Generating Function Algorithms for Symbolic Computation

Robert Andrews Ravenscroft, Jr.
Ph.D. Dissertation

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-91-42
May, 1991

Generating Function Algorithms for Symbolic Computation

by
Robert Andrews Ravenscroft, Jr.
B. S., Trinity College, 1980
M. S., University of Connecticut, 1984
Sc. M., Brown University, 1986

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University.

May 1991

Vita

Robert Andrews Ravenscroft, Jr. was born 26 January, 1958 in New Milford, Connecticut. He grew up in Burlington, Connecticut where he was a 1976 graduate of Lewis S. Mills Regional High School. He graduated from Trinity College in Hartford, Connecticut in 1980 with a B. S. in Mathematics and Computer Coordinate with Mathematics.

Mr. Ravenscroft attended the graduate school at the University of Connecticut from 1980 to 1983. While there, he was a teaching assistant and lecturer in Computer Science. His master's thesis was entitled "Space-Time Considerations in the Implementation of Polymorphic Procedures and Data Types." He received his M. S. in Computer Science in 1984.

Mr. Ravenscroft entered the graduate school at Brown University in 1983. He received his Sc. M. in Computer Science in 1986. While at Brown, he served as a teaching assistant and research assistant. For two years, he taught Computer Science courses for the College of Continuing Education at the University of Rhode Island.

Upon completion of his graduate studies, Mr. Ravenscroft intends to seek employment in industry. His research interests include the design and analysis of algorithms, the design and implementation of programming languages, and the development of mathematical software. Besides his interest in Computer Science, Mr. Ravenscroft enjoys gardening, hiking, cooking, and zymurgy.

Abstract

Various algebraic models have been used to implement symbolic procedures for solving summations and recurrences. Surprisingly, one model that has received little attention in the development of symbolic algorithms is generating functions. Generating function methods for discrete mathematics provide an ad hoc collection of techniques for solving summations and recurrences. In this work we consider their systematic application to symbolic computation.

To demonstrate the feasibility and capabilities of generating functions as an algebraic model of computation, we develop and implement algorithms for manipulating generating functions and the sequences that they encode. In particular, we concentrate on algorithms for solving summations and recurrences. Techniques are developed to handle special functions and hybrid terms, which are terms involving two or more factors.

Given various classes of summations and recurrences we develop a theory to characterize the form of their solution. This allows us to demonstrate the ability of our algorithms to solve problems from these classes. This knowledge also assists us in developing algorithms that know where to look for a solution rather than doing an ad hoc search for a solution.

Acknowledgements

After many months I have finally reached this point. It is hard to believe that all that is left to this project is to thank everyone.

I had two advisors for this thesis. John Savage was the head of my thesis committee. I would like to thank him for his support and encouragement and for allowing me the opportunity to work with Ed Lamagna. John's suggestions and assistance were invaluable in helping me to finish. Ed Lamagna was my day-to-day advisor. I would like to thank him for agreeing to be my advisor, his guidance and encouragement, and especially for his thorough reading of this entire thesis. My gratitude also to Keith Geddes for his comments and feedback on this work.

The staff of the Computer Science Department deserves a big thanks, especially for dealing with all of the paper work needed to defend. Also, the efforts of Marge and Lorraine at at the University of Rhode Island Computer Science Department were greatly appreciated.

There are many friends and acquaintances I would like to mention. From my pre-Brown days, I would like to thank Suwathin Phiansunthon, Tom Grzybowski, Paniporn Phiansunthon, and Bruce Warren for their friendship, support, and encouragement over the years. During my years in Providence, I have had many wonderful and interesting housemates: Gautam Vemuri, Swaminathan Manohar, Urmilla Belliappa, Ajit Ranade, Alan Johnson, Ken Ward, Jasmine Jin, and John Stasko. In addition, I've had many wonderful friends in the Computer Science Department: Manohar, Kevin Brophy, Scott Oaks, John Stasko, Richard Hughey, and Cheryl Harkness.

Special thanks go to Manohar and Cheryl for eating all of those desserts, John for the years of sport prognostications, and Rich for the many Tex challenges and for his willingness to sample my homebrew efforts.

My gratitude to Manohar and his parents, brothers, and sister for making me a part of their family. Also, big hugs to Sathya and the trio: Adithya, Indu, and Yamini. My heartfelt thanks to Jasmine Jin, Alan Johnson, Richard Hughey and Cheryl Harkness. All have been wonderful friends through good and bad times. In particular, Cheryl's friendship and support helped me through many rough times and made the good times even better. Thanks to Alan for the many late night "feasts" that we have had over the years.

I am indebted to Pani for her confidence in my ability to finish this work. Her support and encouragement saw me through the many times when I felt that I was hopelessly bogged down with the writing of this thesis.

Finally, there is no way I can measure the contribution of my family. My mother, Dolores Ravenscroft; my father, Robert Ravenscroft; and my brother Craig Ravenscroft supported me in many ways over the many years it took me to finish. I dedicate this thesis to them.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Computer algebra systems provide an interactive environment to assist in the symbolic solution of many mathematical problems. Unlike most conventional programming languages, which produce results in numeric form, symbolic systems express answers using variable and function names. Such systems benefit from some of the advantages computers offer over hand calculations including increased computational speed, increased accuracy, and elimination of tedious computations. As a result, the user can avoid trivial algebra errors and get fast, reliable results.

MACSYMA, Maple, Mathematica, and Reduce are examples of currently available computer algebra systems.[1] Maple, in particular, has the ability to solve equations, perform integration and differentiation, find Taylor series for functions whose derivatives are known or can be found, and solve some types of summations. Symbolic algorithms for solving some of these problems are well developed. For example, as a result of Risch's method, the symbolic integration problem can be considered closed [BRONSTEIN, RISCH, TRAGER]. In contrast, symbolic algorithms for summations and recurrences are inadequate. They are limited in the problems that they can solve. In particular, they do not handle well special functions or hybrid terms, which are terms involving two or more factors.

Various algebraic models have been used to implement symbolic procedures for discrete mathematics. Surprisingly, one model that has received little attention in the development of symbolic algorithms is generating functions. Generating functions are continuous functions which encode a sequence of values [WILF]. They allow us to solve a wide variety of problems by manipulating a single continuous function rather than a possibly infinite sequence of values. Among their applications are enumerations in combinatorics, moment generating functions in probability, and summations and recurrences in discrete mathematics.

Generating function methods for discrete mathematics provide an ad hoc collection of techniques for solving summations and recurrences. Their typical applications are

[1] MACSYMA is a registered trademark of Symbolics, Inc. Maple is a registered trademark of the University of Waterloo. Mathematica is a registered trademark of Wolfram Research, Inc. Reduce is a registered trademark of Rand Corporation.

to sums involving polynomials and exponentials and linear recurrences with constant coefficients. In this work we will consider their systematic application to symbolic computation. We will develop and implement algorithms that manipulate generating functions and the sequences that they encode. In particular, we will concentrate on algorithms for solving summations and recurrences, especially those that involve special functions and hybrid terms.

1.1. Background and Existing Work

Various algorithms have been implemented to evaluate summations and recurrences. We will consider several types of sums and recurrences and the algorithms that can solve them.

1.1.1. Summation

The indefinite summation problem involves finding a closed form solution, s_n, to the equation

$$s_n = \sum_{k=0}^{n} a_k,$$

where a_k is a function of k but independent of n. This problem is equivalent to solving the difference equation

$$s_n - s_{n-1} = a_n.$$

Three general, algorithmic procedures for solving indefinite summations have been developed.

Karr's algorithm considers the problem as a difference equation and uses field theory to solve it [KARR]. From an initial field of constants, extension fields are built by the addition of symbols that behave like difference equation solutions. Given a problem and an extension field, the algorithm will determine when the sum is a function of the symbols in the field.

Moenck's method considers the summation problem for polynomial or rational function summands in a manner analogous to Risch's method of integration [MOENCK]. The sum is expressed as a function of a rational part and a transcendental part involving polygamma functions. These special functions involve logarithmic derivatives of the gamma function.

Gosper's procedure is based on hypergeometric functions and finds a solution if the ratio s_n/s_{n-1} is rational [GOSPER 77, GOSPER 78]. The linear difference equation is reduced by a change of variable to a system of linear equations. Their solution gives coefficients for a polynomial that yields a closed form solution for s_n. In some instances, this closed form may contain expressions that require iterative methods for evaluation.

Each of these procedures have limitations as to the types of sums they can solve. Karr's is limited by the field extensions that have been made, Moenck's by the form of the summand, and Gosper's by the type of the result. The run time complexity of Karr's algorithm increases with the number of field extensions. The iterative methods

used by Gosper's procedure can in some instances require exponential time. In general, none of these methods handle well summations involving special classes of functions.

One of these special classes of functions is the harmonic numbers,

$$H_n = \sum_{k=1}^{n} \frac{1}{k}.$$

Moenck's procedure when appropriately modified can produce this result. Gosper's algorithm will not since H_n/H_{n-1} is not a rational function. With the correct field extensions, this sum as well as sums involving H_k in the summand can be handled by Karr's procedure. A similar situation exists for sums involving the higher order harmonic numbers, $H_n^{(m)} = \sum_{k=1}^{n} 1/k^m$. Savio, Lamagna, and Liu have considered the problem of symbolic expressions for summations involving harmonic numbers as an extension to Moenck's algorithm [SAVIO].

Another group of special sums of interest are those containing the upper summation limit n in the summand. Such summations involve sequences defined as a function of two parameters. One parameter is held constant while summation occurs over the other parameter. Many sums involving binomial coefficients are good illustrations. Such problems are known as definite summations since they can be evaluated for only special values of the upper limit. For example, $\sum_{k=0}^{m} \binom{n}{k}$ cannot be evaluated for general m. Its value is known only when the upper limit $m \geq n$ or for certain specific values of m like $m = n/2$. Hayden and Lamagna have developed a procedure based on hypergeometric functions to evaluate definite summations involving binomial coefficients [HAYDEN 86].

Wilf has proposed a technique he calls the "Snake Oil" method for handling sums involving multivariate sequences [WILF]. This is one of the few approaches to symbolic summation that makes use of generating functions. Wilf uses generating functions to encode free variables in the summand. These are variables that are independent of the summation index or limit. The result is a sum involving generating functions. This sum is evaluated and the resulting generating function expanded to find the solution to the original sum. The generating function is not used to solve the sum, rather it is used in an attempt to simplify the sum by removing free variables. This method has particular application to sums involving binomial coefficients.

Fibonacci numbers and Stirling numbers are both special classes of functions that play an important role in analyzing algorithms. General procedures for summations involving these numbers do not exist. Hybrid sums, whose summands are products of values from two or more sequences, are another type of special summation not generally solvable by existing procedures.

Russell describes a method based on undetermined coefficients for solving hybrid sums that involve sequences defined by homogeneous linear recurrences [RUSSELL 79]. He defines a standard solution for such sums and proves that if you can solve for the constants in the solution, then you have found the solution to the sum.

1.1.2. Recurrence Relations

A recurrence relation, also known as a difference equation, is a function expressed in terms of itself at multiple values. With sufficient boundary conditions, it defines a sequence of values. When the unknown function occurs only to the first power, the equation is a linear recurrence relation. Recurrence relations can be categorized by the form of their equation.

A first order linear recurrence relation has the general form

$$t_n = f(n)t_{n-1} + a_n. \tag{1.1}$$

Each value of the sequence is defined in terms of its previous value. As noted before, any indefinite summation can be expressed in this way with $f(n) = 1$. The more general case of (1.1) is the extended linear first order recurrence relation, which has the form

$$t_n = f(n)t_{h(n)} + a_n. \tag{1.2}$$

The function $h(n)$ determines the index of the previous value in the sequence that is used to define t_n. Divide-and-conquer recurrences are examples of (1.2) with $h(n) = n/c$, where integer $c \geq 2$.

A linear recurrence relation of order k has the form

$$t_n = \sum_{j=1}^{k} f_j(n)t_{n-j} + a_n. \tag{1.3}$$

The value t_n is defined in terms of its previous k values. The function $f_j(n)$ is dependent not only on n, but j as well. The more general case of (1.3) is the extended linear recurrence of order k. Its form is

$$t_n = \sum_{j=1}^{k} f_j(n)t_{h^{[j]}(n)} + a_n, \tag{1.4}$$

where $h^{[j]}(n)$ is the jth composition of the function $h(n)$ with itself. That is, $h^{[j]}(n) = h(h(\cdots h(n)\cdots))$. Equation (1.3) is the special case of equation (1.4) with $h(n) = n-1$.

When t_n is defined in terms of all of its preceding values, the equation is a full history recurrence relation. Its general form is

$$t_n = \sum_{j=1}^{n} f_j(n)t_{n-j} + a_n. \tag{1.5}$$

We can create linear recurrences that do not look like any of the five types we have described. However, by defining $f_j(n) = 0$ for appropriate values of j we can categorize them as being of one of the forms given by equations (1.3), (1.4), and (1.5).

There are other types of recurrences besides the five examined so far. Systems of linear recurrence relations involve two or more sequences of numbers. The values of each sequence are defined in terms of previous values of that sequence and values of

the other sequences. Multivariate recurrence relations are sequences that are functions of more than one variable. For example, the binomial coefficients, $B_{n,k} = \binom{n}{k}$, satisfy the recurrence $B_{n,k} = B_{n-1,k} + B_{n-1,k-1}$. Nonlinear recurrence relations involve the unknown function raised to powers other than the first.

Algorithmic methods for solving recurrence relations are very limited. Cohen and Katcoff describe a system that uses a modified generating function approach to solve linear recurrence relations with constant coefficients of order four or less, first order linear recurrence relations with variable coefficients, and systems of linear recurrence relations with one index and constant coefficients [COHEN]. Karr's algorithm is based on first order linear difference equations and can handle any recurrence of this type, not only summations [KARR]. Hayden and Lamagna report using Gosper's algorithm to solve certain linear first order recurrences, systems of linear recurrences with constant coefficients, and divide-and-conquer recurrences [HAYDEN 88].

1.2. Goals and Overview

The objective of this work is to develop and analyze symbolic algorithms based on generating functions for solving summations and recurrences. To do so, we have several major goals.

- The design and implementation of such algorithms. We want to demonstrate the feasibility and capabilities of generating functions as an algebraic model for symbolic computation.
- The development of methods for summation and recurrence problems that involve hybrid terms and special functions. Existing algorithms cannot solve such problems. Developing methods for evaluating these problems will allow us to extend the class of problems that can be solved symbolically.
- Analysis of the capabilities of generating function based algorithms. We want to know how our algorithms perform relative to others. In particular, we want to know when generating function algorithms are better, when other methods are better, and what procedures are necessary in a comprehensive symbolic system for solving summations and recurrences.
- Development of a theory regarding solutions to summations and recurrences. Given various classes of summations and recurrences we want to characterize the form of their solution. This will allow us to demonstrate the ability of our algorithms to solve completely a class of problems. Also, by understanding how the problem class affects the form of the solution we can develop "intelligent" algorithms that know where to look for a solution rather than using an ad hoc search for the solution.

As we proceed, we will develop other additional theory and applications. While not directly related to our major goals, they none the less provide useful tools for other aspects of symbolic computation with generating functions.

Chapter 2 looks at generating functions and their application to the solution of summations and recurrences. Generating function methods are developed to handle sums and recurrences involving hybrid terms.

Chapter 3 presents an overview of the design of generating function based algorithms for solving sums and recurrences. Some steps of the algorithms can be implemented using well known techniques. The requirements for these steps are identified. Other steps necessitate the development of new algorithms for manipulating generating functions. The capabilities required by these algorithms are determined.

Chapter 4 considers the theory of rational generating functions. Algebraic structures are defined for these generating functions, the sequences that they encode, and the closed form expressions that define these sequences. We will later make use of these algebraic structures to prove the completeness of our algorithms. We also relate the sequences encoded with rational generating functions to the homogeneous linear recurrences that define them. As a result, we have numerous ways to determine if a sequence has a rational generating function, as well as having several ways to define such a sequence.

Chapter 5 explores the applications of rational generating functions. Properties of the sequences that they encode are examined. Methods to relate sequences with common factors in their generating functions are investigated. The solutions to sums and recurrences involving sequences encoded by rational generating functions are considered. Closure of these problems is demonstrated and the form of their solution is described. Finally, algorithms for encoding and expanding rational generating functions are presented along with other algorithms for symbolic summation with rational generating functions.

Chapter 6 examines multivariate rational generating functions. Among the sequences that they encode are the binomial coefficients. Summations involving binomial coefficients are studied and algorithmic methods for these sums considered.

Chapter 7 explores harmonic numbers. Generating function methods are presented for sums that involve harmonics numbers, products of harmonic numbers with polynomials and rational functions, and rational functions. An algebraic structure is developed that defines the set of rational functions which can be summed.

Chapter 8 discusses the implementation of the generating function based algorithms. The capabilities of the algorithms are analyzed. Complexity issues of the algorithms and the underlying algebra system are considered.

Chapter 9 looks at the new issues raised by this research and summarizes the results and contributions of this work.

Chapter 2

Generating Functions

Generating functions are functions that encode complete information about a sequence. They are an important mathematical tool as they permit the manipulation of a single value rather than an infinite sequence of values. Among their applications are enumerations in combinatorics, moment generating functions in probability, and the solution of summations and recurrences in discrete mathematics.

In this chapter we will look at generating functions, techniques for their manipulation, and their application to summations and recurrences. We will extend the theory of multivariate generating functions to handle the evaluation of some types of sums and recurrences that involve hybrid terms.

2.1. Generating Function Theory

A *sequence* is a function whose domain is the integers. Often, the domain is restricted to the nonnegative or the positive integers. We will be less restrictive and consider sequences whose domain includes all integers greater than or equal to some constant. Throughout this work, we will use the notation $\langle a_n \rangle$ to denote the sequence of values defined by a_n. In this case a_n serves as the name of the sequence. The index n indicates that the sequence is a function of n. Any limits on n define the domain of the sequence. In some cases a name will not be given for the sequence. Rather, an expression involving n will be given. In this case, the expression is the function defining the value of each term in the sequence. For example, $\langle n \rangle$ is the sequence of integers.

In some cases, the name of a sequence will be used without brackets to refer to the value of the sequence. If the subscript is a particular integer, such as a_5, we will be referring to the value of the sequence at that particular index. If the subscript is a parameter, such as a_n, then we will be referring to either the value of the nth term in the sequence or the expression defining the value of the nth term in the sequence.

The ordinary generating function for the sequence $\langle a_0, a_1, a_2, \ldots \rangle$ is

$$G(z) = \sum_{k=0}^{\infty} a_k \, z^k. \tag{2.1}$$

The exponential generating function for the same sequence is

$$H(z) = \sum_{k=0}^{\infty} a_k \frac{z^k}{k!}. \tag{2.2}$$

Multi-dimensional sequences are functions of two or more variables. Their values are encoded by super generating functions, which are found by applying (2.1) or (2.2) to each variable in the sequence. Thus the two-dimensional super generating function

$$G(w,z) \;=\; \sum_{n=0}^{\infty}\sum_{k=0}^{\infty} a_{n,k}\, w^k z^n \tag{2.3}$$

encodes the sequence $\langle a_{n,k}\rangle$ by applying (2.1) twice. Exponential generating functions could have been used to sum over either variable by applying (2.2) instead of (2.1).

Given an ordinary generating function $G(z)$, we need to be able to extract the sequence that it encodes. In general, we usually want to find an expression for a_n, the coefficient of z^n in $G(z)$. To obtain this value, consider the sequence found by taking the nth derivative of $G(z)$ with respect to z. This sequence is

$$\begin{aligned}\frac{d^n G(z)}{dz^n} &= \sum_{k=n}^{\infty} k^{\underline{n}}\, a_k\, z^{k-n} \\ &= n!\, a_n + \sum_{k=1}^{\infty} (n+k)^{\underline{n}}\, a_{n+k}\, z^k.\end{aligned}$$

The function $k^{\underline{n}}$ is the falling factorial function, $k^{\underline{n}} = k(k-1)\cdots(k-n+1)$. Evaluating this sequence at $z = 0$ we find that

$$\left.\frac{d^n G(z)}{dz^n}\right|_{z=0} \;=\; n!\, a_n$$

and therefore

$$a_n \;=\; \frac{1}{n!}\left.\frac{d^n G(z)}{dz^n}\right|_{z=0}. \tag{2.4}$$

It is informative to note that the series

$$G(z) \;=\; \sum_{n=0}^{\infty} a_n\, z^n \;=\; \sum_{n=0}^{\infty} \frac{G^{(n)}(0)\, z^n}{n!}$$

is the Taylor series expansion of $G(z)$ about $z = 0$. A similar approach can be used to extract the nth coefficient from the exponential generating function $H(z)$. Taking the nth derivative of $H(z)$ we find that

$$\begin{aligned}\frac{d^n H(z)}{dz^n} &= \sum_{k=n}^{\infty} \frac{k^{\underline{n}}\, a_k\, z^{k-n}}{k!} \\ &= a_n + \sum_{k=1}^{\infty} \frac{(n+k)^{\underline{n}}\, a_{n+k}\, z^k}{k!}.\end{aligned}$$

Evaluating this sequence at $z = 0$ immediately leads to

$$a_n = \left. \frac{d^n H(z)}{dz^n} \right|_{z=0} . \tag{2.5}$$

2.1.1. Manipulation of Generating Function

While of theoretical interest, equations (2.1) through (2.5) are not of practical use when attempting to find or expand generating functions. In many cases, evaluating the infinite series defined by (2.1) or (2.2) is difficult. For example, finding a solution for the series

$$H(z) = \sum_{k=0}^{\infty} k\, z^k$$

is not easy without the assistance of some additional mathematical tools. Likewise, while (2.4) and (2.5) can be used without difficulty to find the coefficient of the nth term of a generating function for a particular small value of n, it may not be easy to find the general nth term. Consider the generating function

$$H(z) = \frac{z}{(1-z)^2}.$$

Its nth derivative is

$$H^{(n)}(z) = \frac{n!\,(n+z)}{(1-z)^{n+2}}.$$

Deriving this expression required the examination of $H^{(n)}(z)$ for small values of n, a guess of the form of $H^{(n)}(z)$ for general n, and a proof of the form by induction. Having done all of this, we are able to determine that the generating function $H(z)$ encodes the sequence $a_n = n$. If we were unable to guess at the expression for $H^{(n)}(z)$ we would not have been able to find a_n.

Fortunately, there are many practical mathematical methods that can be used to assist in encoding and expanding generating functions. Combinatorial enumerations can be useful. For example, the function $1 + z$ represents membership of an item in a set. The coefficient of z^0 encodes the one case where the item is not in the set and the coefficient of z^1 encodes the one case where the item is in the set. If we have m distinct items, the coefficient of z^n in the generating function $(1 + z)^m$ encodes the number of ways that there could be n items in the set. This quantity is $\binom{m}{n}$, the number of ways to choose n out of m items, which we know from the binomial theorem is the coefficient of z^n in $(1 + z)^m$. Such an approach unfortunately requires deductive reasoning and is not generally applicable to algorithmic methods.

A more practical way to encode and expand generating functions is to use a set of generating function manipulation rules. A generating function manipulation is a mapping of one or more generating functions to a new generating function. The rules relate these generating function mappings to their corresponding sequence mappings.

Generating Function Relationship	Sequence Relationship	
$S(z) = \alpha A(z)$	$s_k = \alpha a_k$	(2.8)
$S(z) = z^m A(z)$	$s_k = a_{k-m}$	(2.9)
$S(z) = \alpha A(z) + \beta B(z)$	$s_k = \alpha a_k + \beta b_k$	(2.10)
$S(z) = A(\alpha z)$	$s_k = \alpha^k a_k$	(2.11)
$S(z) = A(z^m)$, integer $m \geq 2$	$s_k = \begin{cases} a_{k/m} & \text{if } k \bmod m = 0 \\ 0 & \text{otherwise} \end{cases}$	(2.12)
$S(z) = \frac{d}{dz}A(z)$	$s_k = (k+1)a_{k+1}$	(2.13)
$S(z) = \int_0^z A(t)dt$	$s_k = \frac{a_{k-1}}{k}$	(2.14)

Table 2.1. Ordinary Generating Function Manipulations

With them, we can use the construction of a generating function to determine the sequence that it encodes and we can find generating functions for new sequences that we build.

Two common power series used with generating function manipulations are

$$\frac{1}{1-z} = \sum_{k=0}^{\infty} z^k \tag{2.6}$$

and

$$e^z = \sum_{k=0}^{\infty} \frac{z^k}{k!}. \tag{2.7}$$

These are the ordinary generating function (2.1) and the exponential generating function (2.2) for the sequence $\langle a_k \rangle = \langle 1 \rangle$. Application of the binomial theorem,

$$(1+z)^m = \sum_{k=0}^{m} \binom{m}{k} z^k,$$

gives the generating function for the sequence of binomial coefficients $\langle a_k \rangle = \langle \binom{m}{k} \rangle$.

Table 2.1 lists several manipulations for ordinary generating functions. The left column expresses the ordinary generating function $S(z)$ as a function of the ordinary generating functions $A(z)$ and $B(z)$. The right column defines s_k, the kth term of $S(z)$, as a function of the sequences defined by $A(z)$ and $B(z)$. Each of these manipulations can be proved by using (2.1) to expand the generating functions and comparing coefficients of z^k.

We can use (2.8) to scale all terms of a sequence by a constant amount and (2.9) to shift the terms in a sequence m positions left or right. Addition of two sequences

is handled by (2.10). These three rules can be used to multiply a known generating function by any polynomial. For example, if the sequence $\langle a_k \rangle$ is encoded by the generating function $A(z)$ and the generating function of the sequence $\langle s_k \rangle$ is $S(z) = p(z)A(z)$, where $p(z)$ is the degree m polynomial

$$p(z) = \sum_{j=0}^{m} \beta_j z^j,$$

then we have that

$$S(z) = \sum_{j=0}^{m} \beta_j z^j A(z).$$

Applying (2.8), (2.9), and (2.10) we find that

$$S(z) = \sum_{n=0}^{\infty} \sum_{j=0}^{m} (\beta_j a_{n-j}) z^n$$

and so

$$s_k = \sum_{j=0}^{m} \beta_j a_{n-j}. \tag{2.15}$$

Thus, s_k is a linear combination of values from the sequence $\langle a_k \rangle$.

Rule (2.9) also allows to define sequences with domain $n \geq \gamma$, for any integer γ. For example, if $A(z)$ has domain $n \geq 0$, then $A(z)/z^\gamma$ for $\gamma > 0$ has domain $n \geq -\gamma$ and $A(z)/z^\gamma = \sum_{n=-\gamma}^{\infty} a_n z^n$.

Equation (2.11) is a change of variable rule. It is often used to bring the kth power of a constant into a sequence. For example, we can apply (2.11) to (2.6) to find that the generating function of the sequence $\langle \alpha^k \rangle$ is $1/(1 - \alpha z)$. One special application of (2.11) is (2.12). This rule generates a sequence whose nonzero elements occur at each position k that is a multiple of m.

Rules (2.13) and (2.14) are used to bring a factor of k or $1/k$ into a sequence. Differentiating (2.1) and multiplying by z gives the sequence

$$z\frac{d}{dz}A(z) = z\sum_{k=0}^{\infty} \frac{d}{dz} a_k z^k = z\sum_{k=0}^{\infty} (k+1) a_{k+1} z^k = \sum_{k=1}^{\infty} k a_k z^k. \tag{2.16}$$

Integrating (2.1) produces the sequence

$$\int_0^z B(t)dt = \sum_{k=0}^{\infty} \int_0^z b_k t^k dt = \sum_{k=0}^{\infty} \frac{b_k}{k+1} t^{k+1} \Big|_{t=0}^{z} = \sum_{k=0}^{\infty} \frac{b_k}{k+1} z^{k+1} = \sum_{k=1}^{\infty} \frac{b_{k-1}}{k} z^k.$$

In particular, applying (2.13) to the generating function (2.6) and multiplying by z gives

$$z\frac{d}{dz}\frac{1}{1-z} = \frac{z}{(1-z)^2} = z\sum_{k=0}^{\infty} \frac{d}{dz} z^k = \sum_{k=1}^{\infty} k z^k.$$

Applying rule (2.14) to (2.6) results in

$$\int_0^z \frac{1}{1-t}dt \;=\; \ln\left(\frac{1}{1-z}\right) \;=\; \sum_{k=0}^{\infty}\int_0^z t^k dt \;=\; \sum_{k=1}^{\infty}\frac{z^k}{k}.$$

These manipulations have given us the important generating functions for integers and their reciprocals.

2.1.2. Application of Generating Functions

Generating functions have application to the evaluation of summations and recurrence relations. In general, a summation or the particular solution to a recurrence relation is defined in terms of a sequence $\langle a_k \rangle$. The generating function of the solution to the summation or recurrence relation can then be expressed in terms of the generating function of the sequence $\langle a_k \rangle$. If the new generating function can be expanded, the problem has been solved.

Simple Sums and Convolutions.

Perhaps the most important application of generating functions to summation is the convolution [GRAHAM, WILF]. The convolution of the sequences $\langle a_k \rangle$ and $\langle b_k \rangle$ is defined as

$$s_n \;=\; \sum_{k=0}^{n} a_k b_{n-k}.$$

The sequence $\langle s_n \rangle$ is the coefficient of z^n in the generating function

$$S(z) \;=\; A(z)B(z), \tag{2.17}$$

where $A(z)$ is the generating function of $\langle a_k \rangle$ and $B(z)$ is the generating function of $\langle b_k \rangle$. This result is easily obtained by using (2.1) to find $S(z)$, the generating function that encodes s_n. Applying (2.1) we find that

$$\begin{aligned} S(z) &= \sum_{n=0}^{\infty} s_n z^n \\ &= \sum_{n=0}^{\infty}\sum_{k=0}^{n} a_k b_{n-k} z^n. \end{aligned}$$

Changing the order of summation and replacing n with $n+k$ gives

$$\begin{aligned} S(z) &= \sum_{k=0}^{\infty}\sum_{n=k}^{\infty} a_k b_{n-k} z^n \\ &= \sum_{k=0}^{\infty}\sum_{n=0}^{\infty} a_k b_n z^{n+k}. \end{aligned}$$

Rearranging terms and using (2.1) to find the generating functions of the sequences $\langle a_k \rangle$ and $\langle b_n \rangle$ results in

$$\begin{aligned} S(z) &= \sum_{k=0}^{\infty} a_k z^k \sum_{n=0}^{\infty} b_n z^n \\ &= A(z)B(z). \end{aligned}$$

A more intuitive proof of the convolution can be found by considering the product

$$S(z) = A(z)B(z) = \sum_{j=0}^{\infty} a_j z^j \sum_{k=0}^{\infty} b_k z^k. \tag{2.18}$$

The coefficient of z^n in (2.18) is simply the sum of all z^n terms resulting from the product of $A(z)$ and $B(z)$. Only the terms for $0 \le k \le n$ in $A(z)$ can contribute to this sum. Any other terms from $A(z)$ contribute too large an exponent when multiplied by a term from $B(z)$. We need to multiply each of these $n+1$ terms from $A(z)$ with an appropriate term from $B(z)$ and sum to get the desired result. Consider the general kth term from $A(z)$. Its value is $a_k z^k$. To get the z^n term of $S(z)$ we must multiply by the z^{n-k} term from $B(z)$. Summing over all valid values of k gives

$$s_n z^n = \sum_{k=0}^{n} a_k z^k b_{n-k} z^{n-k} = \sum_{k=0}^{n} a_k b_{n-k} z^n,$$

which, by definition, is the convolution.

Example 2.1. Consider the sum

$$s_n = \sum_{k=1}^{n} \alpha^{n-k} F_k,$$

where $\langle F_k \rangle$ is the sequence of Fibonacci numbers. Applying (2.11) to (2.1) we find that the generating function of $\langle \alpha^k \rangle$ is $1/(1-\alpha z)$. The generating function of $\langle F_k \rangle$ is $z/(1-z-z^2)$ [GRAHAM]. By (2.17) the generating function of $\langle s_n \rangle$ is

$$S(z) = \frac{z}{(1-\alpha z)(1-z-z^2)}.$$

Application of partial fractions with respect to z gives

$$S(z) = -\frac{\alpha^2 z}{(1+\alpha-\alpha^2)(1-\alpha z)} + \frac{1+\alpha+\alpha z}{(1+\alpha-\alpha^2)(1-z-z^2)}.$$

By (2.8) and (2.11), the coefficient of z^n in the first term is $-\alpha^{n+1}/(1+\alpha-\alpha^2)$. By (2.15), the coefficient of z^n in the second term is $(F_n + \alpha F_n + \alpha F_{n-1})/(1+\alpha-\alpha^2)$. Recognizing that $F_{n+1} = F_n + F_{n-1}$, we find that

$$s_n = \frac{-\alpha^{n+1} + F_n + \alpha F_{n+1}}{1+\alpha-\alpha^2}. \quad \blacksquare$$

The convolution can be generalized to m factors. The generating function of the convolution

$$s_n = \sum_{\alpha_1+\cdots+\alpha_m=n} a_{1,\alpha_1} + a_{2,\alpha_2} + \cdots + a_{m,\alpha_m}$$

is

$$S(z) = A_1(z)A_2(z)\cdots A_m(z), \tag{2.19}$$

where $A_i(z)$ is the generating function of the sequence $\langle a_{i,k}\rangle$.

Example 2.2. Consider the sum

$$s_n = \sum_{i+j+k=n} i\,\alpha^j F_k.$$

The generating function of the sequence $\langle i\rangle$ is, as we have seen, $z/(1-z^2)$. By (2.19), the generating function of $\langle s_n\rangle$ is

$$S(z) = \frac{z^2}{(1-z)^2(1-\alpha z)(1-z-z^2)}.$$

Applying partial fractions we find that

$$\begin{aligned} S(z) &= \frac{-2+3\alpha+z-2\alpha z}{(1-\alpha)^2(1-z)^2} + \frac{-\alpha^2}{(1-\alpha)^2(1+\alpha-\alpha^2)(1-\alpha z)} \\ &\quad + \frac{3\alpha+2+z+2\alpha z}{(1+\alpha-\alpha^2)(1-z-z^2)}. \end{aligned}$$

Expanding the generating functions we obtain

$$\begin{aligned} s_n &= \frac{(3\alpha-2)(n+1)+(1-2\alpha)n}{(1-\alpha)^2} - \frac{\alpha^{n+2}}{(1-\alpha)^2(1+\alpha-\alpha^2)} \\ &\quad + \frac{(3\alpha+2)F_{n+1}+(1+2\alpha)F_n}{1+\alpha-\alpha^2}. \quad \blacksquare \end{aligned}$$

A special case of the convolution results when $b_k = 1$. In that case we are attempting to evaluate the simple sum

$$s_n = \sum_{k=0}^{n} a_k.$$

From (2.6) we know that the generating function of the sequence $\langle b_k\rangle = \langle 1\rangle$ is $B(z) = 1/(1-z)$. Thus the generating function for s_n, the solution of the simple sum, is

$$S(z) = \sum_{n=0}^{\infty} s_n z^n = \sum_{n=0}^{\infty}\sum_{k=0}^{n} a_k z^n = \frac{1}{1-z}A(z). \tag{2.20}$$

The value of s_n can be found by expanding $S(z)$.

Example 2.3. Consider the sum

$$s_n = \sum_{k=0}^{n} \alpha^k.$$

By (2.20), the generating function of $\langle s_n \rangle$ is

$$S(z) = \frac{1}{(1-z)(1-\alpha z)}.$$

Applying partial fractions we get

$$S(z) = \frac{1}{(1-\alpha)(1-z)} - \frac{\alpha}{(1-\alpha)(1-\alpha z)}$$

Expanding $S(z)$, we find that the coefficient of z^n is

$$s_n = \frac{1}{1-\alpha} - \frac{\alpha^{n+1}}{1-\alpha} = \frac{1-\alpha^{n+1}}{1-\alpha}. \quad \blacksquare$$

Recurrence Relations

Generating functions can be used to solve several classes of recurrence relations. The generating function of the sequence is derived directly from the recurrence relation. Expanding the generating function gives the value of the sequence. The types of recurrence problems that can be solved this way include linear recurrences of order k with constant coefficients, systems of linear recurrences with constant coefficients, and multivariable linear recurrence relations with constant coefficients. The method is similar for all three cases. We will consider the linear recurrence case first. The methods for linear systems and multivariable recurrences are just extensions to this case.

Linear recurrences of order k with constant coefficients. The kth order linear recurrence with constant coefficients is given by

$$t_n = \sum_{i=1}^{k} \alpha_i t_{n-i} + a_n. \tag{2.21}$$

If $a_n = 0$ we have the homogeneous recurrence

$$t_n = \sum_{i=1}^{k} \alpha_i t_{n-i}. \tag{2.22}$$

We can define a_n to be the inhomogeneous part of the recurrence (2.21).

The general solution to (2.21) is

$$t_n^{(g)} = t_n^{(h)} + t_n^{(p)}.$$

The quantity $t_n^{(h)}$ is the solution to the homogeneous difference equation (2.22) and is defined in terms of k arbitrary constants. The quantity $t_n^{(p)}$ is a particular solution to the difference equation (2.21) defined without any arbitrary constants. The quantity $t_n^{(g)}$ defines all solutions to the recurrence relation (2.21). If k boundary conditions are specified, the recurrence is completely defined. We can then solve for the k arbitrary constants in $t_n^{(g)}$ and find t_n, the solution that satisfies the recurrence [LIU, PURDOM].

Example 2.4. Consider the first order recurrence relation

$$t_n = t_{n-1} + 1.$$

The homogeneous solution is $t_n^{(h)} = C$ and a particular solution is $t_n^{(p)} = n$. Thus the total general solution is given by $t_n^{(g)} = C + n$. If we are given the boundary condition $t_0 = 7$, the recurrence is completely defined, and its solution is $t_n = n + 7$ for $n \geq 0$. ∎

Given (1.3) and k boundary conditions it is possible to find directly a generating function for the sequence that they completely define. Expanding the generating function yields the solution to the recurrence. Since this approach takes into account the boundary conditions it is not necessary to find the general solution and solve for k arbitrary constants.

The generating function found by this method can be expressed as

$$T(z) = T^{(h)}(z) + T^{(p)}(z).$$

The generating function $T^{(h)}(z)$ encodes the homogeneous solution and takes into account the boundary conditions of the sequence. The generating function $T^{(p)}(z)$ encodes the particular solution that combines with $T^{(h)}(z)$ to give $T(z)$. Normally we would compute $T(z)$ and not consider the values of $T^{(h)}(z)$ and $T^{(p)}(z)$. However, when we consider recurrences involving hybrid terms we will need to evaluate each generating function separately.

Given the kth order linear recurrence with constant coefficients defined by (2.21) we want to find

$$T(z) = \sum_n t_n z^n. \tag{2.23}$$

The general approach is to multiply both sides of the recurrence by z^n and sum over n to get an equation involving $T(z)$. We can then solve for $T(z)$. However, we do have to take into account the k boundary conditions of the recurrence and the fact that the sequence is defined only for some $n \geq \gamma + 1$. Thus the recurrence is valid only for $n \geq \gamma + k + 1$. One way to do this is to introduce Kronecker delta terms in order to define the recurrence over all n [GRAHAM] We can then multiply by z^n and sum over all n to find a relation defining $T(z)$. Another approach, which we use here, multiplies the recurrence by z^n and sums over those values of n for which the recurrence is defined [KNUTH, PURDOM]. The resulting summations are then manipulated to find a relation defining $T(z)$.

