdot CEqnRoots[2] = 1,$
$\mathrm{alpha} \cdot CEqnRoots[1]^2 + \mathrm{beta} \cdot CEqnRoots[2]^2 = 1\Big\}, \{\mathrm{alpha}, \mathrm{beta}\}\Big)$

$$alphas := \left\{\alpha = \frac{\sqrt{5}}{5}, \beta = -\frac{\sqrt{5}}{5}\right\} \tag{8.12}$$

We use the **subs** command to substitute the values for α and β into the formula **Fn**.

> $Fn := subs(alphas, Fn)$

$$Fn := \frac{\sqrt{5}\left(\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n}{5} - \frac{\sqrt{5}\left(-\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n}{5} \tag{8.13}$$

If we are to use such a formula to repeatedly compute values, then we should use it to define a function. You can enter a new function definition manually, but a more convenient way is to use the **unapply** command. It takes two arguments: an expression and the variable that is to be the argument of the function. It produces a functional operator.

> $Fibonacci2 := unapply(Fn, n)$

$$Fibonacci2 := n \mapsto \frac{\sqrt{5}\left(\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n}{5} - \frac{\sqrt{5}\left(-\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n}{5} \tag{8.14}$$

The procedure **Fibonacci2** is much more efficient than even the optimized recursive procedure **Fibonacci**. To see this, we record the accumulated time for computing the first 100 000 Fibonacci numbers (after, of course, clearing the remember table for the **Fibonacci** procedure with the **forget** command).

> $forget(Fibonacci)$

```
> st := time( ) :
  for i to 100 000 do
     Fibonacci(i)
  end do :
  time ( ) − st
     1.452                                                        (8.15)
```

```
> st := time( ) :
  for i to 100 000 do
     Fibonacci2(i)
  end do :
  time ( ) − st
     0.417                                                        (8.16)
```

A Solver

Now, we will generalize what we have been doing and write a Maple procedure to solve a degree two linear, homogeneous recurrence relation with constant coefficients, provided that the roots of the characteristic polynomial are distinct. We will write a procedure **RecSol2Distinct** which solves the recurrence

$$a_n = ca_{n-1} + da_{n-2}$$

subject to the initial conditions

$$a_1 = u \text{ and } a_2 = v$$

and then returns a procedure that can be used to compute terms of the sequence.

For the moment, we assume that the characteristic polynomial $r^2 - cr - d$ has two distinct roots. Later, we will modify the procedure to relax that restriction. With the assumption that the roots of the characteristic polynomial are distinct, all our procedure needs to do is to repeat the steps we did manually in the examples above.

```
RecSol2Distinct := proc(c, d, u, v)
   local CERoots, alphas, alpha, beta, f, n, r;
   # First solve the characteristic equation
   CERoots := solve(r^2 - c*r-d, r);
   # Next solve the equations derived from initial conditions
   alphas := solve({
      alpha * CERoots[1] + beta * CERoots[2] = u,
      alpha * CERoots[1]^2 + beta * CERoots[2]^2 = v
   }, {alpha, beta});
   # Finally substitute the answers into the general form
   f := subs(alphas, alpha * CERoots[1]^n + beta * CERoots[2]^n);
   return unapply(f, n);
end proc:
```

To see how it works, we check that it gives the same result for the Fibonacci sequence that we obtained by hand. To construct a function for computing the Fibonacci sequence, invoke the new procedure as:

> $f := RecSol2Distinct(1, 1, 1, 1)$

$$f := n \mapsto \frac{\sqrt{5}\left(\frac{\sqrt{5}}{2}+\frac{1}{2}\right)^{n}}{5} - \frac{\sqrt{5}\left(-\frac{\sqrt{5}}{2}+\frac{1}{2}\right)^{n}}{5} \tag{8.17}$$

The procedure **f** can be used to compute the general term of the Fibonacci sequence.

> $f(n)$

$$\frac{\sqrt{5}\left(\frac{\sqrt{5}}{2}+\frac{1}{2}\right)^{n}}{5} - \frac{\sqrt{5}\left(-\frac{\sqrt{5}}{2}+\frac{1}{2}\right)^{n}}{5} \tag{8.18}$$

You can see that is the same formula that we derived above and correctly produces the first 10 Fibonacci numbers.

> $seq(simplify(f(n)), n = 1..10)$

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55 \tag{8.19}$$

A Recurrence with Repeated Roots

We will next create a procedure that can handle the case of repeated roots. First, let us look at an example of a recurrence relation whose characteristic polynomial has a double root. The recurrence relation

$$a_n = 4\,a_{n-1} - 4\,r_{n-2}$$

has the characteristic equation

> $charEqn := r^2 - 4\,r + 4 = 0$

$$charEqn := r^2 - 4\,r + 4 = 0 \tag{8.20}$$

which has roots

> $CERoots := solve(charEqn, r)$

$$CERoots := 2, 2 \tag{8.21}$$

We can clearly see that in this case the root is repeated, but for Maple to recognize it, we need to use the following test.

> $evalb(CERoots[1] = CERoots[2])$

$$true \tag{8.22}$$

Note that the **evalb** command forces evaluation as a Boolean value. This is not necessary in conditional statements (like **if** statements) because the context of those statements causes Maple to evaluate the expression as a Boolean. If we call the double root (2 in this case) r_0, then the recurrence relation has the explicit solution

$$a_n = \alpha\, r_0^n + n\beta\, r_0^n,$$

for all positive integers n, and for some constants α and β. The initial conditions of $a_1 = 1$ and $a_2 = 4$ produce the system of equations

$$\begin{cases} \alpha \cdot 2^1 + 1 \cdot \beta \cdot 2^1 = 1 \\ \alpha \cdot 2^2 + 2 \cdot \beta \cdot 2^2 = 4 \end{cases}$$

As before, we solve this system for α and β.

$$> \mathit{Alphas} := \mathit{solve}\left(\left\{\mathrm{alpha} \cdot \mathit{CERoots}[1] + \mathrm{beta} \cdot \mathit{CERoots}[1] = 1, \right.\right.$$
$$\left.\left.\mathrm{alpha} \cdot \mathit{CERoots}[1]^2 + 2\,\mathrm{beta} \cdot \mathit{CERoots}[1]^2 = 4\right\}, \{\mathrm{alpha}, \mathrm{beta}\}\right)$$

$$\mathit{Alphas} := \{\alpha = 0, \beta = 1/2\} \tag{8.23}$$

And finally, substitute these values into the general form $a_n = \alpha\, r_0^n + n\beta\, r_0^n$,

$$> \mathit{subs}\left(\mathit{Alphas}, \mathrm{alpha} \cdot \mathit{CERoots}[1]^n + n \cdot \mathrm{beta} \cdot \mathit{CERoots}[1]^n\right)$$

$$\frac{n\, 2^n}{2} \tag{8.24}$$

A More General Recurrence Solver

The steps carried out above are quite general and we can write a procedure, **RecSolver2**, which solves a recurrence relation (degree two, linear, homogeneous, with constant coefficients) regardless of whether the characteristic polynomial has distinct roots or not. The following procedure solves the recurrence

$$a_n = ca_{n-1} + da_{n-2},$$

with initial conditions

$$a_1 = u \text{ and } a_2 = v.$$

```
RecSolver2 := proc(c, d, u, v)
   local CERoots, alphas, alpha, beta, f, n, r;
   # First solve the characteristic equation
   CERoots := solve(r^2 - c*r - d, r);
   # Then test if the roots are the same
   if (CERoots[1] = CERoots[2]) then
      # the roots are the same so follow the last example
      alphas := solve({
         alpha * CERoots[1] + beta * CERoots[1] = u,
         alpha * CERoots[1]^2 + 2 * beta * CERoots[1]^2 = v
      },{alpha,beta});
      f := subs(alphas,alpha * CERoots[1]^n + n * beta *
          CERoots[1]^n);
   else
```

```
      # otherwise, use the RecSol2 method
      alphas := solve({
         alpha * CERoots[1] + beta * CERoots[2] = u,
         alpha * CERoots[1]^2 + beta * CERoots[2]^2 = v
      },{alpha,beta});
      f := subs(alphas,alpha * CERoots[1]^n + beta * CERoots[2]^n);
   end if;
   # Finally, use unapply
   return unapply(f, n);
end proc:
```

RecSolver2 first tests for a repeated root and then does the appropriate computation. We test this procedure on the examples we did by hand, such as the Fibonacci sequence:

> *RecSolver2* (1, 1, 1, 1)

$$n \mapsto \frac{\sqrt{5}\left(\frac{\sqrt{5}}{2}+\frac{1}{2}\right)^n}{5} - \frac{\sqrt{5}\left(-\frac{\sqrt{5}}{2}+\frac{1}{2}\right)^n}{5} \qquad \textbf{(8.25)}$$

For the example with a double root:

> *RecSolver2* (4, −4, 1, 4)

$$n \mapsto \frac{n\,2^n}{2} \qquad \textbf{(8.26)}$$

In both of those examples, the result is consistent with what we had obtained before. We will now use the solver to find the first 10 terms of the sequence defined by the following recurrence relation and initial conditions.

$$a_n = 4\,a_{n-1} - 3\,a_{n-2}$$

$$a_1 = 1 \text{ and } a_2 = 2.$$

> *g* := *RecSolver2* (4, −3, 1, 2)

$$g := n \mapsto \frac{3^n}{6} + \frac{1}{2} \qquad \textbf{(8.27)}$$

> *seq* (*simplify* (*g* (*n*)), *n* = 1 ..10)

$$1, 2, 5, 14, 41, 122, 365, 1094, 3281, 9842 \qquad \textbf{(8.28)}$$

As another example, consider the following recurrence relation

$$a_n = -a_{n-1} - a_{n-2},$$

with initial conditions

$$a_1 = 1 \text{ and } a_2 = 2.$$

```
> h := RecSolver2(−1, −1, 1, 2)
```

$$h := n \mapsto \frac{\left(\sqrt{3} - 5I\right)\sqrt{3}\left(-\frac{1}{2} + \frac{I\sqrt{3}}{2}\right)^n}{3(-1 + I\sqrt{3})} + \frac{I\sqrt{3}\left(-2 + 6I\sqrt{3}\right)\left(-\frac{1}{2} - \frac{I\sqrt{3}}{2}\right)^n}{12} \tag{8.29}$$

Notice that the solution to this recurrence is very complicated and requires the use of complex numbers. (Maple uses the name **I** to represent the imaginary unit $\sqrt{-1}$.) However, if we compute the first 10 terms, we notice a very simple pattern emerges.

```
> seq(simplify(h(n)), n = 1..10)
```

$$1,\ 2,\ -3,\ 1,\ 2,\ -3,\ 1,\ 2,\ -3,\ 1 \tag{8.30}$$

Maple can make this pattern explicit if we replace the numerical initial conditions with symbolic constants.

```
> k := RecSolver2(−1, −1, lambda, mu)
```

$$k := n \mapsto \frac{\left(\sqrt{3}\lambda - I\lambda - 2I\mu\right)\sqrt{3}\left(-\frac{1}{2} + \frac{I\sqrt{3}}{2}\right)^n}{3(-1 + I\sqrt{3})} + \frac{I\sqrt{3}\left(2\lambda + 2I\sqrt{3}\lambda + 2I\sqrt{3}\mu - 2\mu\right)\left(-\frac{1}{2} - \frac{I\sqrt{3}}{2}\right)^n}{12} \tag{8.31}$$

```
> seq(simplify(k(n)), n = 1..10)
```

$$\lambda,\ \mu,\ -\lambda - \mu,\ \lambda,\ \mu,\ -\lambda - \mu,\ \lambda,\ \mu,\ -\lambda - \mu,\ \lambda \tag{8.32}$$

Nonhomogeneous Recurrence Relations

So far, we have been restricted to homogeneous linear recurrence relations with constant coefficients. However, the techniques used in solving them may be extended to provide solutions to *nonhomogeneous* linear recurrence relations with constant coefficients. That is, recurrence relations of the form

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

with $c_1,\ c_2,\ ..,\ c_k$ real numbers and $F(n)$ a function depending only on n. To solve this more general form of a recurrence relation, we do two things: (1) find the solutions of the associated homogeneous recurrence relation (the relation obtained by removing $F(n)$); (2) find a particular solution for the nonhomogeneous equation.

Consider the following example:

$$a_n = 6\,a_{n-1} - 9\,a_{n-2} + n\,3^n,$$

from Example 12 of Section 8.2 in the text.

The first step is to find the solutions to the associated homogeneous recurrence relation

$$a_n = 6\,a_{n-1} - 9\,a_{n-2}.$$

To do this, we can use our **RecSolver2**. We will use α and β for the initial conditions so that we get all the solutions.

> $HSolution := RecSolver2\,(6, -9, \text{alpha}, \text{beta})$

$$HSolution := n \mapsto \left(-\frac{\beta}{9} + \frac{2\,\alpha}{3}\right) 3^n + n\left(\frac{\beta}{9} - \frac{\alpha}{3}\right) 3^n \tag{8.33}$$

The second step is to find a particular solution. Theorem 6 in Section 8.2 of the text tells us how to find the form of the particular solution. Note that $F(n) = n\,3^n$ and 3 is a root of the characteristic polynomial with multiplicity 2 (you can verify this by solving the characteristic equation of the associated homogeneous relation; it is also made apparent by the form of **HSolution**). Thus, the theorem tells us that there is a particular solution of the form

$$n^2(pn+q)3^n.$$

We will define a functional operator for the form of the particular solution.

> $PForm := n \rightarrow n^2\,(p \cdot n + q)\,3^n$

$$PForm := n \mapsto n^2\,(p\,n + q)\,3^n \tag{8.34}$$

To find a particular solution, we need to find the values of p and q. To find these values, we substitute the terms of **PForm** into the recurrence relation. This gives us an equation in terms of p and q (and n).