Step 1. Multiply by z^n and sum over $n \geq \gamma + k + 1$. From (2.21) we get

$$\sum_{n=\gamma+k+1}^{\infty} t_n z^n = \sum_{n=\gamma+k+1}^{\infty} \sum_{i=1}^{k} \alpha_i t_{n-i} z^n + \sum_{n=\gamma+k+1}^{\infty} a_n z^n.$$

Step 2. Rearrange the order of summation, factor z^i out of the inner summation, and adjust the limits of summation.

$$\begin{aligned}
\sum_{n=\gamma+k+1}^{\infty} t_n z^n &= \sum_{n=\gamma+k+1}^{\infty} \sum_{i=1}^{k} \alpha_i t_{n-i} z^n + \sum_{n=\gamma+k+1}^{\infty} a_n z^n \\
&= \sum_{i=1}^{k} \alpha_i z^i \sum_{n=\gamma+k+1}^{\infty} t_{n-i} z^{n-i} + \sum_{n=\gamma+k+1}^{\infty} a_n z^n \\
&= \sum_{i=1}^{k} \alpha_i z^i \sum_{n=\gamma+k-i+1}^{\infty} t_n z^n + \sum_{n=\gamma+k+1}^{\infty} a_n z^n.
\end{aligned}$$

Step 3. Add and subtract terms to get the inner summation into the form (2.23) and the second term into the form of its generating function $A(z) = \sum_{n=\gamma_a}^{\infty} a_n z^n$,

$$\sum_{n=\gamma+1}^{\infty} t_n z^n - \sum_{n=\gamma+1}^{\gamma+k} t_n z^n = \sum_{i=1}^{k} \alpha_i z^i \left(\sum_{n=\gamma+1}^{\infty} t_n z^n - \sum_{n=\gamma+1}^{\gamma+k-i} t_n z^n \right) + \sum_{n=\gamma_a}^{\infty} a_n z^n - \sum_{n=\gamma_a}^{\gamma+k} a_n z^n.$$

Substituting for $T(z)$ and $A(z)$ and rearranging gives

$$\begin{aligned}
T(z) - \sum_{n=\gamma+1}^{\gamma+k} t_n z^n &= \sum_{i=1}^{k} \alpha_i z^i \left(T(z) - \sum_{n=\gamma+1}^{\gamma+k-i} t_n z^n \right) + A(z) - \sum_{n=\gamma_a}^{\gamma+k} a_n z^n \\
&= \sum_{i=1}^{k} \alpha_i z^i T(z) - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=\gamma+1}^{\gamma+k-i} t_n z^n + A(z) - \sum_{n=\gamma_a}^{\gamma+k} a_n z^n \\
&= \sum_{i=1}^{k} \alpha_i z^i T(z) - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} t_{\gamma+n} z^{\gamma+n} + A(z) - \sum_{n=\gamma_a}^{\gamma+k} a_n z^n.
\end{aligned}$$

Step 4. Collect terms and solve for $T(z)$.

$$T(z) - \sum_{i=1}^{k} \alpha_i z^i T(z) = \sum_{n=\gamma+1}^{\gamma+k} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} t_{\gamma+n} z^{\gamma+n} + A(z) - \sum_{n=\gamma_a}^{\gamma+k} a_n z^n.$$

Solving for $T(z)$, we find that

$$T(z) = \frac{\sum_{n=\gamma+1}^{\gamma+k} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} t_{\gamma+n} z^{\gamma+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i} + \frac{A(z) - \sum_{n \leq \gamma+k} a_n z^n}{1 - \sum_{i=1}^{k} \alpha_i z^i}. \quad (2.24)$$

Expanding $T(z)$ gives the value of the sequence $\langle t_n \rangle$ that satisfies (2.21) and its k boundary conditions. If we let $\langle a_n \rangle = \langle 0 \rangle$ we find that the generating function of the solution to the homogeneous recurrence is

$$T^{(h)}(z) = \frac{\sum_{n=\gamma+1}^{\gamma+k} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} t_{\gamma+n} z^{\gamma+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i}. \tag{2.25}$$

Thus the generating function of the particular solution is

$$T^{(p)}(z) = \frac{A(z) - \sum_{n=\gamma_a}^{\gamma+k} a_n z^n}{1 - \sum_{i=1}^{k} \alpha_i z^i}. \tag{2.26}$$

Example 2.5. Consider the recurrence

$$t_n = \begin{cases} 2t_{n-1} - t_{n-2} + 1, & \text{if } n \geq 2; \\ 2, & \text{if } n = 1; \\ 1, & \text{if } n = 0. \end{cases} \tag{2.27}$$

This recurrence is of order $k = 2$ and t_n is defined over $n \geq 0$. Thus, $\gamma = -1$ and the recurrence holds for $n \geq 2$.

Step 1. Multiply by z^n and sum over $n \geq \gamma + k + 1$. From (2.27) we get

$$\sum_{n=2}^{\infty} t_n z^n = \sum_{n=2}^{\infty} (2t_{n-1} - t_{n-2}) z^n + \sum_{n=2}^{\infty} z^n.$$

Step 2. Rearrange the order of summation, factor powers of z from each summand, and adjust the limits of summation.

$$\begin{aligned} \sum_{n=2}^{\infty} t_n z^n &= 2 \sum_{n=2}^{\infty} t_{n-1} z^n - \sum_{n=2}^{\infty} t_{n-2} z^n + \sum_{n=2}^{\infty} z^n \\ &= 2z \sum_{n=1}^{\infty} t_n z^n - z^2 \sum_{n=0}^{\infty} t_n z^n + \sum_{n=2}^{\infty} z^n \end{aligned}$$

Step 3. Add and subtract terms to get the last summation into the form of the generating function $\sum_{k=0}^{n} z^n$ and the other summations into the form (2.23),

$$\sum_{n=0}^{\infty} t_n z^n - (2z + 1) = 2z \left(\sum_{n=0}^{\infty} t_n z^n - 1 \right) - z^2 \sum_{n=0}^{\infty} t_n z^n + \sum_{n=0}^{\infty} z^n - (z + 1).$$

Substituting for $T(z)$ and $A(z)$ and rearranging gives

$$\begin{aligned} T(z) - (2z + 1) &= 2zT(z) - 2z - z^2 T(z) + A(z) - (z + 1) \\ &= (2z - z^2) T(z) - 2z + A(z) - z - 1. \end{aligned}$$

Step 4. Collect terms and solve for $T(z)$.

$$T(z) - \left(2z - z^2\right) T(z) = 2z + 1 - 2z + A(z) - z - 1.$$

Solving for $T(z)$ and substituting $A(z) = \sum_{k=0}^{n} z^k = 1/(1-z)$ we find that

$$T(z) = \frac{1}{1-2z+z^2} + \frac{A(z)-z-1}{1-2z-z^2} = \frac{1}{(1-z)^2} + \frac{z^2}{(1-z)^3}.$$

We notice that

$$T^{(h)}(z) = \frac{1}{(1-z)^2}.$$

By (2.13) this is the generating function for the sequence

$$t_n^{(h)} = n+1,$$

which satisfies the homogeneous recurrence $t_n = 2t_{n-1} - t_{n-2}$ for the boundary conditions $t_0 = 1$ and $t_1 = 2$. We also find that the generating function of the particular solution is

$$T^{(p)}(z) = \frac{z^2}{(1-z)^3}.$$

Applying (2.13) and (2.9) we find that this generating function encodes the sequence

$$t_n^{(p)} = \frac{n(n-1)}{2}.$$

This is a particular solution that satisfies the recurrence $t_n = 2t_{n-1} - t_{n-2} + 1$. Combining these two sequences, we find that the complete solution to (2.27) is

$$t_n = t_n^{(h)} + t_n^{(p)} = n+1+\frac{n(n-1)}{2} = \frac{n^2+n+2}{2}. \quad \blacksquare$$

Systems of linear recurrences with constant coefficients. Systems of linear recurrences are two or more sequences whose recursive definitions are mutually dependent. By applying the first three steps that we used with linear recurrences to each of the recurrences in the system we are able to find a linear system of generating functions. Solving the system yields the generating function of each sequence.

Example 2.6. Consider the system of recurrences

$$a_n = \begin{cases} a_{n-1} + b_n, & \text{if } n \geq 1; \\ 1, & \text{if } n = 0; \quad \text{and} \end{cases}$$

$$b_n = \begin{cases} a_{n-1} + b_{n-1}, & \text{if } n \geq 1; \\ 0, & \text{if } n = 0. \end{cases}$$

Both recurrences are of order 1 with their boundary condition at $n = 0$. Thus each recurrence holds over $n \geq 1$. We want to find

$$A(z) = \sum_{n=0}^{\infty} a_n z^n \quad \text{and} \quad B(z) = \sum_{n=0}^{\infty} b_n z^n.$$

Step 1. Multiply both recurrences by z^n and sum over $n \geq 1$.

$$\begin{aligned}\sum_{n=1}^{\infty} a_n z^n &= \sum_{n=1}^{\infty} a_{n-1} z^n + \sum_{n=1}^{\infty} b_n z^n. \\ \sum_{n=1}^{\infty} b_n z^n &= \sum_{n=1}^{\infty} a_{n-1} z^n + \sum_{n=1}^{\infty} b_{n-1} z^n.\end{aligned}$$

Step 2. Factor out z and adjust the limits of summation.

$$\begin{aligned}\sum_{n=1}^{\infty} a_n z^n &= z \sum_{n=0}^{\infty} a_n z^n + \sum_{n=1}^{\infty} b_n z^n. \\ \sum_{n=1}^{\infty} b_n z^n &= z \sum_{n=0}^{\infty} a_n z^n + z \sum_{n=0}^{\infty} b_n z^n.\end{aligned}$$

Step 3. Add and subtract terms to get the summations into the form of $A(z)$ or $B(z)$.

$$\begin{aligned}\sum_{n=0}^{\infty} a_n z^n - a_0 &= z \sum_{n=0}^{\infty} a_n z^n + \sum_{n=0}^{\infty} b_n z^n - b_0. \\ \sum_{n=0}^{\infty} b_n z^n - b_0 &= z \sum_{n=0}^{\infty} a_n z^n + z \sum_{n=0}^{\infty} b_n z^n.\end{aligned}$$

Recognizing $A(z)$ and $B(z)$ gives

$$\begin{aligned}A(z) - a_0 &= zA(z) + B(z) - b_0. \\ B(z) - b_0 &= zA(z) + zB(z).\end{aligned}$$

Step 4. Substitute for $a_0 = 1$ and $b_0 = 0$ and collect terms.

$$\begin{aligned}A(z) - zA(z) &= 1 + B(z). \\ B(z) - zB(z) &= zA(z).\end{aligned}$$

Solving for $A(z)$ and $B(z)$ we find that

$$A(z) = \frac{1 - z}{1 - 3z + z^2} \quad \text{and} \quad B(z) = \frac{z}{1 - 3z + z^2}.$$

These rather interesting generating functions encode the sequences $a_n = F_{2n+1}$ and $b_n = F_{2n}$ [GRAHAM]. ∎

Multivariate linear recurrences with constant coefficients. Multivariate sequences are functions of two or more parameters. They can be recursively defined in terms of previous values of each of their parameters. To find the generating function of an m variable sequence, we can choose one parameter and apply the the first three steps from above to that parameter. The result is a recurrence on $m - 1$ variables defining a generating function. We can repeat the process $m - 1$ times to encode each of the parameters. In the last iteration, we do step 4 and solve for the generating function.

Example 2.7. Consider the recurrence for the binomial coefficients.

$$b_{n,k} = \begin{cases} b_{n-1,k} + b_{n-1,k-1}, & \text{if } k > 0 \text{ and } n > 0; \\ 1, & \text{if } k = 0 \text{ and } n >= 0; \\ 0, & \text{if } k > 0 \text{ and } n = 0. \end{cases} \tag{2.28}$$

We will first use the variable w to find the generating function

$$B_n(w) = \sum_{k=0}^{\infty} b_{n,k} w^k.$$

The recurrence over k is of order 1 and is thus valid for all $k \geq 1$.

Step 1. Multiply by w^k and sum over $k \geq 1$.

$$\sum_{k=1}^{\infty} b_{n,k} w^k = \sum_{k=1}^{\infty} b_{n-1,k} w^k + \sum_{k=1}^{\infty} b_{n-1,k-1} w^k.$$

Step 2. Factor w out of the rightmost summation and adjust the limits.

$$\sum_{k=1}^{\infty} b_{n,k} w^k = \sum_{k=1}^{\infty} b_{n-1,k} w^k + w \sum_{k=0}^{\infty} b_{n-1,k} w^k.$$

Step 3. Add and subtract terms to get the summations into the form of $B_n(w)$.

$$\sum_{k=0}^{\infty} b_{n,k} w^k - b_{n,0} = \sum_{k=0}^{\infty} b_{n-1,k} w^k - b_{n-1,0} + w \sum_{k=0}^{\infty} b_{n-1,k} w^k.$$

Substituting for $B_n(w)$ gives

$$B_n(w) - b_{n,0} = B_{n-1}(w) - b_{n-1,0} + w B_{n-1}(w).$$

We can now repeat the process and find the generating function

$$B(w, z) = \sum_{n=0}^{\infty} B_n(w) z^n.$$

The recurrence over n is also of order 1 with its boundary condition at $n = 0$. Thus it is valid over all $n \geq 1$. For those values of n we can note that $b_{n,0} = b_{n-1,0} = 1$, which allows us to simplify the recurrence to $B_n(w) = (1 + w) B_{n-1}(w)$.

Step 1. Multiply by z^n and sum over $n \geq 1$.

$$\sum_{n=1}^{\infty} B_n(w) z^n = \sum_{n=0}^{\infty} (1 + w) B_{n-1}(w) z^n.$$

Step 2. Factor out z from the right hand summation and adjust the limits.

$$\sum_{n=1}^{\infty} B_n(w) z^n = z(1 + w) \sum_{n=0}^{\infty} B_n(w) z^n.$$

Step 3. Add and subtract terms to get the left hand summation into the form of $B(w,z)$.

$$\sum_{n=0}^{\infty} B_n(w)z^n - B_0(w) \;=\; z(1+w)\sum_{n=0}^{\infty} B_n(w)z^n.$$

We can recognize that $B_0(w) = \sum_{k=0}^{\infty} b_{0,k}w^k = 1$ and substitute for $B(w,z)$ to get

$$B(w,z) - 1 \;=\; z(1-w)B(w,z).$$

Step 4. Collect terms and solve for $B(w,z)$.

$$B(w,z) \;=\; \frac{1}{1-z-wz}. \quad \blacksquare$$

2.2. Hybrid Terms and Generating Functions

The generating function methods that we have examined required that we find the generating function $A(z)$ for a sequence $\langle a_k \rangle$ that appears in the summation or recurrence being solved. We then map $A(z)$ to the generating function of the solution $S(z)$. This approach is limited by our ability to find $A(z)$. One case where we may be unable to find $A(z)$ is for summations and recurrences that involve hybrid terms. As the name implies, hybrid terms are the product of values from two parent sequences. Given a problem involving hybrid terms from the sequence $\langle a_k b_k \rangle$, we would have to find the generating function

$$AB(z) \;=\; \sum_{k=0}^{\infty} a_k b_k z^k,$$

in order to use generating functions to find the solution.

In some cases we can evaluate $AB(z)$. The most obvious is the sequence $\langle a_k \alpha^k \rangle$, where we can apply (2.11). Unfortunately, there are many cases where we are unable to find a generating function for $\langle a_k b_k \rangle$. Other techniques must be used to attempt to solve such summations or recurrences. For example, methods such as summation by parts or Abel's transformation are often used to attack summations involving hybrid terms.

While there is no general way to encode the sequence $\langle a_k b_k \rangle$ in a generating function of one variable, we can encode the sequence $\langle c_{m,k} \rangle = \langle a_m \rangle \langle b_k \rangle$ in a generating function of two variables. Applying (2.3) we find that

$$C(w,z) \;=\; \sum_{k=0}^{\infty}\sum_{m=0}^{\infty} a_m w^m b_k z^k \;=\; \sum_{k=0}^{\infty} A(w)b_k z^k \;=\; A(w)B(z).$$

We now have that $\langle a_k b_k \rangle = \langle c_{k,k} \rangle$. Thus, $a_k b_k = c_{k,k}$ is the coefficient of $w^k z^k$ in $C(w,z)$.

Not all multivariate generating functions encode hybrid terms. To distinguish the different types of multivariate generating functions and the sequences that they encode, we will use the following terminology throughout this work. A *hybrid term* is a term

that is the product of values from two or more sequences. *Hybrid generating functions* are multivariable generating functions that can be expressed as the product of independent generating functions. The sequences that they encode are *hybrid sequences*. For example, $C(w,z) = A(w)B(z)$ is a hybrid generating function of two variables. The sequence that it encodes, $\langle c_{i,n} \rangle = \langle a_m b_n \rangle$ is a hybrid sequence. A *diagonal generating function* is a generating function of m variables whose coefficients are all zero except along the diagonal. An m variable diagonal generating function can be expressed as $F(z_1, \ldots, z_m) = A(z_1 \cdots z_m)$. The sequence that it encodes is given by

$$f_{\alpha_1,\alpha_2,\ldots,\alpha_m} = \begin{cases} a_k, & \text{if } \alpha_1 = \alpha_2 = \cdots = \alpha_m = k; \\ 0, & \text{otherwise.} \end{cases}$$

In the remainder of this section we will consider the application of hybrid generating functions. Our main tool will be the hybrid convolution, which we will use to solve summations and recurrences involving hybrid terms.

2.2.1. Hybrid Convolution

We wish to develop a method for convolutions that involve a sequence of hybrid terms $\langle a_k b_k \rangle$. This sequence is contained in the hybrid sequence $\langle t_{i,j} \rangle = \langle a_i \rangle \langle b_j \rangle$. The generating function of $\langle t_{i,j} \rangle$ is $T(w,z) = A(w)B(z)$, where $A(w)$ and $B(z)$ encode $\langle a_i \rangle$ and $\langle b_j \rangle$. If we cannot find a single variable generating function for $\langle a_k b_k \rangle$ we cannot apply (2.17) to solve the convolution. In an attempt to extend the convolution to a two variable sequence, we can consider convolving $\langle t_{i,j} \rangle$ with another bivariate sequence $\langle c_{i,j} \rangle$ encoded by the generating function $C(w,z)$. Applying the convolution to each variable, we find that the coefficient of $w^n z^n$ in $S(w,z) = C(w,z)T(w,z)$ is

$$s_{n,n} = \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j} t_{n-i,n-j}.$$

This sum has $(n+1)^2$ terms in it. We can see that $t_{n-i,n-i} = a_{n-i} b_{n-i}$ is a factor in these terms when $i = j$. Unfortunately, all terms for $i \neq j$ have undesired factors in them. These factors can be eliminated if $c_{i,j} = 0$, when $i \neq j$. This leads us to consider the convolution of a diagonal bivariate generating function with a bivariate generating function.

Theorem 2.1 (Bivariate Convolution). Let the generating function $T(w,z)$ encode the two-dimensional sequence $\langle t_{m,n} \rangle$ and let $C(wz)$ be the bivariate diagonal generating function that encodes the sequence $\langle c_k \rangle$. The coefficient of $w^m z^n$ in the expansion of

$$G(w,z) = T(w,z)C(wz) \tag{2.29}$$

is

$$g_{m,n} = \sum_{k=0}^{\min(m,n)} c_k t_{m-k,n-k}. \tag{2.30}$$

Proof. By algebraically manipulating (2.29), we have been able to show that the coefficient of $w^m z^n$ is (2.30). A more intuitive proof can be obtained by considering how the coefficient of $w^m z^n$ is obtained from the product of the generating functions $C(wz)$ and $T(w,z)$.

The expansion of the generating function $C(wz)$ is

$$C(wz) = \sum_{k=0}^{\infty} c_k w^k z^k, \tag{2.31}$$

and the expansion of $T(w,z)$ is

$$T(w,z) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} t_{i,j} w^i z^j. \tag{2.32}$$

The coefficient of $w^n z^n$ in (2.29) results from the sum of all $w^m z^n$ terms in the product of (2.31) and (2.32). Only the terms for $k = 0$ to $\min(m,n)$ in (2.31) can contribute to this sum. Any other terms from (2.31) when multiplied by a term from (2.32) will have too large an exponent. We need only multiply each of these $\min(m,n)+1$ terms from (2.31) with an appropriate term from (2.32) and sum to get the desired result.

Consider the general kth term in (2.31). Its value is $c_k w^k z^k$. To obtain an $w^m z^n$ term we must multiply it by the $w^{m-k} z^{n-k}$ term from (2.32). Summing each product over all valid k gives

$$\sum_{k=0}^{\min(m,n)} \left(c_k w^k z^k\right)\left(t_{m-k,n-k} w^{m-k} z^{n-k}\right) = \sum_{k=0}^{\min(m,n)} c_k t_{m-k,n-k} w^m z^n.$$

Thus, (2.30) is the coefficient of $w^m z^n$ in (2.29). □

Actually, all that we have done is to convolve the sequence $\langle c_k \rangle$ with the diagonals of the sequence $\langle t_{i,j} \rangle$ given by

$$\sum_{k=0}^{\infty} t_{m-\eta+k,n-\eta+k} w^{m-\eta+k} z^{n-\eta+k}, \quad \eta = \min(m,n).$$

The convolution of the sequence $\langle c_k \rangle$ with the sequence of hybrid terms $\langle a_j b_j \rangle$, occurs on the main diagonal given by $m = n$.

Corollary 2.1.1 (Hybrid Convolution). The solution to the hybrid convolution

$$s_n = \sum_{k=0}^{n} a_{n-k} b_{n-k} c_k \tag{2.33}$$

is the coefficient of $w^n z^n$ in the expansion of

$$G(w,z) = A(w)B(z)C(wz), \tag{2.34}$$

where $A(z)$, $B(z)$, and $C(z)$ are the ordinary generating functions of $\langle a_k \rangle$, $\langle b_k \rangle$, and $\langle c_k \rangle$.

Proof. Let the sequence $\langle t_{i,j} \rangle = \langle a_i \rangle \langle b_j \rangle$. Then, $T(w,z) = A(w)B(z)$. By Theorem 2.1, the coefficient of $w^m z^n$ in $G(w,z) = T(w,z)C(wz)$ is

$$g_{m,n} = \sum_{k=0}^{\min(m,n)} c_k t_{m-k,n-k} = \sum_{k=0}^{\min(m,n)} c_k a_{m-k} b_{n-k}.$$

Since $s_n = g_{n,n}$, we have that s_n is the coefficient of $w^n z^n$ in $G(w,z)$. □

To solve (2.33), we must expand (2.34) as a function of w and z to determine the coefficient of $w^n z^n$. Unfortunately, we cannot expand over one variable independent of the other due to the presence of $C(wz)$. This problem can be avoided if one of the generating functions is a rational function $B(z) = p(z)/q(z)$, where $p(z)$ and $q(z)$ are polynomials in z. This permits the use of partial fractions to transform (2.34) into a form that we can expand.

Example 2.8. Consider the convolution

$$s_n = \sum_{k=0}^{n} 2^{n-k} k F_k.$$

By Corollary 2.1.1, s_n is the coefficient of $w^n z^n$ in the expansion of

$$G(w,z) = \frac{1}{1-2wz} \frac{w}{(1-w)^2} \frac{z}{1-z-z^2}.$$

Applying partial fractions with respect to w we find that

$$G(w,z) = \frac{z}{1-z-z^2} \left(\frac{2z}{(1-2z)^2(1-2wz)} + \frac{w-2z}{(1-2z)^2(1-w)^2} \right).$$

If we consider the left hand term we see that

$$\begin{aligned} \frac{1}{1-z-z^2} &= \sum_{i=1}^{\infty} F_i z^i, \\ \frac{2z}{(1-2z)^2} &= \sum_{j=1}^{\infty} j 2^j z^j, \quad \text{and} \\ \frac{1}{1-2wz} &= \sum_{k=0}^{\infty} 2^k w^k z^k. \end{aligned}$$

The product of these generating functions is

$$\left(\sum_{i=1}^{\infty} F_i z^i \right) \left(\sum_{j=1}^{\infty} j 2^j z^j \right) \left(\sum_{k=0}^{\infty} 2^k w^k z^k \right). = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \sum_{k=0}^{\infty} F_i j 2^{j+k} w^k z^{i+j+k}.$$

We can see by inspection that the coefficient of $w^n z^n$ in this product occurs when $i = j = 0$. This coefficient is 0. We can find the coefficient of w^n in the right hand

term. The generating function of $1/(1-w)^2$ encodes the sequence $\langle n+1 \rangle$. By (2.15), the coefficient of w^n in the right hand term of $G(w,z)$ is

$$\frac{z}{1-z-z^2}\frac{n-2z(n+1)}{(1-2z)^2}.$$

Applying partial fractions with respect to z gives

$$\frac{z}{1-z-z^2}\frac{n-2z(n+1)}{(1-2z)^2} = \frac{-4(n+6)z+2(n+5)}{(1-2z)^2} - \frac{(n+6)z+2(n+5)}{1-z-z^2}.$$

The generating function $1/(1-2z)^2$ encodes the sequence $\langle (n+1)2^n \rangle$ and $1/(1-z-z^2)$ encodes the sequence $\langle F_{n+1} \rangle$. Applying (2.15) we find that

$$\begin{aligned} s_n &= -4(n+6)n2^{n-1} + 2(n+5)(n+1)2^n - (n+6)F_n - 2(n+5)F(n+1) \\ &= 2^{n+1}(n^2+6n+5-n^2-6n) - (n+6)F_n - 2(n+5)F(n+1) \\ &= 5\,2^{n+1} - (n+6)F_n - 2(n+5)F(n+1). \quad \blacksquare \end{aligned}$$

Just as we were able to apply the convolution to simple sums, we can apply Corollary 2.1.1 to sums involving hybrid terms [RAVENSCROFT].

Corollary 2.1.2 (Hybrid Sum Theorem). The solution to the hybrid sum

$$s_n = \sum_{k=0}^{n} a_k b_k$$

is the coefficient of $w^n z^n$ in the expansion of

$$G(w,z) = \frac{1}{1-wz} A(w)B(z),$$

where $A(z)$ and $B(z)$ are the ordinary generating functions of $\langle a_k \rangle$ and $\langle b_k \rangle$.

Proof. This sum is a hybrid convolution between the hybrid term $a_k b_k$ and the sequence $\langle c_k \rangle = \langle 1 \rangle$. The generating function of the sequence $\langle 1 \rangle$ is $C(y) = 1/(1-y)$. By Corollary 2.1.1, s_n is the coefficient of $w^n z^n$ in the expansion of

$$G(w,z) = A(w)B(z)C(wz) = \frac{1}{1-wz} A(w)B(z). \quad \square$$

We will make frequent use of this corollary. To illustrate its power, we examine its application to a general class of problems.

Example 2.9. Consider the hybrid sum

$$s_n = \sum_{k=0}^{n} c_k b_k,$$

where the sequence $\langle b_k \rangle$ has a generating function of the form $B(z) = D(z)/(1-z)$. This implies that $b_n = \sum_{k=0}^{n} d_k$ and that $d_n = b_n - b_{n-1}$. Let $C(w)$ be the generating function of the sequence $\langle c_k \rangle$. By Corollary 2.1.2, s_n is the coefficient of $w^n z^n$ in

$$G(w,z) = \frac{1}{1-wz} C(w)B(z) = \frac{1}{1-wz} C(w)\frac{D(z)}{1-z}.$$

To find the coefficient of $w^n z^n$, we can apply partial fractions with respect to z to get

$$G(w,z) = \frac{C(w)D(z)}{(1-w)(1-z)} - \frac{wC(w)D(z)}{(1-w)(1-wz)}.$$

Let $A(w) = C(w)/(1-w)$. This implies that $a_n = \sum_{k=0}^{n} c_k$, $a_0 = c_0$, and $c_n = a_n - a_{n-1}$ for $n \geq 1$. Expanding the left hand term, we find that the coefficient of $w^n z^n$ is $a_n b_n$. The right hand term is the generating function of a hybrid sum. The generating function $wC(w)/(1-w)$ encodes the sequence $\langle a_{k-1} \rangle$ and the generating function $D(z)$ encodes the sequence $\langle d_k \rangle$. Since a_k is not defined for $k \leq 0$, the right hand term encodes the summation

$$\sum_{k=1}^{n} a_{k-1} d_k.$$

Taking the coefficient of $w^n z^n$ in $G(w,z)$, we get the identity

$$\sum_{k=0}^{n} c_k b_k = a_n b_n - \sum_{k=0}^{n} a_{k-1} d_k.$$

Substituting for c_k and d_k we get

$$a_0 b_0 + \sum_{k=1}^{n} (a_k - a_{k-1}) b_k = a_n b_n - \sum_{k=1}^{n} a_{k-1}(b_k - b_{k-1}).$$

This is the well known summation by parts transformation. Based on this derivation, we can observe that summations involving a sequence whose generating function has a factor of $1/(1-z)$ are good candidates for the transformation. ∎

2.2.2. Multifactor Hybrid Terms

We do not have to limit ourselves to solving problems that involve hybrid terms with only two factors. We can extend our results to m factor hybrid terms by using m factor hybrid generating functions.

Theorem 2.2 (Multivariate Convolution). Let $T(z_1, \ldots, z_m)$ be the m-variate generating function that encodes the m-dimensional sequence $\langle t_{\alpha_1,\ldots,\alpha_m} \rangle$ and let $C(z_1 \cdots z_m)$ be an m-variate diagonal generating function that encodes the sequence $\langle c_k \rangle$. The coefficient of $(z_1 \cdots z_m)^n$ in the expansion of

$$G(z_1, \ldots, z_m) = T(z_1, \ldots, z_m) C(z_1 \cdots z_m)$$

is

$$g_{\alpha_1,\ldots,\alpha_m} = \sum_{k=0}^{\min(\alpha_1,\ldots,\alpha_m)} c_k t_{\alpha_1-k,\ldots,\alpha_m-k}.$$

Proof. The proof follows that for Theorem 2.1. We match each term from $C(z_1 \cdots z_m)$ for $k = 0$ to $\min(\alpha_1, \ldots, \alpha_m)$ with a term from $T(z_1, \ldots, z_m)$. These are the only terms which can contribute to the coefficient of $(z_1 \cdots z_m)^n$ in $G(z_1, \ldots, z_m)$. The sum of these terms is $g_{\alpha_1,\ldots,\alpha_m}$. □

As a direct result of this theorem, we can evaluate convolutions and summations involving hybrid terms with m factors.

Corollary 2.2.1 (Multifactor Hybrid Convolution). The solution to the multifactor hybrid convolution

$$s_n = \sum_{k=0}^{n} c_k \prod_{i=1}^{m} a_{i,n-k}$$

is the coefficient of $(z_1 \cdots z_m)^n$ in the expansion of

$$G(z_1, \ldots, z_m) = C(z_1 \cdots z_m) \prod_{i=1}^{m} A_i(z_i),$$

where $A_i(z_i)$ is the ordinary generating function of $\langle a_{i,k} \rangle$ and $C(z_1 \cdots z_m)$ is the diagonal generating function of $\langle c_k \rangle$.

Corollary 2.2.2 (Multifactor Hybrid Sum). The solution to the hybrid sum

$$s_n = \sum_{k=0}^{n} \prod_{i=1}^{m} a_{i,k}$$

is the coefficient of $(z_1 \cdots z_m)^n$ in the expansion of

$$G(z_1, \ldots, z_m) = \frac{1}{1 - \prod_{i=1}^{m} z_i} \prod_{i=1}^{m} A_i(z_i),$$

where $A_i(z_i)$ is the ordinary generating function of $\langle a_{i,k} \rangle$.

Example 2.10. Consider the sum

$$s_n = \sum_{k=0}^{n} 2^k k F_k.$$

By Corollary 2.2.2, s_n is the coefficient of $x^n y^n z^n$ in

$$G(x, y, z) = \frac{yz}{(1 - xyz)(1 - 2x)(1 - y)^2(1 - z - z^2)}.$$

Application of partial fractions with respect to x gives

$$G(x,y,z) = \frac{yz}{(1-y)^2(1-z-z^2)}\left(\frac{-yz}{(2-yz)(1-xyz)} + \frac{2}{(2-yz)(1-2x)}\right).$$

The coefficient of $x^n y^n z^n$ in the left hand term is 0. Expanding the right hand term as a function of x and applying partial fractions with respect to y we find that

$$\frac{2^{n+1}yz}{(2-yz)(1-y)^2(1-z-z^2)} = \frac{2^{n-1}yz}{(1-z-z^2)}\left(\frac{z^2}{(2-z)^2(2-yz)} + \frac{yz-2z+2}{(2-z)^2(1-y)^2}\right).$$

Once again we can discard the left hand term. We can expand the right hand term as a function of y and use partial fractions with respect to z to obtain

$$\begin{aligned}&\frac{2^{n+1}z\left((n-1)z-2nz+2n\right)}{(2-z)^2(1-z-z^2)}\\ &\quad = 2^{n+1}z\left(2n-(n+1)z\right)\left(\frac{2+z}{5(1-z-z^2)} + \frac{z-3}{5(2-z)^2}\right).\end{aligned}$$

The generating function $1/(2-z)^2$ encodes the sequence $\langle (n+1)/2^{n+2}\rangle$. Thus, the coefficient of z^n is

$$\begin{aligned}s_n &= \frac{2^{n+1}}{5}(4nF_n - 2(n+1)F_{n-1} + 2nF_{n-1} - (n+1)F_{n-2})\\ &\quad + \frac{2^{n+1}}{5}\left(\frac{2n^2}{2^{n+1}} - \frac{(n+1)(n-1)}{2^n} - \frac{6n(n+1)}{2^{n+2}} + \frac{3(n+1)n}{2^{n+1}}\right)\\ &= \frac{2^{n+1}((3n-1)F_n + (n-1)F_{n-1})}{5} + \frac{2n^2 - 2(n^2-1) - 3n(n+1) + 3(n+1)n}{5}\\ &= \frac{2^{n+1}((3n-1)F_n + (n-1)F_{n-1}) + 2}{5}. \quad \blacksquare\end{aligned}$$

2.2.3. Linear Recurrences and Hybrid Terms

One important application of the convolution was made when finding the generating function of the particular solution (2.26) for linear recurrences with constant coefficients. The multivariate convolution has similar application to linear recurrences involving hybrid terms.

Theorem 2.3. Let t_n be the first order linear recurrence with constant coefficients of order k given by

$$t_n = \sum_{i=1}^{k} \alpha_i t_{n-i} + \prod_{i=1}^{m} a_{i,n} \tag{2.35}$$

and k boundary conditions for $\gamma+1 \le n \le \gamma+k$. Let $A_i(z)$ be the ordinary generating function of the sequence $\langle a_{i,j}\rangle$ for $1 \le i \le m$. The solution of the recurrence is $t_n = t_n^{(h)} + t_n^{(p)}$, where $t_n^{(h)}$ is the coefficient of z^n in

$$T^{(h)}(z) = \frac{\sum_{j=\gamma+1}^{\gamma+k} t_j z^j - \sum_{i=1}^{k} \alpha_i z^i \sum_{j=1}^{k-i} t_{\gamma+j} z^{\gamma+j}}{1 - \sum_{i=1}^{k} \alpha_i z^i}$$

and $t_n^{(p)}$ is the coefficient of $(z_1 \cdots z_m)^n$ in

$$T^{(p)}(z_1, \ldots, z_m) = \frac{\prod_{i=1}^m A_i(z_i)}{1 - \sum_{i=1}^k \alpha_i (z_1 \cdots z_m)^i} - \frac{\sum_{j=\gamma_a}^{\gamma+k} \prod_{i=1}^m a_{i,j} z_i^j}{1 - \sum_{i=1}^k \alpha_i (z_1 \cdots z_m)^i},$$

where $\prod_{i=1}^m a_{i,j}$ is defined for $j \geq \gamma_a$.

Proof. The solution to the homogeneous recurrence $t_n = \sum_{i=1}^k \alpha_i t_{n-i}$ does not involve the hybrid term $a_{1,n} \cdots a_{m,n}$. Thus its generating function is given by (2.25). If we let $a_n = \prod_{i=1}^m a_{i,n}$ we find from (2.26) that the generating function of the particular solution to the recurrence $t_n = \sum_{i=1}^k \alpha_i t_{n-i} + a_n$ is

$$T^{(p)}(z) = \frac{A(z)}{1 - \sum_{i=1}^k \alpha_i z^i} - \frac{\sum_{j=\gamma_a}^{\gamma+k} a_j z^j}{1 - \sum_{i=1}^k \alpha_i z^i},$$

where $A(z) = \sum_{n=0}^{\gamma_a} a_n z^n$. Let c_n be the coefficient of z^n in the expansion of $1/(1 - \sum_{i=1}^k \alpha_i z^i)$. The first term in $T^{(p)}(z)$ is a convolution between the sequences $\langle c_n \rangle$ and $\langle a_n \rangle$. Its value is given by

$$\sum_{j=0}^n c_{n-j} a_j = \sum_{j=0}^n c_{n-j} \prod_{i=1}^m a_{i,j}.$$

By Corollary 2.2.1, the value of this hybrid convolution is the coefficient of $(z_1 \cdots z_m)^n$ in

$$\frac{\prod_{i=1}^m A_i(z_i)}{1 - \sum_{i=1}^k \alpha_i (z_1 \cdots z_m)^i}.$$

The second term in $T^{(p)}(z)$ is the product of the polynomial $\sum_{j=\gamma_a}^{\gamma+k} a_j z^j$ and the generating function $1/(1 - \sum_{i=1}^k \alpha_i z^i)$. It encodes the value

$$\sum_{j=\gamma_a}^{\gamma+k} a_j c_{n-j} = \sum_{j \leq \gamma+k} c_{n-j} \prod_{i=1}^m a_{i,j}.$$

We can encode this sequence in the diagonal generating function

$$\frac{\sum_{j=\gamma_a}^{\gamma+k} \prod_{i=1}^m a_{i,j} z_i^j}{1 - \sum_{i=1}^k \alpha_i (z_1 \cdots z_m)^i}.$$

Thus, we find that the particular solution to (2.35) is the coefficient of $(z_1 \cdots z_m)^n$ in

$$T^{(p)}(z_1, \ldots, z_m) = \frac{\prod_{i=1}^m A_i(z_i)}{1 - \sum_{i=1}^k \alpha_i (z_1 \cdots z_m)^i} - \frac{\sum_{j \leq \gamma+k} \prod_{i=1}^m a_{i,j} z_i^j}{1 - \sum_{i=1}^k \alpha_i (z_1 \cdots z_m)^i}. \quad \square$$

Example 2.11. Consider the linear recurrence

$$t_n = \begin{cases} 2t_{n-1} + nF_n, & \text{if } n \geq 1; \\ 0 & \text{if } n = 0. \end{cases}$$

We want to find the generating function $T(z) = \sum_{n=0}^{\infty} t_n z^n$ and then expand to determine t_n. The recurrence holds for $n \geq 1$. Multiplying by z^n and summing over $n \geq 1$ gives

$$\sum_{n=1}^{\infty} t_n z^n = \sum_{n=1}^{\infty} 2t_{n-1} z^n + \sum_{n=1}^{\infty} nF_n z^n = 2z \sum_{n=1}^{\infty} t_{n-1} z^{n-1} + \sum_{n=1}^{\infty} nF_n z^n.$$

Recognizing $T(z)$ we get

$$T(z) - t_0 = 2zT(z) + \sum_{n=1}^{\infty} nF_n z^n.$$

Substituting $t_0 = 0$ and solving for $T(z)$ we find that

$$T(z) = \frac{1}{1-2z} \sum_{n=1}^{\infty} nF_n z^n.$$

The homogeneous solution to t_n is 0. The particular solution is the convolution encoded by $T(z)$. The value of this convolution is

$$\sum_{k=0}^{n} 2^{n-k} k F_k,$$

which we evaluated using the hybrid convolution in Example 2.8. ∎

2.2.4. Multifactor Convolutions with Hybrid Terms

The hybrid convolutions we have considered so far have involved a hybrid term and one other function. As was done with the convolution, the hybrid convolution can be extended to the product of a hybrid term and more than one additional factor.

Theorem 2.4. Let $T(z_1, \ldots, z_m)$ be the m-variate generating function that encodes the m-dimensional sequence $\langle t_{\alpha_1,\ldots,\alpha_m} \rangle$ and let $C_i(z)$ be the ordinary generating function encoding the sequence $\langle c_{i,k} \rangle$ for $0 \leq i \leq p$. The multifactor hybrid convolution

$$g_{\alpha_1,\ldots,\alpha_m} = \sum_{\beta_1+\ldots+\beta_p+k=\eta} t_{\alpha_1-\eta+k,\ldots,\alpha_m-\eta+k} \prod_{i=1}^{p} c_{i,\beta_i}, \quad \eta = \min(\alpha_1, \ldots, \alpha_m),$$

is the coefficient of $(z_1 \cdots z_m)^n$ in the expansion of

$$G(z_1, \ldots, z_m) = T(z_1, \ldots, z_m) \prod_{i=1}^{p} C_i(z_1 \cdots z_m).$$

Proof. We can express the sum $g_{\alpha_1,\ldots,\alpha_m}$ as

$$g_{\alpha_1,\ldots,\alpha_m} = \sum_{k=0}^{\eta} t_{\alpha_1-k,\ldots,\alpha_m-k} \sum_{\beta_1+\ldots+\beta_p=k} \prod_{i=1}^{p} c_{i,\beta_i}, \quad \eta = \min(\alpha_1,\ldots,\alpha_m).$$

By (2.17), the inner sum is the coefficient of z^n in $C(z) = C_1(z)\cdots C_p(z)$. If we let c_k be this quantity, we get

$$g_{\alpha_1,\ldots,\alpha_m} = \sum_{k=0}^{\eta} t_{\alpha_1-k,\ldots,\alpha_m-k}\, c_k, \quad \eta = \min(\alpha_1,\ldots,\alpha_m).$$

By Theorem 2.2, this is the coefficient of $(z_1\cdots z_m)^n$ in

$$G(z_1,\ldots,z_m) = T(z_1,\ldots,z_m)C(z_1\cdots z_m) = T(z_1,\ldots,z_m)\prod_{i=1}^{p} C_i(z_1\cdots z_m). \quad \square$$

Example 2.12. Consider the sum

$$s_n = \sum_{i+j+k=n} i2^j k F_k.$$

By Theorem 2.4, s_n is the coefficient of $w^n z^n$ in

$$G(w,z) = \frac{wz}{(1-wz)^2(1-2wz)}\,\frac{w}{(1-w)^2}\,\frac{z}{1-z-z^2}.$$

Performing partial fractions with respect to w gives

$$G(w,z) = \frac{wz}{1-z-z^2}\left(\frac{z^2\,(2z-wz^2-4+3wz)}{(1-z)^3(1-wz)^2} + \frac{8z^2}{(1-2z)^2(1-2wz)} + \frac{z\,(6z^2-4wz^2-4z+wz+w)}{(1-2z)^2(1-z)^3(1-w)^2}\right).$$

The first two terms do not have a $w^n z^n$ in their expansion. Finding the coefficient of w^n and performing partial fractions with respect to z we get

$$\begin{aligned}&\frac{z\,((2n+4)z^3-(3n+1)z^2+(n-1)z)}{(1-2z)^2(1-z)^3(1-z-z^2)}\\ &= \frac{4(n-3)z}{1-z} + \frac{(n-5)z}{(1-z)^2} + \frac{-2z}{(1-z)^3} + \frac{8(n+5)z}{1-2z} + \frac{-8z}{(1-2z)^2}\\ &\quad - \frac{z\,(8(n+1)z+13(n+1))}{1-z-z^2}.\end{aligned}$$

Expanding as a function of z, we find that

$$\begin{aligned}s_n &= 4(n-3)+n(n-5)-n(n+1)+4(n+5)2^n-4n2^n\\ &\quad -13(n+1)F_n-8(n+1)F_{n-1}\\ &= 5\,2^{n+2}-2(n+6)-13(n+1)F_n-8(n+1)F_{n-1}. \quad \blacksquare\end{aligned}$$

Chapter 3

Algorithm Design

The generating function methods of Chapter 2 can be used to implement symbolic algorithms for solving summations and recurrences. In this chapter, we will consider the design of algorithms based on generating functions for solving linear recurrences with constant coefficients, systems of linear recurrences with constant coefficients, summations, and convolutions. The issues discussed here will guide us as we develop and implement various algorithms in later chapters.

Section 1 examines the general procedure for solving each problem and identifies the steps for which generating function based algorithms must be designed and developed. Section 2 considers in more detail the issues involved in designing algorithms to encode and expand generating functions.

3.1. A General Algorithm

The general algorithm for solving summations and recurrences can be broken down to four basic steps:

1. Parse the input.
2. Encode the input generating functions.
3. Transform the input generating functions to the generating function of the solution.
4. Expand the generating function of the solution.

Of the four steps, parsing and transformation of the input do not require the development of generating function based algorithms. Parsing techniques are well known. Implementation of this step is dependent on the input environment and the information that must be obtained from the input. The transformation of the input generating functions to those of the solution is a mapping based on the input values. The various mappings are defined in Chapter 2 and can be implemented directly.

Algorithms need to be developed for encoding and expanding generating functions. The issue here is the determination of which functions can be encoded and what types

of generating functions will need to be expanded. We have seen that the solution generating functions are a mapping based on the input. Thus, if we examine the effects of the mapping, we can determine the possible forms of the solution generating functions and identify the capabilities needed to expand them.

3.1.1. Recurrences

Linear recurrences with constant coefficients

When parsing a linear recurrence or a system of linear recurrences we need to determine the following pieces of information:

- the index variable used in the recurrence,
- the index and coefficient of each term within the homogeneous part of the recurrence relation,
- the range and value of boundary conditions,
- the inhomogeneous part of the recurrence.

Once the input generating functions are encoded, we can find the generating function of the solution using (2.25) and (2.26) or Theorem 2.3. If there are insufficient boundary conditions, we have some choices. The algorithm could issue an error message and fail, it could introduce sufficient symbolic constants, or it could use a sufficient number of constants of value zero.

We can now consider the generating functions that will have to be expanded to find the solution. The homogeneous generating function will be a rational function. The particular generating function will involve either a convolution or a hybrid convolution. If it involves a convolution it will be the product of a rational function and the generating function of the inhomogeneous part of the recurrence. If it involves a hybrid convolution, the diagonal generating function part of the convolution will be a rational function. The hybrid generating function part of the convolution will be the product of the generating functions of the factors of the inhomogeneous part of the recurrence.

Systems of linear recurrences with constant coefficients

Parsing systems of linear recurrences is more involved than a single linear recurrence. We must identify the name of each sequence that is defined by a recurrence relation and the range and values of its boundary conditions. For each recurrence, we need to find the following information:

- the index variable,
- the sequence name, index, and coefficient of each term in the homogeneous part of the recurrence,
- the inhomogeneous part of the recurrence.

To find the generating functions of each sequence defined by the system, each recurrence relation is mapped to a relation involving the generating functions. Each recurrence can be mapped in the same way. Let z be the generating function variable and let n be the index variable. Then, the steps involved are:

1. Determine the lower bound of the index variable n for which the recurrence holds.
2. Multiply both sides of the equation by z^n.
3. Sum over the valid values of n.
4. Distribute the summation over each term.
5. Factor powers of z from each summation and adjust the limits of summation so that $\sum_{n=n_0}^{\infty} t_{n-\alpha} z^n = z^\alpha \sum_{n=n_0-\alpha}^{\infty} t_n z^n$.
6. Add and subtract boundary values in order to find generating functions from the summations.
7. Gather terms.

Solving the resulting system yields the generating functions for each sequence. The difficult step in this process is the determination of the lower bound of the index value for which the recurrence holds. The following steps can be used to find the appropriate value for each recurrence relation:

1. For each sequence determine the smallest index for which it is defined.
2. Repeat for each recurrence relation:
   (a) Find each occurrence of a sequence in the relation.
   (b) Determine the smallest value of the index variable for which that occurrence is defined.
   (c) The maximum of these values is the lower bound of the index variable for which the recurrence holds.

For example, let t_n be defined for $n \geq \beta$. If $t_{n-\alpha}$ occurs in a recurrence, it will be defined if $n - \alpha \geq \beta$ or, equivalently for $n \geq \alpha + \beta$. The algorithm finds these values for all sequence terms in the recurrence and chooses the maximum. This becomes the lower bound for the index variable and assures that recurrence is valid over the limits of summation.

In order to expand the generating functions that result from solving this system of equations, we need to determine their form. To do so, we can consider the mapping for one of the recurrences. The ith equation in a system of m simultaneous recurrences has the general form

$$\sum_{j=1}^{m} \alpha_{i,j} t_{j,n} = \sum_{j=1}^{m} \sum_{l=1}^{k_i} \beta_{i,j,l} t_{j,n-l} + a_{i,n}, \tag{3.1}$$

where $a_{i,n}$ is the inhomogeneous part of the recurrence. The algorithm multiplies by z^n and sums over valid values of n. It then factors out powers of z, adds and subtracts terms to complete the generating functions of the sequences, and then collect terms. If there are m recurrences, the ith relation will have the form

$$\sum_{j=1}^{m} p_{i,j}(z)T_j(z) \;=\; q_i(z) + H_i(z), \tag{3.2}$$

where $T_j(z)$ is the generating function of the jth sequence, $H_i(z)$ is the generating function of the inhomogeneous part of the ith recurrence, and $p_{i,j}(z)$ and $q_i(z)$ are polynomials in z.

The polynomials $p_{i,j}(z)$ and $q_i(z)$ arise when we gather terms in step 7. Before we gather terms, each occurrence of the generating function $T_j(z)$ is multiplied by a constant from the recurrence relation and a power of z from step 5. When we gather together the terms with $T_j(z)$ in them, we get the polynomials $p_{i,j}(z)$. In step 6, when terms are added to complete the generating functions, the same terms are subtracted in order to maintain equality. These terms are each a constant multiplied by a power of z and when collected form $q_i(z)$.

To find the values of $T_i(z)$ we solve the system of m equations given by (3.2) for i from 1 to m. The following theorem considers the form of the solution of the system of equations given by a generalized version of (3.2).

Theorem 3.1. Let

$$\sum_{j=1}^{m} p_{i,j}(z)T_j(z) \;=\; A_i(z), \tag{3.3}$$

be the ith equation in a system of m equations defining the m unknown functions $T_j(z)$, where $p_{i,j}(z)$ is a rational function in z and $A_i(z)$ is any function in z. If the equations are linearly independent, then the solution to each $T_i(z)$ for $1 \le i \le m$ is of the form

$$T_i(z) \;=\; \sum_{k=1}^{m} g_{i,k}(z)A_k(z), \tag{3.4}$$

where $g_{i,k}(z)$ is a rational function of z.

Proof. The proof is by induction. The base case is $m = 2$. The solution to the system is

$$\begin{aligned} T_1(z) &= \frac{p_{2,2}(z)}{p_{1,1}(z)p_{2,2}(z) - p_{1,2}(z)p_{2,1}(z)}A_1(z) - \frac{p_{1,2}(z)}{p_{1,1}(z)p_{2,2}(z) - p_{1,2}(z)p_{2,1}(z)}A_2(z); \\ T_2(z) &= \frac{p_{1,1}(z)}{p_{1,1}(z)p_{2,2}(z) - p_{1,2}(z)p_{2,1}(z)}A_2(z) - \frac{p_{2,1}(z)}{p_{1,1}(z)p_{2,2}(z) - p_{1,2}(z)p_{2,1}(z)}A_1(z). \end{aligned}$$

Since the equations are linearly independent, the denominators are nonzero.

To prove the inductive step assume that (3.4) is true for $m = m_0$ and prove it true for $m = m_0 + 1$. Given a system of $m_0 + 1$ equations of the form (3.3) in $m_0 + 1$

unknowns, choose an equation whose coefficient of $T_{m_0+1}(z)$ is nonzero and let this be equation m_0+1. Divide this equation by $p_{m_0+1,m_0+1}(z)$ to get

$$T_{m_0+1}(z) = -\sum_{j=1}^{m_0} \frac{p_{m_0+1,j}(z)}{p_{m_0+1,m_0+1}(z)} T_j(z) + \frac{1}{p_{m_0+1,m_0+1}(z)} A_{m_0+1}(z). \qquad (3.5)$$

Substitute for $T_{m_0+1}(z)$ in the remaining m_0 equations of form (3.3). The ith equation will now be

$$\sum_{j=1}^{m_0} p_{i,j}(z) T_i(z)$$
$$- p_{i,m_0+1} \left(\sum_{j=1}^{m_0} \frac{p_{m_0+1,j}(z)}{p_{m_0+1,m_0+1}(z)} T_j(z) - \frac{1}{p_{m_0+1,m_0+1}(z)} A_{m_0+1}(z) \right) = A_i(z).$$

Rearranging we get

$$\sum_{j=1}^{m_0} \left(p_{i,j}(z) - \frac{p_{i,m_0+1}(z) p_{m_0+1,j}(z)}{p_{m_0+1,m_0+1}(z)} \right) T_j(z) = B_i(z),$$

where

$$B_i(z) = A_i(z) - \frac{p_{i,m_0+1}(z)}{p_{m_0+1,m_0+1}(z)} A_{m_0+1}(z).$$

We now have a system of m_0 equations in m_0 unknowns where the ith equation is of the form (3.3). Since the original equations are linearly independent, the new system of equations is linearly independent. Applying the induction hypothesis and substituting for $B_k(z)$, we find that the solution to $T_i(z)$, for $1 \le i \le m_0$, is

$$\begin{aligned} T_i(z) &= \sum_{k=1}^{m_0} g_{i,k}(z) B_k(z) \\ &= \sum_{k=1}^{m_0} g_{i,k}(z) \left(A_k(z) - \frac{p_{k,m_0+1}(z)}{p_{m_0+1,m_0+1}(z)} A_{m_0+1}(z) \right). \end{aligned}$$

Substituting into (3.5) and rearranging we further find that

$$\begin{aligned} T_{m_0+1}(z) &= -\sum_{j=1}^{m_0} \frac{p_{m_0+1,j}(z)}{p_{m_0+1,m_0+1}(z)} \sum_{k=1}^{m_0} g_{j,k}(z) \left(A_k(z) - \frac{p_{k,m_0+1}(z)}{p_{m_0+1,m_0+1}(z)} A_{m_0+1}(z) \right) \\ &\quad + \frac{1}{p_{m_0+1,m_0+1}(z)} A_{m_0+1}(z) \\ &= -\sum_{j=1}^{m_0} \frac{p_{m_0+1,j}(z)}{p_{m_0+1,m_0+1}(z)} \sum_{k=1}^{m_0} g_{j,k}(z) A_k(z) \\ &\quad + \sum_{j=1}^{m_0} \frac{p_{m_0+1,j}(z)}{p_{m_0+1,m_0+1}(z)} \sum_{k=1}^{m_0} g_{j,k}(z) \frac{p_{k,m_0+1}(z)}{p_{m_0+1,m_0+1}(z)} A_{m_0+1}(z) \end{aligned}$$

$$
\begin{aligned}
&+\frac{1}{p_{m_0+1,m_0+1}(z)}A_{m_0+1}(z) \\
= &-\sum_{k=1}^{m_0}\sum_{j=1}^{m_0}\frac{g_{j,k}(z)p_{m_0+1,j}(z)}{p_{m_0+1,m_0+1}(z)}A_k(z) \\
&+\left(\frac{1}{p_{m_0+1,m_0+1}(z)}+\sum_{j=1}^{m_0}\sum_{k=1}^{m_0}\frac{g_{j,k}(z)p_{m_0+1,j}(z)p_{k,m_0+1}(z)}{p^2_{m_0+1,m_0+1}(z)}\right)A_{m_0+1}(z).
\end{aligned}
$$

Thus, $T_i(z)$ satisfies (3.4) for $1 \leq i \leq m_0 + 1$. □

We can now apply Theorem 3.1 to the system of equations given by (3.2). Since $A_j(z) = q_j(z) + H_j(z)$ we find that

$$
\begin{aligned}
T_i(z) &= \sum_{j=1}^{m} g_{i,j}(z)\left(q_j(z) + H_j(z)\right) \\
&= \sum_{j=1}^{m} g_{i,j}(z)q_j(z) + \sum_{j=1}^{m} g_{i,j}(z)H_j(z).
\end{aligned}
$$

The first term is a rational function in z. The second term is the sum of products between rational functions and the generating functions of the inhomogeneous parts of the recurrences. If the recurrences are all homogeneous, then each $T_i(z)$ is a rational function. Thus, we find that we need to develop algorithms to expand rational generating functions and algorithms to expand convolutions between rational functions and other types of functions.

3.1.2. Summations

When parsing a summation, we need to determine the following pieces of information:

- whether or not the sum is a convolution,
- the index variable,
- the limits of summation,
- the summand.

Having determined this information, we have four possible transformations from which to choose:

- simple sums,
- hybrid sums,
- convolutions,
- hybrid convolutions.

The mappings for each transformation are defined in Chapter 2. For the simple sum and the convolution they are given by (2.20) and (2.19). To find the generating function for the solution to the hybrid sum we can use Corollary 2.2.2. Corollary 2.2.1 defines the solution for the hybrid convolution.

As with recurrences, algorithms to expand these generating functions must be developed. The mappings will produce four possible types of generating functions to expand. For the simple sum, we will have a convolution between the generating function $1/(1-z)$ and the generating function of the summand. For the hybrid sum, we will have a hybrid convolution. The diagonal generating function will be $1/(1-z)$ and the hybrid generating function will be the product of the generating functions of the factors in the summand. For convolutions, we will have the product of the generating functions of the factors in the summand. For the hybrid convolution, the hybrid generating function will come from the product of the factors of the hybrid term in the summand. The diagonal generating function will come from the product of the generating functions of the other factors in the summand.

3.2. Algorithm Design Issues

Of the four general steps in the algorithms, we have seen that parsing the input and transforming the input generating functions to the output generating functions are implementation problems. We will not consider these problems further. However, for both recurrences and summations, we need to develop and implement algorithms for encoding and expanding generating functions.

The encoding algorithm is the same for all problems: given an expression (or function), find its generating function. The decoding algorithms can be grouped into three broad categories:

- rational functions,
- convolutions,
- hybrid convolutions.

The types of convolutions and hybrid convolutions that will have to be expanded depend on the types of generating functions that we are able to encode. As procedures to encode generating functions are developed, procedures to expand convolutions and hybrid convolutions that involve them must be developed.

3.2.1. Encoding Generating Functions

We must encode generating functions for both the inhomogeneous parts of recurrences and for the summands of summations. There are two situations that we encounter. The inhomogeneous term or the summand could be an algebraic expression involving the index variable or variable of summation. We can treat this expression as the value of a sequence and attempt to find its generating function. Or, we could be

given an expression that involves a *special* function. This is simply the case where a named function of n, such as a sequence, is part of the expression. We need to find a generating function for the special function and then a generating function for the expression involving it.

An important issue in the design of these algorithms is how knowledge is built into them. One extreme would be to "memorize" everything and resort to table lookup. The other extreme would be to "deduce" everything and attempt to compute the generating function from (2.1). Our goal is to be as algorithmic as we can and to minimize table lookup as much a possible. The algorithms should know a few basic generating functions which they can look up and then use manipulation rules to encode additional sequences.

Once implemented, we need to know what class of functions the routine will handle. This information is needed to design procedures to expand generating functions. We want to know if the algorithm will handle any function from the class without further extension. We also want to be sure that the algorithm will fail gracefully when given a function it cannot encode. It is better to return failure than an incorrect result.

Algebraic Expressions

We can consider various classes of algebraic expressions a_n involving the symbol n and determine if we can compute their generating functions in an algorithmic way. If we can, we determine the initial generating function we must start with and the manipulations needed to derive the generating function $A(z)$ for a_n from the initial generating function.

- The expression for a_n is constant. We can apply rule (2.8) to the generating function $1/(1-z)$.

- The expression for a_n is exponential. We can apply (2.11) to the generating function $1/(1-z)$.

- The expression for a_n is a polynomial. Each term of the polynomial can be processed individually. To find the generating function for n^k, (2.16) can be applied k times to the generating function $1/(1-z)$. The corresponding coefficient can be handled by (2.8). The generating function for the individual terms can be added together to find the generating function for the polynomial.

- The expression for a_n is a rational function. A general method for encoding rational functions is not known. For example, the generating function of $1/k$ is $-\ln(1-z)$. Unfortunately, we cannot apply (2.14) to bring another factor of $1/k$ into the sequence and we are unable to find an ordinary generating function for $1/k^2$.

- The expression for a_n involves sums, differences, and products of polynomials and exponentials. We can immediately handle the product of a polynomial $p(n)$ with an exponential α^n using (2.11). For more complicated expressions of this type,

such as $(p(n)\alpha^n + q(n)\beta^n)r(n)\gamma^n$, we can expand the expression to a sum of products form and process each term individually. Adding the resulting generating functions gives the generating function for a_n.

Special Functions

Dealing with special functions requires some form of table lookup. We need to go from the name for the special function to its generating function. When dealing with special functions we want to maintain flexibility in how information about a function is specified. We also want to get maximum capability from the fewest definitions.

The simplest way to handle special functions would be to add their generating function to a table and look them up as needed. This puts the burden of finding a generating function for a new function on the user. A more desirable method is to allow the generating function to be computed from the definition of the special function. The definition could be a recursive definition, a summation, or an algebraic expression possibly involving other special functions. From the definition the generating function or hybrid generating function for the special function could be computed. This approach gives more flexibility to the user in how he adds new information to the system.

If the special function is a member of some larger class of functions, a procedure for encoding the whole class can be written. The generating function of the special function can then be determined algorithmically from the function's parameters using some initial table lookup. In this way we can get more capability than we could by just looking up the generating function or the special function's definition.

3.2.2. Decoding Generating Functions

The desired result of our algorithms is to find a "closed form solution." This is a widely understood but poorly defined concept. In a general sense we want to express the solution in terms of a finite number of well known and well defined quantities. This could be an algebraic expression or might include special functions. The question is, "What form is more important to the user?" If a solution involves Fibonacci numbers, should the system return the symbol F_n or the expression $((1+\sqrt{5})/2)^n/\sqrt{5} - ((1-\sqrt{5})/2)^n/\sqrt{5}$? As procedures for decoding generating functions are designed we must address the question of what is a proper closed form solution and how should special functions be used in expressing closed form solutions.

The algorithms to be developed will depend on the type of generating functions that need to be expanded. We have already found that to solve linear recurrences we will have to expand generating functions which are rational. We have also found that the other types of generating functions that we will have to expand depend on the type of input for which we can encode generating functions. Thus as we develop routines to encode classes of generating functions we will also develop routines for expanding convolutions and hybrid convolutions involving the generating functions of those classes.

Closed Forms

We will define the *closed form* of a function s_n to be an expression, valid for any n, that defines s_n as an algebraic function of the symbol n using a fixed number of the operations addition, subtraction, multiplication, division, and exponentiation. For example, the expression we used for the Fibonacci numbers above meets this definition.

It is often more informative to express the closed form of a function in terms of other well known functions. In some cases, such as the Fibonacci numbers, these are functions which have closed forms but whose names are considered more informative. In other cases these are functions that do not have known closed forms, including harmonic numbers, Stirling numbers, and Euler numbers. To handle such cases we will define an *extended closed form* to be an expression, valid for any n, that defines s_n as an algebraic function of the symbol n and a set of well defined functions of n using a fixed number of the operations addition, subtraction, multiplication, division, and exponentiation.

The algorithmic use of extended closed form solutions for recurrences and summations raises two important issues. We must decide *when* to look for a special function and *where* to look for it. As we develop new expansion algorithms we must take these issues into account. While we do not want to do an exhaustive search for special functions in the expansion of a generating function, it might be more informative to include functions used within the context of the problem being solved: for example, functions used to define a summand or the inhomogeneous part of a recurrence being considered. We can thus use the problem input to guide us where to look for special functions in the solution.

Chapter 4

Theory of Rational Generating Functions

Rational generating functions play an important role in discrete math. As we have already seen, they encode the solution to homogeneous linear recurrences with constant coefficients. In addition, they encode many important classes of sequences, including polynomials and exponentials. The set of rational generating functions, as well as the set of sequences that they encode, form well defined algebraic structures. By studying these structures, we can characterize properties that apply to all members of the set or to some well defined subset. We can also show that many manipulations of members of either set are closed on the set. We will make important use of these algebraic structures to prove the capabilities of our algorithms for rational generating functions.

In this chapter we look from a theoretical perspective at rational generating functions and the sequences that they encode. We define algebraic structures for the set of rational generating functions, the set of sequences that they encode, and the closed form expressions that define those sequences. We then relate homogeneous linear recurrences to rational generating functions.

4.1. Field of Rational Generating Functions

Before we consider the algebraic properties of rational generating functions we review some important algebraic structures [BURTON]. The pair $(\mathcal{F}, \star)$ is a *semigroup* if

- $\mathcal{F}$ is a nonempty set,
- $\star$ is a binary associative operation defined on $\mathcal{F}$,
- $\mathcal{F}$ is closed under $\star$.

The semigroup $(\mathcal{F}, \star)$ is a *group* if and only if

- $(\mathcal{F}, \star)$ is a semigroup with identity,
- each element in $\mathcal{F}$ has an inverse with respect to $\star$.

If the operation $\star$ is commutative, then $(\mathcal{F}, \star)$ is a commutative group. The system $(\mathcal{G}, \oplus, \odot)$ is a *ring* if

- $\mathcal{G}$ is a nonempty set,
- $\oplus$ and $\odot$ are binary operations defined on $\mathcal{G}$,
- $(\mathcal{G}, \oplus)$ is a commutative group,
- $(\mathcal{G}, \odot)$ is a semigroup,
- the operation $\odot$ distributes over the operation $\oplus$.

A ring $(\mathcal{G}, \oplus, \odot)$ is a *field* if

- $(\mathcal{G} - \{0\}, \odot)$ forms a commutative group, where 0 is the identity element of the operation $\oplus$.

Rational generating functions with coefficients from the set $\mathcal{F}$ form a set which we call $RAT_{\mathcal{F}}$. If the system $(\mathcal{F}, +, \cdot)$, where $+$ and $\cdot$ are the field operations of addition and multiplication, is a field then $(RAT_{\mathcal{F}}, +, \cdot)$ is a field. Addition is closed over this field and forms a commutative group with identity 0. The additive inverse of each $f \in RAT_{\mathcal{F}}$ is $-f$. Multiplication is also closed over $RAT_{\mathcal{F}}$ and forms a semigroup. If we eliminate the element 0, then each $f \in RAT_{\mathcal{F}}$ has $1/f$ as its multiplicative inverse and $(RAT_{\mathcal{F}} - \{0\}, \cdot)$ forms a commutative group with identity 1.

The set of functions $RAT_{\mathcal{F}}$ is commonly named $\mathcal{F}(z)$, the rational functions of z with coefficients from the field $\mathcal{F}$. We have chosen the RAT notation over the $\mathcal{F}(z)$ notation to highlight the fact that we are discussing a set of generating functions that encode sequences rather than a set of functions of z. However, we will continue to use the related notation $\mathcal{F}[z]$ to denote the set of polynomials in z with coefficients from set $\mathcal{F}$.

There are many possible coefficient fields for the rational functions. The more common are $\mathbb{Q}$, the rational numbers; $\mathbb{R}$, the real numbers; and $\mathbb{C}$, the complex numbers. We note that $RAT_{\mathbb{Q}} \subset RAT_{\mathbb{R}} \subset RAT_{\mathbb{C}}$. We also note that the integers form only a ring $(\mathbb{Z}, +, \cdot)$ since every member of $\mathbb{Z}$ does not have a multiplicative inverse in $\mathbb{Z}$. However, every member of $RAT_{\mathbb{Q}}$ can be expressed as a ratio of polynomials with integer coefficients so we find that the set $RAT_{\mathbb{Z}}$ is equivalent to the set $RAT_{\mathbb{Q}}$. Thus $(RAT_{\mathbb{Z}}, +, \cdot)$ is a field even though $(\mathbb{Z}, +, \cdot)$ is a ring. Since this is the case, we will keep the more stringent requirement that the set of coefficients for the rational generating functions form a field with respect to addition and multiplication.

Throughout this chapter we will refer to both the set $RAT_{\mathcal{F}}$ and the set RAT. In the former case the coefficient field will be important to the result, and we will restrict our discussion to the particular coefficient set $\mathcal{F}$. In the latter case we will be discussing a property that applies to a generating function that is rational with coefficients from any field defined with respect to addition and multiplication.

To discuss rational generating functions we need to be able to manipulate them and the polynomials from which they are constructed. Table 4.1 lists several functions that

numer$(G(z))$	numerator of rational function $G(z)$
denom$(G(z))$	denominator of rational function $G(z)$
degree$(p(z))$	degree of polynomial $p(z)$
ldegree$(p(z))$	degree of lowest order term of polynomial $p(z)$
coeff$(p(z), i)$	coefficient of z^i in polynomial $p(z)$
lcoeff$(p(z))$	coefficient of lowest order term of polynomial $p(z)$
lterm$(p(z))$	lcoeff$(p(z))z^{\text{ldegree}(p(z))}$ for polynomial $p(z)$

Table 4.1. Functions for manipulating rational functions and polynomials.

will be helpful. Unless otherwise indicated, we will assume that a rational function is in *normal* form. That is, it has been expressed as a ratio of two polynomials. We will define a *trivial* rational generating function to be a function $G(z)$ where

$$\text{ldegree}(\text{denom}(G(z)) = \text{degree}(\text{denom}(G(z)).$$

These are generating functions whose denominator is a trivial polynomial involving a single power of z. When

$$\text{ldegree}(\text{denom}(G(z)) < \text{degree}(\text{denom}(G(z)),$$

we will define $G(z)$ to be a *nontrivial* rational generating function. We will also define a *finite* sequence to be a sequence with a finite number of nonzero terms and a *infinite* sequence to be a sequence with an infinite number of nonzero terms.

4.2. Sequences with Rational Generating Functions

The set of rational generating functions encodes a set of sequences. These sequences form a well defined algebraic structure that can be related to RAT. We consider manipulation of members of this set and formally define the closed form expressions of these sequences.

4.2.1. Field of Sequences

Each generating function in the set RAT uniquely encodes a sequence. We call this set of sequences RAT_SEQ. The system $(RAT\_SEQ, +, *)$, where $*$ is the convolution operation, forms a field. The operations $+$ and $*$ are the sequence manipulations caused by the corresponding operations in the field $(RAT, +, \cdot)$.

The two fields can be algebraically related. A *ring homomorphism* is a mapping F from the ring $(\mathcal{F}, \oplus, \odot)$ to the ring $(\mathcal{G}, \boxplus, \boxdot)$ such that for every $f, g \in \mathcal{F}$,

$$F(f \oplus g) = F(f) \boxplus F(g)$$

and

$$F(f \odot g) = F(f) \boxdot F(g).$$

If the homomorphism F is one-to-one, the rings are *isomorphic.* This means that the homomorphism preserves the algebraic properties of the ring $(\mathcal{F}, \oplus, \odot)$ when mapping it onto the ring $(\mathcal{G}, \boxplus, \boxdot)$. The two structures can generally be treated as being the same algebraically.

Not surprisingly, the fields $(RAT_{\mathcal{F}}, +, \cdot)$ and $(RAT\_SEQ_{\mathcal{F}}, +, *)$ are isomorphic for the field $(\mathcal{F}, +, \cdot)$. Let 'expand()' be the function that expands a function in $RAT_{\mathcal{F}}$ to find the sequence that it encodes. Since a generating function uniquely encodes one sequence, 'expand()' is one-to-one. In addition, if $F(z), G(z) \in RAT_{\mathcal{F}}$ encode the sequences $\langle f_n \rangle$ and $\langle g_n \rangle$ then

$$\text{expand}(F(z) + G(z)) = \text{expand}(F(z)) + \text{expand}(G(z)) = \langle f_n \rangle + \langle g_n \rangle$$

and

$$\text{expand}(F(z)G(z)) = \text{expand}(F(z)) * \text{expand}(G(z)) = \langle f_n \rangle * \langle g_n \rangle.$$

Thus the mapping 'expand()' is a one-to-one homomorphism and the fields $(RAT_{\mathcal{F}}, +, \cdot)$ and $(RAT\_SEQ_{\mathcal{F}}, +, *)$ are isomorphic. It is also possible to show that the mapping which encodes the generating function of a sequence in $RAT\_SEQ_{\mathcal{F}}$, 'gf()', is a one-to-one homomorphism.

4.2.2. Sequence Manipulations

With the exception of (2.14), the manipulations defined in Table 2.1 are closed over RAT. In each case, applying the manipulation to generating functions in RAT produces a generating function in RAT. Thus, the corresponding sequence manipulations define new sequences that are in the set RAT_SEQ.

If we restrict the rational functions to a coefficient field $(\mathcal{F}, +, \cdot)$, we can show that the relations (2.8) to (2.13) are closed on $RAT_{\mathcal{F}}$ under certain conditions. Relations (2.8), (2.10), and (2.11) are closed on $RAT_{\mathcal{F}}$ if $\alpha, \beta \in \mathcal{F}$. The following theorem determines the conditions under which relation (2.13) is closed on $RAT_{\mathcal{F}}$.

Theorem 4.1. Let $(\mathcal{F}, +, \cdot)$ be a field defined over the operations addition and multiplication. If $\mathbb{Z} \subseteq \mathcal{F}$, then for every $G(z) \in RAT_{\mathcal{F}}$, its derivative $G'(z) \in RAT_{\mathcal{F}}$.

Proof. We can express $G(z)$ in normal form as $p(z)/q(z)$, where $p(z), q(z) \in \mathcal{F}[z]$. Differentiating, we find that $G'(z) = (p'(z)q(z) - p(z)q'(z))/q^2(z)$. We see that this expression is closed on $RAT_{\mathcal{F}}$ if, for any $p(z) \in \mathcal{F}[z]$ the derivative $p'(z)$ is also in $\mathcal{F}[z]$. Any polynomial can be expressed as

$$p(z) = \sum_{i=0}^{\text{degree}(p(z))} p_i z^i.$$

Differentiating with respect to z gives

$$p'(z) = \sum_{i=1}^{\text{degree}(p(z))} i p_i z^{i-1}.$$

The coefficients p_i for i from 1 to degree$(p(z))$ are in $\mathcal{F}$. If $\mathbb{Z} \subseteq \mathcal{F}$, then $ip_i \in \mathcal{F}$ and $p'(z) \in \mathcal{F}[z]$. □

4.2.3. Closed Form Expressions

We characterize the value of each term in a sequence according to the coefficient field of the sequence's generating function.

Theorem 4.2. Let $\langle f_n \rangle$ be the sequence encoded by $F(z)$. If $F(z) \in RAT_{\mathcal{F}}$ for coefficient field $(\mathcal{F}, +, \cdot)$ and $\mathbb{Z} \subseteq \mathcal{F}$ then $f_n \in \mathcal{F}$.

Proof. If $F(z)$ has z as a factor, we can write $F(z) = z^m G(z)$, where z is not a factor of $G(z)$. By (2.9), $\langle f_n \rangle = \langle g_{n-m} \rangle$, where $\langle g_n \rangle$ is the sequence encoded by $G(z)$. If we can prove that $g_n \in \mathcal{F}$ then we have proven that $f_n \in \mathcal{F}$. We have two cases to consider: $G(z)$ is a polynomial and $G(z)$ is a nontrivial rational generating function.

Case 1. $G(z)$ is a polynomial. The generating function can be expressed as

$$G(z) = \sum_{k=0}^{\text{degree}(G(z))} g_k z^k.$$

Each coefficient g_k of $G(z)$ is in $\mathcal{F}$. Thus, each term of the sequence

$$\langle g_n \rangle = \langle g_0, g_1, \ldots, g_{\text{degree}(G(z))}, 0, 0, \ldots \rangle$$

is in $\mathcal{F}$.

Case 2. $G(z)$ is a nontrivial rational function. The generating function is the ratio

$$G(z) = \frac{p(z)}{q(z)},$$

where $p(z), q(z) \in \mathcal{F}[z]$. From (2.4) we find that

$$g_n = \frac{1}{n!} \left. \frac{d^n G(z)}{dz^n} \right|_{z=0}.$$

By Theorem 4.1 $RAT_{\mathcal{F}}$ is closed under differentiation. Thus, differentiating a rational function from $RAT_{\mathcal{F}}$ n times yields a rational function that is in $RAT_{\mathcal{F}}$. Evaluating at $z = 0$ will produce the ratio of the constant terms in the numerator and denominator polynomials of the normalized rational function. Each of these constants is in $\mathcal{F}$ and their ratio is in $\mathcal{F}$. Since $n!$ is an integer, $\mathbb{Z} \subseteq \mathcal{F}$, and $G^{(n)}(z)|_{z=0} \in \mathcal{F}$ we find that $g_n \in \mathcal{F}$. □

We can see why powers of z were factored out of $F(z)$. If the denominator had z as a factor then the denominator of $F^{(n)}(z)$ would have z as a factor and the constant term of the denominator of $F^{(n)}(z)$ would be 0. In addition, based on the power of z that is factored out of $F(z)$, we find that the first nonzero term in $\langle f_n \rangle$ occurs at $n = \text{ldegree}(\text{numer}(F(z))) - \text{ldegree}(\text{denom}(F(z)))$. We also note that trivial rational generating functions encode only finite sequences and nontrivial rational generating functions encode infinite sequences.

While Theorem 4.2 has characterized the field where the terms of a sequence $\langle g_n \rangle$ encoded by the generating function $G(z) \in RAT_{\mathcal{F}}$ can be found, we would like to know more about the closed form expression for g_n. If $G(z)$ is a trivial rational function, then the denominator of $G(z)$ is some power of z and the generating function encodes a sequence with a finite number of nonzero terms. We can find values for g_n by determining the coefficients of the polynomial in the numerator and using the denominator to shift and scale the sequence. For $n > \text{degree}(\text{numer}(G(z))) - \text{degree}(\text{denom}(G(z)))$ we have $g_n = 0$. Thus the closed form of a finite sequence is zero.

If $G(z)$ is a nontrivial rational function, then the denominator of $G(z)$ is a polynomial with terms involving more than a single power of z. We can let

$$p(z) = \text{numer}(G(z)) \quad \text{and} \quad q(z) = \text{denom}(G(z)).$$

If $\langle q_n \rangle$ is the sequence encoded by $1/q(z)$ then

$$g_n = \sum_{i=\text{ldegree}(p(z))}^{\text{degree}(p(z))} \text{coeff}(p(z), i) q_{n-i}.$$

We note that $\langle g_n \rangle$ is defined by the convolution $\langle g_n \rangle = \langle p_n \rangle * \langle q_n \rangle$. However, since $\langle p_n \rangle$ has a finite number of terms, the convolution defining g_n has a finite number of terms. For $n - \text{ldegree}(q(z)) < \text{degree}(p(z))$, some of the values q_{n-i} are zero.

To expand $1/q(z)$ we can let

$$\mathcal{G}(z) = \frac{1}{q(z)/\text{lterm}(q(z))} = \frac{\text{lcoeff}(q(z)) z^{\text{ldegree}(q(z))}}{q(z)}.$$

Thus

$$\frac{1}{q(z)} = \frac{\mathcal{G}(z)}{\text{lcoeff}(q(z)) z^{\text{ldegree}(q(z))}},$$

and

$$q_n = \frac{\mathcal{g}_{n+\text{ldegree}(q(z))}}{\text{lcoeff}(q(z))},$$

where $\langle \mathcal{g}_n \rangle$ is the sequence encoded by $\mathcal{G}(z)$.

It is possible to derive formally and prove a closed form expression for $\mathcal{g}_n$ [GRAHAM, PURDOM]. However, our interest is in the form of this expression so we will look only at the method for finding it.

We can factor $\mathcal{G}(z)$ over the complex numbers as

$$\mathcal{G}(z) = \frac{1}{(1 - r_1 z)^{d_1} \cdots (1 - r_m z)^{d_m}}, \tag{4.1}$$

where $d_1 + \ldots + d_m$ = degree(denom($G(z)$)). Application of partial fractions with respect to z gives

$$G(z) = \sum_{i=1}^{m} \sum_{j=1}^{d_i} \frac{a_{i,j}}{(1 - r_i z)^j},$$

where the values $a_{i,j}$ are constant with respect to z. The generating function $1/(1-\alpha z)^k$ encodes the sequence $\langle \binom{k+n-1}{n} \alpha^n \rangle$. Thus,

$$g_n = \sum_{i=1}^{m} \sum_{j=1}^{d_i} \binom{j+n-1}{n} r_i^j,$$

and substituting back we find an expression for g_n.

By the Fundamental Theorem of Algebra, every polynomial of degree m with complex coefficients can be factored into m factors. Thus, this method can be applied to any nontrivial rational generating function to find the closed form expression of the sequence that it encodes. The only limitation to finding this expression is the ability to completely factor the denominator of the generating function.

Having characterized the closed forms of sequences in RAT_SEQ, we would like to know what expressions can be encoded by a rational generating function. Obviously, any sequence with a finite number of nonzero terms and a closed form of zero can be encoded with a trivial rational generating function. Of more interest are those sequences that are encoded by nontrivial rational generating functions. We will build an algebraic structure and show that it defines the closed forms of sequences encoded by nontrivial rational generating functions. This will allow us to define formally the expressions that can be encoded with nontrivial rational generating functions.

Lemma 4.1. Let $(\mathcal{F}, +, \cdot)$ be a field defined for the operations addition and multiplication. Let $(\mathcal{F}[n], +, \cdot)$ be the ring of polynomials of symbol n with coefficients from the set $\mathcal{F}$. Let $f(n)$ be the closed form of the sequence $\langle f(n) \rangle$ defined for $n \geq 0$. If $f(n) \in \mathcal{F}[n]$ and $\mathbb{Z} \subseteq \mathcal{F}$, then $F(z)$, the generating function of $\langle f(n) \rangle$, is in $RAT_{\mathcal{F}}$.