> $Peqn := PForm\,(n) = 6 \cdot PForm\,(n-1) - 9 \cdot PForm\,(n-2) + n \cdot 3^n$

$$\begin{aligned}Peqn := n^2\,(p\,n+q)\,3^n = 6\,(n-1)^2\,(p\,(n-1)+q)\,3^{n-1}\\ -9\,(n-2)^2\,(p\,(n-2)+q)\,3^{n-2} + n\,3^n\end{aligned} \tag{8.35}$$

> $Peqn := simplify\,(Peqn)$

$$Peqn := n^2\,(p\,n+q)\,3^n = 3^n\,\left(n^3 p + n^2 q - 6\,p\,n + n + 6\,p - 2\,q\right) \tag{8.36}$$

The second line in the pair of commands above replaces the equation **Peqn** with its simplified form. This explicit use of the **simplify** command is necessary for Maple to be able to solve the equation.

Next, we have Maple solve that equation for p and q. In order to indicate that we want Maple to find the values of p and q that satisfy the equation for all values of n, we will make use of the **identity** command within the **solve** command as follows.

The **identity** command can only be used within the first argument of **solve**. It requires two arguments. The first is an equation, such as **Peqn**, or an expression which is assumed to be equated to 0. The second argument is the name of the variable that the identity is in terms of.

```
> Pvals := solve(identity(Peqn, n), {p, q})
```

$$Pvals := \left\{p = \frac{1}{6}, q = \frac{1}{2}\right\} \tag{8.37}$$

Thus, the particular solution is

```
> subs(Pvals, PForm(n))
```

$$n^2 \left(\frac{n}{6} + \frac{1}{2}\right) 3^n \tag{8.38}$$

Putting it all together, we see that all solutions to the recurrence relation $a_n = 6\,a_{n-1} - 9\,a_{n-2} + n\,3^n$ are of the form

```
> HSolution(n) + subs(Pvals, PForm(n))
```

$$\left(-\frac{\beta}{9} + \frac{2\,\alpha}{3}\right) 3^n + n\left(\frac{\beta}{9} - \frac{\alpha}{3}\right) 3^n + n^2 \left(\frac{n}{6} + \frac{1}{2}\right) 3^n \tag{8.39}$$

Maple's Recurrence Solver

Now that we have seen how to use Maple to implement an algorithm to solve simple recurrence relations, it is time to introduce Maple's command for solving recurrence relations.

We have already seen the Maple command **solve** for working with polynomial equations and systems of equations. Similarly, there is a Maple command **rsolve**, which is specially engineered for dealing with recurrence relations. It is a much more sophisticated version of our **RecSolver2** procedure and can deal with recurrence relations of arbitrary degree, repeated roots, and nonlinear recurrence relations. To use **rsolve**, you need to tell it what the recurrence relation is and some initial conditions. You must also specify the name of the recursive function to solve for.

For example, to solve the Fibonacci recurrence, you enter the following statement.

```
> unassign(F):
  rsolve({F(0) = 0, F(1) = 1, F(n) = F(n - 1) + F(n - 2)}, F(n))
```

$$\frac{\sqrt{5}\left(\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n}{5} - \frac{\sqrt{5}\left(-\frac{\sqrt{5}}{2} + \frac{1}{2}\right)^n}{5} \tag{8.40}$$

(We use the **unassign** command to clear the name **F** of anything that may have been stored in it before. Using **unassign**, or a command like **F := 'F'**, is a good idea before using **rsolve**, since if the function name is already storing a value, **rsolve** may return an error.)

The **rsolve** command will let us solve nonhomogeneous recurrence relations like the Tower of Hanoi problem very easily. Recall that the Tower of Hanoi problem has the recurrence relation

$$H_n = 2\,H_{n-1} + 1,$$

with initial condition $H_1 = 1$.

```
> unassign(H):
  rsolve({H(1) = 1, H(n) = 2 H(n - 1) + 1}, H(n))
```

$$2^n - 1 \tag{8.41}$$

It is not actually necessary to specify the initial conditions for a recurrence relation. If they are not present, Maple will still solve the equation, inserting symbolic constants (e.g., **G(0)** and **G(1)**) in place of numeric values, as the following example illustrates.

$$> \mathit{unassign}(G):$$
$$\mathit{rsolve}\,(G\,(n) = 2\,G\,(n-1) - 6\,G\,(n-2)\,, G\,(n))$$

$$-\frac{I\left(-G\,(1)\,\sqrt{5} + G\,(0)\,\sqrt{5} + 5\,I\,G\,(0)\right)\left(-I\,\sqrt{5}+1\right)^{n}}{10} - \frac{I\left(G\,(1)\,\sqrt{5} - G\,(0)\,\sqrt{5} + 5\,I\,G\,(0)\right)\left(I\,\sqrt{5}+1\right)^{n}}{10} \tag{8.42}$$

The function **rsolve** can handle different kinds of recurrence relations, including:

- linear recurrence relations with constant coefficients,
- systems of linear recurrence relations with constant coefficients,
- divide-and-conquer recurrence relations with constant coefficients,
- many first order linear recurrence relations, and
- some nonlinear first order recurrence relations.

The capabilities of **rsolve**, like other Maple functions, are constantly being enhanced and extended. However, **rsolve** is not a panacea—you can easily find recurrence relations that it is incapable of solving. When **rsolve** is unable to solve a recurrence relation, it simply returns unevaluated, as below.

$$> \mathit{unassign}(u):$$
$$\mathit{rsolve}\left(u\,(n) = (u\,(n-1))^2 - \mathrm{e}^{2\,u(n-2)}, u\,(n)\right)$$

$$\mathit{rsolve}\left(u\,(n) = (u\,(n-1))^2 - \mathrm{e}^{2\,u(n-2)}, u\,(n)\right) \tag{8.43}$$

Problem Solving with Maple and Recurrence Relations

It is often the case that a problem, as presented, gives no clue that a solution may be found using recurrence. Let us see how we can use Maple to solve a problem that is not explicitly expressed as one requiring the use of recurrence for its solution.

Here is our problem: into how many regions is the plane divided by 1000 lines, assuming that no two of the lines are parallel, and no three are coincident? Such a situation may arise in an attempt to model fissures in the ocean floor.

To start, we might try to discover the answer for smaller numbers of lines. To generalize the problem, we may ask for the number of regions produced by n lines, where n is some positive integer. It is fairly obvious that a single line (corresponding to the case $n = 1$) divides the plane into two regions. Two lines, if they are not parallel, can easily be seen to divide the plane into four regions. (Two parallel lines produce only three regions.) If we call the number of regions produced by n lines, no two of which are parallel and no three of which are coincident, R_n, then $R_1 = 2$ and $R_2 = 4$.