Proof. We can express the polynomial as

$$f(n) = \sum_{i=0}^{\text{degree}(f(n))} \alpha_i n^i,$$

where $\alpha_i \in \mathcal{F}$. If the generating function of $\langle n^i \rangle$ is $G_i(z) = \sum_{n=0}^{\infty} n^i z^n$ then the generating function of $\langle f(n) \rangle$ is

$$F(z) = \sum_{i=0}^{\text{degree}(f(n))} \alpha_i G_i(z).$$

We can prove by induction that

$$G_i(z) = \frac{n_i(z)}{(1-z)^{i+1}}, \tag{4.2}$$

where $n_i(z) \in \mathbb{Z}[z]$. The base case is $G_0(z) = \sum_{n=0}^{\infty} 1z^n = 1/(1-z)$. Thus $n_0(z) = 1$. To prove the inductive step assume that (4.2) is true for $i = i_0$ and prove it true for $i = i_0 + 1$. Applying (2.16) and the induction hypothesis we get

$$G_{i_0+1}(z) = zG'_{i_0}(z) = z\frac{d}{dz}\frac{n_{i_0}(z)}{(1-z)^{i_0+1}} = \frac{z(n'_{i_0}(z)(1-z) + (i_0+1)n_{i_0}(z))}{(1-z)^{i_0+2}}.$$

Since $n_{i_0}(z) \in \mathbb{Z}[z]$, $n_{i_0+1}(z) = z(n'_{i_0}(z)(1-z) + (i_0+1)n_{i_0}(z)) \in \mathbb{Z}[z]$.

Substituting, we find that

$$F(z) = \sum_{i=0}^{\text{degree}(f(n))} \alpha_i \frac{n_i(z)}{(1-z)^{i+1}}.$$

If $\mathbb{Z} \subseteq \mathcal{F}$, then $\alpha_i n_i(z) \in \mathcal{F}[z]$ and $F(z) \in RAT_{\mathcal{F}}$. □

This lemma proves that sequences defined by polynomials over the domain of non-negative integers have rational generating functions. We now build an algebraic structure that defines expressions that involve exponential terms or factors.

Lemma 4.2. Let $\mathcal{T}_{\mathcal{F}}$ be the set $\{\sum_{i=1}^{k} x_i \cdot y_i^n | x_i \in \mathcal{F}[n], y_i \in \mathcal{F}, \text{integer } k > 0\}$ where $(\mathcal{F}, +, \cdot)$ is a field, n is a symbol, and y^n represents an integer power of y. The system $(\mathcal{T}_{\mathcal{F}}, +, \cdot)$ is a commutative ring.

Proof. The proof follows directly from the definition of addition and multiplication over field $(\mathcal{F}, +, \cdot)$ and ring $(\mathcal{F}[n], +, \cdot)$. For every $y \in \mathcal{F}$, the value y^n is defined in terms of the operation multiplication on the field $(\mathcal{F}, +, \cdot)$. For any value that n can represent, we can treat y^n as if it were in $\mathcal{F}$. Using this fact we can show by algebraic manipulation on $\mathcal{F}$ and $\mathcal{F}[n]$ that

- addition on $\mathcal{T}_{\mathcal{F}}$ is associative, commutative, and closed,
- the additive inverse is defined on $\mathcal{T}_{\mathcal{F}}$,
- multiplication on $\mathcal{T}_{\mathcal{F}}$ is associative, commutative, and closed,
- multiplication distributes over addition on $\mathcal{T}_{\mathcal{F}}$. □

We now show that a sequence defined by a member of $\mathcal{T}_{\mathcal{F}}$ for $n \geq 0$ has a rational generating function.

Theorem 4.3. Let $(\mathcal{T}_{\mathcal{F}}, +, \cdot)$ be the ring defined in Lemma 4.2 for field $(\mathcal{F}, +, \cdot)$. Let f_n be the closed form expression of the sequence $\langle f_n \rangle$ defined for $n \geq 0$. If $f_n \in \mathcal{T}_{\mathcal{F}}$ and $\mathbb{Z} \subseteq \mathcal{F}$, then $F(z)$, the generating function of $\langle f_n \rangle$, is in $RAT_{\mathcal{F}}$.

Proof. Since $f_n \in \mathcal{T}_\mathcal{F}$, we can distribute multiplication over addition to express f_n in sum of products form. Each term in the sum can be expressed in the form $t_n = p(n)\alpha^n$, where $p(n) \in \mathcal{F}[n]$ and $\alpha \in \mathcal{F}$. Since $\mathbb{Z} \subseteq \mathcal{F}$ and $p(n) \in \mathcal{F}[n]$, by Lemma 4.1 we find that $P(z)$, the generating function of $\langle p(n) \rangle$, is in $RAT_\mathcal{F}$. By (2.11), the generating function of $\langle t_n \rangle$ is $T(z) = P(\alpha z)$. Since $\alpha \in \mathcal{F}$, we can see that the coefficients of $T(z)$ are in $\mathcal{F}$ and the generating function of each term in the sum of products expansion of f_n is in $RAT_\mathcal{F}$. Thus $F(z)$, which is the sum of these generating functions, is in $RAT_\mathcal{F}$. □

Theorem 4.3 considers only sequences defined by members of $\mathcal{T}_\mathcal{F}$ for $n \geq 0$. We use generating function shifting techniques to show that a sequence defined by a member of $\mathcal{T}_\mathcal{F}$ for $n \geq \gamma$ has a rational generating function.

Corollary 4.3.1. Let $(\mathcal{T}_\mathcal{F}, +, \cdot)$ be the ring defined in Lemma 4.2 for coefficient field $(\mathcal{F}, +, \cdot)$. Let f_n be the closed form expression of the sequence $\langle f_n \rangle$ defined for $n \geq \gamma$. If $f_n \in \mathcal{T}_\mathcal{F}$ and $\mathbb{Z} \subseteq \mathcal{F}$, then $F(z)$, the generating function of $\langle f_n \rangle$, is in $RAT_\mathcal{F}$.

Proof. Let $\langle g_n \rangle = \langle f_{n+\gamma} \rangle$ for $n \geq 0$. By (2.9), $F(z) = G(z)/z^\gamma$, where $G(z)$ is the generating function of $\langle g_n \rangle$. If $G(z) \in RAT_\mathcal{F}$ then $F(z) \in \mathcal{F}$. To prove that $G(z) \in RAT_\mathcal{F}$ we find the closed form of g_n and show that it is in the set $\mathcal{T}_\mathcal{F}$.

Each term in the sum of products form of f_n can be expressed as $p(n)\theta^n$, where $p(n) \in \mathcal{F}[n]$ and $\theta \in \mathcal{F}$. Substituting, we find that each term in the expression for g_n is of the form $p(n-\gamma)\theta^{n-\gamma} = \theta^{-\gamma}p(n-\gamma)\theta^n$. If $\mathbb{Z} \subseteq \mathcal{F}$ then $\theta^{-\gamma}p(n-\gamma) \in \mathcal{F}[n]$ and $\theta^{-\gamma}p(n-\gamma)\theta^n \in \mathcal{T}_\mathcal{F}$. Thus, $G(z) \in RAT_\mathcal{F}$ and $F(z) \in RAT_\mathcal{F}$. □

Corollary 4.3.1 considers only sequences for which the closed form expression is valid for all terms in the sequence. By definition, a finite number of terms in the sequence need not be defined by the closed form expression. We extend the corollary to any sequence that has a closed form in $\mathcal{T}_\mathcal{F}$.

Corollary 4.3.2. Let $(\mathcal{T}_\mathcal{F}, +, \cdot)$ be the ring defined in Lemma 4.2 for coefficient field $(\mathcal{F}, +, \cdot)$. Let n_0 and γ be integers such that $n_0 \geq \gamma$. Let $\langle f_n \rangle$ be a sequence defined for all $n \geq \gamma$ with closed form expression c_n that defines f_n for all $n \geq n_0$. If $f_n \in \mathcal{F}$ for all $n \geq \gamma$, $c_n \in \mathcal{T}_\mathcal{F}$, and $\mathbb{Z} \subseteq \mathcal{F}$ then $F(z)$, the generating function of $\langle f_n \rangle$, is in $RAT_\mathcal{F}$.

Proof. By the definition of a closed form, we can express f_n as

$$f_n = \begin{cases} c_n + g_n, & \text{if } \gamma \leq n \leq n_0 - 1; \\ c_n, & \text{if } n \geq n_0, \end{cases}$$

where $g_n = f_n - c_n$. If $\mathbb{Z} \subseteq \mathcal{F}$ then $f_n, c_n \in \mathcal{F}$ for all n and $g_n \in \mathcal{F}$. The generating function of $\langle f_n \rangle$ is $F(z) = C(z) + G(z)$, where $C(z) = \sum_{n=\gamma}^{\infty} c_n z^n$ and $G(z) = \sum_{n=\gamma}^{n_0-1} g_n z^n$. By Corollary 4.3.1, $C(z) \in RAT_\mathcal{F}$. $G(z)$ is a trivial rational generating function in $RAT_\mathcal{F}$. Thus, by the closure of addition on the field $(RAT_\mathcal{F}, +, \cdot)$, $F(z) \in RAT_\mathcal{F}$. □

We have now shown that every sequence with a closed form expression in $\mathcal{T}_{\mathcal{F}}$ for some coefficient field $(\mathcal{F}, +, \cdot)$ has a rational generating function. If we look at the coefficient field $(\mathbb{C}, +, \cdot)$, we can see that the set $\mathcal{T}_{\mathbb{C}}$ defines the closed form expression for every sequence with a generating function in $RAT_{\mathbb{C}}$. Trivial generating functions have closed form zero which is in the set. By the Fundamental Theorem of Algebra, the characteristic generating function of any nontrivial rational generating function in $RAT_{\mathbb{C}}$ can be factored as in (4.1). This allows us to apply partial fractions to express the characteristic generating function in the form (4.2). Expanding the generating function leads to a closed from expression for the sequence that is in $\mathcal{T}_{\mathbb{C}}$.

4.2.4. Linear Transformations and Products

We have already seen methods for constructing new sequences in RAT_SEQ from existing sequences. The manipulation rules (2.8) through (2.13) when applied to sequences in RAT_SEQ yield sequences in RAT_SEQ. In addition, the convolution operation is closed on RAT_SEQ. The following two theorems define additional ways in which members of RAT_SEQ can be used to construct new sequences that are in RAT_SEQ.

Theorem 4.4. Let $\langle g_n \rangle = \langle f_{\alpha n+\beta} \rangle$ for $\alpha, \beta \in \mathbb{Z}$. If $\langle f_n \rangle \in RAT\_SEQ$ then $\langle g_n \rangle \in RAT\_SEQ$.

Proof. If $\langle f_n \rangle$ is a finite sequence then $\langle g_n \rangle$ is a finite sequence and its generating function is a trivial rational function. Otherwise, a closed form expression for the sequence exists and has the form

$$f_n = \sum_i p_i(n)\theta_i^n.$$

Since this closed form exists, there is some field $(\mathcal{F}, +, \cdot)$ such that $\mathbb{Z} \subseteq \mathcal{F}$ and $f_n \in \mathcal{T}_{\mathcal{F}}$, where $\mathcal{T}_{\mathcal{F}}$ is the set defined in Lemma 4.2. Substituting, we find that

$$g_n = f_{\alpha n+\beta} = \sum_i p_i(\alpha n+\beta)\theta_i^{\alpha n+\beta} = \sum_i p_i(\alpha n+\beta)\theta_i^{\beta}\left(\theta_i^{\alpha}\right)^n.$$

Since $\mathbb{Z} \in \mathcal{F}$, the composition $p_i(\alpha n + \beta) \in \mathcal{F}[n]$ and $p_i(\alpha n + \beta)\theta_i^{\beta} \in \mathcal{F}[n]$. By the definition of exponentiation, $\theta_i^{\alpha} \in \mathcal{F}$ for integer α. We thus find that $p_i(\alpha n+\beta)\theta_i^{\alpha n+\beta} \in \mathcal{T}_{\mathcal{F}}$, and so, by Corollary 4.3.2, the generating function of $\langle g_n \rangle$ is a rational function and $\langle g_n \rangle \in RAT\_SEQ$. □

Example 4.1. Consider the sequence $\langle s_n \rangle$ encoded by the generating function $S(z) = \gamma z(1+\gamma z)/(1-\gamma z)^3$. Its closed form is $s_n = n^2\gamma^n$. The sequence $\langle t_n \rangle = \langle s_{\alpha n+\beta} \rangle$ has closed form

$$t_n = \left(\alpha^2 n + 2\alpha\beta n + \beta^2\right)\gamma^{\alpha n+\beta}.$$

The generating function of $\langle t_n \rangle$ is

$$\begin{aligned} T(z) &= \sum_{n=0}^{\infty} t_n z^n \\ &= \sum_{n=0}^{\infty} \left(\alpha^2 n + 2\alpha\beta n + \beta^2\right) \gamma^{\alpha n+\beta} z^n \\ &= \sum_{n=0}^{\infty} \gamma^{\beta} \left(\alpha^2 n + 2\alpha\beta n + \beta^2\right) (\gamma^{\alpha})^n z^n. \end{aligned}$$

The generating function of $\langle \alpha^2 n + 2\alpha\beta n + \beta^2 \rangle$ is $((\alpha^2 - 2\alpha\beta + \beta^2)z^2 + (\alpha^2 + 2\alpha\beta - 2\beta^2)z + \beta^2)/(1-z)^3$. Applying (2.11) and (2.8), we find that

$$T(z) = \gamma^{\beta} \frac{(\alpha^2 - 2\alpha\beta + \beta^2)(\gamma^{\alpha})^2 z^2 + (\alpha^2 + 2\alpha\beta - 2\beta^2)\gamma^{\alpha} z + \beta^2}{(1-\gamma^{\alpha} z)^3}. \quad \blacksquare$$

Theorem 4.5. Let $\langle s_n \rangle = \langle f_n g_n \rangle$. If $\langle f_n \rangle, \langle g_n \rangle \in RAT\_SEQ$ then $\langle s_n \rangle \in RAT\_SEQ$.

Proof. If $\langle f_n \rangle$ or $\langle g_n \rangle$ is a finite sequence, then $\langle s_n \rangle$ is a finite sequence and $\langle s_n \rangle$ has a trivial rational generating function. Otherwise, the closed forms of the sequences $\langle f_n \rangle$ and $\langle g_n \rangle$ must be in the ring $(\mathcal{T}_{\mathcal{F}}, +, \cdot)$ for some field $(\mathcal{F}, +, \cdot)$. By the closure of multiplication, the product of their closed forms must be in the ring $(\mathcal{T}_{\mathcal{F}}, +, \cdot)$. Thus by Corollary 4.3.2, the hybrid sequence $\langle s_n \rangle = \langle f_n g_n \rangle$ is in RAT_SEQ. $\square$

Example 4.2. Consider the sequences $\langle a_n \rangle = \langle F_n \alpha^n \rangle$, $\langle b_n \rangle = \langle nF_n \rangle$, and $\langle c_n \rangle = \langle F_n^2 \rangle$. We can show that each has a rational generating function. By (2.11) we find that the generating function of $\langle a_n \rangle$ is

$$A(z) = \frac{\alpha z}{1 - \alpha z - \alpha^2 z^2}$$

and by (2.16) we find that the generating function of $\langle b_n \rangle$ is

$$B(z) = \frac{z + z^3}{(1 - z - z^2)^2}.$$

To find the generating function of $\langle c_n \rangle$, we first note that the closed form of the Fibonacci numbers is defined for $n \geq 0$. Applying (2.1) we find that the generating function of $\langle c_n \rangle$ is

$$\begin{aligned} C(z) &= \sum_{n=0}^{\infty} \left(\frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}\right)^2 z^n \\ &= \sum_{n=0}^{\infty} \frac{1}{5} \left(\phi^{2n} - 2\phi^n \hat{\phi}^n + \hat{\phi}^{2n}\right) z^n \\ &= \frac{1}{5} \sum_{n=0}^{\infty} \left(\phi^{2n} - 2(-1)^n + \hat{\phi}^{2n}\right) z^n \end{aligned}$$

$$= \frac{1}{5}\left(\frac{1}{1-\phi^2 z} - \frac{2}{1+z} + \frac{1}{1-\hat{\phi}^2 z}\right)$$
$$= \frac{z^2 - z}{(1+z)(1-3z+z^2)}. \quad \blacksquare$$

While the product of two infinite sequences generally produces an infinite sequence, there are some cases where the hybrid sequence is finite.

Example 4.3. Consider the sequences encoded by the generating functions $A(z) = 1/(1-z^2)$ and $B(z) = z/(1-z^2)$. The sequence $\langle a_n \rangle$ encoded by $A(z)$ has the closed form

$$a_n = \frac{1+(-1)^n}{2}, \quad \text{for } n \geq 0.$$

The sequence $\langle b_n \rangle$ encoded by $B(z)$ has the closed form

$$b_n = \frac{1-(-1)^n}{2}, \quad \text{for } n \geq 1.$$

The closed form of the hybrid sequence $\langle c_n \rangle = \langle a_n b_n \rangle$ is

$$c_n = \frac{1+(-1)^n}{2}\frac{1-(-1)^n}{2} = \frac{1-(-1)^{2n}}{4} = 0.$$

Thus, $\langle c_n \rangle$ is a finite sequence. $\blacksquare$

4.3. Linear Recurrences and Rational Generating Functions

We have already seen in (2.25) that the generating function of a sequence defined by a homogeneous linear recurrence with constant coefficients is rational. We consider more formally the relationship between homogeneous linear recurrences and nontrivial rational generating functions.

We will use the phrase *recurrence relation* in two contexts. We will say that a sequence *satisfies* a recurrence if substituting appropriate values of the sequence into the recurrence relation for all $n \geq n_0$ yields a true equation. For example, the Fibonacci numbers satisfy the recurrence $t_n = t_{n-1} + t_{n-2}$ for $n \geq 2$. We will say a sequence is *defined* by a recurrence relation if we are given a recurrence relation and sufficient boundary conditions to generate the terms in the sequence. The Fibonacci numbers are defined by the recurrence $t_n = t_{n-1} + t_{n-2}$, for $n \geq 2$, with $t_1 = 1$ and $t_0 = 0$.

4.3.1. Homogeneous Linear Recurrences

Let the nontrivial rational generating function $G(z)$ encode the sequence $\langle g_n \rangle$. We define the *characteristic generating function* of $G(z)$ to be

$$\mathcal{G}(z) = \frac{1}{q(z)/\operatorname{lterm}(q(z))},$$

where $q(z) = \text{denom}(G(z))$. Likewise, we define the sequence $\langle g_n \rangle$ encoded by $\mathcal{G}(z)$ to be the characteristic sequence of $\langle g_n \rangle$. Note that the denominator of a characteristic generating function is a polynomial whose coefficient of z^0 is 1. We can now relate every characteristic generating function to a homogeneous linear recurrence.

Lemma 4.3. The characteristic generating function

$$\mathcal{G}(z) = \frac{1}{1 - \sum_{i=1}^{k} \alpha_i z^i}$$

encodes the sequence defined by the recurrence

$$g_n = \begin{cases} \sum_{i=1}^{k} \alpha_i g_{n-i}, & \text{if } n \geq 1; \\ 1, & \text{if } n = 0; \\ 0, & \text{if } -k+1 \leq n \leq -1. \end{cases}$$

Proof. We can apply (2.25) directly to find that

$$\mathcal{G}(z) = \frac{\sum_{n=-k+1}^{0} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} t_{-k+n} z^{-k+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i}.$$

Substituting the boundary conditions we get

$$\begin{aligned} \mathcal{G}(z) &= \frac{\sum_{n=-k+1}^{-1} 0 z^n + 1 z^0 - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} 0 z^{-k+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i} \\ &= \frac{1}{1 - \sum_{i=1}^{k} \alpha_i z^i}. \quad \square \end{aligned}$$

Now that we have found the recurrence that defines $\langle g_n \rangle$ we can see why $\mathcal{G}(z)$ is called a characteristic generating function. Its denominator is equivalent to the characteristic polynomial that can be used to solve a homogeneous linear recurrence [LIU]. As we will see throughout this chapter, the behavior of any nontrivial rational generating function is determined by its characteristic generating function. Likewise, the behavior of a sequence encoded by a nontrivial rational generating is determined by its characteristic sequence.

Lemma 4.4. Let

$$\mathcal{G}(z) = \frac{1}{(1 - \sum_{i=1}^{k} \alpha_i z^i)}$$

be the characteristic generating function of the generating function $G(z)$. Let

$$G(z) = p(z)\, \mathcal{G}(z),$$

where $p(z) = \text{numer}(G(z)))/\,\text{lterm}(\text{denom}(G(z)))$. The sequence $\langle g_n \rangle$ encoded by $G(z)$ satisfies the homogeneous linear recurrence

$$g_n = \sum_{i=1}^{k} \alpha_i g_{n-i}.$$

Proof. By definition, the generating function $\mathcal{G}(z)$ encodes the sequence $\langle g_n \rangle$. By (2.15), $g_n = \sum_{j=\text{ldegree}(p(z))}^{\text{degree}(p(z))} \text{coeff}(p(z), j) g_{n-j}$. Applying the recurrence for $\langle g_n \rangle$ from Lemma 4.3, we find that

$$g_n = \sum_{j=\text{ldegree}(p(z))}^{\text{degree}(p(z))} \text{coeff}(p(z), j) \alpha_i \sum_{i=1}^{k} g_{n-i-j}.$$

Reversing the order of summation we get

$$\begin{aligned} g_n &= \sum_{i=1}^{k} \sum_{j=\text{ldegree}(p(z))}^{\text{degree}(p(z))} \alpha_i \, \text{coeff}(p(z), j) g_{n-i-j} \\ &= \sum_{i=1}^{k} \alpha_i g_{n-i}. \quad \square \end{aligned}$$

We can now relate every nontrivial rational generating function to a homogeneous linear recurrence.

Theorem 4.6. Every nontrivial rational generating function $F(z)$ encodes a sequence $\langle f_n \rangle$ that is defined by a linear homogeneous recurrence with constant coefficients.

Proof. Let

$$\mathcal{F}(z) = \frac{1}{1 + \alpha_1 z + \alpha_2 z^2 + \cdots + \alpha_k z^k}$$

be the characteristic generating function of $F(z)$. We can express $F(z)$ as

$$F(z) = \frac{p(z)}{\text{lterm}(\text{denom}(F(z)))} \mathcal{F}(z).$$

By Lemma 4.4, $\langle f_n \rangle$ must satisfy the linear recurrence

$$f_n = \sum_{i=1}^{k} \alpha_i f_{n-i}.$$

Applying (2.4), we can find sufficient boundary conditions to define $\langle f_n \rangle$. $\square$

4.3.2. Boundary Conditions

It is known that an order k recurrence is completely defined by k boundary conditions [Liu]. In (2.25) we let the indices of the boundary conditions be $\gamma + 1 \leq n \leq \gamma + k$. By inspection we see that the numerator of the generating function defined by (2.25) will be the sum of terms involving $x^{\gamma+1}, \ldots, x^{\gamma+k}$, while the denominator is the sum of terms involving $x^0, \ldots, x^k$. An $x^{\gamma+1}$ can be factored out of the numerator. We find that if $\gamma < -1$, the denominator will have a power of z as a factor and if $\gamma > -1$, the

numerator will have a power of z as a factor. We also find that the generating function $F(z)$ for an order k homogeneous linear recurrence with k boundary conditions has

$$\begin{aligned}&\text{degree}(\text{denom}(F(z))) - \text{ldegree}(\text{denom}(F(z))) = k \quad \text{and}\\ &\text{degree}(\text{numer}(F(z))) - \text{ldegree}(\text{numer}(F(z))) \leq k-1.\end{aligned}$$

The cases where $\text{degree}(\text{numer}(F(z))) - \text{ldegree}(\text{numer}(F(z))) < k-1$ arise when some of the initial boundary conditions are 0.

Many nontrivial rational generating functions have

$$\text{degree}(\text{numer}(F(z))) - \text{ldegree}(\text{numer}(F(z))) \geq k.$$

The following theorem characterizes the recurrences that define the sequences encoded by such generating functions.

Theorem 4.7. Let $T(z)$ be a nontrivial rational generating function where

$$\begin{aligned}&\text{degree}(\text{denom}(T(z))) - \text{ldegree}(\text{denom}(T(z))) = k \quad \text{and}\\ &\text{degree}(\text{numer}(T(z))) - \text{ldegree}(\text{numer}(T(z))) \geq k.\end{aligned}$$

Let $\delta = \text{degree}(\text{numer}(T(z))) - \text{ldegree}(\text{numer}(T(z))) - k + 1$. The sequence $\langle t_n \rangle$ encoded by $T(z)$ is defined by the order k recurrence encoded by the characteristic generating function of $T(z)$ and $k+\delta$ boundary conditions.

Proof. Factor any powers of z out of $T(z)$ to obtain

$$Q(z) = \frac{\text{numer}(T(z))/z^{\text{ldegree}(\text{numer}(T(z)))}}{\text{denom}(T(z))/z^{\text{ldegree}(\text{denom}(T(z)))}}.$$

The degree of the numerator is $k+\delta-1$ and the degree of the denominator is k. Polynomial division of the numerator gives

$$Q(z) = p(z) + \frac{r(z)}{\text{denom}(T(z))/z^{\text{ldegree}(\text{denom}(T(z)))}},$$

where $p(z)$ and $r(z)$ are polynomials in z such that the coefficient of z^0 in each is nonzero. We have

$$\text{degree}(p(z)) = \delta \quad \text{and} \quad \text{degree}(r(z)) = k-1.$$

Let $\langle r_n \rangle$ be the sequence defined by $r(z)/(\text{denom}(T(z))/z^{\text{ldegree}(\text{denom}(T(z)))})$. Expanding generating functions we find that the sequence encoded by $Q(z)$ is

$$q_n = \begin{cases} \text{coeff}(p(z), n) + r_n, & \text{if } 0 \leq n \leq \delta - 1; \\ r_n, & \text{if } n \geq \delta. \end{cases}$$

By Lemma 4.4, $\langle q_n \rangle$ and $\langle r_n \rangle$ both satisfy the recurrence $t_n = \sum_{i=1}^{k} \alpha_i t_{n-i}$. The boundary values for $\langle r_n \rangle$ are at $0 \leq n \leq k-1$ and the recurrence for $\langle r_n \rangle$ holds

for $n \geq k$. From the recursive definition of $\langle q_n \rangle$, we see that the recurrence is valid only for $n \geq k + \delta$. If it held for any smaller values of n then we would have that $\text{lcoeff}(p(z), \delta) = 0$, which is a contradiction.

To define the recurrence for $\langle q_n \rangle$, we need boundary values for $\delta + 1 \leq n \leq \delta + k - 1$ in order for the recurrence to be valid at $n = k + \delta$. We also need boundary values for $0 \leq n \leq \delta$ for the δ initial values of q_n which do not affect the recurrence. Thus we need a total of $k + \delta$ boundary conditions to define $\langle q_n \rangle$. Since $\langle t_n \rangle$ is obtained by shifting $\langle q_n \rangle$, $k + \delta$ boundary conditions are also needed to define $\langle t_n \rangle$. □

Given the recurrence definition

$$t_n = \sum_{i=1}^{k} \alpha_i t_{n-k}$$

with $k + \delta$ boundary terms defined for $\gamma + 1 \leq n \leq \gamma + k + \delta$, we can modify the steps followed in Chapter 2 to find the generating function $T(z) = \sum_{n=\gamma+1}^{\infty} t_n z^n$.

Step 1. Multiply by z^n and sum over $n \geq \gamma + k + \delta + 1$. This yields

$$\sum_{n=\gamma+k+\delta+1}^{\infty} t_n z^n = \sum_{n=\gamma+k+\delta+1}^{\infty} \sum_{i=1}^{k} \alpha_i t_{n-i} z^n.$$

Step 2. Rearrange the order of summation, factor z^i out of the inner summation, and adjust the limits of summation.

$$\begin{aligned} \sum_{n=\gamma+k+\delta+1}^{\infty} t_n z^n &= \sum_{i=1}^{k} \alpha_i z^i \sum_{n=\gamma+k+\delta+1}^{\infty} t_{n-i} z^{n-i} \\ &= \sum_{i=1}^{k} \alpha_i z^i \sum_{n=\gamma+k+\delta-i+1}^{\infty} t_n z^n. \end{aligned}$$

Step 3. Add and subtract terms to get the sums into the form of the sum for $T(z)$,

$$\sum_{n=\gamma+1}^{\infty} t_n z^n - \sum_{n=\gamma+1}^{\gamma+\delta+k} t_n z^n = \sum_{i=1}^{k} \alpha_i z^i \left(\sum_{n=\gamma+1}^{\infty} t_n z^n - \sum_{n=\gamma+1}^{\gamma+k+\delta-i} t_n z^n \right).$$

Substituting for $T(z)$ and rearranging gives

$$\begin{aligned} T(z) - \sum_{n=\gamma+1}^{\gamma+\delta+k} t_n z^n &= \sum_{i=1}^{k} \alpha_i z^i \left(T(z) - \sum_{n=\gamma+1}^{\gamma+k+\delta-i} t_n z^n \right) \\ &= \sum_{i=1}^{k} \alpha_i z^i T(z) - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=\gamma+1}^{\gamma+k+\delta-i} t_n z^n \\ &= \sum_{i=1}^{k} \alpha_i z^i T(z) - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k+\delta-i} t_{\gamma+n} z^{\gamma+n}. \end{aligned}$$

Step 4. Collect terms and solve for $T(z)$.

$$T(z) - \sum_{i=1}^{k} \alpha_i z^i T(z) = \sum_{n=\gamma+1}^{\gamma+k+\delta} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k+\delta-i} t_{\gamma+n} z^{\gamma+n}.$$

Doing so, we find that

$$T(z) = \frac{\sum_{n=\gamma+1}^{\gamma+k+\delta} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k+\delta-i} t_{\gamma+n} z^{\gamma+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i}.$$

It is possible to specify too many boundary conditions. For example, boundary conditions could be given for $\gamma + 1 \leq n \leq \gamma + k + \delta$ but at $n = \gamma + k + \delta$ we find that

$$t_{\gamma+k+\delta} = \sum_{i=1}^{k} \alpha_i t_{\gamma+k+\delta-i}.$$

If we isolate the $z^{\gamma+k+\delta}$ terms from each summation in the numerator of the generating function $T(z)$ we get

$$\begin{aligned} T(z) &= \frac{\sum_{n=\gamma+1}^{\gamma+k+\delta-1} t_n z^n + t_{\gamma+k+\delta} z^{\gamma+k+\delta}}{1 - \sum_{i=1}^{k} \alpha_i z^i} \\ &\quad - \frac{\sum_{i=1}^{k} \alpha_i z^i \left(\sum_{n=1}^{k+\delta-i-1} t_{\gamma+n} z^{\gamma+n} + t_{\gamma+k+\delta-i} z^{\gamma+k+\delta-i} \right)}{1 - \sum_{i=1}^{k} \alpha_i z^i} \\ &= \frac{\sum_{n=\gamma+1}^{\gamma+k+\delta-1} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k+\delta-i-1} t_{\gamma+n} z^{\gamma+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i} \\ &\quad + \frac{t_{\gamma+k+\delta} z^{\gamma+k+\delta} - \sum_{i=1}^{k} \alpha_i t_{\gamma+k+\delta-i} z^{\gamma+k+\delta}}{1 - \sum_{i=1}^{k} \alpha_i z^i} \\ &= \frac{\sum_{n=\gamma+1}^{\gamma+k+\delta-1} t_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k+\delta-i-1} t_{\gamma+n} z^{\gamma+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i}. \end{aligned}$$

Thus we see that having unnecessary boundary values will not change the generating function.

A related problem occurs when a specific set of boundary values leads to cancellation in the generating function. That is, a particular set of boundary conditions for an order k recurrence creates a sequence that satisfies an order $k-1$ recurrence.

Example 4.4. Consider the recurrence

$$s_n = \begin{cases} (2+\sqrt{2})s_{n-1} - 2\sqrt{2}s_{n-2}, & \text{if } n \geq 2; \\ 2, & \text{if } n = 1; \\ 1, & \text{if } n = 0. \end{cases}$$

The generating function of the sequence $\langle s_n \rangle$ is

$$\begin{aligned} S(z) &= \frac{2z + 1 - (2 - \sqrt{2})z}{1 - (2 + \sqrt{2})z + 2\sqrt{2}z^2} \\ &= \frac{1 - \sqrt{2}z}{1 - (2 + \sqrt{2})z + 2\sqrt{2}z^2} \\ &= \frac{1}{1 - 2z}. \quad \blacksquare \end{aligned}$$

Notice that the boundary conditions $s_0 = 1$ and $s_1 = 2$ satisfy a recurrence, $t_n = 2t_{n-1}$, whose characteristic generating function is a factor of the characteristic generating function for $\langle s_n \rangle$. As a result, cancellation occurs. This observation leads to the following theorem.

Theorem 4.8. Let $\langle g_n \rangle$ be a recurrence that satisfies a homogeneous linear recurrence with constant coefficients. The sequence $\langle g_n \rangle$ satisfies infinitely many homogeneous linear recurrences with constant coefficients.

Proof. Let $G(z)$ be the generating function of the sequence $\langle g_n \rangle$. Let $\mathcal{G}(z)$ be the characteristic generating function of $G(z)$ such that

$$G(z) = \frac{p(z)}{\text{lterm}(\text{denom}(G(z)))} \mathcal{G}(z),$$

where $p(z)$ is a polynomial in z. Let $\mathcal{F}(z)$ be a characteristic generating function with $\mathcal{G}(z)$ as a factor, such that

$$\text{degree}(\text{denom}(\mathcal{F}(z))) > \text{degree}(\text{denom}(\mathcal{G}(z))).$$

There are infinitely many such $\mathcal{F}(z)$. If $\mathcal{F}(z) = \mathcal{G}(z)/q(z)$,

$$G(z) = \frac{p(z)q(z)}{\text{lterm}(\text{denom}(G(z)))} \mathcal{F}(z).$$

We can express g_n as a linear combination of values from $\langle f_n \rangle$, the sequence encoded by $\mathcal{F}(z)$. Duplicating the proof of Lemma 4.4, we can show that g_n satisfies the recurrence for f_n. $\square$

Chapter 5

Applications of Rational Generating Functions

Having developed a theory of rational generating functions and the sequences that they encode, we now consider applications of rational generating functions. In this chapter we will look at relationships involving sequences in RAT_SEQ, then consider the solution of summations and recurrences, and finally investigate algorithms for manipulating rational generating functions. We will determine which summations and recurrences involving sequences from RAT_SEQ can be solved, where their solutions can be found, and the form in which their solutions can be expressed.

5.1. Sequence Relationships

The sequences in RAT_SEQ have several interesting properties that relate terms within a sequence and relate terms between sequences with common factors in their characteristic generating functions. These relationships are useful when manipulating or simplifying expressions that involve sequences from RAT_SEQ.

5.1.1. Relationships Within Sequences

We can relate recursively any term in a sequence defined by an order k linear recurrence with constant coefficients to any previous term in the sequence.

Theorem 5.1. Let $\langle g_n \rangle$ satisfy the kth order linear recurrence $g_n = \sum_{i=1}^{k} \alpha_i g_{n-i}$. If $\langle g_n \rangle$ is encoded by the characteristic generating function

$$G(z) = \frac{1}{1 - \sum_{i=1}^{k} \alpha_i z^i},$$

of $\langle g_n \rangle$ then

$$g_n = g_j g_{n-j} + \sum_{m=1}^{k-1} \sum_{i=1}^{k-m} \alpha_{i+m} g_{j-i} g_{n-j-m}. \tag{5.1}$$

Proof. The sequence $\langle g_n \rangle$ is defined by the recurrence

$$g_n = \begin{cases} \sum_{i=1}^{k} \alpha_i g_{n-i}, & \text{if } n \geq 1; \\ 1, & \text{if } n = 0; \\ 0, & \text{if } -1 \geq n \geq -m+1. \end{cases} \tag{5.2}$$

We can now prove (5.1) by induction on j. The base case is $j = 0$. We get

$$\begin{aligned} g_n &= g_0 g_n + \sum_{m=1}^{k-1} \sum_{i=1}^{k-m} \alpha_{i+m} g_{-i} g_{n-m} \\ &= g_0 g_n, \end{aligned}$$

since $g_{-i} = 0$ for $1 \leq i \leq m-1$. To prove the inductive step, assume that (5.1) is true for $j = j_0$ and prove that it is true for $j = j_0 + 1$. Applying the recurrence for g_{n-j_0} to the induction hypothesis we have

$$\begin{aligned} g_n &= g_{j_0} g_{n-j_0} + \sum_{m=1}^{k-1} \sum_{i=1}^{k-m} \alpha_{i+m} g_{j_0-i} g_{n-j_0-m} \\ &= \sum_{m=1}^{k} \alpha_m g_{j_0} g_{n-j_0-m} + \sum_{m=1}^{k-1} \sum_{i=1}^{k-m} \alpha_{i+m} g_{j_0-i} g_{n-j_0-m}. \end{aligned}$$

Combining the summations and simplifying gives

$$\begin{aligned} g_n &= \alpha_k g_{j_0} g_{n-j_0-k} + \sum_{m=1}^{k-1} \left(\alpha_m g_{j_0} + \sum_{i=1}^{k-m} \alpha_{i+m} g_{j_0-i} \right) g_{n-j_0-m} \\ &= \alpha_k g_{j_0} g_{n-j_0-k} + \sum_{m=1}^{k-1} \sum_{i=0}^{k-m} \alpha_{i+m} g_{j_0-i} g_{n-j_0-m}. \end{aligned}$$

We can break the summation over m into the cases where $m = 1$ and $2 \leq m \leq k-1$, change the limits of the summations over i and m, and rearrange to find that

$$\begin{aligned} g_n &= \alpha_k g_{j_0} g_{n-j_0-k} + \sum_{m=1}^{1} \sum_{i=0}^{k-m} \alpha_{i+m} g_{j_0-i} g_{n-j_0-m} + \sum_{m=2}^{k-1} \sum_{i=0}^{k-m} \alpha_{i+m} g_{j_0-i} g_{n-j_0-m} \\ &= \alpha_k g_{j_0} g_{n-j_0-k} + \sum_{i=0}^{k-1} \alpha_{i+1} g_{j_0-i} g_{n-(j_0+1)} + \sum_{m=2}^{k-1} \sum_{i=0}^{k-m} \alpha_{i+m} g_{j_0-i} g_{n-j_0-m} \\ &= \sum_{i=1}^{k} \alpha_i g_{j_0-(i-1)} g_{n-(j_0+1)} + \sum_{m=1}^{k-2} \sum_{i=0}^{k-m-1} \alpha_{i+m+1} g_{j_0-i} g_{n-j_0-(m+1)} + \alpha_k g_{j_0} g_{n-j_0-k} \\ &= \sum_{i=1}^{k} \alpha_i g_{j_0+1-i} g_{n-(j_0+1)} + \sum_{m=1}^{k-2} \sum_{i=1}^{k-m} \alpha_{i+m} g_{j_0-(i-1)} g_{n-(j_0+1)+m} + \alpha_k g_{j_0} g_{n-j_0-k}. \end{aligned}$$

The third term can be expressed as $\alpha_k g_{j_0} g_{n-j_0-k} = \alpha_{(k-1)+1} g_{(j_0+1)-1} g_{n-(j_0+1)+(k-1)}$, which satisfies the summation in the second term for $m = k-1$. Absorbing this term

into the summation and applying recurrence (5.2), we have

$$\begin{aligned} g_n &= \left(\sum_{i=1}^{k} \alpha_i g_{j_0+1-i}\right) g_{n-(j_0+1)} + \sum_{m=1}^{k-1} \sum_{i=1}^{k-m} \alpha_{i+m} g_{j_0+1-i} g_{n-(j_0+1)+m} \\ &= g_{j_0+1} g_{n-(j_0+1)} + \sum_{m=1}^{k-1} \sum_{i=1}^{k-m} \alpha_{i+m} g_{j_0+1-i} g_{n-(j_0+1)+m}. \quad \square \end{aligned}$$

This theorem can be used to relate a sequence to its characteristic sequence. Let $\langle g_n \rangle$ satisfy the homogeneous linear recurrence $g_n = \sum_{i=1}^{k} \alpha_i g_{n-i}$ for $n \geq n_0$. Let the boundary conditions for g_n be defined for $n_0 - k - \delta \leq n \leq n_0 - 1$, where δ is some nonnegative integer. If we let $j = n - n_0 + 1$, we obtain

$$g_n = g_{n-n_0+1} g_{n_0-1} + \sum_{m=1}^{k-1} \sum_{i=1}^{k-m} \alpha_{i+m} g_{n-n_0+1-i} g_{n_0-1-m}.$$

Thus we find that the sequence $\langle g_n \rangle$ is completely defined by its characteristic generating function and its boundary conditions.