What does the situation look like when $n = 3$? Figure 8.1 is representative of this situation. In this case, the number of regions is 7, so $R_3 = 7$. To find R_4 we must add a fourth line to the diagram. This suggests trying to compute R_4 in terms of R_3 so that we begin to think of $\{R_n\}$ as a recurrence relation. Figure 8.2 shows what the situation looks like when a fourth line is added to the three

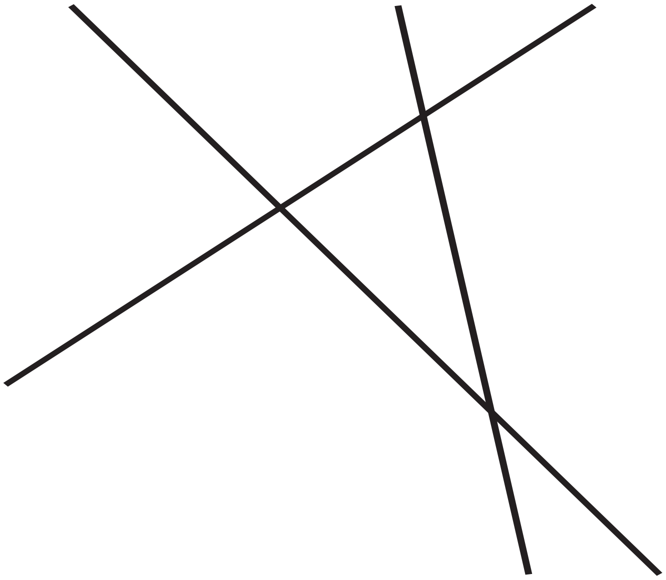

Figure 8.1: Three lines dividing the plane

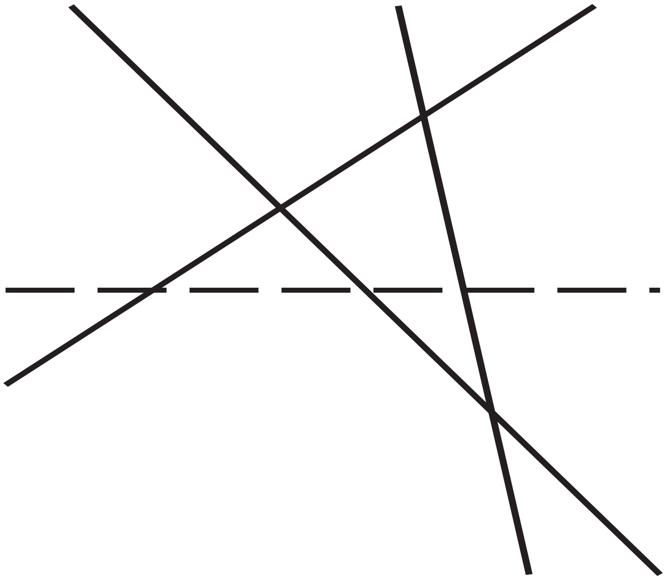

Figure 8.2: Four lines dividing the plane

existing lines. From the assumptions that no two lines are parallel and no three pass through a single point, it follows that the new line must intersect each of the existing three lines in exactly one point. This means that the new line passes through exactly four of the regions formed by the original three lines. Each region that it passes through is divided into two regions, so the total number of new regions added by the fourth point is 4. Thus, $R_4 = R_3 + 4$. Similar arguments for a general configuration of lines reveals that R_n satisfies the recurrence relation $R_n = R_{n-1} + n$.

Furthermore, we have already computed the initial condition $R_1 = 2$. This is enough to solve the recurrence.

```
> unassign(R):
  rsolve({R(1) = 2, R(n) = R(n − 1) + n}, R(n))
```

$$(n+1)\left(\frac{n}{2}+1\right)-n \tag{8.44}$$

```
> simplify((8.44))
```

$$\frac{1}{2}n^2+\frac{n}{2}+1 \tag{8.45}$$

To answer the question: how many regions is the plane divided by 1000 lines with no two parallel and no three coincident?

```
> R := unapply((8.44), n)
```

$$R := n \mapsto (n+1)\left(\frac{n}{2}+1\right)-n \tag{8.46}$$

```
> R(1000)
```

$$500\,501 \tag{8.47}$$

8.3 Divide-and-Conquer Algorithms and Recurrence Relations

A very good example of divide-and-conquer relations is the one provided by the binary search algorithm. Here, we consider a practical application of this algorithm in an implementation of a binary search on a sorted list of integers. This is an implementation of the algorithm described in Algorithm 3 in Section 3.1 of the text and first presented in Section 3.1 of this manual.

```
binarysearch := proc(x::integer, A::list(integer))
   local n, i, j, m, location;
   n := nops(A);
   i := 1;
   j := n;
   while i < j do
      m := floor((i+j)/2);
      if x > A[m] then
         i := m + 1;
      else
         j := m;
      end if;
   end do;
   if x = A[i] then
      location := i;
   else
      location := 0;
   end if;
   return location;
end proc:
```

The variable **A** is the list of integers to search, which is assumed to be sorted in increasing order, and **x** is the integer to search for. The local variables **j** and **i** are initialized to the number of elements in the list and 1, respectively. The while loop continues as long as **i** and **j** are different from each other. Each step of the loop serves to narrow the difference between them by calculating the middle of the list, represented by **m**, and determining which half **x** is in. Eventually, the search will focus in on one location in the list, which is either **x** or, if not, the search has failed and the algorithm returns 0.

Let us now do an analysis of the algorithm to see how divide-and-conquer recurrence relations are generated. In general, a divide-and-conquer type recurrence relation has the form

$$f(n) = af(n/b) + g(n).$$

Each iteration of the while loop of **binarysearch** produces a single list half the size of the original. Therefore, $a = 1$ and $b = 2$. The function $g(n)$, which measures the comparisons added in implementing the reduction, is identically 2. This is because one comparison is added to see which half of the list the key is on, and one is added to see if the while loop needs to continue. Hence, for the **binarysearch** algorithm, the recurrence relation is

$$f(n) = f(n/2) + 2.$$

Additionally, we can see that $f(1) = 2$, because if the list is of length 1, then the algorithm will do one comparison to determine that the while loop is unnecessary and one comparison to make sure that the element being searched for is the one element in the list. We can now use **rsolve** to solve this recurrence.

```
> unassign(bin):
  rsolve({bin(1) = 2, bin(n) = bin(n/2) + 2}, bin(n))
```

$$2 + \frac{2\ln(n)}{\ln(2)} \qquad \textbf{(8.48)}$$

8.4 Generating Functions

Generating functions are a powerful tool for manipulating sequences of numbers and for solving a variety of counting problems. In this section, we will see how Maple can be used to represent and manipulate generating functions.

The generating function $G(x)$ for a sequence $\{a_k\}$ is the formal power series

$$\sum_{k=0}^{\infty} a_k x^k = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_k x^k + \cdots.$$

It is called *formal* because we are not interested in evaluating it as a function of x. Our focus is on finding a formula for its coefficients. In particular, this means that there are no convergence issues to be considered.

Power Series Tools

Maple provides extensive facilities for manipulating formal power series. They belong to the Maple package **powseries**.

```
> with(powseries):
```

The first thing we need to do is to learn how to create a power series with Maple, which is done with the command **powcreate**. This command can be used in two ways: either by specifying a closed form for the general coefficient or with a recurrence relation and initial conditions.

For example, to create the generating function for the sequence $\{3^k\}$, we use the following code.

```
> unassign(e) :
  powcreate(e(k) = 3^k)
```

Note that we use the **unassign** command with **powcreate**, just as we did with **rsolve**. While it is not necessary in most situations, the **unassign** command can help avoid errors that may occur if these commands are reexecuted.