From a more practical standpoint, this theorem is useful for relating a term in a sequence to any previous value in the sequence. It can be used to simplify occurrences of a sequence where the index is a sum of two or more variables, such as t_{n+m} or t_{3n}.

Example 5.1. Consider the Fibonacci numbers, which satisfy the homogeneous recurrence $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. The characteristic sequence for the Fibonacci numbers is defined by

$$\mathcal{F}_n = \begin{cases} \mathcal{F}_{n-1} + \mathcal{F}_{n-2}, & \text{if } n \geq 1; \\ 1, & \text{if } n = 0; \\ 0, & \text{if } n = -1. \end{cases}$$

By Theorem 5.1 we have

$$F_n = \mathcal{F}_j F_{n-j} + \mathcal{F}_{j-1} F_{n-j-1}. \tag{5.3}$$

The Fibonacci numbers are obtained by shifting their characteristic sequence, that is $F_n = \mathcal{F}_{n-1}$ for $n \geq 0$. Thus we can substitute for $\mathcal{F}_j$ and $\mathcal{F}_{j-1}$ in (5.3) to find that

$$F_n = F_{j+1} F_{n-j} + F_j F_{n-j-1}. \quad \blacksquare$$

This identity for the Fibonacci numbers appears frequently in the literature. Among its applications it has been used to prove that F_{kn} is a multiple of F_n [GRAHAM, page 280]. Evaluating F_{kn} gives

$$F_{kn} = F_{n+1} F_{kn-n} + F_n F_{kn-n-1}.$$

Using $k = 2$ as a base case, the result can be proved by induction on k.

We now consider some properties of sequences defined by second order linear recurrences with constant coefficients.

Theorem 5.2. If $\langle t_n \rangle$ satisfies the linear homogeneous recurrence of order 2

$$t_n = \alpha t_{n-1} + \beta t_{n-2},$$

then there exists some constant γ such that

$$t_{n+1}t_{n-1} - t_n^2 = \gamma(-\beta)^n.$$

Proof. Applying the recurrence for t_n we find that

$$\begin{aligned} t_{n+1}t_{n-1} - t_n^2 &= (\alpha t_n + \beta t_{n-1})t_{n-1} - (\alpha t_{n-1} + \beta t_{n-2})t_n \\ &= \beta t_{n-1}^2 - \beta t_n t_{n-2} \\ &= -\beta(t_n t_{n-2} - t_{n-1}^2). \end{aligned} \tag{5.4}$$

We can find the minimum $n = n_0$ such that the recurrence $t_{n_0+1} = \alpha t_{n_0} + \beta t_{n-1}$ holds for all $n \geq n_0$. Let $t_{n_0+1}t_{n_0-1} - t_{n_0}^2 = \gamma(-\beta)^{n_0}$. Thus $\gamma = (t_{n_0+1}t_{n_0-1} - t_{n_0}^2)/(-\beta)^{n_0}$. Applying (5.4) $n - n_0$ times we find that

$$t_{n+1}t_{n-1} - t_n^2 = (-\beta)^{n-n_0}(t_{n_0+1}t_{n_0-1} - t_{n_0}^2) = \gamma(-\beta)^n$$

for any $n \geq n_0$. □

Example 5.2. Consider the Fibonacci numbers, which satisfy the homogeneous linear recurrence $F_n = F_{n-1} + F_{n-2}$. By Theorem 5.2, $F_{n+1}F_{n-1} - F_n^2 = \gamma(-1)^n$. The recurrence for F_{n+1} holds for $n \geq 1$. Thus $\gamma = (F_2F_0 - F_1^2)/(-1) = 1$ and

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n. \quad \blacksquare$$

This equation is known as Cassini's identity [GRAHAM, KNUTH]. We made frequent use of it in Chapter 2 to simplify the results of hybrid summations involving Fibonacci numbers.

We can generalize the results of Theorem 5.2. Applying the recurrence to the expression $t_{m+1}t_{n-1} - t_m t_n$, we find that

$$\begin{aligned} t_{m+1}t_{n-1} - t_m t_n &= (\alpha t_m + \beta t_{m-1})t_{n-1} - t_m(\alpha t_{n-1} + \beta t_{n-2}) \\ &= \beta t_{m-1}t_{n-1} - \beta t_m t_{n-2} \\ &= -\beta(t_m t_{n-2} - t_{m-1}t_{n-1}). \end{aligned}$$

As a direct consequence we have that

$$t_{m+1}t_{n-1} - t_m t_n = (-\beta)^c(t_{m-c+1}t_{n-c-1} - t_{m-c}t_{n-c}),$$

for positive integer c. By choosing an appropriate value of m or n, we can define the constant γ as we did in Theorem 5.2 and find that

$$t_{m+1}t_{n-1} - t_m t_n = \gamma(-\beta)^{\min(m,n)}.$$

As a further extension, we find that if s_n and t_n both satisfy the second order homogeneous recurrence $f_n = \alpha f_{n-1} + \beta f_{n-2}$ then

$$\begin{aligned} s_{m+1}t_{n-1} - s_m t_n &= (\alpha s_m + \beta s_{m-1})t_{n-1} - s_m(\alpha t_{n-1} + \beta t_{n-2}) \\ &= \beta s_{m-1}t_{n-1} - \beta s_m t_{n-2} \\ &= -\beta(s_m t_{n-2} - s_{m-1}t_{n-1}). \end{aligned}$$

This leads to the identity

$$s_{m+1}t_{n-1} - s_m t_n = \gamma(-\beta)^{\min(m,n)},$$

for some constant γ.

Theorems 5.1 and 5.2 provide good examples of why it is desirable to treat the rational generating functions as an algebraic structure. Application of these theorems to the Fibonacci numbers have been widely cited in the literature as interesting and useful properties of the Fibonacci numbers. However, by examining the larger set of nontrivial rational generating functions, we are able to show that these properties apply to more than one particular sequence.

The final relationship within a sequence that we will examine is the simplification of expressions involving sequences that satisfy homogeneous linear recurrences. Let $\langle t_n \rangle$ be a sequence that satisfies the kth order homogeneous linear recurrence $t_n = \sum_{i=1}^{k} \alpha_i t_{n-i}$. Given an expression involving a fixed number $m > k$ of consecutive terms from the sequence $\langle t_n \rangle$, we can simplify the expression to involve only k consecutive terms from the sequence $\langle t_n \rangle$. First, pick the k values $t_n, \ldots, t_{n-k+1}$ that the answer is to involve. Repeated applications of the recurrence can be used to convert occurrences of the sequence to the desired index values and leads to the following algorithm.

1. While there are indices greater than n repeat the following step: Find the largest index greater than n and call this t_{n+c}. Apply the recurrence to express t_{n+c} as a function of $t_{n+c-1}, \ldots, t_{n+c-k}$.

2. While there are indices less than $n - k + 1$ repeat the following step: Find the term with the smallest index less than $n - k + 1$ and call this t_{n-b}. Rearrange the recurrence and substitute to express this term as $t_{n-b} = (t_{n-b+k} - \sum_{i=1}^{k-1} \alpha_i t_{n-b+k-i})/\alpha_k$.

This algorithm gives a method to reduce the number of terms from a sequence in an expression. While there could be an expression involving fewer terms from the sequence, the method guarantees that no more than k terms are needed to express a sequence defined by an order k recurrence.

Example 5.3. Consider the expression

$$s_n = n^2 F_{n+1} + (n^2 - n)F_n + (2n - 1)F_{n-1} + 3F_{n-2}.$$

We can simplify s_n to an expression that involves only F_n and F_{n-1}. F_{n+1} can be expressed as $F_n + F_{n-1}$ and F_{n-2} as $F_n - F_{n-1}$. Substituting we find that

$$\begin{aligned} s_n &= n^2(F_n + F_{n-1}) + (n^2 - n)F_n + (2n-1)F_{n-1} + 3(F_n - F_{n-1}) \\ &= (2n^2 - n + 3)F_n + (n^2 + 2n - 4)F_{n-1}. \quad \blacksquare \end{aligned}$$

As a result of this simplification algorithm, we see that expressions involving the product of two or more sequences can have a large number of terms. An expression involving the sum of products of two sequences, one defined by an order k recurrence and the other by an order m recurrence, can have as many as km terms that are products between terms of the two sequences. Expressions involving products of terms from a single sequence may have fewer terms in their simplified form. If an expression involves the product of m terms defined by an order k recurrence, it can be simplified to include products of the form $t_{n-\alpha_1} t_{n-\alpha_2} \cdots t_{n-\alpha_m}$ for $0 \leq \alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_m \leq k-1$. For example, an expression involving the product of three Fibonacci numbers when simplified would only need the four products $F_nF_nF_n$, $F_nF_nF_{n-1}$, $F_nF_{n-1}F_{n-1}$, and $F_{n-1}F_{n-1}F_{n-1}$ instead of the eight products that three different order 2 sequences might require.

5.1.2. Relationships Between Sequences

When manipulating sequences with nontrivial rational generating functions, it is not uncommon to encounter generating functions with common factors in their denominators. In such cases, it is sometimes desirable to find an extended closed form for one of these sequences that involves terms from the other sequences. That is, given a sequence $\langle f_n \rangle$ encoded by the nontrivial rational generating function $F(z)$ and a nontrivial rational generating function $G(z)$ that has common factors in its denominator with the denominator of $F(z)$, we wish to find a closed form expression for g_n that involves terms from $\langle f_n \rangle$.

We consider first the relationship between a sequence and its characteristic sequence. Let $\langle g_n \rangle$ be the sequence encoded by the nontrivial rational function $G(z)$ and let $\langle \mathcal{g}_n \rangle$ be its characteristic sequence encoded by the generating function $\mathcal{G}(z)$. If $G(z) = p(z)/q(z)$, we can use (2.15) to express g_n as a function of terms from $\langle \mathcal{g}_n \rangle$. A more difficult problem is to find an extended closed form expression for $\mathcal{g}_n$ that involves terms from $\langle g_n \rangle$.

Since the generating function $\mathcal{G}(z) = \text{lcoeff}(q(z))z^{\text{ldegree}(q(z))}/q(z)$, we find that $\mathcal{g}_n = \text{lcoeff}(q(z))q_{n-\text{ldegree}(q(z))}$, where $\langle q_n \rangle$ is the sequence encoded by the generating function $1/q(z)$. To find $1/q(z)$ we can use partial fractions on z to get

$$\frac{1}{p(z)q(z)} = \frac{s(z)}{p(z)} + \frac{t(z)}{q(z)}.$$

Multiplying through by $p(z)$ we find that

$$\frac{1}{q(z)} = s(z) + \frac{t(z)p(z)}{q(z)}.$$

The generating function $t(z)p(z)/q(z)$ encodes the sequence

$$r_n = \sum_{k=0}^{\text{degree}(t(z))} \text{coeff}(t(z),k)g_{n-k}.$$

The polynomial $s(z)$ encodes a finite sequence. Thus we find that

$$q_n = \begin{cases} r_n, & \text{if } n > \text{degree}(s(z)); \\ r_n + \text{coeff}(s(z),n), & \text{if } 0 \le n \le \text{degree}(s(z)). \end{cases}$$

The polynomial $s(z)$ serves to adjust the boundary conditions of q_n. Shifting and scaling q_n gives an extended closed form for g_n that involves a linear combination of terms from $\langle g_n \rangle$.

Example 5.4. Consider the generating function $G(z) = z(1+z)/(1-z)^3$ which encodes the sequence $\langle g_n \rangle$. We want to find a closed form expression for $\langle g_n \rangle$, the characteristic sequence of $\langle g_n \rangle$, that involves terms from $\langle g_n \rangle$. The characteristic generating function is $\mathcal{G}(z) = 1/(1-z)^3$. Applying partial fractions over z we get

$$\frac{1}{z(1+z)(1-z)^3} = \frac{1}{8}\frac{8+7z}{z(1+z)} + \frac{1}{8}\frac{17-20z+7z^2}{(1-z)^3}.$$

Multiplying through by $z(1+z)$ gives

$$\frac{1}{(1-z)^3} = \frac{8+7z}{8} + \frac{17-20z+7z^2}{8}\frac{z(1+z)}{(1-z)^3}.$$

Expanding the generating functions we find that

$$g_n = \begin{cases} (17g_n - 20g_{n-1} + 7g_{n-2})/8, & \text{if } m \ge 2; \\ (7 + 17g_1 - 20g_0)/8, & \text{if } n = 1; \\ (8 + 17g_0)/8, & \text{if } n = 0. \end{cases} \tag{5.5}$$

The sequence encoded by $G(z)$ is

$$g_n = n^3$$

and the sequence encoded by $\mathcal{G}(z)$ is

$$g_n = \frac{n^2+3n+2}{2}.$$

In particular, $g_0 = 1$ and $g_1 = 3$. To verify (5.5) we can substitute values of $\langle g_n \rangle$ to find that

$$g_n = \frac{17n^3 - 20(n-1)^3 + 7(n-2)^3}{8} = \frac{n^2+3n+2}{2},$$

$$g_1 = \frac{7+17(1)^3}{8} = 3, \quad \text{and}$$

$$g_0 = \frac{8}{8} = 1. \quad \blacksquare$$

We now know how to express a sequence as a function of its characteristic sequence and a characteristic sequence as a function of a sequence that satisfies its homogeneous linear recurrence. In the remaining relationships, we will consider how to relate sequences defined by characteristic generating functions. That is, all generating functions will have the form $1/(1-\sum_{i=1}^{k}\alpha_i z^i)$. Given that $\mathcal{G}(z)$ is related to $\mathcal{F}(z)$, we want to find an extended closed form expression for $\langle g_n\rangle$, the sequence encoded by $\mathcal{G}(z)$, that involves terms from $\langle f_n\rangle$, the sequence encoded by $\mathcal{F}(z)$. Once the sequences encoded by the characteristic generating functions have been related, we can extend the relationship to any two sequences characterized by those generating functions.

We first consider the case where $\mathcal{G}(z)$ is a factor of $\mathcal{F}(z)$. Let $\mathcal{F}(z) = 1/(p(z)q(z))$, where $p(z)$ and $q(z)$ are polynomials of degree at least one and let $\mathcal{G}(z) = 1/p(z)$. We see that

$$\mathcal{G}(z) \;=\; \frac{1}{p(z)} \;=\; q(z)\,\mathcal{F}(z).$$

Thus, by (2.15)

$$g_n \;=\; \sum_{k=0}^{\text{degree}(q(z))} \text{coeff}(q(z),k)\,f_{n-k}, \tag{5.6}$$

and we find that g_n can be expressed as a linear combination of terms from $\langle f_n\rangle$.

Example 5.5. Express the values ϕ^n and $\hat{\phi}^n$, where $(1-\phi z)(1-\hat{\phi}z) = 1-z-z^2$, as functions of the Fibonacci numbers. The characteristic generating function of the Fibonacci numbers is $\mathcal{F}(z) = 1/(1-z-z^2)$ and encodes the sequence $\langle F_{n+1}\rangle$. The sequence $\langle\phi^n\rangle$ is encoded by the generating function $1/(1-\phi z) = (1-\hat{\phi}z)\,\mathcal{F}(z)$. Thus, by (5.6)

$$\phi^n \;=\; F_{n+1} - \hat{\phi}F_n \;=\; F_{n+1} - (1-\phi)F_n \;=\; \phi F_n + F_{n-1}.$$

Likewise, the sequence $\langle\hat{\phi}^n\rangle$ is encoded by the generating function $1/(1-\hat{\phi}z) = (1-\phi z)\,\mathcal{F}(z)$ and

$$\hat{\phi}^n \;=\; F_{n+1} - \phi F_n \;=\; F_{n+1} - (1-\hat{\phi})F_n \;=\; \hat{\phi}F_n + F_{n-1}. \quad \blacksquare$$

We next consider the case where $\mathcal{G}(z)$ is a power of $\mathcal{F}(z)$. We can differentiate $\mathcal{F}(z)$ to obtain a generating function whose denominator has $\mathcal{G}(z)$ as a factor and then relate the new generating function to $\mathcal{G}(z)$. This approach, however, leads us to consider a more general situation. Let $\mathcal{F}(z) = 1/p(z)$ and $\mathcal{G}(z) = 1/q(z)$. Let each factor of $q(z)$ be a factor of $p(z)$ and let the degree of at least one factor in $q(z)$ be greater than the degree of that same factor in $p(z)$. We can use the following steps to express g_n as a function of terms from $\langle f_n\rangle$.

1. Let $S(z) = \mathcal{F}(z)$.
2. Let $i = 0$.
3. Repeat until $q(z)$ is a factor of denom$(S(z))$.

(a) Let $S(z) = dS(z)/dz$.

(b) Let $i = i + 1$.

4. Let $s_n = (n+i)^{\underline{i}} f_{n+i}$.

5. Use the previous method to express g_n as a function of s_n.

The first three steps of this algorithm calculate

$$S(z) = \frac{d^{(i)}}{dz} \frac{1}{p(z)}.$$

Every time a rational function is differentiated, the degree of the factors in the denominator increase. Thus we differentiate until the denominator of $S(z)$ contains $q(z)$ as a factor. The generating function $S(z)$ is $\sum_n s_n z^n$. After step 3 we find that

$$S(z) = \frac{d^{(i)}}{dz} \sum_n f_n z^n = \sum_n \frac{d^{(i)}}{dz} f_n z^n = \sum_n n^{\underline{i}} f_n z^{n-i}.$$

Thus the coefficient of z^n in $S(z)$ is $s_n = (n+i)^{\underline{i}} s_{n+i}$, which is determined in step 4. In step 5, we find an extended closed form expression for g_n that involves a linear combination of terms from $\langle s_n \rangle$. Since s_n can be defined in terms of the sequence $\langle f_n \rangle$, we can substitute to find an extended closed form expression for g_n that involves terms from $\langle f_n \rangle$. This expression will be a linear combination of terms that involve products between polynomials and terms from $\langle f_n \rangle$.

Example 5.6. Consider the generating function $G(z) = 1/(1-z-z^2)^2$. We wish to find an extended closed form expression for g_n that involves the Fibonacci numbers. The generating function

$$S(z) = \frac{d}{dz} \frac{1}{1-z-z^2} = \frac{1+2z}{(1-z-z^2)^2}$$

encodes the sequence $\langle (n+1)F_{n+2} \rangle$. We now want to express the characteristic sequence $\langle g_n \rangle$ as a function of terms from this sequence. To do so, we use partial fractions to obtain

$$\frac{1}{(1+2z)(1-z-z^2)^2} = \frac{16}{25(1+2z)} + \frac{9+14z-12z^2-8z^3}{25(1-z-z^2)^2}.$$

Multiplying by $1+2z$ we find that

$$G(z) = \frac{16}{25} + \frac{9+14z-12z^2-8z^3}{25} S(z).$$

Expanding generating functions we get the extended closed form expression

$$g_n = \frac{9s_n + 14s_{n-1} - 12s_{n-2} - 8s_{n-3}}{25}.$$

We can substitute $s_n = (n+1)F_{n+2}$ and simplify to obtain

$$\begin{aligned} g_n &= \frac{9(n+1)F_{n+2} + 14nF_{n+1} - 12(n-1)F_n - 8(n-2)F_{n-1}}{25} \\ &= \frac{(4n+6)F_n + (3n+5)F_{n-1}}{5}. \quad \blacksquare \end{aligned}$$

Using (2.15) and the three methods discussed here, we have the tools needed to relate any two sequences encoded by nontrivial rational generating functions with common factors in their characteristic generating functions. If none of the methods are directly applicable, partial fractions can be used to isolate common factors in their characteristic generating functions. The previous methods can then be applied to these common factors to relate the sequences.

5.2. Summation

We now consider the summation of sequences from RAT_SEQ. First we determine the types of sums in RAT_SEQ that can be solved. We then use the partial fraction expansions of Appendix A to describe the form of the solutions to these sums and examine ways to express their results.

5.2.1. Closure of Summation

We can use the convolution to show that summation is closed over RAT_SEQ.

Theorem 5.3. Let $\langle a_{i,j} \rangle \in RAT\_SEQ$ for i from 1 to m. The sequence $\langle s_n \rangle$ defined by

$$s_n = \sum_{\alpha_1+\alpha_2+\cdots+\alpha_m=n} \prod_{i=1}^{m} a_{i,\alpha_i}$$

is a member of RAT_SEQ.

Proof. The generating functions $A_i(z)$ of the sequences $\langle a_{i,j} \rangle$ are in RAT. By (2.17) the generating function of $\langle s_n \rangle$ is

$$S(z) = \prod_{i=1}^{m} A_i(z).$$

Since RAT is closed over multiplication, $S(z)$ is in RAT and the sequence $\langle s_n \rangle$ is in RAT_SEQ. $\square$

While this seems to be an obvious result, its importance becomes more apparent when it is considered along with Theorems 4.4 and 4.5. We find that along with simple sums and convolutions, hybrid sums and hybrid convolutions involving only sequences from RAT_SEQ have their solution in RAT_SEQ. In addition, convolutions involving multiple hybrid terms also have a solution in RAT_SEQ. Thus, for any summation or convolution involving only sequences from RAT_SEQ we know that the generating function of the solution will be found in RAT.

5.2.2. Form of Solution

For any summation or convolution within RAT_SEQ, if the solution has a nontrivial generating function then we need only to expand the generating function in order to find a closed form expression for the solution. We have seen that such an expression exists for any nontrivial rational generating function with coefficients from the field of complex numbers.

Often it is more informative if the result of a summation can be expressed in terms of the functions in its summand. We will consider the various types of summations within RAT_SEQ and show that their solutions have extended closed forms involving the sequences in the summand.

Summations

We first consider the solution to simple sums and then extend the result to hybrid sums.

Theorem 5.4. Let

$$s_n = \sum_{k=0}^{n} a_k,$$

where $\langle a_n \rangle$ is in RAT_SEQ. If $\langle a_k \rangle$ has a nontrivial rational generating function then s_n has an extended closed form expression involving terms from the sequence $\langle a_n \rangle$.

Proof. By (2.17), the generating function of $\langle s_n \rangle$ is $S(z) = A(z)/(1-z)$, where $A(z)$ is the nontrivial rational generating function of $\langle a_n \rangle$. We consider the case where $A(z)$ does not have $1/(1-z)$ as a factor and the case where $A(z)$ has $1/(1-z)$ as a factor.

Case 1. $A(z)$ does not have a factor of $1-z$ in its denominator. Let $A(z) = p(z)/q(z)$, where

$$p(z) = \sum_{i=0}^{\text{degree}(p(z))} p_i z^i \quad \text{and} \quad q(z) = \sum_{i=0}^{\text{degree}(q(z))} q_i z^i.$$

Using the partial fraction expansion given in (A.8),

$$S(z) = p(z)\left(\frac{1}{q(1)(1-z)} + \frac{r(z)}{q(1)q(z)}\right) = \frac{p(z)}{q(1)(1-z)} + \frac{r(z)}{q(1)}A(z),$$

where

$$r(z) = \sum_{i=1}^{\text{degree}(q(z))-1} \sum_{k=0}^{i} q_{\text{degree}(q(z))+k-i} z^k.$$

The first term encodes a sequence whose closed form is a constant. The second term is a product between the polynomial $r(z)/q(1)$ and the generating function $A(z)$. The sequence it encodes is a linear combination of terms from the sequence $\langle a_n \rangle$.

Case 2. $A(z)$ has a factor of $1-z$ in its denominator. Let $A(z) = p(z)/((1-z)^m q(z))$, where

$$p(z) = \sum_{i=0}^{\text{degree}(p(z))} p_i z^i \quad \text{and} \quad q(z) = \sum_{i=0}^{\text{degree}(q(z))} q_i z^i.$$

Using the partial fraction expansion given in (A.9),

$$S(z) = p(z)\left(\frac{1}{q(1)\,(1-z)^{m+1}} + \frac{r(z)}{(1-z)^m q(1)\,q(z)}\right) = \frac{p(z)}{q(1)\,(1-z)^{m+1}} + \frac{r(z)}{q(1)}A(z),$$

where

$$r(z) = \sum_{i=1}^{\text{degree}(q(z))-1} \sum_{k=0}^{i} q_{\text{degree}(q(z))+k-i} z^k.$$

The first term encodes a sequence whose closed form is a polynomial. As in Case 1, the second term encodes a sequence whose extended closed form involves terms from the sequence $\langle a_n \rangle$. □

We note that if the summand is a finite sequence it will have only a finite number of nonzero terms. The generating function of the solution will be a polynomial in z divided by $1 - z$. The closed form of the sequence encoded by this generating function is a constant and reflects the fact that once the last nonzero term in the summand has been added in, the sum remains constant.

We now use Theorem 5.4 to determine the form of the solution to a hybrid sum.

Corollary 5.4.1. Let

$$s_n = \sum_{k=0}^{n} a_{1,k} a_{2,k} \cdots a_{m,k},$$

where each sequence $\langle a_{i,n} \rangle$ for $1 \leq i \leq n$ is in RAT_SEQ. If the hybrid sequence $\langle a_{1,k} a_{2,k} \cdots a_{m,k} \rangle$ is an infinite sequence, then s_n has an extended closed form expression involving terms from the m sequences $\langle a_{1,n} \rangle$ to $\langle a_{m,n} \rangle$.

Proof. The sequence $\langle b_k \rangle = \langle a_{1,k} a_{2,k} \cdots a_{m,k} \rangle$ has an infinite number of nonzero terms. By Theorem 4.5 we find that $B(z)$, the generating function of $\langle b_k \rangle$, is in RAT. Therefore, by Theorem 5.4, s_n has an extended closed form expression involving terms from the sequence $\langle b_n \rangle$. Since $b_n = a_{1,n}, a_{2,n}, \ldots, a_{m,n}$, we can substitute for b_n to get an extended closed form expression involving terms from the sequences $\langle a_{1,n} \rangle$ to $\langle a_{m,n} \rangle$. □

As a result of these two proofs, we can determine the form of the solution to any simple sum or hybrid sum involving sequences from RAT_SEQ. Let the summand have $m \geq 1$ factors, each a nontrivial sequence from RAT_SEQ. Let each sequence $\langle a_{i,n} \rangle$ satisfy an order k_i recurrence. We can see from the proofs that the solution will involve products between terms from the m sequences. The expression can be simplified so

that each sequence has only k_i terms in the solution. Thus we find that the form of the solution is

$$s_n = \sum_{i_1=0}^{k_1-1} \sum_{i_2=0}^{k_2-1} \cdots \sum_{i_m=0}^{k_m-1} c_{i_1,i_2,\ldots,i_m} a_{1,n-i_1} a_{2,n-i_2} \cdots a_{m,n-i_m} + p(n), \qquad (5.7)$$

where $p(n)$ is a polynomial in n and each of the values $c_{i_1,i_2,\ldots,i_m}$ is constant with respect to n.

Of interest are the summands that cause $p(n)$ to be a constant and those that cause $p(n)$ to vary with n. If we consider the generating function of the summand, we see that $p(n)$ is not constant only when the generating function has a factor of $1-z$ in its denominator. If this is to occur, the closed form for the summand must include a polynomial term. In terms of the set $\mathcal{T}_{\mathcal{F}}$, which defines the closed form for members of RAT_SEQ, a term of the form $q(n)$ must be present. The exponential factor of this term is one. For the simple sum, such summands are obvious from inspection of their generating function. They can be much more difficult to find for hybrid sums.

Example 5.7. Consider the sequence $\langle s_n \rangle = \langle F_n^4 \rangle$. The sequence is nonzero for all $n \geq 1$ and its closed form is

$$\begin{aligned} s_n &= \left(\frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} \right)^4 \\ &= \frac{\phi^{4n} - 4\phi^{3n}\hat{\phi}^n + 6\phi^{2n}\hat{\phi}^{2n} - 4\phi^n\hat{\phi}^{3n} + \hat{\phi}^{4n}}{25} \\ &= \frac{\phi^{4n} + 4\phi^{2n} + 6 + 4\hat{\phi}^{2n} + \hat{\phi}^{4n}}{25}, \end{aligned}$$

where $\phi = (1+\sqrt{5})/2$ and $\hat{\phi} = (1-\sqrt{5})/2$. The third term of the numerator is a trivial polynomial that occurs because $\phi^2\hat{\phi}^2 = 1$. The generating function of $\langle s_n \rangle$ is

$$\begin{aligned} S(z) &= \frac{1}{25}\left(\frac{2}{2-7z-3\sqrt{5}z} - \frac{8}{2+3z+\sqrt{5}z} + \frac{6}{1-z} \right. \\ &\qquad \left. - \frac{8}{2+3z-\sqrt{5}z} + \frac{2}{2-7z+3\sqrt{5}z} \right) \\ &= \frac{z(1+z)(1-5z+z^2)}{(1-z)(1-7z+z^2)(1+3z+z^2)}. \end{aligned}$$

The $1-z$ factor in the denominator is contributed by the polynomial term in the closed form expression for s_n. ∎

From the example we see that when a hybrid sequence has a polynomial in its closed form expression its generating function will involve powers of $1-z$ in its denominator. Consider the hybrid term $b_n = a_{1,n}a_{2,n}\cdots a_{m,n}$. If $a_{i,n}$ has a closed form expression

$$a_{i,n} = \sum_{j_i=1}^{t_i} p_{i,j_i}(n)\theta_{i,j_i}^n$$

involving t_i terms, then

$$b_n = \prod_{i=1}^{m} \sum_{j_i=1}^{t_i} p_{i,j_i}(n)\theta_{i,j_i}^n.$$

The closed form expression for b_n has $t_1 t_2 \cdots t_m$ terms. If any one of these terms is a polynomial, then the generating function of $\langle b_n \rangle$ will have a power of $1 - z$ in its denominator. For one of the terms to be a polynomial, there must exist some set of values $j_1, j_2, \ldots, j_m$ for $1 \leq j_i \leq t_i$ such that the product $\theta_{1,j_1} \cdots \theta_{m,j_m} = 1$. For the sequence $\langle F_n^4 \rangle$, the product was $\phi\phi\hat{\phi}\hat{\phi} = 1$.

Russell has developed a method based on undetermined coefficients for solving indefinite hybrid sums involving sequences defined by homogeneous linear recurrences [RUSSELL 79, RUSSELL 82]. He defines a *standard sum* to be an expression of the form (5.7), where $p(n)$ is a constant. He proves that if you are able to solve for the constants $c_{i_1,i_2,\ldots,i_m}$ in the standard sum, then you have found the solution to the indefinite sum. He also analyzes those summations for which a solution for the constants cannot be obtained. While he is unable to describe the solution of these sums, he is able to show that for an m factor summand $a_{1,n} \cdots a_{m,n}$, there is a root θ_i of the characteristic polynomial of each $\langle a_{i,n} \rangle$, such that $\theta_1 \cdots \theta_m = 1$. Since the characteristic polynomial of a sequence corresponds to its characteristic generating function, we find that the sum of products expansion of the closed forms of these summands have at least one polynomial term. The polynomial $p(n)$ in the solution (5.7) of these sums has degree at least one. Thus, adding the nth term to the summation contributes to the value of $p(n)$. This requires that the coefficients of $p(n)$ be included in the method of undetermined coefficients in order to find a correct solution. Since Russell does not include the coefficients of $p(n)$, his method does not find a correct solution.

To overcome some of the limitations in his method, Russell studied the special cases of sums of sequences defined by second order linear recurrences and sums of the squares of such sequences [RUSSELL 81]. He derived closed form expressions for the sums based on the coefficients in the recurrences. By relating the coefficients, he was able to determine those cases where the closed form expression of the summand involves a polynomial and introduces a polynomial into the solution.

Convolutions

We can also find extended closed form expressions for convolutions that involve factors from the summand.

Theorem 5.5. Let

$$s_n = \sum_{\alpha_1 + \cdots + \alpha_m = n} a_{1,\alpha_1} a_{2,\alpha_2} \cdots a_{m,\alpha_m},$$

where each sequence $\langle a_{i,k} \rangle$ for $1 \leq i \leq m$ is in RAT_SEQ. If $\langle s_n \rangle$ is an infinite sequence, then s_n has an extended closed form involving terms from the sequences $\langle a_{i,k} \rangle$.

Proof. By (2.17), the sequence $\langle s_n \rangle$ is encoded by the generating function

$$S(z) = A_1(z)A_2(z) \cdots A_m(z),$$

where $A_i(z)$ is the generating function of $\langle a_{i,k} \rangle$. If $\langle s_n \rangle$ is an infinite sequence, then $S(z)$ is a nontrivial rational generating function. We can use partial fractions on z to expand $S(z)$ as

$$\sum_j S_j(z).$$

Each $S_j(z)$ can be related to one or more of the generating functions $A_i(z)$. If none of the generating functions $A_i(z)$ have common factors in their characteristic generating functions, then each $S_j(z)$ can be related to exactly one $A_i(z)$. The sequence encoded by each $S_i(z)$ can be expressed as a linear combination of terms from the sequence encoded by the related $A_i(z)$. Thus, we find that the extended closed form solution for s_n involves linear combinations of the sequences in the summand.

If any of the generating functions $A_i(z)$ have common factors in their characteristic generating functions, the extended closed form solution is more complicated. Each $S_j(z)$ will either be related to one sequence in the summand or to at least two sequences in the summand. In the first case, the sequence encoded by $S_j(z)$ can be expressed as a linear combination of terms from the related sequence. In the second case, we have to pick which $A_i(z)$ we want to relate with $S_j(z)$. If the characteristic generating function of $S_j(z)$ is a factor of the characteristic generating function of $A_i(z)$, we can express the sequence encoded by $S_j(z)$ as a linear combination of terms from $\langle a_{i,n} \rangle$. It is possible for the degree of the characteristic generating function of $S_j(z)$ to be greater than the degree of the corresponding factor in the characteristic generating function of $A_i(z)$. In this case, we differentiate $A_i(z)$ in order to relate it to $S_j(z)$. The extended closed form of the sequence encoded by $S_j(z)$ will involve a linear combination of terms that are products between polynomials and terms from $\langle a_{i,n} \rangle$. □

Example 5.8. Consider the convolution

$$s_n = \sum_{k=0}^{n} a_k b_{n-k},$$

where the generating function of $\langle a_k \rangle$ is $A(z) = 1/((1-\alpha z)(1-\beta z))$ and the generating function of $\langle b_k \rangle$ is $B(z) = 1/((1-\alpha z)(1-\gamma z))$ The generating function of the sequence $\langle s_n \rangle$ is

$$\begin{aligned} S(z) &= \frac{1}{(1-\alpha z)^2(1-\beta z)(1-\gamma z)} \\ &= \frac{\alpha^2(2\beta\gamma - \alpha\beta - \alpha\gamma)}{(\alpha-\gamma)^2(\alpha-\beta)^2(1-\alpha z)} + \frac{\alpha^2}{(\alpha-\gamma)(\alpha-\beta)(1-\alpha z)^2} \\ &\quad + \frac{\beta^3}{(\beta-\gamma)(\alpha-\beta)^2(1-\beta z)} + \frac{\gamma^3}{(\gamma-\beta)(\alpha-\gamma)^2(1-\gamma z)}. \end{aligned}$$

By (5.6), we find that $1/(1-\alpha z)$ encodes the sequence $\langle a_n - \beta a_{n-1} \rangle$, $1/(1-\beta z)$ encodes the sequence $\langle a_n - \alpha a_{n-1} \rangle$, and $1/(1-\gamma z)$ encodes the sequence $\langle b_n - \alpha b_{n-1} \rangle$. Differentiating, we find that the generating function $\alpha/(1-\alpha z)^2$ encodes the sequence

$\langle (n+1)(a_{n+1} - \beta a_n) \rangle$. Thus, the generating function $1/(1-\alpha z)^2$ encodes the sequence $\langle (n+1)(a_{n+1} - \beta a_n)/\alpha \rangle$. Expanding $S(z)$ we find that

$$\begin{aligned} s_n &= \frac{\alpha^2(2\beta\gamma - \alpha\beta - \alpha\gamma)}{(\alpha-\gamma)^2(\alpha-\beta)^2}(a_n - \beta a_{n-1}) + \frac{\alpha}{(\alpha-\gamma)(\alpha-\beta)}(n+1)(a_{n+1} - \beta a_n) \\ &\quad + \frac{\beta^3}{(\beta-\gamma)(\alpha-\beta)^2}(a_n - \alpha a_{n-1}) + \frac{\gamma^3}{(\gamma-\beta)(\alpha-\gamma)^2}(b_n - \alpha b_{n-1}). \quad \blacksquare \end{aligned}$$

5.3. Linear Recurrences

We have shown that homogeneous linear recurrences with constant coefficients define sequences in RAT_SEQ. Likewise, we can show that systems of homogeneous linear recurrences with constant coefficients define a system of sequences that are all members of RAT_SEQ. Using the methods of Chapter 2, we can find rational generating functions for the solutions in both cases. Expanding these generating functions gives a closed form for the solution to the corresponding recurrence.

It is much more difficult to find an extended closed form for recurrences involving known sequences from RAT_SEQ. Unlike summation, where we knew to look at the sequences defining the summand, with homogeneous recurrences we do not know which sequences to use. If such a closed form is desired, we are reduced to looking for sequences whose characteristic generating functions have common factors with the generating function we are trying to expand.

Of more interest are the solutions to inhomogeneous linear recurrences with constant coefficients and systems of these recurrences. In this section we will consider the solutions to such recurrences when the inhomogeneous part of the recurrences are from RAT_SEQ.

5.3.1. Single Linear Recurrences

We can show that if the inhomogeneous part of a linear recurrence with constant coefficients is in RAT_SEQ, then the solution to the recurrence is in RAT_SEQ.

Theorem 5.6. Let the sequence $\langle s_n \rangle$ be defined by the order k linear recurrence with constant coefficients

$$s_n = \sum_{i=1}^{k} \alpha_i s_{n-i} + a_i,$$

and at least k boundary conditions. If $\langle a_i \rangle \in RAT\_SEQ$ then $\langle s_n \rangle \in RAT\_SEQ$.

Proof. Let the boundary conditions be defined over $\gamma + 1 \leq n \leq \gamma + k + \delta$, for integer $\delta \geq 0$. Let $\langle t_n \rangle$ be the sequence defined by the recurrence for s_n and the boundary conditions of s_n over $\gamma + \delta + 1 \leq n \leq \gamma + k + \delta$. By (2.24), the generating function of $\langle t_n \rangle$ is

$$T(z) = \frac{\sum_{n=\gamma+\delta+1}^{\gamma+\delta+k} s_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} t_{\gamma+\delta+n} z^{\gamma+\delta+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i} + \frac{A(z) - \sum_{n=\gamma_a}^{\gamma+\delta+k} a_n z^n}{1 - \sum_{i=1}^{k} \alpha_i z^i},$$

where $A(z) = \sum_{n=\gamma_a}^{\infty} a_n z^n$ and $a_n = 0$ for $n < \gamma_a$. By (2.1) we find that

$$S(z) = \sum_{n=\gamma+1}^{\infty} s_n z^n = \sum_{n=\gamma+1}^{\gamma+\delta} s_n z^n + T(z). \tag{5.8}$$

The generating function $A(z)$ is a rational function, therefore $T(z)$ is a rational function and $S(z)$ is a rational function. Thus, we find that $\langle s_n \rangle \in RAT\_SEQ$. □

We can now consider the form of the solution to inhomogeneous linear recurrences with constant coefficients.

Theorem 5.7. Let the sequence $\langle s_n \rangle$ be defined by the order k linear recurrence with constant coefficients

$$s_n = \sum_{i=1}^{k} \alpha_i s_{n-i} + a_i,$$

and at least k boundary conditions. If $\langle a_i \rangle$ has a nontrivial rational generating function, then s_n has an extended closed form expression that involves terms from $\langle a_n \rangle$.

Proof. By (5.8) the generating function of the sequence $\langle s_n \rangle$ is

$$S(z) = \sum_{n=\gamma+1}^{\gamma+\delta} s_n z^n + \frac{\sum_{n=\gamma+\delta+1}^{\gamma+\delta+k} s_n z^n - \sum_{i=1}^{k} \alpha_i z^i \sum_{n=1}^{k-i} t_{\gamma+\delta+n} z^{\gamma+\delta+n}}{1 - \sum_{i=1}^{k} \alpha_i z^i} + \frac{A(z) - \sum_{n=\gamma_a}^{\gamma+\delta+k} a_n z^n}{1 - \sum_{i=1}^{k} \alpha_i z^i},$$

where $A(z) = \sum_{n=\gamma_a}^{\infty} a_n z^n$ and $a_n = 0$ for $n < \gamma_a$. If $A(z)$ is a nontrivial rational generating function then the third term in $S(z)$ can be related to the generating function $A(z)$. Thus, an extended closed form expression for s_n involving terms from $\langle a_n \rangle$ can be found. □

The sequence $\langle a_n \rangle$ can be generated using Theorems 4.4 and 4.5. Thus, if a_n is a hybrid term, we can substitute and find an extended closed form expression for s_n that involves terms from the factors of a_n.