Note that the **powcreate** command does not have a return value but has made **e** into a procedure which is now Maple's representation of the power series. We can confirm this by having Maple display the first few terms of the series using the **tpsform** command (for truncated power series). This command takes three arguments: the name of the power series, the variable to be used in the expression, and the order. Maple displays the terms of the series whose degree is smaller than the given order.

```
> tpsform(e, x, 5)
```

$$1 + 3x + 9x^2 + 27x^3 + 81x^4 + O\left(x^5\right) \tag{8.49}$$

We can also define a power series using a recurrence relation together with initial conditions. (Note: If you do not provide sufficiently many initial conditions to guarantee a unique solution to the recurrence, an error will be raised.)

For example, to create the generating function for the Fibonacci sequence, which is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2},$$

with initial conditions

$$F_0 = 1 \text{ and } F_1 = 1,$$

we enter the following command.

```
>  unassign(Fibonacci3) :
  powcreate(Fibonacci3(n) = Fibonacci3(n − 1) + Fibonacci3(n − 2),
    Fibonacci3(0) = 1, Fibonacci3(1) = 1)
```

We can then have Maple display the first few terms of the generating function for the Fibonacci numbers.

```
> tpsform(Fibonacci3, x, 7)
```

$$1 + x + 2x^2 + 3x^3 + 5x^4 + 8x^5 + 13x^6 + O\left(x^7\right) \tag{8.50}$$

Maple also provides a way to access an arbitrary coefficient in a formal power series. To Maple, each formal power series is, in fact, a procedure that takes integer arguments. The value returned when given the integer n as an argument is the coefficient of the x^n term. For example, the 50th Fibonacci number can be computed as follows.

```
> Fibonacci3 (50)
    20 365 011 074                                        (8.51)
```

Solving Problems with Generating Functions

Generating functions are more than just a convenient way to represent numerical sequences. They are a powerful tool for solving recurrence relations, as well as other kinds of counting problems. This power stems from our ability to manipulate them like ordinary power series from Calculus and to interpret those manipulations. To illustrate Maple's facilities for manipulating generating functions, consider Example 12 from Section 8.4 of the text.

> Use generating functions to determine the number of ways to insert tokens worth \$1, \$2, and \$5 into a vending machine to pay for an item that costs r dollars in both the cases when the order in which the tokens are inserted does not matter and when the order does matter.

Following the text, the solution to the problem when order does matter is the coefficient of x^r in the generating function

$$\left(1+x+x^2+x^3+\cdots\right)\left(1+x^2+x^4+x^6+\cdots\right)\left(1+x^5+x^{10}+x^{15}+\cdots\right).$$

To solve the problem, we need to create the three power series and multiply them together. To create the three series, we could use the **powcreate** command as above, but the **evalpow** command provides an approach that is often easier.

The **evalpow** command accepts one argument, an algebraic expression which may include formal power series, polynomials, the usual arithmetic operators, and some functions compatible with the power series package (such as **powdiff** and **powint** for differentiation and integration). From Table 1 of Section 8.4 of the text, we see that our three generating functions can be written as rational expressions:

$$\frac{1}{1-x^r}=1+x^r+x^{2\,r}+x^{3\,r}+\cdots.$$

The three generating functions that we are interested in all share this same form with $r=1$, $r=2$, and $r=5$, respectively. Therefore, we can use **evalpow** to create them.

$$> \ Token1D := evalpow\left(\frac{1}{1-x}\right)$$

$$Token1D := \mathbf{proc}\,(powparm)\ldots\mathbf{end\ proc} \tag{8.52}$$

$$> \ Token2D := evalpow\left(\frac{1}{1-x^2}\right)$$

$$Token2D := \mathbf{proc}\,(powparm)\ldots\mathbf{end\ proc} \tag{8.53}$$

$$> \ Token5D := evalpow\left(\frac{1}{1-x^5}\right)$$

$$Token5D := \mathbf{proc}\,(powparm)\ldots\mathbf{end\ proc} \tag{8.54}$$

(It is worth noting that the **evalpow** command returns a procedure, which is how Maple represents power series.)

We use **tpsform** to confirm that these are the generating functions that we were trying to create.

> $\mathit{tpsform}\left(\mathit{Token1D}, x, 5\right)$

$$1 + x + x^2 + x^3 + x^4 + O\left(x^5\right) \tag{8.55}$$

> $\mathit{tpsform}\left(\mathit{Token2D}, x, 10\right)$

$$1 + x^2 + x^4 + x^6 + x^8 + O\left(x^{10}\right) \tag{8.56}$$

> $\mathit{tpsform}\left(\mathit{Token5D}, x, 25\right)$

$$1 + x^5 + x^{10} + x^{15} + x^{20} + O\left(x^{25}\right) \tag{8.57}$$

Now, we can use **evalpow** again to multiply the three series together.

> $\mathit{Tokens} := \mathit{evalpow}\left(\mathit{Token1D} \cdot \mathit{Token2D} \cdot \mathit{Token5D}\right)$

$$\mathit{Tokens} := \mathbf{proc}\left(\mathit{powparm}\right) \ldots \mathbf{end\ proc} \tag{8.58}$$

> $\mathit{tpsform}\left(\mathit{Tokens}, x, 8\right)$

$$1 + x + 2x^2 + 2x^3 + 3x^4 + 4x^5 + 5x^6 + 6x^7 + O\left(x^8\right) \tag{8.59}$$

We can see from the above that there are six ways to pay for a \$7 item (since the coefficient of x^7 is 6), just as was computed in the text. If we wanted to know the number of ways to pay for an item costing \$234, all we would need to do is find the coefficient of x^{234}.

> $\mathit{Tokens}\left(234\right)$

$$2832 \tag{8.60}$$

For the second part, the case where the order does matter, the text explains that the generating function we need is

$$1 + \left(x + x^2 + x^5\right) + \left(x + x^2 + x^5\right)^2 + \cdots = \frac{1}{1 - (x + x^2 + x^5)}.$$

We can create this power series in Maple just like we did above.

> $\mathit{Tokens2} := \mathit{evalpow}\left(\frac{1}{1 - (x + x^2 + x^5)}\right)$

$$\mathit{Tokens2} := \mathbf{proc}\left(\mathit{powparm}\right) \ldots \mathbf{end\ proc} \tag{8.61}$$

> $\mathit{tpsform}\left(\mathit{Tokens2}, x, 8\right)$

$$1 + x + 2x^2 + 3x^3 + 5x^4 + 9x^5 + 15x^6 + 26x^7 + O\left(x^8\right) \tag{8.62}$$

We see that the coefficient of x^7 is 26, so there are 26 ways to pay for a \$7 item when order does matter.

8.5 Inclusion–Exclusion

In this section, we will apply the principle of inclusion and exclusion. At the heart of the principle of inclusion and exclusion is the formula

$$|A \cup B| = |A| + |B| - |A \cap B|,$$

which says that, for two finite sets A and B, the number of elements in the union $A \cup B$ of the two sets may be found by adding the sizes of A and B and then subtracting the number of elements common to both A and B, which would otherwise be counted twice. This formula can be generalized to count the number of elements in the union of any finite number of finite sets.

Recall that we define a set in Maple by enclosing the comma-separated list of elements in braces.

```
> A := {1, 2, 3}
    A := {1, 2, 3}                                                    (8.63)
```

To find the cardinality, or size of a set, we can use the command **nops** or **numelems**.

```
> nops(A)
    3                                                                 (8.64)

> numelems(A)
    3                                                                 (8.65)
```

The set operations of **union** and **intersect** are represented in Maple by writing out their names:

```
> X := {1, 2, 3, 4, 5} :
  Y := {4, 5, 6, 7, 8} :

> X union Y
    {1, 2, 3, 4, 5, 6, 7, 8}                                          (8.66)

> X intersect Y
    {4, 5}                                                            (8.67)
```

The set theoretic difference is computed by the Maple operator **minus**.

```
> X minus Y
    {1, 2, 3}                                                         (8.68)
```

Let us use the operations to verify the principle of inclusion and exclusion in a particular example.

```
> Flintstones := {"Fred", "Pebbles", "Wilma"}
    Flintstones := {"Fred", "Pebbles", "Wilma"}                       (8.69)

> Rubbles := {"BamBam", "Barney", "Betty"}
    Rubbles := {"BamBam", "Barney", "Betty"}                          (8.70)

> Husbands := {"Barney", "Fred"}
    Husbands := {"Barney", "Fred"}                                    (8.71)
```

```
> Wives := {"Betty", "Wilma"}
    Wives := {"Betty", "Wilma"}                                        (8.72)

> Kids := {"BamBam", "Pebbles"}
    Kids := {"BamBam", "Pebbles"}                                      (8.73)
```

If this were a complete census, then the number of children living in Bedrock would be

```
> nops(Kids)
    2                                                                  (8.74)
```

while the number of Bedrock inhabitants who are either Flintstones or children is

```
> nops(Flintstones union Kids)
    4                                                                  (8.75)
```

According to the principle of inclusion and exclusion, this number should be the same as

```
> nops(Flintstones) + nops(Kids) − nops(Flintstones intersect Kids)
    4                                                                  (8.76)
```

which, of course, it is.

As another example, consider the problem of determining the number of positive integers less than or equal to 100 that are not divisible by either 2 or 11. First, we generate the set of positive integers less than or equal to 100.

```
> hundred := {$1 ..100}
    hundred := {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
       21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
       41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
       61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
       81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100}     (8.77)
```

Next, we remove those elements that are divisible by 2:

```
> DivBy2 := hundred minus {seq(2 · i, i = 1 ..100)}
    DivBy2 := {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
       35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71,
       73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99}           (8.78)
```

and those that are divisible by 11:

```
> DivBy11 := hundred minus {seq(11 · i, i = 1 ..100)}
    DivBy11 := {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19,
       20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40,
       41, 42, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61,
       62, 63, 64, 65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 78, 79, 80, 81, 82,
       83, 84, 85, 86, 87, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 100}   (8.79)
```

(Note the combined use of the **minus** and **seq** operators; they work very conveniently together in this example.)

We are looking for integers that belong to either or both of A and B, that is, to their union, so we want the size of $A \cup B$, which is

```
> nops(DivBy2 union DivBy11)
    96                                                              (8.80)
```

According to the principle of inclusion and exclusion, this value could also be computed as

```
> nops(DivBy2) + nops(DivBy11) − nops(DivBy2 intersect DivBy11)
    96                                                              (8.81)
```

8.6 Applications of Inclusion–Exclusion

In this section, we will explore the following problem: Three sets of twins, Ashley and Amanda Abel; Brandon and Benjamin Bernoulli; and Christopher and Courtney Cartan (none of whom bear any relation to the mathematicians with the same surname), are to be seated in a row. List the ways in which they can be seated so that no person sits next to his or her twin.

The principle of inclusion–exclusion gives us insight into how we might accomplish this task. Rather than attempting to generate the seating arrangements in which no person sits next to his or her twin, it will be easier to consider all the possible arrangements of the twins and then exclude those that do not satisfy the condition. To begin, we define lists to store the names of the twins.

```
> Abels := ["Ashley", "Amanda"] :
  Bernoullis := ["Brandon", "Benjamin"] :
  Cartans := ["Christopher", "Courtney"] :
    Abels := ["Ashley", "Amanda"]
    Bernoullis := ["Brandon", "Benjamin"]
    Cartans := ["Christopher", "Courtney"]                          (8.82)
```

```
> twins := [Abels, Bernoullis, Cartans]
    twins := [["Ashley", "Amanda"], ["Brandon", "Benjamin"],
      ["Christopher", "Courtney"]]                                  (8.83)
```

We solve the problem using two procedures. The main procedure will consider each possible arrangement in turn and populate the list of those which satisfy the condition. The main procedure will rely on a second procedure to test whether an arrangement satisfies the condition of having no twins seated next to each other. We begin by writing the second procedure.

To test whether a given arrangement has a pair of twins seated next to one another, we will consider the five pairs of seats, 1 and 2, 2 and 3, ..., 5 and 6, and check to see if the people seated in those positions are twins.

```
testSeating := proc(seating::list)
  global twins;
  local i, twinpair;
```

```
    for i from 1 to 5 do
        for twinpair in twins do
            if ((seating[i] in twinpair) and (seating[i+1] in twinpair))
                then
                return false;
            end if;
        end do;
    end do;
    return true;
end proc:
```

The procedure is passed a list of the six people's names, representing a seating, so that the person in the third seat is **seating[3]**. The **for** loop indexed by **i** goes through the five pairs of seats. The inner **for** loop avoids having to duplicate the **if** statement. We could have written one **if** statement checking to see if the people in seats **i** and **i+1** are both Abels, and then a second if statement to see if they are both Bernoullis, and then a third to see if they are both Cartans. Instead, the loop sets the **twinpair** variable to each of the lists in **twins** in turn. Thus, **twinpair** represents, at each step in the loop, one of the families. And then the **if** statement checks to see whether the people in the seats **i** and **i+1** are members of that family, using the **in** operator. If any of these **if** statements are true, that the people in the pair of consecutive seats are from the same family, then the procedure immediately returns false, indicating that the seating is not acceptable. If the seating survives all of the **if** statements, then the procedure returns true.

To check the **TestSeating** procedure, consider the following potential seatings.

```
> seating1 := ["Ashley", "Amanda", "Brandon", "Benjamin",
    "Christopher", "Courtney"]
    seating1 := ["Ashley", "Amanda", "Brandon", "Benjamin",
        "Christopher", "Courtney"]                                   (8.84)

> seating2 := ["Ashley", "Brandon", "Christopher", "Amanda",
    "Benjamin", "Courtney"]
    seating2 := ["Ashley", "Brandon", "Christopher", "Amanda",
        "Benjamin", "Courtney"]                                      (8.85)
```

We see that the first seating fails but the second passes, as they should.

```
> testSeating (seating1)
    false                                                            (8.86)

> testSeating (seating2)
    true                                                             (8.87)
```

We now have a procedure to test a potential seating for the condition of not having twins seated next to each other. To generate a list of all of such seatings, we use Maple's **permute** command (from the **combinat** package) to generate all the possible permutations of the people and then test to see which are valid and which should be discarded. The **permute** command takes a list and returns all the possible permutations of the objects. (Refer to Chapter 6 of this manual for more information about the use of the commands in the **combinat** package.)