5.3.2. Systems of Linear Recurrences

We can extend the results involving inhomogeneous linear recurrences to systems of inhomogeneous linear recurrences.

Theorem 5.8. Let the sequences $\langle t_{i,n} \rangle$ for $1 \leq i \leq m$ be defined by a set of m simultaneous linear recurrences with constant coefficients where the ith recurrence has the form (3.1). If each sequence $\langle a_{i,n} \rangle$ is in RAT_SEQ and the system of recurrences can be solved, then $\langle t_{i,n} \rangle$ is in RAT_SEQ.

Proof. By Theorem 3.1 we have seen that the generating function of the sequence $\langle t_{i,n} \rangle$ has the form

$$T_i(z) = Q_i(z) + \sum_{j=1}^{m} g_{i,j}(z) A_j(z), \tag{5.9}$$

where $Q_i(z)$ and each $g_{i,j}(z)$ are rational functions, and the $A_j(z)$ are the generating functions of the sequences $\langle a_{j,n} \rangle$. If each $\langle a_{j,n} \rangle \in RAT\_SEQ$, then each $A_j(z) \in RAT$. Thus, each $T_i(z) \in RAT$ and each $\langle t_{i,n} \rangle \in RAT\_SEQ$. □

We now look at the extended closed form expressions for the solutions to systems of inhomogeneous linear recurrences with constant coefficients.

Theorem 5.9. Let the sequences $\langle t_{i,n} \rangle$ for $1 \leq i \leq m$ be defined by a set of m simultaneous linear recurrences with constant coefficients where the ith recurrence has the form (3.1). If the system of recurrences can be solved and any of the $\langle a_{i,n} \rangle$ are infinite sequences, then the sequences $\langle t_{i,n} \rangle$ have extended closed form expressions involving terms from the sequences $\langle a_{i,n} \rangle$.

Proof. The generating function of each sequence $\langle t_{i,n} \rangle$ has the form given by (5.9). Each generating function $A_j(z)$ that is a nontrivial rational generating function can be related to the function $g_{i,j}(z)A_j(z)$ by common factors in their denominators. Thus $T_i(z)$ can be expanded to yield an extended closed form involving terms from those sequences $\langle a_{j,n} \rangle$. □

If any of the sequences $\langle a_{j,n} \rangle$ are hybrid terms, we can find their generating functions using Theorem 4.5. We can substitute for hybrid terms $a_{j,n}$ in the solution to find a closed form expression involving the factors of $\langle a_{j,n} \rangle$.

5.4. Algorithms for Rational Generating Functions

We first present algorithms for encoding and expanding rational generating functions and consider extensions and applications of these algorithms. We then look at methods for symbolic summation involving sequences in RAT_SEQ.

5.4.1. Generating Function Algorithms

We start by developing algorithms for mapping between the set of nontrivial rational functions and the set of closed form expressions. We then apply these algorithms to encode any sequence whose closed form is in $\mathcal{T}_{\mathcal{F}}$. Application of the algorithms to encoding generating functions for sequences defined by linear indexing or hybrid terms is considered.

Input: f_n, the expression to encode.
Output: $F(z)$, the generating function of $\langle f_n \rangle$ for $n \geq 0$.

1. Expand f_n in sum of products form.
2. Let $F(z) \leftarrow 0$.
3. Repeat for each term t_n in the sum of products expansion of f_n:
   (a) Express t_n as $p(n)\theta^n$.
   (b) Find $P(z)$, the generating function of $\langle p(n) \rangle$.
   (c) Let $F(z) \leftarrow F(z) + P(\theta z)$.
4. Return $F(z)$.

Algorithm 5.1. Rational/Encode. Encodes rational generating functions.

Algorithms for Encoding and Expanding

Algorithm 5.1 encodes rational generating functions. This algorithm takes as input an expression $f_n \in \mathcal{T}_{\mathcal{F}}$ as defined by Lemma 4.2. It calculates the generating function $F(z)$ of the sequence $\langle f_n \rangle$ defined for $n \geq 0$.

The expression is converted to a sum of products form in Step 1. Step 3 of the algorithm does the actual calculation of generating functions. Each term in the sum of products expansion of f_n is expressed as the product between a polynomial $p(n)$ and an exponential θ^n. The generating function $P(z)$ of $\langle p(n) \rangle$ is found. By (2.11) the generating function of $\langle p(n)\theta^n \rangle$ is $P(\theta z)$. Summing the generating functions for each term in the expansion of f_n gives $F(z)$. Figure 5.1 traces the execution of Algorithm 5.1 on the input $f_n = (n2^n + 3^n)(n-2)5^n$.

Step 3(b) computes the generating function of a polynomial. There are several options when implementing an algorithm for this step. If $p(n) = p_0 + p_1 n + \cdots + p_m n^m$, the naive approach would be to encode each term $p_i n^i$ individually. We would apply (2.16) i times to find the generating function of each $\langle n^i \rangle$, multiply by p_i, and then sum over all i. While such an algorithm would find the correct generating function, it is inefficient. To calculate the generating function of $\langle n^i \rangle$, it recalculates the generating function of $\langle n^{i-1} \rangle$.

Algorithms 5.2 and 5.3 present more efficient ways to compute the generating function of a polynomial. Algorithm 5.2 applies (2.16) to the variable $g(z)$ to compute, in order, the generating functions of the sequences $\langle 1 \rangle$, $\langle n \rangle$, $\langle n^2 \rangle$, ..., $\langle n^m \rangle$. Step 3(a) uses $g(z)$ to add in the generating function of $\langle p_i n^i \rangle$. Figure 5.2 traces the execution of Algorithm 5.2 on the input $p(n) = an^2 + bn + c$.

Algorithm 5.3 modifies Horner's rule for the evaluation of polynomials [PURDOM]. Horner's method expresses the polynomial $p(n) = p_0 + p_1 n + \cdots + p_m n^m$, as

Input: $f_n = (n2^n + 3^n)(n-2)5^n$.

1. $f_n \leftarrow (n^2 - 2n)10^n - (n-2)15^n$.
2. $F(z) \leftarrow 0$.
3. Repeat for each term t_n in $(n^2 - 2n)10^n - (n-2)15^n$:
   (a) $t_n \leftarrow (n^2 - 2n)10^n$.
   (b) $P(z) \leftarrow (-z + 3z^2)/(1-z)^3$.
   (c) $F(z) \leftarrow 0 + (-10z + 300z^2)/(1-10z)^3$.

   (a) $t_n \leftarrow (n-2)15^n$.
   (b) $P(z) \leftarrow (-2 + 3z)/(1-z)^2$.
   (c) $F(z) \leftarrow (-10z + 300z^2)/(1-10z)^3 + (-2 + 45z)/(1-15z)^2$.
4. Return $(-2 + 95z - 1350z^2 + 4250z^3 + 22500z^4)/((1-10z)^3(1-15z)^2))$.

Figure 5.1. Example trace of Algorithm 5.1.

Input: Polynomial $p(n)$.
Output: $P(z)$, the generating function of $\langle p(n) \rangle$ for $n \geq 0$.

1. Let $P(z) \leftarrow 0$.
2. Let $g(z) \leftarrow 1/(1-z)$.
3. Repeat for i from 0 to degree$(p(n)) - 1$:
   (a) Let $P(z) \leftarrow P(z) + \text{coeff}(p(n), i)g(z)$.
   (b) Let $g(z) \leftarrow zg'(z)$.
4. Let $P(z) \leftarrow P(z) + \text{coeff}(p(n), \text{degree}(p(n)))g(z)$.
5. Return $P(z)$.

Algorithm 5.2. Rational/Polynomial1. Encodes generating functions of polynomials.

Input: $p(n) = an^2 + bn + c$.

1. $P(z) \leftarrow 0$.
2. $g(z) \leftarrow 1/(1-z)$.
3. Repeat for i from 0 to 1:
   (a) $P(z) \leftarrow 0 + c/(1-z)$.
   (b) $g(z) \leftarrow z/(1-z)^2$.
   (a) $P(z) \leftarrow c/(1-z) + bz/(1-z)^2$.
   (b) $g(z) \leftarrow z(1+z)/(1-z)^3$.
4. $P(z) \leftarrow (c + bz - cz)/(1-z)^2 + a(z+z^2)/(1-z)^3$.
5. Return $((a-b+c)z^2 + (a+b-2c)z + c)/(1-z)^3$.

Figure 5.2. Example trace of Algorithm 5.2.

Input: Polynomial $p(n)$.
Output: $P(z)$, the generating function of $\langle p(n) \rangle$ for $n \geq 0$.

1. Let $P(z) \leftarrow \text{coeff}(p(n), \text{degree}(p(n)))/(1-z)$.
2. Repeat for i from $\text{degree}(p(n)) - 1$ down to 0:
   (a) Let $P(z) \leftarrow zP'(z) + \text{coeff}(p(n), i)/(1-z)$.
3. Return $P(z)$.

Algorithm 5.3. Rational/Polynomial2. Encodes generating functions of polynomials.

Input: $p(n) = an^2 + bn + c$.

1. $P(z) \leftarrow a/(1-z)$.
2. Repeat for i from 1 down to 0:
   (a) $P(z) \leftarrow za/(1-z)^2 + b/(1-z)$.
   (a) $P(z) \leftarrow z(a + az + b - bz)/(1-z)^3 + c/(1-z)$.
3. Return $((a-b+c)z^2 + (a+b-2c)z + c)/(1-z)^3$.

Figure 5.3. Example trace of Algorithm 5.3.

$$p(n) = (\cdots(p_m n + p_{n-1})n + \cdots + p_1)n + p_0.$$

By (2.16) we find that if $Q(z)$ is the generating function of $\langle q(n)\rangle$, then $zQ'(z)$ is the generating function of $\langle nq(n)\rangle$. The algorithm begins with $P(z) = p_n/(1-z)$, which is the generating function of $\langle p_n\rangle$. Repetition i of Step 2 starts with $P(z)$ being the generating function of $\langle p_m n^{i-1} + p_{m-1}n^{i-2} + \cdots + p_{m-i+1}\rangle$. Thus, $zP'(z)$ is the generating function of $\langle p_m n^i + p_{m-1}n^{i-1} + \cdots + p_{m-i+1}n\rangle$ and step 2 computes the generating function of the polynomial $\langle p_m n^i + p_{m-1}n^{i-1} + \cdots + p_{m-i}\rangle$. After m iterations of Step 2 we have computed the generating function of $\langle p(n)\rangle$. Figure 5.3 traces the execution of Algorithm 5.3 on the input $p(n) = an^2 + bn + c$.

Algorithm 5.4 expands a nontrivial rational generating function $F(z)$ to find a closed form expression for the sequence that it encodes. In Steps 1 through 5 it expresses the generating function as

$$F(z) = \frac{n(z)}{cz^d} G(z),$$

where $cz^d = \text{lterm}(\text{denom}(F(z)))$ and $G(z)$ is the partial fraction expansion of the characteristic generating function of $F(z)$. Since $G(z)$ has the form (4.1) it has a partial fraction expansion of the form (4.2). Each term can be written as $\alpha/(1-\beta z)^{m+1}$, where α and β are constants and m is a positive integer. This generating function encodes the sequence $\langle \alpha\binom{n+m}{m}\beta^n\rangle$. We could choose to calculate this value from the generating function $1/(1-z)$ using the manipulation rules for generating functions. However, since this is the basic sequence on which the closed form expansion of rational generating functions is built, it is more efficient to map the generating function directly to the sequence rather than derive it every time it occurs. Step 7 of the algorithm expands each of these terms and sums the resulting expressions. The net result is g_n, the closed form expression of the characteristic sequence of $\langle f_n\rangle$. Step 8 shifts and scales g_n to find the closed form of the sequence encoded by the generating function $G(z)/(cz^d)$. Step 9 applies (2.15), using the numerator polynomial $n(z)$ to shift and scale g_n, to find f_n. Figure 5.4 traces the execution of Algorithm 5.4 on the input $F(z) = (2-z)/((3z(1-3z)^2((1-2z))$.

Input: Nontrivial rational generating function $F(z)$.
Output: f_n, the closed form expression of the sequence $\langle f_n \rangle$ encoded by $F(z)$.

1. Put $F(z)$ in normal form.
2. Let $n(z) \leftarrow \text{numer}(F(z))$.
3. Let $p(z) \leftarrow \text{denom}(F(z))$
4. Let $d \leftarrow \text{ldegree}(p(z))$ and $c \leftarrow \text{lcoeff}(p(z))$.
5. Let $G(z)$ be the partial fraction of $\text{lterm}(p(z))/p(z)$.
6. Let $g_n \leftarrow 0$.
7. Repeat for each term $T(z)$ in $G(z)$:
   (a) Express $T(z)$ as $\alpha/(1-\beta z)^{m+1}$.
   (b) Let $t_n \leftarrow \alpha\binom{n-m}{m}\beta^n$.
   (c) Let $g_n \leftarrow g_n + t_n$.
8. Let $g_n \leftarrow g_{n+d}/c$.
9. Let $f_n \leftarrow \sum_{i=0}^{\text{degree}(n(z))} \text{coeff}(n(z), i) g_{n-i}$.
10. Return f_n.

Algorithm 5.4. Rational/Expand. Expands nontrivial rational generating functions.

Input: $F(z) = (2-z)/(3z(1-3z)^2(1-2z))$.

1. $F(z) \leftarrow (2-z)/(3z(1-3z)^2(1-2z))$.
2. $n(z) \leftarrow 2-z$.
3. $p(z) \leftarrow 3z(1-3z)^2(1-2z)$.
4. $d \leftarrow 1$ and $c \leftarrow 3$.
5. $G(z) \leftarrow 3/(1-3z)^2 - 6/(1-3z) + 4/(1-2z)$.
6. $g_n \leftarrow 0$.
7. Repeat for term $T(z)$ in $3/(1-3z)^2 - 6/(1-3z) + 4/(1-2z)$:
   (a) $T(z) \leftarrow 3/(1-3z)^2$.
   (b) $t_n \leftarrow 3(n+1)3^n$.
   (c) $g_n \leftarrow 0 + (n+1)3^{n+1}$.

   (a) $T(z) \leftarrow -6/(1-3z)$.
   (b) $t_n \leftarrow -6\,3^n$.
   (c) $g_n \leftarrow (n+1)3^{n+1} - 2\,3^{n+1}$.

   (a) $T(z) \leftarrow 4/(1-2z)$.
   (b) $t_n \leftarrow 4\,2^n$.
   (c) $g_n \leftarrow (n-1)3^{n+1} + 2^{n+2}$.
8. $g_n \leftarrow (n3^{n+2} + 2^{n+3})/3$.
9. $f_n \leftarrow 2(n3^{n+2} + 2^{n+3})/3 - ((n-1)3^{n+1} + 2^{n+2})/3$.
10. Return $(5n+1)3^n + 2^{n+2}$.

Figure 5.4. Example trace of Algorithm 5.4.

Application of Encoding and Expanding Algorithms

Algorithm 5.1 finds the generating function for a sequence defined by a closed form expression for $n \geq 0$. This algorithm corresponds to the proof of Theorem 4.3. We would like to generalize the algorithm to correspond to the proofs of Corollary 4.3.1 and Corollary 4.3.2.

Corollary 4.3.1 allows a sequence to be defined by a closed form expression over $n \geq \gamma$ instead of over $n \geq 0$. This can be implemented with Algorithm 5.1 by duplicating the proof of the corollary. To find the generating function $F(z)$ of the sequence defined by the expression f_n over $n \geq \gamma$, we can do the following:

1. Let $g_n = f_{n-\gamma}$. The sequence $\langle g_n \rangle$ will be defined over $n \geq 0$.
2. Find $G(z)$, the generating function of $\langle g_n \rangle$ over $n \geq 0$.
3. Let $F(z) = z^\gamma G(z)$.

We can take a different approach to finding $F(z)$. The generating function we want is

$$F(z) = \sum_{n=\gamma}^{\infty} f_n z^n.$$

Algorithm 5.1 returns the quantity

$$G(z) = \sum_{n=0}^{\infty} f_n z^n.$$

Thus, we have

$$F(z) = \begin{cases} \sum_{n=\gamma}^{-1} f_n z^n + G(z), & \text{if } \gamma < 0; \\ G(z), & \text{if } \gamma = 0; \\ G(z) - \sum_{n=0}^{\gamma-1} f_n z^n, & \text{if } \gamma > 0. \end{cases}$$

Corollary 4.3.2 allows a finite number of terms in a sequence to have a value different from that given by its closed form. Let $\langle f_n \rangle$ be defined over $n \geq \gamma$ and let its closed form expression c_n be valid only over $n \geq n_0$, where $n_0 \geq \gamma$. We can duplicate the proof of the corollary to find $F(z)$, the generating function of $\langle f_n \rangle$. Let $g_n = f_n - c_n$ for $\gamma \leq n < n_0$. The finite sequence $\langle g_n \rangle$ represents the amount each term in $\langle f_n \rangle$ differs from its closed form expression. The modified encoding algorithm can be applied as above to find $C(z)$, the generating function of $\langle c_n \rangle$ over $n \geq \gamma$. Then

$$F(z) = C(z) + \sum_{n=\gamma}^{n_0-1} g_n z^n.$$

We can use the algorithms for encoding and decoding rational generating functions to implement Theorem 4.4, which uses linear indexing of an existing sequence to define a new sequence, and Theorem 4.5, which defines hybrid sequences. Again, the algorithms follow the proof of each theorem. A new sequence is defined as $\langle s_n \rangle = \langle a_{\alpha n+\beta} \rangle$ for

integer α and β or as $\langle s_n \rangle = \langle a_n b_n \rangle$. The closed form for s_n is found from the closed forms for a_n and b_n. The generating function encoding algorithm is used to find the generating function of the sequence defined by s_n. Finally, the generating function is adjusted for any terms in $\langle s_n \rangle$ that do not satisfy its closed form expression.

5.4.2. Summation Algorithms

The algorithms we have developed provide us with all of the tools that we need to solve sums and convolutions in RAT_SEQ. We outline the steps involved when solving either type of problem.

To evaluate a summation

$$s_n = \sum_{k=0}^{n} a_k$$

we do the following:

1. Find the closed form of the summand using the algorithms for linear indexing or hybrid terms if needed.
2. Find the generating function $A(z)$ of the summand using the appropriate encoding algorithm.
3. Find the generating function of the solution, $S(z) = A(z)/(1-z)$.
4. Expand $S(z)$ to find the closed form of s_n.

To evaluate a convolution

$$s_n = \sum_{\alpha_1+\alpha_2+\cdots+\alpha_m=n} a_{1,\alpha_1} a_{2,\alpha_2} \cdots a_{m,\alpha_m}$$

we do the following:

1. Repeat for each factor $a_{i,k}$:
   (a) Find the closed form of $a_{i,k}$ using the algorithms for linear indexing or hybrid terms if needed.
   (b) Find the generating function $A_i(z)$ of $\langle a_{i,k} \rangle$ using the appropriate encoding algorithm.
2. Find the generating function of the solution, $S(z) = A_1(z)A_2(z)\cdots A_m(z)$.
3. Expand $S(z)$ to find the closed form of s_n.

These steps will find a closed form expression for any summation or convolution involving sequences from RAT_SEQ. However, we are more interested in expressing the solutions in extended closed forms that involve the sequences defining the summand.

For convolutions, we follow the proof of Theorem 5.5 and use partial fractions to expand the generating function of the solution. Each term in this expansion can be

related to a factor of the summand. This will allow us to express the solution in terms of the factors in the convolution. The convolution algorithm also provides a way to find extended closed form solutions for inhomogeneous recurrences and systems of inhomogeneous recurrences. The generating functions encoding the solutions to these recurrences involve the convolution of a rational function with the generating function of the inhomogeneous terms. These algorithms can be used to expand those generating functions.

For simple sums, we use partial fractions to express $S(z)$, the generating function of the solution, in the form that we used in the proof of Theorem 5.4. We then express the solution as a function of terms from $\langle a_n \rangle$, the sequence defining the summand. For a hybrid sum, we use an approach like that in the proof of Corollary 5.4.1. We find the closed form of the hybrid term using Theorem 4.5 and then treat the sum as if it were a simple sum.

This method for solving hybrid sums is limited by the ability to find a closed form for the hybrid term. In some cases, the hybrid term may have a factor in the summand for which we cannot find a closed form expression. Thus we may not be able to obtain the generating function of the summand. Algorithm 5.5 implements the hybrid sum method based on Corollary 2.2.2. The algorithm uses the hybrid generating function for the summand to evaluate the sum.

The key to this algorithm is Step 2. To demonstrate the ability of the algorithm, we can consider the case where the summand has $m = 2$ factors and then inductively apply this result to $m \geq 2$. To evaluate the $m = 2$ variable case, we want to find the coefficient of $w^n z^n$ in $C(wz)G(w)F(z)$, where $C(wz)$ is a diagonal rational function of wz and $F(z)$ and $G(w)$ are rational functions. The hybrid sum method applies partial fractions with respect to z, discards the term without a $w^n z^n$ term in its expansion, and finds the coefficient of z^n in the remaining term. We can follow Step 2 to show that the algorithm does this as well.

In Step 2(a) we let $n(wz) = \text{numer}(C(wz))$, and in Step 2(b) we perform partial fractions on the denominators with respect to z. By (A.10) we find that

$$\begin{aligned}
&\frac{n(wz)\,\text{numer}(F(z))}{\text{denom}(C(wz))\,\text{denom}(F(z))} \\
&\quad = \quad n(wz)\,\text{numer}(F(z))\left(\frac{r(w,z)}{q(w)\,\text{denom}(C(wz))} + \frac{p(w,z)}{q(w)\,\text{denom}(F(z))}\right) \\
&\quad = \quad \frac{r(w,z)\,\text{numer}(F(z))}{q(w)}C(wz) + \frac{p(w,z)n(wz)}{q(w)}F(z),
\end{aligned}$$

where $p(w,z)$, $q(w)$, and $r(w,z)$ are polynomials. The left hand term does not have a $w^n z^n$ term in its expansion, so we discard it in Step 2(c). The right hand term is the quantity found in Step 2(d). This term is a product of the polynomial $p(w,z)n(wz)$, the rational function $1/q(w)$, and $F(z)$. Expanding as a function of z results in $t(w)/q(w)$, where $t(w)$ is the polynomial in w obtained by using the polynomial $p(w,z)n(wz)$ to shift and scale terms from the sequence $\langle f_n \rangle$ encoded by $F(z)$. This polynomial is found in Step 2(e). Finally, in Step 2(f) we let $C(w) = t(w)/q(w)$.

Input: A list of m generating functions $A_i(z_i)$ and the sequences $\langle a_{i,n} \rangle$ that they encode.
Output: An extended closed form solution for $s_n = \sum_{k=0}^{n} a_{1,k} \cdots a_{m,k}$ involving terms from the sequences $\langle a_{1,n} \rangle$ to $\langle a_{m,n} \rangle$.

1. Let $C(z_1 \cdots z_m) \leftarrow 1/(1 - z_1 \cdots z_m)$.
2. Repeat for i from 1 to $m - 1$:
   (a) Let $n(z_i \cdots z_m) \leftarrow \text{numer}(C(z_i \cdots z_m))$.
   (b) Apply partial fractions over z_i to $1/(\text{denom}(C(z_i \cdots z_m))\,\text{denom}(A_i(z_i)))$.
   (c) Discard terms with no coefficient of $(z_i \cdots z_m)^n$ in their expansion.
   (d) Normalize the remaining term. It will have the form
   $$\frac{p(z_i, z_{i+1} \cdots z_m)}{q(z_{i+1} \cdots z_m)\,\text{denom}(A_i(z_i))},$$
   where $p(z_i, z_{i+1} \cdots z_m)$ is a polynomial.
   (e) Multiply this term by $n(z_i \cdots z_m)$ and $\text{numer}(A_i(z_i))$. Find the coefficient of z_i^n in
   $$n(z_i \cdots z_m)p(z_i, z_{i+1} \cdots z_m)A_i(z_1).$$
   Use $n(z_i \cdots z_m)p(z_i, z_{i+1} \cdots z_m)$ to shift and scale terms from the sequence $\langle a_{i,n} \rangle$. The result is a polynomial $t(z_{i+1} \cdots z_m)$.
   (f) Let $C(z_{i+1} \cdots z_m) \leftarrow t(z_{i+1} \cdots z_m)/q(z_{i+1} \cdots z_m)$.
3. Apply partial fractions to the denominator of $C(z_m)A_m(z_m)$.
4. Find coefficient of z_m^n. Relate terms in the partial fraction expansion to either $C(z_m)$ or $A_m(z_m)$. Expand terms related to $A_m(z_m)$ as a function of terms from the sequence $\langle a_{m,n} \rangle$. Find coefficient of z_m^n in the other terms.

Algorithm 5.5. Rational/Hybrid_Sum. Computes hybrid sums over RAT_SEQ.

Input: Generating functions $A_1(z_1) = 1/(1-2z_1)$ and $A_2(z_2) = z_2/(1-z_2-z_2^2)$. Sequences $\langle a_{1,n} \rangle = \langle 2^n \rangle$ and $\langle a_{2,n} \rangle = \langle F_n \rangle$.

1. $C(z_1 z_2) \leftarrow 1/(1-z_1 z_2)$
2. Repeat for i from 1 to 1:
   (a) $n(z_1 z_2) = 1$.
   (b) Partial fractions with respect to z_1 on $1/((1-z_1 z_2)(1-2z_1))$ gives
   $$\frac{z_2}{(z_2-2)(1-z_1 z_2)} + \frac{2}{(2-z_2)(1-2z_1)}.$$
   (c) Discard $z_2/((z_2-2)(1-z_1 z_2))$.
   (d) Keep $2/((2-z_2)(1-2z_1))$. We find that $p(z_1, z_2) = 2$ and $q(z_2) = 2 - z_2$.
   (e) $t(z_2) \leftarrow 2^{n+1}$.
   (f) $C(z_2) \leftarrow 2^{n+1}/(2-z_2)$.
3. Partial fractions with respect to z_2 on $2^{n+1} z_2/((2-z_2)(1-z_2-z_2^2))$ gives
   $$2^{n+1} z_2 \left(\frac{-1}{5(2-z_2)} + \frac{3+z_2}{5(1-z_2-z_2^2)} \right).$$
4. The coefficient of z_2^n is
   $$\frac{2^{n+1}}{5}(-2^{-n} + 3F_n + F_{n-1}) = \frac{3\,2^{n+1} F_n + 2^{n+1} F_{n-1} - 2}{5}.$$

Figure 5.5. Example trace of Algorithm 5.5.

In the two variable case, one pass through Step 2 gives us a generating function in w which we then proceed to expand in Steps 3 and 4. For $m \geq 3$, we have to make multiple passes through Step 2. We can inductively apply the above result to show that the algorithm finds the desired coefficient of $(z_1 \cdots z_m)^n$. At each iteration of Step 2, we let $w = z_{i+1} \cdots z_m$, $z = z_i$, and $F(z) = A_i(z_i)$. The base case is $C(wz) = 1/(1-wz)$ and $F(z) = A_1(z_1)$. As we showed before, performing Steps 2(a) through 2(f) on $C(wz)$ and $F(z)$ will find the coefficient of z^n. Therefore, it correctly finds the coefficient of z_i^n. We are then left with $C(w) = C(z_{i+1} \cdots z_m)$ and the generating functions $F_{i+1}(z_{i+1})$ to $F_m(z_m)$. The problem size is reduced by one variable and we repeat the process until there is only one variable remaining. Figure 5.5 traces the execution of Algorithm 5.5 for the hybrid sum $s_n = \sum_{k=0}^{n} 2^k F_k$.

We can easily modify this algorithm to handle multifactor hybrid convolutions. One factor in the convolution will be a hybrid term. The factors of the hybrid terms will

provide the generating functions $A_i(z_i)$. We multiply the generating functions of the remaining factors in the summand to get $B(w)$. In Step 1, we let the initial $C(z_1 \cdots z_m)$ be $B(z_1 \cdots z_m)$. By Theorem 2.4, the coefficient of $(z_1 \cdots z_m)^n$ will be the solution to the multifactor hybrid convolution. The algorithm proceeds to find this value.

Chapter 6

Super Generating Functions and Binomial Summation

Super generating functions encode sequences of two or more variables. Among the sequences that they can be used to encode are the binomial coefficients, the Stirling numbers, and the Euler numbers. Of these, the binomial coefficients are perhaps the most important in discrete mathematics. They have a natural physical interpretation since the binomial coefficient $\binom{n}{k}$ defines the number of ways to choose k distinct items from a set of n items.

The generating function of the binomial coefficients is a simple instance of a rational function of two variables. As we did with RAT, we can define many properties of multivariate rational generating functions. We show that various types of sums involving binomial coefficients have solutions and indicate where the solutions can be found. This will allow us to define classes of summations involving binomial coefficients that can be solved using generating functions.

6.1. Multivariate Rational Generating Functions

Multivariate rational generating functions are functions of m variables that are rational with respect to each variable. We discuss some of their algebraic properties, examine methods for expanding them to find the sequences that they encode, and consider extensions to the definition of closed form that enable us to include these sequences.

6.1.1. Field of Multivariate Rational Generating Functions

Given the field $(\mathcal{F}, +, \cdot)$, the set $\mathcal{F}(y)$ of rational functions of the symbol y forms a field $(\mathcal{F}(y), +, \cdot)$. The set $RAT_{\mathcal{F}(y)}$ of rational generating functions with coefficients from $\mathcal{F}(y)$ forms a field $(RAT_{\mathcal{F}(y)}, +, \cdot)$. Continuing the process, we find that the rational functions of $m-1$ variables form a field $(\mathcal{F}(y_1)(y_2)\cdots(y_{m-1}), +, \cdot)$ and thus the rational generating functions with coefficients from $\mathcal{F}(y_1)(y_2)\cdots(y_{m-1})$ form a field $(RAT_{\mathcal{F}(y_1)(y_2)\cdots(y_{m-1})}, +, \cdot)$.

Many of the properties that we proved for RAT hold for multivariate rational generating functions. If $G(y_1, y_2, \ldots, y_{m-1}, z) \in RAT_{\mathcal{F}(y_1)(y_2)\cdots(y_{m-1})}$ and $\mathbb{Z} \subseteq \mathcal{F}$, then the coefficient of z^n for a particular n in the expansion of $G(y_1, y_2, \ldots, y_{m-1}, z)$ is in $\mathcal{F}(y_1)(y_2)\cdots(y_m)$. Likewise, we can show that the coefficient of $y_1^{\alpha_1}\cdots y_{m-1}^{\alpha_{m-1}} z^n$ of $G(y_1, y_2, \ldots, y_{m-1}, z)$ is in $\mathcal{F}$.

Relationships between sequences with common factors in their generating functions can be shown using our existing repertoire of methods. Relationships within a sequence are harder to express. In particular, simplification of expressions involving adjacent terms from a multivariate sequence is not as well defined as in the single variable case.

The definition of a characteristic generating function can be extended to multivariate rational generating functions. Each nontrivial multivariate generating function has a characteristic generating function which maps directly to a multivariate homogeneous linear recurrence with constant coefficients. All sequences with this characteristic generating function will satisfy the recurrence that it encodes.

Every multivariate homogeneous linear recurrence with constant coefficients has a multivariate rational generating function. A systematic method to solve these recurrences depends on our ability to expand rational super generating functions.

6.1.2. Generating Function Expansion

Unlike $RAT_{\mathbb{C}}$, there are no known methods for finding the general term in the expansion of a multivariate rational generating function. Rather, we must treat each generating function individually. We will consider two special cases that are of importance to summation. The generating function

$$G(y, z) = \frac{1}{1 - y/p(z)},$$

where $p(z)$ is a polynomial relates to convolutions of sequences from RAT_SEQ. The characteristic generating function

$$\mathcal{H}(y, z) = \frac{1}{1 - \alpha y - \beta z - \gamma yz}$$

is the general form of the generating function for the binomial coefficients. For each of these generating functions, we consider methods to determine the coefficient of $y^m z^n$.

Expansion of $G(y, z) = 1/(1 - y/p(z))$

We can easily find the coefficient of y^m in $G(y, z)$,

$$G(y, z) = \sum_{m=0}^{\infty} \left(\frac{1}{p(z)}\right)^m y^m.$$

The generating function $1/p(z)^m$ encodes the convolution defined by

$$g_{m,n} = \sum_{\alpha_1+\alpha_2+\cdots+\alpha_m=n} p_{\alpha_1}\cdots p_{\alpha_m},$$

where $\langle p_n \rangle$ is the sequence encoded by $1/p(z)$.

If m is a particular integer constant, we know how to use repeated differentiation of $1/p(z)$ to relate $g_{m,n}$ to the sequence $\langle p_n \rangle$. Or, we can use Algorithm 5.4 to expand $1/p(z)^m$ and find a closed form expression for $g_{m,n}$. However, in this case m is a symbolic parameter. Technically, $1/p(z)^m$ is not a rational function. Rather, it is a function of z and m. Thus, we cannot expand it as we did rational generating functions.

Depending on the degree of $p(z)$ we can attack the generating function in a variety of ways. We can factor out $\text{lterm}(p(z))^m = (\text{lcoeff}(p(z))z^{\text{ldegree}(p(z))})^m$ to obtain

$$\frac{1}{p(z)^m} = \frac{1}{(\text{lcoeff}(p(z))z^{\text{ldegree}(p(z))})^m q(z)^m},$$

where $q(z) = p(z)/\text{lterm}(p(z))$. If $\text{degree}(q(z)) = 1$ then $q(z) = 1 + \beta z$ and

$$\begin{aligned}\frac{1}{p(z)^m} &= \frac{1}{(\text{lcoeff}(p(z))z^{\text{ldegree}(p(z))})^m (1+\beta z)^m} \\ &= \frac{1}{(\text{lcoeff}(p(z))z^{\text{ldegree}(p(z))})^m} \sum_{n=0}^{\infty} \binom{n+m-1}{m-1} \beta^n z^n.\end{aligned}$$

Thus, we find that

$$g_{m,n} = \frac{\binom{n+\delta+m-1}{m-1}\beta^{n+\delta}}{\text{lcoeff}(p(z))^m},$$

where $\delta = \text{ldegree}(p(z))$.

If $\text{degree}(q(z)) = 2$, we can choose from two approaches. Since m is a parameter, we cannot perform partial fractions on $1/q(z)^m$. We can, however, determine what the effect of partial fractions would be. If $q(z) = (1-az)(1-bz)$ then by (A.2) we find that

$$\frac{1}{q(z)^m} = a^m \sum_{k=0}^{m-1} \frac{(-b)^k \binom{m-1+k}{m-1}}{(a-b)^{m+k}(1-az)^{m-k}} + b^m \sum_{k=0}^{m-1} \frac{(-a)^k \binom{m-1+k}{m-1}}{(b-a)^{m+k}(1-bz)^{m-k}}.$$

We can expand this generating function and shift and scale in order to find a closed form expression for $g_{m,n}$. If we want an extended closed form expression, we relate the generating functions $1/(1-az)$ and $1/(1-bz)$ to the generating function $1/p(z)$. This will allow us to find an extended closed form for the sequences $\langle a^n \rangle$ and $\langle b^n \rangle$ that involves terms from the sequence $\langle p_n \rangle$. Substituting, we get an extended closed form for the sequence $\langle g_{m,n} \rangle$.

Example 6.1. Consider the convolution

$$s_n = \sum_{\alpha_1+\alpha_2+\cdots+\alpha_m=n} F_{\alpha_1} F_{\alpha_2} \cdots F_{\alpha_m},$$

where $m \geq 2$.

By (2.17), the generating function of the sequence $\langle s_n \rangle$ is $S(z) = z^m/(1-z-z^2)^m$. We will consider the expansion of the generating function $A(z) = 1/(1-z-z^2)^m =$

$1/((1-\phi z)(1-\hat{\phi}z))^m$, and then recognize that $S(z) = z^m A(z)$ implies $s_n = a_{n-m}$. To find a_n, we need to determine the partial fraction expansion of $A(z)$.

Factoring $A(z)$ and applying (A.2) gives

$$A(z) = \phi^m \sum_{k=0}^{m-1} \frac{(-\hat{\phi})^k \binom{m-1+k}{m-1}}{(\phi-\hat{\phi})^{m+k}(1-\phi z)^{m-k}} + \hat{\phi}^m \sum_{k=0}^{m-1} \frac{(-\phi)^k \binom{m-1+k}{m-1}}{(\hat{\phi}-\phi)^{m+k}(1-\hat{\phi}z)^{m-k}}.$$

Rearranging the terms and recognizing that $\phi - \hat{\phi} = \sqrt{5}$ and $\phi\hat{\phi} = -1$ we have

$$\begin{aligned} A(z) &= \sum_{k=0}^{m-1} \frac{\phi^m (-\hat{\phi})^k \binom{m-1+k}{m-1}}{(\sqrt{5})^{m+k}(1-\phi z)^{m-k}} + \sum_{k=0}^{m-1} \frac{\hat{\phi}^m (-\phi)^k \binom{m-1+k}{m-1}}{(-\sqrt{5})^{m+k}(1-\hat{\phi}z)^{m-k}} \\ &= \sum_{k=0}^{m-1} \frac{\phi^{m-k} \binom{m-1+k}{m-1}}{(\sqrt{5})^{m+k}(1-\phi z)^{m-k}} + \sum_{k=0}^{m-1} \frac{\hat{\phi}^{m-k} \binom{m-1+k}{m-1}}{(-\sqrt{5})^{m+k}(1-\hat{\phi}z)^{m-k}}. \end{aligned}$$

The coefficient of z^n in the generating function $1/(1-\alpha z)^m$ is $\binom{m-1+n}{m-1}\alpha^n$ [WILF]. Expanding the generating function $A(z)$ and rearranging terms, we find that the coefficient of z^n is

$$\begin{aligned} a_n &= \sum_{k=0}^{m-1} \frac{\phi^{m-k} \binom{m-1+k}{m-1}}{(\sqrt{5})^{m+k}} \binom{m-k-1+n}{m-k-1} \phi^n \\ &\quad + \sum_{k=0}^{m-1} \frac{\hat{\phi}^{m-k} \binom{m-1+k}{m-1}}{(-\sqrt{5})^{m+k}} \binom{m-k-1+n}{m-k-1} \hat{\phi}^n \\ &= \sum_{k=0}^{m-1} \frac{\binom{m-1+k}{m-1}\binom{m-k-1+n}{m-k-1}}{(\sqrt{5})^{m+k}} \left(\phi^{m-k+n} + (-1)^{m+k}\hat{\phi}^{m-k+n}\right). \end{aligned}$$

Recognizing that $\phi^n = \phi F_n + F_{n-1}$ and $\hat{\phi}^n = \hat{\phi}F_n + F_{n-1}$ and rearranging terms gives

$$\begin{aligned} a_n &= \sum_{k=0}^{m-1} \frac{\binom{m-1+k}{m-1}\binom{m-k-1+n}{m-k-1}}{(\sqrt{5})^{m+k}} \\ &\quad \left(\phi F_{m-k+n} + F_{m-k+n-1} + (-1)^{m+k}\hat{\phi}F_{m-k+n} + (-1)^{m+k}F_{m-k+n-1}\right) \\ &= \sum_{k=0}^{m-1} \frac{\binom{m-1+k}{m-1}\binom{m-k-1+n}{m-k-1}}{(\sqrt{5})^{m+k}} \\ &\quad \left((\phi + (-1)^{m+k}\hat{\phi})F_{m-k+n} + (1+(-1)^{m+k})F_{m-k+n-1}\right). \end{aligned}$$

Finally, we shift the terms in the sequence $\langle a_n \rangle$ by $-m$ to find that

$$\begin{aligned} s_n &= a_{n-m} \\ &= \sum_{k=0}^{m-1} \frac{\binom{m-1+k}{k}\binom{n-k-1}{m-k-1}}{\left(\sqrt{5}\right)^{m+k}} \left((\phi + (-1)^{m+k}\hat{\phi})F_{n-k} + (1+(-1)^{m+k})F_{n-k-1}\right). \end{aligned}$$

It is useful to note that $s_n = 0$ for $n < m$. Also, the quantity $\binom{n-k-1}{m-k-1}$ is a polynomial in n of degree $m-k-1$. When $m+k$ is odd, the expression in parentheses simplifies to $\sqrt{5}F_{n-k}$, and when $m+k$ is even it simplifies to $F_{n-k}+2F_{n-k-1}$. Thus, s_n is the sum of the products between Fibonacci numbers and polynomials in n. ∎

A second way to find $g_{m,n}$ when degree$(q(z)) = 2$ is to express the polynomial as $q(z) = 1-\beta_1 z-\beta_2 z^2$ and expand $1/q(z)^m$ as powers of $(\beta_1+\beta_2 z)z$,

$$\begin{aligned}\frac{1}{q(z)^m} &= \frac{1}{(1-(\beta_1+\beta_2 z)z)^m} \\ &= \sum_{i=0}^{\infty}\binom{i+m-1}{m-1}(\beta_1+\beta_2 z)^i z^i \\ &= \sum_{i=0}^{\infty}\binom{i+m-1}{m-1}\beta_1^i\left(1+\frac{\beta_2}{\beta_1}z\right)^i z^i.\end{aligned}$$

We can next use the binomial theorem to obtain

$$\frac{1}{q(z)^m} = \sum_{i=0}^{\infty}\binom{i+m-1}{m-1}\beta_1^i\sum_{j=0}^{i}\binom{i}{j}\left(\frac{\beta_2}{\beta_1}\right)^j z^{i+j}.$$

The coefficient of z^n occurs when $i+j=n$ and so we find that

$$q_{m,n} = \sum_{i=0}^{n}\binom{i+m-1}{m-1}\binom{i}{n-i}\beta_1^i\left(\frac{\beta_2}{\beta_1}\right)^{n-i}. \tag{6.1}$$

We can then shift and scale $q_{m,n}$ to determine $p_{m,n}$.

Example 6.2. Consider the generating function

$$G(z) = \left(\frac{z}{1-z-z^2}\right)^m.$$

We can express this generating function as

$$G(z) = \frac{1}{(1-z(1+z))^m}z^m.$$

Thus, $\beta_1=\beta_2=1$ and by (6.1), the coefficient of z^n in $1/(1-z-z^2)^m$ is

$$\sum_{k=0}^{n}\binom{k+m-1}{m-1}\binom{k}{n-k}.$$

Shifting by $-m$, we find that the coefficient of z^n in $G(z)$ is

$$g_n = \sum_{k=0}^{n-m}\binom{k+m-1}{m-1}\binom{k}{n-m-k}.$$

We note that this equation gives a binomial coefficient identity for the convolution of m Fibonacci numbers. ∎

To expand $G(y,z)$ for degree$(q(z)) \geq 3$ we can continue with either approach. If degree$(q(z)) = 3$ and $q(z) = (1-az)(1-bz)(1-cz)$, we use (A.2) to find the expansion of $1/((1-az)(1-bz))^m$, multiply the result by $1/(1-cz)^m$, and apply (A.2) to each term. This approach can be extended to any degree for $q(z)$. The resulting sums are unwieldy but the algorithm is consistent at every step.

If $q(z) = 1 - \beta_1 z - \beta_2 z^2 - \cdots - \beta_d z^d$, we can expand $1/q(z)^m$ to obtain

$$\begin{aligned}\frac{1}{(1-\beta_1 z - \beta_2 z^2 - \cdots - \beta_d z^d)^m} &= \frac{1}{(1 - z(\beta_1 + \beta_2 z + \cdots + \beta_d z^{d-1})^m} \\ &= \sum_{i=0}^{\infty} \binom{i+m-1}{m-1} (\beta_1 + \beta_2 z + \cdots + \beta_d z^{d-1})^i z^i.\end{aligned}$$

For $d \geq 2$, continuing to expand this generating function also proves to be unwieldy.

Expansion of $\mathcal{H}(y,z) = 1/(1-\alpha y - \beta z - \gamma yz)$

To expand $\mathcal{H}(y,z)$ we consider various cases for α, β, and γ. The cases where $\alpha = \gamma = 0$, $\beta = \gamma = 0$, $\alpha = \beta = 0$, or $\alpha = \beta = \gamma = 0$ are trivial.

The case where $\alpha = 0$, $\beta \neq 0$, and $\gamma \neq 0$ is equivalent to the case where $\alpha \neq 0$, $\beta = 0$, and $\gamma \neq 0$. We consider only the former. Expanding as a function of y and then of z we find that

$$\begin{aligned}\frac{1}{1-\beta z - \gamma yz} &= \frac{1}{1-\beta z} \frac{1}{1 - \frac{\gamma z}{1-\beta z} y} \\ &= \sum_{m=0}^{\infty} \frac{(\gamma z)^m}{(1-\beta z)^{m+1}} y^m \\ &= \sum_{m=0}^{\infty} \gamma^m z^m \sum_{n=0}^{\infty} \binom{n+m}{m} \beta^n z^n y^m \\ &= \sum_{m=0}^{\infty} \sum_{n=m}^{\infty} \gamma^m \binom{n}{m} \beta^{n-m} z^n y^m.\end{aligned}$$

If $\beta = \gamma = 1$ we have the super generating function of the binomial coefficients.

If $\alpha \neq 0$, $\beta \neq 0$ and $\gamma = 0$, we can expand as a function of y and then of z to obtain

$$\begin{aligned}\frac{1}{1-\alpha y - \beta z} &= \frac{1}{1-\beta z} \frac{1}{1 - \frac{\alpha}{1-\beta z} y} \\ &= \sum_{m=0}^{\infty} \frac{\alpha^m}{(1-\beta z)^{m+1}} y^m \\ &= \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \alpha^m \binom{n+m}{m} \beta^n z^n y^m.\end{aligned}$$

If $\alpha \neq 0$, $\beta \neq 0$, and $\gamma \neq 0$ we have two cases to consider. When $\alpha\beta = -\gamma$ we have a hybrid generating function which expands to

$$\frac{1}{1-\alpha y - \beta z + \alpha\beta yz} = \frac{1}{(1-\alpha y)(1-\beta z)}$$

$$= \sum_{m=0}^{\infty} \sum_{n=m}^{\infty} \alpha^m \beta^n z^n y^m.$$

Otherwise, we can expand as a function of y to obtain

$$\begin{aligned} \frac{1}{1-\alpha y-\beta z-\gamma yz} &= \frac{1}{1-\beta z} \frac{1}{1-\frac{(\alpha+\gamma z)}{1-\beta z} y} \\ &= \sum_{m=0}^{\infty} \frac{(\alpha+\gamma z)^m}{(1-\beta z)^{m+1}} y^m. \end{aligned}$$

The generating function $(\alpha+\gamma z)^m$ encodes the sequence $\langle \binom{m}{k} \alpha^k \gamma^{m-k} \rangle$ and $(1-\beta z)^{m+1}$ encodes the sequence $\langle \binom{k+m}{k} \beta^k \rangle$. To expand $(\alpha+\gamma z)^m/(1-\beta z)^{m+1}$, we recognize the convolution

$$\begin{aligned} \frac{(\alpha+\gamma z)^m}{(1-\beta z)^{m+1}} &= \sum_{k=0}^{m} \binom{m}{k} \alpha^k \gamma^{m-k} z^k \sum_{n=0}^{\infty} \binom{n+m}{m} \beta^n z^n \\ &= \sum_{k=0}^{m} \sum_{n=0}^{\infty} \binom{m}{k} \alpha^k \gamma^{m-k} \binom{n+m}{m} \beta^n z^{n+k} \\ &= \sum_{k=0}^{m} \sum_{n=k}^{\infty} \binom{m}{k} \alpha^k \gamma^{m-k} \binom{n-k+m}{m} \beta^{n-k} z^n \\ &= \sum_{n=0}^{\infty} \sum_{k=0}^{\min(m,n)} \binom{m}{k} \alpha^k \gamma^{m-k} \binom{n-k+m}{m} \beta^{n-k} z^n. \end{aligned}$$

Thus, we find that

$$\frac{1}{1-\alpha y-\beta z-\gamma yz} = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \sum_{k=0}^{\min(m,n)} \binom{m}{k} \alpha^k \gamma^{m-k} \binom{n-k+m}{m} \beta^{n-k} z^n y^m.$$

While there were similarities in the approaches we used to expand each of these generating functions, we see that each had to be handled in a slightly different way. Generating functions of higher degree or more variables become even more difficult to expand.

6.1.3. Closed Form Expressions

When dealing with sequences of a single variable, we were able to define a closed form expression in such a way that the definition held for all sequences with rational generating functions. If we consider the binomial coefficients, we see that they do not satisfy this definition.

The binomial coefficients can be defined as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n^{\underline{k}}}{k!}.$$

The factorial definition uses $(n-1)+(k-1)+(n-k-1)+2 = 2n-1$ multiplications and divisions. The falling factorial definition uses $(k-1)+(k-1)+1 = 2k-1$ multiplications and divisions. We apparently cannot compute $\binom{n}{k}$ using a number of operations independent of both n and k.

Ideally, we would like to extend the definition of closed form to multivariate sequences encoded by rational generating functions. Unfortunately, since we do not know a general method for expanding multivariate rational generating functions, we cannot be sure if there is a definition that will hold for all sequences that they encode. Therefore, we will not attempt to extend formally the definition of closed form. We will examine instead some issues that should be considered when defining closed forms for multivariate sequences.

We can consider two changes to the definition of a closed form that the binomial coefficients will satisfy. Our previous definition considered θ^n to be a closed form even though it is defined as $n-1$ multiplications. We could extend the definition to include factorial or falling factorial as acceptable operations. A second option for m variable recurrences is to define the number of operations to be dependent on only $m-1$ of the variables. In this way, the "complexity" of the closed form is less than that needed to compute the sequence from its recursive definition. For example, the recursive definition of the binomial coefficients involves two parameters, whereas the number of operations defining them using either factorials or falling factorials involves only one of the parameters.

The problem domain should also be considered when defining closed form. Generating functions of the form

$$\frac{1}{p(z)^m}$$

are actually partially expanded bivariate rational generating functions that encode sequences defined by a convolution over m factors. For each value of m, the generating function m encodes a different convolution, each of which defines a different sequence. For a particular m, the same expression defines the coefficient of z^n for all n and the number of operations is independent of n. When regarded as the solution of a convolution of m factors, the coefficient of z^n satisfies the definition of closed form. In this case, m is a parameter of the problem and not a variable relevant to the limits of the summation.

6.2. Definite Binomial Sums

A definite binomial sum is defined as

$$s_n = \sum_{k=0}^{n} \binom{m}{k} a_k.$$

Because of the symmetry of the binomial coefficients,

$$\sum_{k=0}^{n} \binom{n}{k} a_k = \sum_{k=0}^{n} \binom{n}{n-k} a_k = \sum_{k=0}^{n} \binom{n}{k} a_{n-k}.$$

Thus, if we can solve the definite binomial sum, then we can solve the corresponding convolution.

Theorem 6.1. Let $\langle a_k \rangle$ be in RAT_SEQ. The sequence $\langle s_n \rangle$ defined by

$$s_n = \sum_{k=0}^{n} \binom{n}{k} a_k$$

is in RAT_SEQ.

Proof. Applying (2.1) we find

$$\begin{aligned} S(z) &= \sum_{n=0}^{\infty} s_n z^n \\ &= \sum_{n=0}^{\infty} \sum_{k=0}^{n} \binom{n}{k} a_k z^n \\ &= \sum_{k=0}^{\infty} \sum_{n=k}^{\infty} \binom{n}{k} a_k z^n \\ &= \sum_{k=0}^{\infty} \sum_{n=0}^{\infty} \binom{n+k}{k} a_k z^{n+k}. \end{aligned}$$

The presence of the summation's upper limit in the summand affects the derivation. We can recognize that

$$\sum_{n=0}^{\infty} \binom{n+k}{k} z^n = \frac{1}{(1-z)^{k+1}}.$$

Rearranging terms and using this result gives

$$\begin{aligned} S(z) &= \sum_{k=0}^{\infty} a_k z^k \sum_{n=0}^{\infty} \binom{n+k}{k} z^n \\ &= \sum_{k=0}^{\infty} a_k z^k \left(\frac{1}{1-z} \right)^{k+1} \\ &= \frac{1}{1-z} \sum_{k=0}^{\infty} a_k \left(\frac{z}{1-z} \right)^k. \end{aligned}$$

This sum is just (2.1) evaluated at $z = z/(1-z)$. We find that

$$S(z) = \frac{1}{1-z} A\left(\frac{z}{1-z} \right). \tag{6.2}$$

Thus, $A(z/(1-z)) \in RAT$, $S(z) \in RAT$, and $\langle s_n \rangle \in RAT\_SEQ$. □

We can use (6.2) with any sequence $\langle a_k \rangle$ for which we know the ordinary generating function. The only restriction to finding the solution to the definite binomial sum is our ability to expand the resulting generating function. However, for sequences from RAT_SEQ, including hybrid terms whose factors are from RAT_SEQ, we know how to expand the generating function of the solution.

Example 6.3. Consider the sum

$$s_n = \sum_{k=0}^{n} \binom{n}{k} F_k.$$

By (6.2), the generating function of $\langle s_n \rangle$ is

$$S(z) = \frac{1}{1-z} F\left(\frac{z}{1-z}\right) = \frac{1}{1-z} \frac{\frac{z}{1-z}}{1 - \left(\frac{z}{1-z}\right) - \left(\frac{z}{1-z}\right)^2} = \frac{z}{1-3z+z^2}.$$

Applying partial fractions with respect to z we find that

$$S(z) = \frac{1}{\sqrt{5}} \left(\frac{1}{1 - \frac{3+\sqrt{5}}{2} z} - \frac{1}{1 - \frac{3-\sqrt{5}}{2} z} \right).$$

Therefore

$$s_n = \frac{1}{\sqrt{5}} \left(\left(\frac{3+\sqrt{5}}{2}\right)^n - \left(\frac{3-\sqrt{5}}{2}\right)^n \right). \quad \blacksquare$$

Notice that this closed form can be obtained directly by applying the binomial theorem to the closed form expression for the Fibonacci numbers. As we noted in Chapter 2, $S(z)$ is the generating function of the sequence $\langle F_{2n} \rangle$. By (5.3), $F_{2n} = F_{n+1}F_n + F_nF_{n-1}$. It is natural to wonder if in general the sequence $\langle s_n \rangle$ defined by the definite binomial sum has an extended closed form that involves terms from the sequence in the summand. The answer is no, as we can demonstrate with the binomial theorem,

$$\sum_{k=0}^{n} \binom{n}{k} c^k = (1+c)^n.$$

The solution $(1+c)^n$ does not have an extended closed form expression involving a fixed number of terms from $\langle c^n \rangle$.

We can also apply the hybrid sum method to evaluate the definite binomial sum

$$s_n = \sum_{k=0}^{n} \binom{n}{k} a_k.$$

The generating function of the binomial coefficients is

$$B_n(z) = \sum_{k=0}^{n} \binom{n}{k} = (1+z)^n.$$

By Corollary 2.1.2, s_n is the coefficient of $w^n z^n$ in

$$G(w,z) = \frac{1}{1-wz}(1+z)^n A(w),$$

where $A(w)$ is the generating function of $\langle a_k \rangle$. If $A(w)$ is a rational generating function, we can write $A(w) = p(w)/q(w)$, where $p(w)$ and $q(w)$ are polynomials in w. By (A.4), we find that applying partial fractions with respect to w gives

$$G(w,z) = (1+z)^n p(w)\left(\frac{1}{q(1/z)\,(1-wz)} + \frac{s(w,z)}{z^m q(1/z) q(w)}\right),$$

where $s(w,z)$ is a polynomial of w and z and m is the degree of $q(w)$. We can discard the left hand term. Since $(1+z)^n$ is a trivial rational generating function, we cannot apply partial fractions with respect to z to the right hand term

$$\frac{(1+z)^n p(w) s(w,z)}{z^m q(1/z) q(w)}.$$

Expanding as a function of z results in an $n+1$ term convolution encoded by the generating function $(1+z)^m/(z^m q(1/z))$. However, super generating functions can help us out. The function $(1+z)^n$ is the coefficient of $y^n z^n$ in $1/(1-y-yz)$. Thus we can encode $(1+z)^n$ with its nontrivial rational super generating function, apply partial fractions with respect to z to

$$\frac{p(w) s(w,z)}{z^m q(1/z) q(w)\,(1-y-yz)},$$

and expand the generating functions. The summation index is encoded by the generating function variables w and z. The summation limit is $k = n$. Thus we want to find the coefficient of $w^n z^n$. The generating function variable y was used to encode $(1+z)^n$ as a super generating function. The desired term is the coefficient of y^n.

We now apply this method to the definite binomial sum of the Fibonacci numbers.

Example 6.4. Again consider the sum

$$s_n = \sum_{k=0}^{n} \binom{n}{k} F_k.$$

Applying Corollary 2.1.2 and super generating function techniques, we find that s_n is the coefficient of $w^n y^n z^n$ in

$$H(w,y,z) = \frac{1}{1-wz}\,\frac{w}{1-w-w^2}\,\frac{1}{1-y-yz}.$$

Applying partial fractions with respect to w gives

$$H(w,y,z) = \frac{w}{1-y-yz}\left(\frac{-z^2}{1+z-z^2}\,\frac{1}{1-wz} + \frac{1+z+wz}{1+z-z^2}\,\frac{1}{1-w-w^2}.\right)$$

We can discard the left hand term. Expanding the right hand term as a function of w and applying partial fractions with respect to z gives

$$\frac{F_n + zF_n + zF_{n-1}}{(1+z-z^2)(1-y-yz)}$$
$$= (F_n + zF_n + zF_{n-1})\left(\frac{1-2y+yz}{(1-3y+y^2)(1+z-z^2)} - \frac{y^2}{(1-3y+y^2)(1-y-yz)}\right).$$

The generating function $1/(1+z-z^2)$ encodes the sequence $\langle(-1)^n F_{n+1}\rangle$. Thus, the coefficient of z^n is

$$\frac{y(-1)^n(F_{n+1}F_{n-1} - F_n^2)}{1-3y+y^2} - \frac{F_n y^{n+2} + F_{n+1}y^{n+1}}{(1-3y+y^2)(1-y)^{n+1}}. \tag{6.3}$$

The first nonzero term in the expansion of the right hand term is at y^{n+1} and so the coefficient of y^n in the right hand term is zero. By Cassini's identity the left hand term is $y/(1-3y+y^2)$ and encodes the sequence

$$s_n = F_{2n}. \quad \blacksquare$$

Notice that if we tried to solve the sum

$$s_n = \sum_{k=0}^{n} \binom{m}{k} F_k$$

by this method, we would want to find the coefficient of y^m in (6.3). If $m \leq n$, this is the definite binomial sum and the coefficient of y^m in the right hand term of (6.3) is zero. However, if $m > n$, then the coefficient of y^m in the right hand term is not zero. Instead, it involves the sum of $m - n + 1$ terms from the convolution encoded by (6.3). Thus we see that the hybrid sum method works because of the presence of the factor of y^{n+1} in the numerator of the right hand term of (6.3).

We can attack the definite binomial sum in yet a third way. By symmetry of the binomial coefficients

$$s_n = \sum_{k=0}^{n} \binom{n}{k} a_k = \sum_{k=0}^{n} \binom{n}{n-k} a_k.$$

Applying (2.17) and super generating function techniques, we find that s_n is the coefficient of $y^n z^n$ in

$$S(z) = \frac{1}{1-y-yz} A(z),$$

where $A(z)$ is the generating function of $\langle a_k \rangle$. If $A(z)$ is rational we can apply partial fractions with respect to z and expand the result to find the coefficient of $y^n z^n$.

Example 6.5. Consider the sum

$$s_n = \sum_{k=0}^{n} \binom{n}{k} F_k = \sum_{k=0}^{n} \binom{n}{n-k} F_k.$$

By (2.17) and super generating function techniques, s_n is the coefficient of $y^n z^n$ in

$$S(y,z) \;=\; \frac{z}{(1-y-yz)(1-z-z^2)}.$$

Applying partial fractions with respect to z gives

$$S(y,z) \;=\; \frac{y+z-yz}{(1-y-y^2)(1-z-z^2)} - \frac{(1-y)y}{(1-y-y^2)(1-y-yz)}.$$

If we expand the right hand term as a function of z we find that the numerator has a factor of y^{n+1}. Therefore the coefficient of $y^n z^n$ in the right hand term is zero. If we expand the left hand term as a function of z we get

$$S(y) \;=\; \frac{yF_{n+1}+(1-y)F_n}{1-y-y^2}.$$

Expanding $S(y)$ as a function of y and simplifying gives

$$s_n \;=\; 2F_nF_{n+1}-F_n^2 \;=\; F_{n+1}F_n+F_nF_{n-1} \;=\; F_{2n}. \quad \blacksquare$$

These three methods give us the tools to evaluate many types of definite binomial sums. The multifactor hybrid sum

$$s_n \;=\; \sum_{k=0}^{n} \binom{n}{k} a_{1,k}\cdots a_{m,k}$$

can be evaluated by the hybrid sum technique. We can also find the generating function of the hybrid sequence $\langle a_{1,k}\cdots a_{m,k}\rangle$ and apply one of the other approaches. The multifactor convolution

$$s_n \;=\; \sum_{\alpha_1+\alpha_2+\cdots+\alpha_m=n} \binom{n}{\alpha_1} a_{2,\alpha_2}\cdots a_{m,\alpha_m}$$

can be written as

$$\begin{aligned}
s_n &= \sum_{\alpha_1=0}^{n} \binom{n}{\alpha_1} \sum_{\alpha_2+\cdots+\alpha_m=n-\alpha_1} a_{2,\alpha_2}\cdots a_{m,\alpha_m} \\
&= \sum_{\alpha_1=0}^{n} \binom{n}{n-\alpha_1} \sum_{\alpha_2+\cdots+\alpha_m=n-\alpha_1} a_{2,\alpha_2}\cdots a_{m,\alpha_m} \\
&= \sum_{\alpha_1=0}^{n} \binom{n}{n-\alpha_1} \sum_{\alpha_2+\cdots+\alpha_m=\alpha_1} a_{2,\alpha_2}\cdots a_{m,\alpha_m} \\
&= \sum_{\alpha_1=0}^{n} \binom{n}{\alpha_1} \sum_{\alpha_2+\cdots+\alpha_m=\alpha_1} a_{2,\alpha_2}\cdots a_{m,\alpha_m}.
\end{aligned}$$

We can first solve the inner convolution or find its generating function. We then apply one of the previously described methods to the resulting definite binomial sum. If all of the sequences $\langle a_{i,k}\rangle$ for $2 \le i \le m$ are in RAT_SEQ, then their convolution is in RAT_SEQ and $\langle s_n\rangle$ is in RAT_SEQ.

6.3. Indefinite Binomial Sums

As sequences indexed by n, the binomial coefficients $\binom{n+m}{m}$ and $\binom{n}{m}$ are polynomials in n of degree m. Their generating functions are

$$\sum_{n=0}^{\infty} \binom{n+m}{m} z^n = \frac{z^m}{(1-z)^{m+1}}$$

and

$$\sum_{n=m}^{\infty} \binom{n}{m} z^n = \frac{1}{(1-z)^{m+1}}.$$

As we can see, the two sequences are similar since their generating functions differ only by a factor of z^m. From their denominators, we see that they both satisfy the same order $m+1$ recurrence. Their denominators also indicate that they are polynomials of degree m.

For particular values of m, $\langle \binom{n+m}{m} \rangle$ and $\langle \binom{n}{m} \rangle$ are in RAT_SEQ. Any hybrid term involving the product between one such binomial and a sequence in RAT_SEQ will also be in RAT_SEQ. Thus, we can solve hybrid sums involving these binomials and express the answer in an extended closed form that involves $m+1$ consecutive terms from $\langle \binom{n+m}{m} \rangle$ or $\langle \binom{n}{m} \rangle$.

We can also find generating functions for hybrid terms involving $\langle \binom{n+m}{m} \rangle$ and $\langle \binom{n}{m} \rangle$ for particular values of m. The generating function of $\langle s_n \rangle = \langle \binom{n}{m} a_n \rangle$ is

$$S(z) = \frac{z^m}{m!} \frac{d^m}{dz^m} A(z), \tag{6.4}$$

where $A(z)$ is the generating function of $\langle a_n \rangle$. We can use (2.1) to prove (6.4),

$$\begin{aligned} S(z) &= \frac{z^m}{m!} \frac{d^m}{dz^m} A(z) \\ &= \frac{z^m}{m!} \frac{d^m}{dz^m} \sum_{n=0}^{\infty} a_n z^n \\ &= \frac{z^m}{m!} \sum_{n=m}^{\infty} a_n n^{\underline{m}} z^{n-m} \\ &= \sum_{n=m}^{\infty} a_n \frac{n^{\underline{m}}}{m!} z^n \\ &= \sum_{n=m}^{\infty} a_n \binom{n}{m} z^n. \end{aligned}$$

The generating function of $\langle s_n \rangle = \langle \binom{n+m}{m} a_n \rangle$ is

$$S(z) = \frac{d^m}{dz^m} \frac{z^m}{m!} A(z), \tag{6.5}$$

where $A(z)$ is the generating function of $\langle a_n \rangle$. We can again use (2.1) to prove (6.5),

$$\begin{aligned} S(z) &= \frac{d^m}{dz^m}\frac{z^m}{m!}A(z) \\ &= \frac{1}{m!}\frac{d^m}{dz^m}\sum_{n=0}^{\infty} a^n z^{n+m} \\ &= \frac{1}{m!}\sum_{n=0}^{\infty} a_n (n+m)^{\underline{m}} z^n \\ &= \sum_{n=0}^{\infty} a_n \frac{(n+m)^{\underline{m}}}{m!} z^n \\ &= \sum_{n=0}^{\infty} a_n \binom{n+m}{m} z^n. \end{aligned}$$

Either (6.4) or (6.5) can be used to solve hybrid sums involving these binomial coefficients. We employ (6.4) or (6.5) to find the generating function of the summand, apply (2.20) to find the generating function of the solution, then expand to find the solution.

If possible, we would like to find a solution to hybrid sums involving the sequences $\langle \binom{n+m}{m} \rangle$ and $\langle \binom{n}{m} \rangle$ for all m, rather than for particular values of m. We apply the hybrid sum technique to one such sum.

Example 6.6. Consider the hybrid sum

$$s_n = \sum_{k=0}^{n} \binom{k+m}{m} 2^k.$$

Table 6.1 presents solutions to this sum for $1 \leq m \leq 5$. We would like to find an expression for the sum involving the symbol m. There is no apparent single closed form that is valid for all m. While there is some regularity to the form of these solutions, there appears to be no way to predict the constants in the expressions. Since the sequence $\langle \binom{n+m}{m} \rangle$ satisfies an order $m+1$ recurrence, the solution to this sum may require $m+1$ consecutive terms from the sequence $\langle \binom{n+m}{m} \rangle$.

We can apply the hybrid sum method in an attempt to find such an expression. By Corollary 2.1.2, s_n is the coefficient of $w^n z^n$ in

$$H(w,z) = \frac{1}{(1-wz)(1-2w)(1-z)^{m+1}}.$$

Applying partial fractions with respect to w gives

$$H(w,z) = \frac{1}{(1-z)^{m+1}}\left(\frac{2}{(2-z)(1-2w)} - \frac{z}{(2-z)(1-wz)}\right).$$

We can discard the right hand term. The coefficient of w^n in the left hand term is

$$\frac{2^{n+1}}{(2-z)(1-z)^{m+1}}.$$

$$\sum_{k=0}^{n}\binom{n+1}{1}2^k = 2^{n+1}\binom{n}{1}+1$$

$$\sum_{k=0}^{n}\binom{n+2}{2}2^k = 2^{n+1}\left(\binom{n+2}{2}-\binom{n+1}{2}+\binom{n}{2}\right)-1$$

$$\sum_{k=0}^{n}\binom{n+3}{3}2^k = 2^{n+1}\left(2\binom{n+2}{3}-2\binom{n+1}{3}+\binom{n}{3}\right)+1$$

$$\sum_{k=0}^{n}\binom{n+4}{4}2^k = 2^{n+1}\left(\binom{n+4}{4}-2\binom{n+3}{4}+4\binom{n+2}{4}-3\binom{n+1}{4}+\binom{n}{4}\right)-1$$

$$\sum_{k=0}^{n}\binom{n+5}{5}2^k = 2^{n+1}\left(3\binom{n+4}{5}-6\binom{n+3}{5}+7\binom{n+2}{5}-4\binom{n+1}{5}+\binom{n}{5}\right)+1$$

Table 6.1. Solutions to $\sum_{k=0}^{n}\binom{k+m}{m}2^k$ for $1 \le m \le 5$

The coefficient of z^n is a convolution defined by the product of the generating functions $1/(2-z)$ and $1/(1-z)^{m+1}$. This convolution has $n+1$ terms, which is not a desirable. We could use (A.1) to perform partial fractions over z. However, we want to find a general method for solving any hybrid sum involving binomial coefficients. While we have $2-z$ in the denominator, we can expect in general to have a polynomial of any degree. In the general case we could determine the partial fraction expansion for the polynomial, multiply by $1/(1-z)^{m+1}$, and use (A.1) to apply partial fractions over z for each term. While such an approach works in principle, it quickly becomes complicated.

Fortunately, super generating functions can help us out. The value $1/(1-z)^{m+1}$ is the coefficient of y^m in the super generating function $1/(1-y-z)$. Thus, using the super generating function we find that s_n is the coefficient of $y^m z^n$ in

$$\frac{2^{n+1}}{(2-z)(1-y-z)}.$$

Applying partial fractions with respect to z gives

$$\frac{2^{n+1}}{(2-z)(1-y-z)} = \frac{-2^{n+1}}{(1+y)(2-z)} + \frac{2^{n+1}}{(1+y)(1-y-z)}. \tag{6.6}$$

Expanding the left hand term we find that the coefficient of y^m in $1/(1+y)$ is $(-1)^m$ and the coefficient of z^m in $1/(2-z)$ is $1/2^{n+1}$. Thus the left hand term expands to $(-1)^{m+1}$. Expanding the right hand term as a function of z gives

$$\frac{2^{n+1}}{(1+y)(1-y)^{n+1}}. \tag{6.7}$$

The generating function $1/(1+y)$ encodes the sequence $\langle(-1)^m\rangle$ and the generating function $1/(1-y)^{n+1}$ encodes the sequence $\langle\binom{m+n}{m}\rangle$. Thus, the generating function in (6.7) encodes the convolution

$$2^{n+1}\sum_{i=0}^{m}(-1)^i\binom{n+m-i}{m-i},$$

and

$$s_n = (-1)^{m+1} + 2^{n+1}\sum_{i=0}^{m}(-1)^i\binom{n+m-i}{m-i}.$$

The binomial coefficient encodes a polynomial in n of degree $m-i$, and so the sum evaluates to a polynomial of degree m. This is a correct solution which only involves $m+1$ binomial coefficients. Unfortunately, they are not the binomial coefficients that we are looking for.

We can find an expression involving the desired binomial coefficients as follows. If we expand the right hand term of (6.6) as a function of y we find that the coefficient of y^m is the convolution

$$2^{n+1}\sum_{i=0}^{m}(-1)^i\frac{1}{(1-z)^{m+1-i}}.$$

Rearranging this generating function gives

$$\begin{aligned}
2^{n+1}\sum_{i=0}^{m}(-1)^i\frac{1}{(1-z)^{m+1-i}} &= 2^{n+1}\frac{1}{(1-z)^{m+1}}\sum_{i=0}^{m}(z-1)^i \\
&= 2^{n+1}\frac{1}{(1-z)^{m+1}}\sum_{i=0}^{m}\sum_{j=0}^{i}\binom{i}{j}z^j(-1)^{i-j} \\
&= 2^{n+1}\frac{1}{(1-z)^{m+1}}\sum_{j=0}^{m}\left(\sum_{i=j}^{m}(-1)^i\binom{i}{j}\right)(-1)^j z^j.
\end{aligned}$$

Expanding as a function of z and adding in the expansion of the left hand term of (6.6) we find that

$$s_n = (-1)^{m+1} + 2^{n+1}\sum_{j=0}^{m}\left(\sum_{i=j}^{m}(-1)^i\binom{i}{j}\right)(-1)^j\binom{n-j+m}{m}.$$

We have found two expressions for s_n. Note that for a given value of m, both solutions involve a fixed number of operations that is independent of n. Thus, each result meets our definition of closed form. We also note that super generating functions have given us a method to perform partial fractions when the denominator involves a polynomial raised to a power given by a symbolic parameter. ∎

We now look at a more complicated sum involving binomial coefficients.

$$\sum_{k=0}^{n}\binom{k+1}{1}F_k = 1+F_n\left(-\binom{n+1}{1}+3\binom{n}{1}\right) + F_{n-1}\left(-\binom{n+1}{1}+2\binom{n}{1}\right)$$

$$\sum_{k=0}^{n}\binom{k+2}{2}F_k = -2+F_n\left(4\binom{n+2}{2}-7\binom{n+1}{2}+5\binom{n}{2}\right) + F_{n-1}\left(2\binom{n+2}{2}-4\binom{n+1}{2}+3\binom{n}{2}\right)$$

$$\sum_{k=0}^{n}\binom{k+3}{3}F_k = 3+F_n\left(-4\binom{n+3}{3}+17\binom{n+3}{3}-19\binom{n+1}{3}+8\binom{n}{3}\right) + F_{n-1}\left(-3\binom{n+3}{3}+11\binom{n+2}{3}-12\binom{n+1}{3}+5\binom{n}{3}\right)$$

$$\sum_{k=0}^{n}\binom{k+4}{4}F_k = -5+F_n\left(9\binom{n+4}{4}-35\binom{n+3}{4}+59\binom{n+2}{4}-44\binom{n+1}{4}+13\binom{n}{4}\right) + F_{n-1}\left(5\binom{n+4}{4}-21\binom{n+3}{4}+36\binom{n+2}{4}-27\binom{n+1}{4}+8\binom{n}{4}\right)$$

Table 6.2. Solution to $\sum_{k=0}^{n}\binom{k+m}{m}F_k$ for $1 \le m \le 4$.

Example 6.7. Consider the sum

$$s_n = \sum_{k=0}^{n}\binom{k+m}{m}F_k.$$

Table 6.2 presents the solution to s_n for $1 \le m \le 4$. As we would expect, the solutions involve $2(m+1)$ terms that are products between Fibonacci numbers and binomial coefficients. We can use the hybrid sum method to find an expression for s_n for general m. By Corollary 2.1.2 and super generating function techniques, s_n is the coefficient of $w^n y^m z^n$ in

$$H(w,y,z) = \frac{w}{(1-wz)(1-w-w^2)(1-y-z)}.$$

Applying partial fractions over w we get

$$H(w,y,z) = \frac{w}{1-y-z}\left(\frac{-z^2}{(1+z-z^2)(1-wz)} + \frac{1+z+wz}{(1+z-z^2)(1-w-w^2)}\right).$$

We can discard the left hand term. Expanding the right hand term as a function of w

and applying partial fractions over z gives

$$\begin{aligned}
&\frac{F_n + zF_n + zF_{n-1}}{(1+z-z^2)(1-y-z)} \\
&\quad = (F_n + zF_n + zF_{n-1})\left(\frac{y-z}{(1+y-y^2)(1+z-z^2)} + \frac{1}{(1+y-y^2)(1-y-z)}\right).
\end{aligned}$$

Expanding the left hand term as a function of z and simplifying with Cassini's identity we obtain

$$\frac{(-1)^{n+1}(F_{n+1}F_{n-1} - F_n^2)}{1+y-y^2} = \frac{-1}{1+y-y^2}.$$

The coefficient of y^m in this generating function is $(-1)^{m+1}F_{m+1}$. As a function of y, $1/((1+y-y^2)(1-y-z))$ encodes a convolution between the sequences $\langle(-1)^k F_{k+1}\rangle$ and $\langle 1/(1-z)^{k+1}\rangle$. Expanding the right hand term as a function of y and rearranging gives

$$\begin{aligned}
&(1+z)F_n \sum_{i=0}^{m}(-1)^i F_{i+1}\frac{1}{(1-z)^{m+1-i}} + zF_{n-1}\sum_{i=0}^{m}(-1)^i F_{i+1}\frac{1}{(1-z)^{m+1-i}} \\
&\quad = F_n \sum_{i=0}^{m}(-1)^i F_{i+1}\frac{1}{(1-z)^{m+1-i}} + z(f_n + F_{n-1})\sum_{i=0}^{m}(-1)^i F_{i+1}\frac{1}{(1-z)^{m+1-i}} \\
&\quad = F_n\frac{1+z}{(1-z)^{m+1}}\sum_{i=0}^{m}(-1)^i F_{i+1}(1-z)^i + F_{n+1}\frac{z}{(1-z)^{m+1}}\sum_{i=0}^{m}(-1)^i F_{i+1}(1-z)^i \\
&\quad = F_n\frac{1+z}{(1-z)^{m+1}}\sum_{i=0}^{m}\sum_{j=0}^{i} F_{i+1}\binom{i}{j}z^j(-1)^{i-j} \\
&\qquad + F_{n+1}\frac{z}{(1-z)^{m+1}}\sum_{i=0}^{m}\sum_{j=0}^{i} F_{i+1}\binom{i}{j}z^j(-1)^{i-j} \\
&\quad = F_n\frac{1+z}{(1-z)^{m+1}}\sum_{j=0}^{m}\left(\sum_{i=j}^{m}(-1)^i F_{i+1}\binom{i}{j}\right)(-1)^j z^j \\
&\qquad + F_{n+1}\frac{z}{(1-z)^{m+1}}\sum_{j=0}^{m}\left(\sum_{i=j}^{m}(-1)^i F_{i+1}\binom{i}{j}\right)(-1)^j z^j.
\end{aligned}$$

The coefficient of z^n in this generating function is

$$\begin{aligned}
&F_n\sum_{j=0}^{m}\left(\sum_{i=j}^{m}(-1)^i F_{i+1}\binom{i}{j}\right)(-1)^j\binom{n+m-j}{m} \\
&\quad + F_{n+1}\sum_{j=0}^{m}\left(\sum_{i=j}^{m}(-1)^i F_{i+1}\binom{i}{j}\right)(-1)^j\binom{n+m-j-1}{m}.
\end{aligned}$$

Thus, we find that

$$s_n = (-1)^{m+1}F_{m+1}$$

$$+ F_n \sum_{j=0}^{m} \left(\sum_{i=j}^{m} (-1)^i F_{i+1} \binom{i}{j} \right) (-1)^j \binom{n+m-j}{m}$$
$$+ F_{n+1} \sum_{j=0}^{m} \left(\sum_{i=j}^{m} (-1)^i F_{i+1} \binom{i}{j} \right) (-1)^j \binom{n+m-j-1}{m}.$$

We note that some simplification is possible. This solution involves the $m+2$ binomial coefficients $\binom{n-1}{m}, \ldots, \binom{n+m}{m}$. For a particular value of m, we can simplify this to $m+1$ consecutive binomial coefficients. ∎

Chapter 7

Harmonic Numbers

In this chapter we demonstrate the application of generating functions to summations involving harmonic numbers in their summands or solutions. The *harmonic numbers* are the sequence $\langle H_n \rangle$ defined by

$$H_n = \sum_{k=1}^{n} \frac{1}{k}.$$

As a result of Karr's work, it can be shown that there is no closed form expression involving rational functions of n or exponentials of n that defines H_n [DESAI, KARR]. Thus, the solutions to summations that involve the reciprocals of integers require extended closed forms that use the symbol H_n.

The *generalized harmonic numbers* are defined by

$$H_n^{(m)} = \sum_{k=1}^{n} \frac{1}{k^m},$$

for integer $m \geq 1$. Again, for any m there is no closed form expression for the summation.

The generating function of the sequence $\langle 1/k \rangle$ is

$$G(z) = \ln\left(\frac{1}{1-z}\right).$$

By (2.20), the generating function of $\langle H_n \rangle$ is

$$H(z) = \frac{1}{1-z} G(z) = \frac{1}{1-z} \ln\left(\frac{1}{1-z}\right).$$

For $m \geq 2$, there is no known generating function for $\langle H_n^{(m)} \rangle$. However, we can encode $H_n^{(m)}$ using hybrid generating functions. By Corollary 2.2.2, $H_n^{(m)}$ is the coefficient of $(z_1 \cdots z_m)^n$ in the expansion of

$$H(z_1, \ldots, z_m) = \frac{1}{1 - z_1 \cdots z_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1-z_i}\right). \tag{7.1}$$

We note that there is no way to rearrange this generating function in order to find an expression for $H_n^{(m)}$. We cannot use partial fractions as there is only one polynomial in the denominator. Thus, when we expand this generating function, we must recognize it as encoding a harmonic number.