Note that the **permute** command expects a list of objects, so we will need a list of all the people. We use the **Flatten** command to turn our **twins** list into a list of all the names. **Flatten** takes a list and removes any nesting of lists so that the result is a list of the objects.

```
> ListTools[Flatten](twins)
    ["Ashley", "Amanda", "Brandon", "Benjamin", "Christopher",
      "Courtney"]                                                  (8.88)
```

Here is the procedure to generate all of the valid seating arrangements.

```
ListSeatings := proc()
   global twins;
   local possibles, seating, OKseatings;
   OKseatings := [];
   possibles := combinat[permute](ListTools[Flatten](twins));
   for seating in possibles do
      if testSeating(seating) then
         OKseatings := [op(OKseatings),seating];
      end if;
   end do;
   return OKseatings;
end proc:
```

This procedure initializes **OKseatings**, which is what will be returned, to the empty list, and it initializes the **possibles** variable to all the permutations of the six people. It then checks each possible permutation using the **testSeating** procedure, and the valid arrangements are added to **OKseatings**. Finally, the procedure returns **OKseatings**.

We run this procedure and store its output in the variable **twinSeatings** but suppress the output. Then, we use **nops** to check how many possible seatings there are. (It is generally a good idea to suppress the output of a procedure that is listing what may be a very large number of possibilities until you know how many there are. Otherwise you may have to wait a very long time for Maple to print all of them out.)

```
> twinSeatings := ListSeatings():
```

```
> nops(twinSeatings)
    240                                                            (8.89)
```

```
> twinSeatings[123]
    ["Benjamin", "Ashley", "Christopher", "Amanda", "Brandon",
      "Courtney"]                                                  (8.90)
```

We see that there are 240 possible seatings and have displayed the 123rd seating.

Solutions to Computer Projects and Computations and Explorations

Computer Projects 12

> Given positive integers m and n, find the number of onto functions from a set with m elements to a set with n elements.

Solution: We have a very convenient formula:

$$\sum_{k=0}^{n-1} (-1)^k \, C(n,k)\,(n-k)^m.$$

This is the number of onto functions from a set of m elements to a set of n elements, assuming $m \geq n$. This formula is derived in the textbook from the principle of inclusion–exclusion (see Theorem 1 of Section 8.6). The only input required in this formula are the integer parameters m and n, which represent the sizes of the domain and codomain, respectively. Maple has a command that corresponds to summations like the one above called **add**. The **add** command can take a few different forms, but the one similar to standard summation notation has two arguments: an expression in terms of an index of summation, for example, **k**, and an argument of the form **k=a..b** indicating the bounds of the summation. For example, to compute

$$\sum_{k=3}^{8} \frac{1}{k},$$

you would enter the following command.

> $add\left(\frac{1}{k}, k = 3\,..8\right)$

$$\frac{341}{280} \tag{8.91}$$

The **add** command will form the heart of our procedure, which will take the sizes of the domain and codomain as input, check to make sure that the assumption $m \geq n$ is satisfied, and then apply the formula.

```
OntoFunctions := proc(m::posint, n::posint)
   local k;
   if m < n then
      return 0;
   end if;
   return add((-1)^k * combinat[numbcomb](n,k) * (n-k)^m, k=0..n-1);
end proc:
```

The if statement is necessary—and makes sense mathematically—because there are no onto functions from a set to a larger set. For example:

> $OntoFunctions\,(4, 9)$

$$0 \tag{8.92}$$

As an example, we can use our function to compute the number of onto functions from a set with 100 elements to a set with 20 elements.

```
> OntoFunctions(100, 20)
    11 238 195 910 319 657 928 539 447 038 143 170 285 517 894 975 095
      769 496 294 319 007 413 091 913 959 828 334 936 464 196 298 192
      508 890 182 316 163 261 067 934 269 440 000                          (8.93)
```

Computations and Explorations 2

Find the smallest Fibonacci number greater than 1 000 000, greater than 1 000 000 000, and greater than 1 000 000 000 000.

Solution: We can solve this quite easily with Maple using a simple **while** loop. In this chapter, we have seen several ways to compute Fibonacci numbers, including the procedure **Fibonacci** in Section 8.1, the formula **Fibonacci2** in Section 8.2, and the generating function **Fibonacci3** in Section 8.4. For this exercise, we will use Maple's built-in function **fibonacci** in the **combinat** package. (Caution: there are other commands also called "fibonacci" in Maple, within the **StringTools** package and the (deprecated) **linalg** package.)

The idea is to compute Fibonacci numbers until the value exceeds the target. The **while** loop is well-suited to this sort of problem. We will create a procedure that takes the target values as input and prints out the desired Fibonacci number and its index.

```
FindFib := proc(target::integer)
   local n;
   n := 1;
   while combinat[fibonacci](n) < target do
      n := n + 1;
   end do;
   printf("The %dth Fibonacci number is %d", n, combinat[fibonacci](n));
end proc:
```

As long as the n th Fibonacci number is smaller than the target value, the index n is increased. Once the target has been exceeded, the **printf** statement displays the index and the value of the Fibonacci number.

The numbers called for by the question are:

```
> FindFib(1 000 000)
    The 31th Fibonacci number is 1 346 269
```

```
> FindFib(1 000 000 000)
    The 45th Fibonacci number is 1 134 903 170
```

```
> FindFib(1 000 000 000 000)
    The 60th Fibonacci number is 1 548 008 755 920
```

Computations and Explorations 3

Find as many prime Fibonacci numbers as you can. It is unknown whether there are infinitely many of these.

Solution: Using Maple, this sort of problem becomes fairly straightforward. We can simply use the Maple procedure **fibonacci** from the **combinat** package to generate Fibonacci numbers and use the **isprime** function to test each for primality. We will wrap this in a procedure so that we can use it in conjunction with the **timelimit** function. The number of Fibonacci numbers tested will depend on your computer and patience.

```
PrimeFib := proc()
   global primefibs;
   local i, temp;
   primefibs := NULL;
   for i from 1 do
      temp := combinat[fibonacci](i);
      if isprime(temp) then
         primefibs := primefibs, temp;
      end if;
   end do;
   return primefibs;
end proc:
```

This produces fairly large values relatively quickly, so we limit it to a tenth of a second.

```
> try
    timelimit(0.1, PrimeFib( ));
  catch "time expired" :
    [primefibs];
  end try;
    [2, 3, 5, 13, 89, 233, 1597, 28 657, 514229, 433 494 437, 2 971 215 073,
      99 194 853 094 755 497, 1066340417491710595814572169,
      19 134 702 400 093 278 081 449 423 917,
      475 420 437 734 698 220 747 368 027 166 749 382 927 701 417 016 557 193 662 268 716\
      376 935 476 241,
      529 892 711 006 095 621 792 039 556 787 784 670 197 112 759 029 534 506 620 905 162\
      834 769 955 134 424 689 676 262 369,
      1 387 277 127 804 783 827 114 186 103 186 246 392 258 450 358 171 783 690 079 918\
      032 136 025 225 954 602 593 712 568 353,
      3 061 719 992 484 545 030 554 313 848 083 717 208 111 285 432 353 738 497 131 674\
      799 321 571 238 149 015 933 442 805 665 949,
      10 597 999 265 301 490 732 599 643 671 505 003 412 515 860 435 409 421 932 560 009\
      680 142 974 347 195 483 140 293 254 396 195 769 876 129 909,
      36 684 474 316 080 978 061 473 613 646 275 630 451 100 586 901 195 229 815 270 242\
      868 417 768 061 193 560 857 904 335 017 879 540 515 228 143 777 781 065 869,
      96 041 200 618 922 553 823 942 883 360 924 865 026 104 917 411 877 067 816 822 264\
      789 029 014 378 308 478 864 192 589 084 185 254 331 637 646 183 008 074 629]
                                                                              (8.94)
```

Computations and Explorations 11

> Compute the probability that a permutation of n objects is a derangement for all positive integers not exceeding 20 and determine how quickly these probabilities approach the number $1/e$.

Solution: To solve this problem, we will make use of the formula which gives the number of derangements of n objects, namely,

$$D_n = n!\left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}\right].$$

The total number of permutations of n objects is, of course, $n!$, so the probability that one of them is a derangement is the ratio $\frac{D_n}{n!}$, which is given by the expression

$$1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}.$$

A very simple Maple function will compute these values for us.

```
DerProb := proc(n::posint)
   local k;
   return add((-1)^k * (1/k!),k=0..n);
end proc:
```

The probabilities that a permutation of n objects is a derangement for $n \leq 20$ are:

```
> seq(DerProb(n), n = 1..20)
```

$$0, \frac{1}{2}, \frac{1}{3}, \frac{3}{8}, \frac{11}{30}, \frac{53}{144}, \frac{103}{280}, \frac{2119}{5760}, \frac{16687}{45360}, \frac{16481}{44800}, \frac{1468457}{3991680}, \frac{16019531}{43545600}, \frac{63633137}{172972800}, \frac{2467007773}{6706022400}, \frac{34361893981}{93405312000}, \frac{15549624751}{42268262400}, \frac{8178130767479}{22230464256000}, \frac{138547156531409}{376610217984000}, \frac{92079694567171}{250298560512000}, \frac{4282366656425369}{11640679464960000} \tag{8.95}$$

To see how these probabilities differ from $1/e$, we will multiply them by e and subtract 1. To represent the number e in Maple, we use the **exp** command with argument 1, that is, e^1.

```
> seq(evalf[25](exp(1) · DerProb(n) − 1), n = 1..20)
```

$$-1.0, 0.3591409142295226176801 44, -0.09390605718031825487990 44, 0.01935568567214196326010 8, -0.00329666289835008036789 47, 0.00047872853006526023677 2, -0.00006061310256550270675 17, 6.804601513342661189\ 10^{-6}, -6.862544954179352489\ 10^{-7}, 6.2831105458124395\ 10^{-8}, -5.2675855306083001\ 10^{-9}, 4.07305385119424\ 10^{-10}, -2.92246853211696\ 10^{-11}, 1.956033996016\ 10^{-12}, -1.226806251301\ 10^{-13}, 7.239038692\ 10^{-15}, -4.032944743\ 10^{-16}, 2.127959000\ 10^{-17}, -1.066413000\ 10^{-18}, 5.08870000010^{-20} \tag{8.96}$$

(Note: the optional form **evalf[n](expr)** of **evalf** specifies that Maple should use **n** digits of precision when evaluating the expression **expr**. The default is 10 digits of precision.)

Exercises

Exercise 1. Implement a procedure to find the optimal schedule that maximizes total attendance.

Exercise 2. Implement a dynamic programming algorithm for finding the maximum sum of consecutive terms of a sequence of real numbers. (See Exercise 56 in Section 8.1.)

Exercise 3. Implement a dynamic programming algorithm for optimally computing matrix-chain multiplication. (See Exercise 57 in Section 8.1.)

Exercise 4. Use Maple to solve the following recurrence relations.

a) $a_n = a_{n-1} - a_{n-2}$, $a_1 = 1$, $a_2 = 1$;

b) $a_n = 15\,a_{n-1} + \frac{1}{2}\,a_{n-2}$, $a_1 = \frac{23}{22}$, $a_2 = \frac{7}{2}$.

Exercise 5. Use Maple to solve each of the recurrence relations in Exercise 1 in Section 8.2 of the textbook. (Solve even those that are *not* linear homogeneous recurrence relations with constant coefficients.)

Exercise 6. Write a general solver in Maple for linear homogeneous recurrence relations with constant coefficients of degree three with distinct roots. Your solver should check that the roots are in fact distinct and, if they are not, should return **FAIL**, which is the standard return value for a Maple function when it cannot complete a computation for some reason.

Exercise 7. Use Maple to investigate the behavior of the limit

$$\lim_{n\to\infty} \frac{\varphi_n}{\psi_n},$$

where φ_n is defined to be the number of prime Fibonacci numbers less than or equal to n, and ψ_n is defined to be the number of Fermat numbers less than or equal to n.

Exercise 8. Implement the recursive algorithm described in Example 12 of Section 8.3 of the text for solving the closest-pair problem.

Exercise 9. Use Maple to find the number of square-free integers less than 100 000 000.

Exercise 10. Use Maple to find the number of onto functions from a set with 1 000 000 elements to a set with 1000 elements.

Exercise 11. It is probably obvious that the number of onto functions from one set to another increases with the sizes of either the domain or the range. Using Maple to experiment, explore whether an increase in the size of the domain or the size of the range has the greater impact on the number of onto functions.

Exercise 12. To generate the *lucky numbers* start with the positive integers and delete every second integer in the list, starting the count with 1 (e.g., delete 2, 4, 6, etc., leaving 1, 3, 5, 7, ...). Other than 1, the smallest integer left is 3. Continue by deleting every third integer from those that remain, starting the counting with 1 (since 1, 3, 5, 7, 9, ... remain, 1 is the first number left, 3 is the second one left, 5 is the third left and gets deleted, and so on). Continue the process where at each stage, every kth integer is deleted, where k is the smallest integer left, other than the previous values of k. The integers that remain are the lucky numbers. Develop a Maple procedure that generates the lucky numbers up to n.

Exercise 13. Can you make any conjectures about lucky numbers by looking at a list of the first 1000 of them? For example, what sort of conjectures can you make about twin lucky numbers? What evidence do you have for your conjectures?

Exercise 14. Generalize the **ListSeatings** procedure to accept one argument, a list of lists (the same structure as the **twins** list), and determines the arrangements such that no two from the same sublist are seated next to one another.

Exercise 15. Further generalize the **ListSeatings** procedure so that it takes two arguments: a list of lists as before and a number n. The procedure should determine the arrangements of the people such that no n from the same sublist are seated together.