7.1. Poly-Log Generating Functions

To solve summations involving products between harmonic numbers and sequences from RAT_SEQ, we have to expand generating functions that are of the form

$$G(z) = \frac{p(z)}{q(z)} \ln\left(\frac{1}{1-z}\right),$$

where $p(z)$ and $q(z)$ are polynomials. The polynomial $p(z)$ serves to shift and scale the sequence encoded by $H(z) = \ln(1/(1-z))/q(z)$. Thus we need only consider the expansion of $H(z)$. We can further simplify the class of generating functions that we need to expand by applying partial fractions with respect to z to $1/q(z)$, which allows us to write $H(z)$ as a linear combination of terms of the form

$$\frac{1}{(1-\alpha z)^m} \ln\left(\frac{1}{1-z}\right).$$

This generating function is an example of a more general situation involving products between rational and non-rational generating functions. We describe a general method that can be applied in an attempt to expand such generating functions and then apply it to poly-log generating functions. Let $G(z)$ be a non-rational generating function that encodes the sequence $\langle g_n \rangle$ and let

$$F(z) = \frac{1}{(1-\alpha z)^m} G(z).$$

We want to find the sequence $\langle f_n \rangle$ encoded by $F(z)$.

To do so, let

$$S(z) = \frac{1}{(1-\alpha z)^{m-1}} G(z)$$

encode the sequence $\langle s_n \rangle$. By differentiating with respect to z we have

$$T(z) = \frac{d}{dz} S(z) = \frac{(m-1)\alpha G(z)}{(1-\alpha z)^m} + \frac{1}{(1-\alpha z)^{m-1}} \frac{d}{dz} G(z).$$

Substituting, we find that

$$\frac{d}{dz} S(z) = (m-1)\alpha F(z) + \frac{1}{(1-\alpha z)^{m-1}} \frac{d}{dz} G(z).$$

Expanding the generating functions gives

$$(n+1)s_{n+1} = (m-1)\alpha f_n + a_n,$$

where $\langle a_n \rangle$ is the sequence encoded by $G'(z)/(1-\alpha z)^{m-1}$. We then solve for f_n.

For this approach to work, we need to be able to expand $S(z)$ to find s_n and expand $G'(z)/(1-\alpha z)^{m-1}$ to find a_n. To expand $S(z)$ we apply the method recursively. The base condition for the recursion is reached after $m-1$ iterations and requires that we be able to expand

$$\frac{1}{1-\alpha z} G(z).$$

This may be a known generating function. If not, we recognize that this is the generating function that encodes a convolution and apply (2.17) to find that the coefficient of z^n is

$$\sum_{k=0}^{n} g_k \alpha^{n-k} = \alpha^n \sum_{k=0}^{n} \frac{g_k}{\alpha^k}.$$

We then attempt to find a solution for this summation. Thus, this method reduces the problem of expanding $G(z)/(1-\alpha z)^m$ to that of expanding the generating functions $G(z)/(1-\alpha z)$ and $G'(z)/(1-\alpha z)^{m-1}$.

With logarithmic generating functions, we find that this method works if we are able to expand

$$\frac{1}{1-\alpha z} \ln\left(\frac{1}{1-z}\right)$$

and

$$\frac{1}{(1-\alpha z)^{m-1}} \frac{d}{dz} \ln\left(\frac{1}{1-z}\right) = \frac{1}{(1-\alpha z)^{m-1}} \frac{1}{1-z}.$$

The latter generating function is rational and so we know how to expand it. The former encodes the convolution

$$\sum_{k=1}^{n} \frac{\alpha^{n-k}}{k}.$$

The value of this sum is known only when $\alpha = 1$, in which case it evaluates to H_n. Thus, we can use this approach to expand generating functions of the form

$$\frac{1}{(1-z)^m} \ln\left(\frac{1}{1-z}\right).$$

Example 7.1. Consider the generating function

$$F_m(z) = \frac{1}{(1-z)^m} \ln\left(\frac{1}{1-z}\right).$$

We will prove by induction on m that it encodes the sequence

$$\langle f_{m,n} \rangle = \left\langle \binom{n+m-1}{m-1} (H_{n+m-1} - H_{m-1}) \right\rangle, \tag{7.2}$$

for $m \geq 1$.

The base case is $F_1(z) = \ln(1/(1-z))/(1-z)$, which encodes the sequence $\langle H_n \rangle$. Substituting, we find that $f_{1,n} = \binom{n}{0}(H_n - H_0)$. Recognizing $H_0 = \sum_{k=1}^{0} 1/k = 0$, we find that $f_{1,n} = H_n$.

To prove the inductive step, assume that (7.2) is true for $m = m_0 - 1$ and prove it true for $m = m_0$. We want to expand the generating function

$$F_{m_0}(z) = \frac{1}{(1-z)^{m_0}} \ln\left(\frac{1}{1-z}\right).$$

If we differentiate $F_{m_0-1}(z)$ we obtain

$$\begin{aligned}
\frac{d}{dz}F_{m_0-1}(z) &= \frac{d}{dz}\left(\frac{1}{(1-z)^{m_0-1}} \ln\left(\frac{1}{1-z}\right)\right) \\
&= \frac{(m_0-1)}{(1-z)^{m_0}} \ln\left(\frac{1}{1-z}\right) + \frac{1}{(1-z)^{m_0}} \\
&= (m_0-1)F_{m_0}(z) + \frac{1}{(1-z)^{m_0}}.
\end{aligned}$$

By the induction hypothesis, the generating function $F_{m_0-1}(z)$ encodes the sequence $\langle \binom{n+m_0-2}{m_0-2}(H_{n+m_0-2} - H_{m_0-2})\rangle$. Therefore, by (2.13) $dF_{m_0-1}(z)/dz$ encodes the sequence $\langle (n+1)\binom{n+m_0-1}{m_0-2}(H_{n+m_0-1} - H_{m_0-2})\rangle$. Expanding the generating functions we find that

$$\left\langle (n+1)\binom{n+m_0-1}{m_0-2}(H_{n+m_0-1} - H_{m_0-2})\right\rangle = \langle (m_0-1)f_{m_0,n}\rangle + \left\langle \binom{n+m_0-1}{m_0-1}\right\rangle.$$

Solving for $f_{m_0,n}$ gives

$$\begin{aligned}
f_{m_0,n} &= \frac{(n+1)\binom{n+m_0-1}{m_0-2}(H_{n+m_0-1} - H_{m_0-2}) - \binom{n+m_0-1}{m_0-1}}{m_0-1} \\
&= \frac{n+1}{m_0-1}\binom{n+m_0-1}{m_0-2}H_{n+m_0-1} \\
&\quad - \frac{n+1}{m_0-1}\binom{n+m_0-1}{m_0-2}H_{m_0-2} - \frac{1}{m_0-1}\binom{n+m_0-1}{m_0-1}.
\end{aligned}$$

The first term can be simplified to $\binom{n+m_0-1}{m_0-1}H_{n+m_0-1}$. We can simplify the second term and combine it with the third term to get

$$\begin{aligned}
&\frac{n+1}{m_0-1}\binom{n+m_0-1}{m_0-2}H_{m_0-2} + \frac{1}{m_0-1}\binom{n+m_0-1}{m_0-1} \\
&= \binom{n+m_0-1}{m_0-1}H_{m_0-2} + \frac{1}{m_0-1}\binom{n+m_0-1}{m_0-1} \\
&= \binom{n+m_0-1}{m_0-1}\left(H_{m_0-2} + \frac{1}{m_0-1}\right) \\
&= \binom{n+m_0-1}{m_0-1}H_{m_0-1}.
\end{aligned}$$

Thus, we find that

$$\begin{aligned} f_{m_0,n} &= \binom{n+m_0-1}{m_0-1} H_{n+m_0-1} - \binom{n+m_0-1}{m_0-1} H_{m_0-1} \\ &= \binom{n+m_0-1}{m_0-1} (H_{n+m_0-1} - H_{m_0-1}). \quad \blacksquare \end{aligned}$$

7.2. Harmonic Sums

We now apply the hybrid sum technique to several classes of summations involving harmonic numbers. Savio, Lamagna, and Liu have studied many of these sums and defined the form of their solution [SAVIO]. We show how the same results can be achieved using generating functions.

To use the hybrid sum method on summations of the form

$$s_n = \sum_{k=1}^{n} a_k H_k^{(m)},$$

we must be able to find generating functions for the sequences $\langle a_k \rangle$ and $\langle H_k^{(m)} \rangle$. If these generating functions are $A(w)$ and $H(z)$ then, by Corollary 2.1.2, s_n is the coefficient of $w^n z^n$ in $A(w)H(z)/(1-wz)$. When $m > 1$, there is no known single variable generating function for the sequence $\langle H_k^{(m)} \rangle$. However, (7.1) defines an m variable generating function $H(z_1, \ldots, z_m)$ for $\langle H_k^{(m)} \rangle$. We see that the coefficient of $(wz_1 \cdots z_m)^k$ in the hybrid generating function $A(w)H(z_1, \ldots, z_m)$ is $a_k H_k^{(m)}$. Thus, by Corollary 2.2.2, s_n is the coefficient of $(wz_1 \cdots z_m)^n$ in the expansion of

$$A(w)H(z_1, \ldots, z_m)/(1 - wz_1 \cdots z_m). \tag{7.3}$$

7.2.1. Harmonic Number Sums

Consider the sum

$$s_n = \sum_{k=1}^{n} H_k^{(m)}.$$

The sequence $\langle H_k^{(m)} \rangle$ is the coefficient of $(z_1 \cdots z_m)^k$ in the expansion of

$$H(z_1, \ldots, z_m) = \frac{1}{1 - z_1 \cdots z_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1-z_i}\right).$$

By Corollary 2.2.2, s_n is the coefficient of $(z_1 \cdots z_m)^n$ in the expansion of

$$G(z_1, \ldots, z_m) = \frac{1}{1 - z_1 \cdots z_m} H(z_1, \ldots, z_m) = \frac{1}{(1 - z_1 \cdots z_m)^2} \prod_{i=1}^{m} \ln\left(\frac{1}{1-z_i}\right).$$

We can gain some insight into expanding this generating function by considering the case where $m = 1$. By (2.20), the solution to

$$s_n = \sum_{k=1}^{n} H_k$$

is the coefficient of z^n in $H(z)/(1-z)$, where $H(z) = \ln(1/(1-z))/(1-z)$ is the generating function of $\langle H_k \rangle$. We have seen how to expand this generating function using the differentiation method. Differentiating $H(z)$ we have

$$\frac{d}{dz}H(z) = \frac{d}{dz}\frac{1}{1-z}\ln\left(\frac{1}{1-z}\right) = \frac{1}{(1-z)^2}\ln\left(\frac{1}{1-z}\right) + \frac{1}{(1-z)^2}.$$

Expanding the generating functions and equating coefficients of z^n gives

$$(n+1)H_{n+1} = s_n + n + 1.$$

Thus,

$$s_n = (n+1)H_{n+1} - (n+1) = (n+1)(H_{n+1} - 1).$$

Consider differentiating a bivariate generating function $F(w,z)$ with respect to w. If

$$F(w,z) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} a_{i,j} w^i z^j,$$

then

$$\frac{d}{dw}F(w,z) = \sum_{i=1}^{\infty}\sum_{j=0}^{\infty} i a_{i,j} w^{i-1} z^j.$$

A similar result is obtained if we differentiate an m-variate generating function with respect to one of its variables.

To expand $G(z_1,\ldots,z_m)$, we differentiate $H(z_1,\ldots,z_m)$ with respect to z_1 to obtain

$$\begin{aligned}
&\frac{d}{dz_1}H(z_1,\ldots,z_m) \\
&= \frac{d}{dz_1}\left(\frac{1}{1-z_1\cdots z_m}\ln\left(\frac{1}{1-z_1}\right)\prod_{i=2}^{m}\ln\left(\frac{1}{1-z_i}\right)\right) \\
&= \left(\frac{z_2\cdots z_m}{(1-z_1\cdots z_m)^2}\ln\left(\frac{1}{1-z_1}\right) + \frac{1}{1-z_1\cdots z_m}\frac{1}{1-z_1}\right)\prod_{i=2}^{m}\ln\left(\frac{1}{1-z_i}\right).
\end{aligned}$$

The coefficient of $(z_1\cdots z_m)^n$ in the generating function $H(z_1,\ldots,z_m)$ is $H_n^{(m)}$ and so the coefficient of $z_1^{n-1}(z_2\cdots z_m)^n$ in $dH(z_1,\ldots,z_m)/dz_1$ is $nH_n^{(m)}$. If we multiply this generating function by z_1 we shift the sequence and find that the coefficient of $(z_1\cdots z_m)^n$ in $z_1 dH(z_1,\ldots,z_m)/dz_1$ is $nH_n^{(m)}$. Thus,

$$z_1\frac{d}{dz_1}H(z_1,\ldots,z_m)$$

$$= \frac{z_1 \cdots z_m}{(1 - z_1 \cdots z_m)^2} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right) + \frac{1}{1 - z_1 \cdots z_m} \frac{z_1}{1 - z_1} \prod_{i=2}^{m} \ln\left(\frac{1}{1 - z_i}\right)$$

$$= z_1 \cdots z_m G(z_1, \ldots, z_m) + \frac{1}{1 - z_1 \cdots z_m} \frac{z_1}{1 - z_1} \prod_{i=2}^{m} \ln\left(\frac{1}{1 - z_i}\right).$$

The coefficient of $(z_1 \cdots z_m)^n$ in $z_1 \cdots z_m G(z_1, \ldots, z_m)$ is s_{n-1}. By Corollary 2.2.2 the coefficient of $(z_1 \cdots z_m)^n$ in

$$\frac{1}{1 - z_1 \cdots z_m} \frac{z_1}{1 - z_1} \prod_{i=2}^{m} \ln\left(\frac{1}{1 - z_i}\right)$$

is the hybrid sum

$$\sum_{k=1}^{n} \frac{1}{k^{m-1}} = H_n^{(m-1)}.$$

Expanding the generating functions and equating coefficients of $(wz_1 \cdots z_m)^n$, we have

$$nH_n^{(m)} = s_{n-1} - H_n^{(m-1)}.$$

Solving for s_{n-1} gives $s_{n-1} = nH_n^{(m)} - H_n^{(m-1)}$, which implies that

$$s_n = (n+1)H_{n+1}^{(m)} - H_{n+1}^{(m-1)}.$$

7.2.2. Harmonic Sums with Polynomials

Consider the sum

$$s_n = \sum_{k=1}^{n} p(k) H_k^{(m)},$$

where $p(k)$ is a polynomial in k. A common approach to such sums is to apply summation by parts. As we have seen, the hybrid sum method is equivalent to summation by parts when one of the sequences defining the hybrid term has a factor of $1 - z$ in the denominator of its generating function.

By (7.3), s_n is the coefficient of $(wz_1 \cdots z_m)^n$ in

$$G(w, z_1, \ldots, z_m) = \frac{1}{1 - wz_1 \cdots z_m} P(w) \frac{1}{1 - z_1 \cdots z_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right),$$

where $P(w)$ is the generating function of $\langle p(k) \rangle$. By the proof of Lemma 4.1, we saw that polynomials of degree d have rational generating functions whose denominator is $(1 - w)^{d+1}$. Thus,

$$P(w) = \frac{q(w)}{(1 - w)^{d+1}}$$

and

$$G(w, z_1, \ldots, z_m) = \frac{1}{1 - wz_1 \cdots z_m} \frac{q(w)}{(1 - w)^{d+1}} \frac{1}{1 - z_1 \cdots z_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right),$$

where $q(w)$ is a polynomial. Using (A.1) to apply partial fractions with respect to w, we have

$$\begin{aligned} & G(w, z_1, \ldots, z_m) \\ &= \frac{q(w)}{1 - z_1 \cdots z_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right) \\ &\quad \left(\frac{(z_1 \cdots z_m)^{d+1}}{(z_1 \cdots z_m - 1)(1 - w z_1 \cdots z_m)} + \sum_{j=0}^{d} \frac{(-1)^j (z_1 \cdots z_m)^j}{(1 - z_1 \cdots z_m)^{j+1} (1 - w)^{d+1-j}} \right). \end{aligned}$$

The left hand term can be discarded. The right hand term can be expressed as

$$\begin{aligned} & \frac{q(w)}{1 - z_1 \cdots z_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right) \sum_{j=0}^{d} \frac{(-1)^j (z_1 \cdots z_m)^j}{(1 - z_1 \cdots z_m)^{j+1} (1 - w)^{d+1-j}} \\ &= N(w, z_1 \cdots z_m) \frac{q(w)}{(1 - w)^{d+1}} \frac{1}{(1 - z_1 \cdots z_m)^{d+1}} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right) \\ &= N(w, z_1 \cdots z_m) P(w) \frac{1}{(1 - z_1 \cdots z_m)^{d+1}} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right), \end{aligned}$$

where

$$N(w, z_1 \cdots z_m) = \sum_{j=0}^{d} (-1)^j (z_1 \cdots z_m)^j (1 - z_1 \cdots z_m)^{d-j-1} (1 - w)^j.$$

This is a polynomial in w and $z_1 \cdots z_m$. It serves to shift and scale the sequences encoded by $P(w)$ and $\prod_{i=1}^{m} \ln\left(\frac{1}{1-z_i}\right) / (1 - z_1 \cdots z_m)^{d+1}$. We know the expansion of $P(w)$ and to expand the hybrid generating function

$$\frac{1}{(1 - z_1 \cdots z_m)^{d+1}} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right),$$

we apply the differentiation method that we used with poly-log generating functions.

Example 7.2. Consider the sum

$$s_n = \sum_{k=1}^{n} k H_k^{(3)}.$$

By (7.3), s_n is the coefficient of $(w z_1 z_2 z_3)^n$ in

$$\begin{aligned} & G(w, z_1, z_2, z_3) \\ &= \frac{1}{1 - w z_1 z_2 z_3} \frac{w}{(1 - w)^2} \frac{1}{1 - z_1 z_2 z_3} \ln\left(\frac{1}{1 - z_1}\right) \ln\left(\frac{1}{1 - z_2}\right) \ln\left(\frac{1}{1 - z_3}\right). \end{aligned}$$

Applying partial fractions with respect to w gives

$$\begin{aligned}
&G(w,z_1,z_2,z_3)\\
&= \frac{w}{1-z_1z_2z_3}\ln\left(\frac{1}{1-z_1}\right)\ln\left(\frac{1}{1-z_2}\right)\ln\left(\frac{1}{1-z_3}\right)\\
&\quad\left(\frac{(z_1z_2z_3)^2}{(1-z_1z_2z_3)^2(1-wz_1z_2z_3)}+\frac{-2z_1z_2z_3+wz_1z_2z_3+1}{(1-z_1z_2z_3)^2(1-w)^2}\right)\\
&= \left(\frac{w(z_1z_2z_3)^2}{(1-z_1z_2z_3)^3(1-wz_1z_2z_3)}+\frac{-2z_1z_2z_3+wz_1z_2z_3+1}{(1-z_1z_2z_3)^3}\frac{w}{(1-w)^2}\right)\\
&\quad\ln\left(\frac{1}{1-z_1}\right)\ln\left(\frac{1}{1-z_2}\right)\ln\left(\frac{1}{1-z_3}\right).
\end{aligned}$$

We can discard the left hand term. The generating function $w/(1-w)^2$ encodes the sequence $\langle n\rangle$. Thus, the coefficient of w^n in the right hand term is

$$\frac{-2nz_1z_2z_3+(n-1)z_1z_2z_3+n}{(1-z_1z_2z_3)^3}\ln\left(\frac{1}{1-z_1}\right)\ln\left(\frac{1}{1-z_2}\right)\ln\left(\frac{1}{1-z_3}\right). \tag{7.4}$$

We use the differentiation method for poly-log generating functions to find f_n, the coefficient of $(z_1z_2z_3)^n$ in

$$F_3(z_1,z_2,z_3) = \frac{1}{(1-z_1z_2z_3)^3}\ln\left(\frac{1}{1-z_1}\right)\ln\left(\frac{1}{1-z_2}\right)\ln\left(\frac{1}{1-z_3}\right).$$

To find f_n, we differentiate the generating function

$$F_2(z_1,z_2,z_3) = \frac{1}{(1-z_1z_2z_3)^2}\ln\left(\frac{1}{1-z_1}\right)\ln\left(\frac{1}{1-z_2}\right)\ln\left(\frac{1}{1-z_3}\right)$$

with respect to z_1 and multiply by z_1. This gives

$$\begin{aligned}
&z_1\frac{d}{dz_1}F_2(z_1,z_2,z_3)\\
&= \left(\frac{2z_1z_2z_3}{(1-z_1z_2z_3)^3}\ln\left(\frac{1}{1-z_1}\right)+\frac{1}{(1-z_1z_2z_3)^2}\frac{z_1}{1-z_1}\right)\ln\left(\frac{1}{1-z_2}\right)\ln\left(\frac{1}{1-z_3}\right)\\
&= \frac{2z_1z_2z_3}{F_3(z_1,z_2,z_3)}+\frac{1}{(1-z_1z_2z_3)^2}\frac{z_1}{1-z_1}\ln\left(\frac{1}{1-z_2}\right)\ln\left(\frac{1}{1-z_3}\right).
\end{aligned}$$

The generating function $F_2(z_1,z_2,z_3)$ encodes the sequence $\langle(n+1)H^{(3)}_{n+1}-H^{(2)}_{n+1}\rangle$ and so $z_1dF_2(z_1,z_2,z_3)/dz_1$ encodes the sequence $n(n+1)H^{(3)}_{n+1}-nH^{(2)}_{n+1}$. By Corollary 2.2.2, the right hand term encodes the hybrid sum $\sum_{k=1}^n H^{(2)}_k=(n+1)H^{(2)}_{n+1}-H_{n+1}$. Relating coefficients of $(z_1z_2z_3)^n$ we have

$$n(n+1)H^{(3)}_{n+1}-nH^{(2)}_{n+1} = 2f_{n-1}+(n+1)H^{(2)}_{n+1}-H_{n+1}.$$

Solving for f_{n-1} and shifting gives

$$f_n = \frac{(n+1)(n+2)}{2} H^{(3)}_{n+2} - \frac{2n+3}{2} H^{(2)}_{n+2} + \frac{1}{2} H_{n+2}.$$

Thus, expanding (7.4) we find that

$$\begin{aligned} s_n &= -2nf_{n-1} + (n-1)f_{n-1} + nf_n \\ &= nf_n - (n+1)f_{n-1} \\ &= \frac{n(n+1)(n+2)}{2} H^{(3)}_{n+2} - \frac{n(2n+3)}{2} H^{(2)}_{n+2} + \frac{n}{2} H_{n+2} \\ &\quad - \frac{n(n+1)^2}{2} H^{(3)}_{n+1} + \frac{(n+1)(2n+1)}{2} H^{(2)}_{n+1} - \frac{n+1}{2} H_{n+1} \\ &= \frac{n(n+1)}{2} H^{(3)}_n + \frac{1}{2} H^{(2)}_n - \frac{1}{2} H_n. \quad \blacksquare \end{aligned}$$

To expand the hybrid generating function

$$G(w, z_1, \ldots, z_m) = \frac{1}{1 - wz_1 \cdots z_m} P(w) \frac{1}{1 - z_1 \cdots z_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right),$$

we first applied partial fractions with respect to w. We could, instead, have applied partial fractions with respect to z_1 to obtain

$$\begin{aligned} &G(w, z_1, \ldots, z_m) \\ &= \left(\frac{1}{1-w} \frac{1}{1 - z_1 \cdots z_m} + \frac{w}{w-1} \frac{1}{1 - wz_1 \cdots z_m}\right) P(w) \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right) \\ &= \frac{1}{1 - z_1 \cdots z_m} \frac{1}{1-w} P(w) \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right) \\ &\quad - \frac{1}{1 - wz_1 \cdots z_m} \frac{w}{1-w} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - z_i}\right) P(w). \end{aligned}$$

Notice that in this case we cannot discard the term with the factor of $1 - wz_1 \cdots z_m$ in its denominator as this term has a coefficient of $(wz_1 \cdots z_m)^n$ in its expansion. The generating function $P(w)/(1-w)$ encodes the sum $\sum_{k=0}^{n} p(k)$ and $wP(w)/(1-w)$ encodes the sum $\sum_{k=0}^{n-1} p(k)$. We recognize the right hand term as encoding a hybrid sum. Expanding the generating functions we find that

$$s_n = H^{(m)}_n \sum_{k=0}^{m} p(k) - \sum_{j=1}^{n} \frac{1}{j^m} \sum_{k=0}^{j-1} p(k). \tag{7.5}$$

In this case the hybrid sum technique is not able to find a closed form solution for the sum and functions instead as a transformation. The resulting sum is the same as that obtained using summation by parts.

We can easily evaluate (7.5). If $p(k)$ is a polynomial of degree d, then its sum is a polynomial $q(n)$ of degree $d+1$. We find that

$$s_n = H_n^{(m)} q(n) - \sum_{j=1}^{n} \frac{1}{j^m} q(j-1).$$

The polynomial $q(j-1)$ is a polynomial in j of degree $d+1$. We express it as

$$q(j-1) = \sum_{i=0}^{d+1} q_i j^i.$$

Substituting, we find that

$$\begin{aligned} s_n &= H_n^{(m)} q(n) - \sum_{j=1}^{n} \frac{1}{j^m} \sum_{i=0}^{d+1} q_i j^i \\ &= H_n^{(m)} q(n) - \sum_{i=0}^{d+1} q_i \sum_{j=1}^{n} j^{i-m}. \end{aligned}$$

For particular values of q_i and m, we can evaluate these summations using generating functions. If $i < m$ then the sum over j evaluates to $H_n^{(m-i)}$, otherwise it evaluates to a polynomial.

Example 7.3. Consider again the sum

$$s_n = \sum_{k=1}^{n} k H_k^{(3)}.$$

Applying (7.5) and evaluating the summations we find that

$$\begin{aligned} s_n &= H_n^{(3)} \sum_{k=0}^{m} k - \sum_{j=1}^{n} \frac{1}{j^3} \sum_{k=0}^{j-1} k \\ &= H_n^{(3)} \frac{n(n+1)}{2} - \sum_{j=1}^{n} \frac{1}{j^3} \frac{j^2 - j}{2} \\ &= \frac{n(n+1)}{2} H_n^{(3)} - \frac{1}{2} \sum_{j=1}^{n} \frac{1}{j} + \frac{1}{2} \sum_{j=1}^{n} \frac{1}{j^2} \\ &= \frac{n(n+1)}{2} H_n^{(3)} + \frac{1}{2} H_n^{(2)} - \frac{1}{2} H_n. \end{aligned}$$

Note that we were able to evaluate each summation by inspection. However, any one of them could have been evaluated using generating functions. ∎

This second approach to evaluating $\sum_{k=1}^{n} k H_k^{(3)}$ was much easier and results from the type of transformation done by the hybrid sum technique. In the first example, the sum was transformed to a new sum involving a harmonic number. In the second, the sum was transformed to sums of powers of integers, which are easier to evaluate with generating functions than sums of harmonic numbers.

7.2.3. Harmonic Sums with Integer Reciprocals

Consider the sum

$$s_n = \sum_{k=1}^{n} \frac{1}{k^m} H_k^{(m)}.$$

The sequence $\langle 1/k^m \rangle$ for $m > 1$ does not have an ordinary generating function. Thus, we cannot use (7.3) to find s_n. However, we can encode $\langle 1/k^m \rangle$ with the hybrid generating function

$$\prod_{i=1}^{m} \ln\left(\frac{1}{1-w_i}\right).$$

By Corollary 2.2.2, s_n is the coefficient of $(w_1 \cdots w_m z_1 \cdots z_m)^n$ in

$$\begin{aligned} & G(w_1, \ldots, w_m, z_1, \ldots, z_m) \\ &= \frac{1}{1 - w_1 \cdots w_m z_1 \cdots z_m} \prod_{i=1}^{n} \ln\left(\frac{1}{1-w_i}\right) \frac{1}{1 - z_1 \cdots z_m} \prod_{j=1}^{n} \ln\left(\frac{1}{1-z_j}\right). \end{aligned}$$

Applying partial fractions with respect to z_1 we have

$$\begin{aligned} & G(w_1, \ldots, w_m, z_1, \ldots, z_m) \\ &= \frac{1}{1 - w_1 \cdots w_m} \frac{1}{1 - z_1 \cdots z_m} \prod_{i=1}^{n} \ln\left(\frac{1}{1-w_i}\right) \prod_{j=1}^{n} \ln\left(\frac{1}{1-z_j}\right) \\ &\quad - \frac{1}{1 - w_1 \cdots w_m z_1 \cdots z_m} \frac{w_1 \cdots w_m}{1 - w_1 \cdots w_m} \prod_{i=1}^{n} \ln\left(\frac{1}{1-w_i}\right) \prod_{j=1}^{n} \ln\left(\frac{1}{1-z_j}\right). \end{aligned}$$

We cannot apply partial fractions again. Instead, we expand the generating functions and hope that the summation has been transformed to a form that can be evaluated. The right hand term encodes a hybrid sum. Expanding it, we obtain

$$s_n = \sum_{k=1}^{n} \frac{1}{k^m} H_k^{(m)} = H_n^{(m)} H_n^{(m)} - \sum_{k=1}^{n} \frac{1}{k^m} H_{k-1}^{(m)},$$

where $H_0^{(m)} = 0$. We cannot evaluate this sum further. However, we note that both sides of the equation have a similar sum. We add and subtract terms to the second summation to find that

$$\begin{aligned} s_n &= H_n^{(m)} H_n^{(m)} - \sum_{k=1}^{n} \frac{1}{k^m} \left(H_k^{(m)} - \frac{1}{k^m} \right) \\ &= H_n^{(m)} H_n^{(m)} - \sum_{k=1}^{n} \frac{1}{k^m} H_k^{(m)} + \sum_{k=1}^{n} \frac{1}{k^m} \frac{1}{k^m} \\ &= H_n^{(m)} H_n^{(m)} - s_n + \sum_{k=1}^{n} \frac{1}{k^m} \frac{1}{k^m}. \end{aligned}$$

We recognize that the third term is $H_n^{(2m)}$. If needed, we could have evaluated the sum using Corollary 2.2.2. We solve for s_n to find that

$$s_n = \frac{1}{2}H_n^{(2m)} + \frac{1}{2}H_n^{(m)}H_n^{(m)}.$$

Once again, we get a result that could be obtained using summation by parts. We note that this method works only for summands of the form $H_k^{(m)}/k^m$. The transformation of summands of the form $H_k^{(m)}/k^p$, where $m \neq p$, does not result in an equation with the summation for s_n on both sides. Thus, we are unable to solve for s_n.

We also note that evaluating this sum requires that we recognize s_n on both sides of the equation. As a result, we have to do additional manipulation in order to solve the sum. A similar situation arises for many hybrid sums involving harmonic numbers. We first use the hybrid sum method to transform the sum and then manipulate the resulting expression to obtain simpler sums that we can solve using generating functions.

This method can be extended to summands of the form $H_k^{(m)}/(k \pm \alpha)^m$, for positive integer α, by encoding $1/(k \pm \alpha)^m$ in a hybrid term using shifting techniques for generating functions. By Corollary 2.2.2, the solution to the sum

$$s_n = \sum_{k=1}^{n} \frac{1}{(k \pm \alpha)^m} H_k^{(m)}$$

is the coefficient of $(w_1 \cdots w_m z_1 \cdots z_m)^n$ in

$$\begin{aligned} &G(w_1, \ldots, w_m, z_1, \ldots, z_m) \\ &= \frac{(w_1 \cdots w_m)^{\mp\alpha}}{1 - w_1 \cdots w_m z_1 \cdots z_m} \prod_{i=1}^{n} \ln\left(\frac{1}{1-w_i}\right) \frac{1}{1 - z_1 \cdots z_m} \prod_{j=1}^{n} \ln\left(\frac{1}{1-z_j}\right). \end{aligned}$$

We can attack this generating function as before. The hybrid sum method is used to transform the sum. The resulting expression is manipulated to obtain simpler sums that can be solved using generating functions. As a result, we find that

$$\sum_{k=1}^{n} \frac{1}{(k+\alpha)^m} H_k^{(m)} = H_{n+\alpha}^{(m)} H_n^{(m)} - \frac{1}{2}H_n^{(2m)} - \frac{1}{2}H_n^{(m)}H_n^{(m)} - \sum_{k=1}^{n} \frac{1}{k^m} \sum_{j=1}^{\alpha-1} \frac{1}{(k+j)^m}$$

and

$$\sum_{k=1}^{n} \frac{1}{(k-\alpha)^m} H_k^{(m)} = \frac{1}{2}H_{n-\alpha}^{(2m)} + \frac{1}{2}H_{n-\alpha}^{(m)}H_{n-\alpha}^{(m)} + \sum_{k=1}^{n-\alpha} \frac{1}{k^m} \sum_{j=1}^{\alpha} \frac{1}{(k+j)^m}.$$

For particular values of α and m we can completely evaluate the summations in these expressions using generating functions.

7.2.4. Other Harmonic Sums

We can apply the hybrid sum method to other sums involving harmonic numbers, polynomials, and integer reciprocals. In most cases the method only transforms the sum. We must then determine if the new sum is one that we can solve. Quite often, these summations do not have solutions.

For example, consider the summation

$$s_n = \sum_{k=1}^{n} H_k^{(m)} H_k^{(p)}.$$

By (7.3), s_n is the coefficient of $(w_1 \cdots w_m z_1 \cdots z_p)$ in

$$G(w_1, \ldots, w_m, z_1, \ldots, z_p) = \frac{1}{1 - w_1 \cdots w_m z_1 \cdots z_p} \frac{1}{1 - w_1 \cdots w_m} \prod_{i=1}^{m} \ln\left(\frac{1}{1 - w_i}\right) \frac{1}{1 - z_1 \cdots z_p} \prod_{i=1}^{p} \ln\left(\frac{1}{1 - z_i}\right).$$

Applying partial fractions with respect to w_1 gives

$$\begin{aligned} &G(w_1, \ldots, w_m, z_1, \ldots, z_p) \\ &= \prod_{i=1}^{m} \ln\left(\frac{1}{1 - w_i}\right) \frac{1}{1 - z_1 \cdots z_p} \prod_{i=1}^{p} \ln\left(\frac{1}{1 - z_i}\right) \\ &\quad \left(\frac{1}{1 - z_1 \cdots z_p} \frac{1}{1 - w_1 \cdots w_m} - \frac{z_1 \cdots z_p}{1 - z_1 \cdots z_p} \frac{1}{1 - w_1 \cdots w_m z_1 \cdots z_p}\right). \end{aligned}$$

Both terms encode hybrid sums. Expanding we find that

$$\begin{aligned} s_n &= H_n^{(m)} \sum_{k=1}^{n} H_k^{(p)} - \sum_{k=2}^{n} \frac{1}{k^m} \sum_{i=1}^{k-1} H_i^{(p)} \\ &= H_n^{(m)} \left((n+1)H_{n+1}^{(p)} - H_{n+1}^{(p-1)}\right) - \sum_{k=2}^{n} \frac{1}{k^m} \left(kH_k^{(p)} - H_k^{(p-1)}\right) \\ &= (n+1)H_n^{(m)} H_{n+1}^{(p)} - H_n^{(m)} H_{n+1}^{(p-1)} - \sum_{k=2}^{n} \frac{1}{k^{m-1}} H_k^{(p)} + \sum_{k=2}^{n} \frac{1}{k^m} H_k^{(p-1)}. \end{aligned}$$

Solutions for s_n are known only for a few particular values of m and p. If $p = 1$ then

$$s_n = (n+1)H_n^{(m)} H_{n+1} - (n+1)H_n^{(m)} - \sum_{k=2}^{n} \frac{1}{k^{m-1}} H_k + \sum_{k=2}^{n} \frac{1}{k^{m-1}}.$$

If $m = 1$ or $m = 2$ we can evaluate s_n. Likewise, we can solve s_n if $m = 1$ and $p = 2$. However, there are no other apparent cases for which s_n has a closed form solution.

7.3. Summation of Rational Functions

The harmonic numbers are the sums of a power of an integer reciprocal. This definition allows us to solve many summations involving rational functions.

Consider the sum

$$s_n = \sum_{k=\gamma}^{n} \frac{p(k)}{q(k)},$$

where $q(k) = q_0 + q_1 k + \cdots + q_m k^m$. We use a lower limit of $\gamma \geq 0$ for the sum so that the range of summation does not include a zero of the polynomial $q(k)$. Using polynomial division, we express $p(k)/q(k)$ as

$$\frac{p(k)}{q(k)} = r(k) + \frac{s(k)}{q(k)},$$

where $r(k)$ and $s(k)$ are polynomials and degree$(s(k)) <$ degree$(q(k))$.

Partial fractions then allows us to write $s(k)/q(k)$ as

$$\frac{s(k)}{q(k)} = \frac{1}{q_m} \sum_{i=1}^{r} \sum_{h=1}^{d_i} \frac{\beta_{i,h}}{(k-\alpha_i)^h},$$

where $q(k)$ has r roots such that

$$q(k) = q_m (k-\alpha_1)^{d_1} \cdots (k-\alpha_r)^{d_r}$$

and $d_1 + d_2 + \cdots + d_r = m$. We find that

$$s_n = \sum_{k=\gamma}^{n} r(k) + \frac{1}{q_m} \sum_{i=1}^{r} \sum_{h=1}^{d_i} \sum_{k=\gamma}^{n} \frac{\beta_{i,h}}{(k-\alpha_i)^h}.$$

The left hand term is the sum of a polynomial. To sum the right hand term we recognize that

$$\sum_{k=\alpha+1}^{n} \frac{1}{(k-\alpha)^m} = H_{n-\alpha}^{(m)},$$

if α is an integer. Thus we find that the innermost summation evaluates to

$$\sum_{k=\gamma}^{n} \frac{\beta_{i,h}}{(k-\alpha_i)^h} = \beta_{i,h} H_{n-\alpha_i}^{(h)} - \beta_{i,h} \sum_{k=\alpha_i+1}^{\gamma-1} \frac{1}{(k-\alpha_i)^h}.$$

We can then expand the outer summations and simplify the resulting expression.

We would like to classify the set of rational functions which can be summed with this technique. We will call this set of rational functions $\mathcal{RF}$. Note that the numerator can be any polynomial and the constant $1/q_m$ can be absorbed into the numerator. The denominator must be a polynomial with integer roots. Thus, we find that

$$\mathcal{RF} = \{p(k)/q(k) | p(k) \in \mathcal{F}[k], q(k) = \prod_{i=1}^{m} (k-\alpha_i), \alpha_i \in \mathbb{Z}\}$$

for some coefficient field $(\mathcal{F}, +, \cdot)$. It can algebraically be shown that $(\mathcal{RF}, +, \cdot)$ is a ring.

The hybrid sum method can be used to evaluate the summation of rational functions from $\mathcal{RF}$. Application of Corollary 2.2.2 to the sum acts as a transformation method and does not lead directly to a solution. Rather, we must apply partial fractions to the denominator and multiply each term of the expansion by $p(k)$, the numerator polynomial. Each term in the expansion is of the form $p(k)/(k-\alpha)^m$ for integer α and integer $m \geq 1$. We then sum each term with the hybrid sum method. The steps are similar to those for summing the product of a harmonic number and a polynomial.

7.4. Algorithms for Harmonic Numbers

The techniques described here can be used to implement algorithms for evaluating harmonic number sums. However, for sums involving the sequences $\langle H_n^{(m)} \rangle$ and $\langle 1/k^m \rangle$ for $m \geq 2$, the hybrid generating functions become complicated due to the number of logarithmic functions as factors.

Other methods provide better approaches to evaluating these sums. The modifications of Savio, Lamagna, and Liu to Moenck's algorithm can handle any of the sums discussed in Sections 2 and 3. Karr's algorithm will also solve these sums, though to do so requires that a field extension be defined for every $H_n^{(m)}$ in the summand or solution.

We note that in the case of rational functions from the set $\mathcal{RF}$, we can compute much of the solution directly from the summand by inspection. We first expand the summand using partial fractions. Each term in the result is either a polynomial term or the product of a constant and a power of an integer reciprocal. We can use an existing algorithm to solve the sum of the polynomial terms. The sum of the reciprocal terms are, by definition, harmonic numbers and can be mapped directly to their solution.

Chapter 8

Algorithm Implementation and Analysis

Symbolic algorithms for manipulating generating functions and the sequences that they encode were implemented in Maple. In this chapter we describe these procedures, compare their capabilities with other symbolic algorithms, and consider factors that affect their run time performance.

8.1. Maple Routines

Various generating function based procedures were implemented in version 4.3 of Maple to demonstrate the capabilities of generating functions as an algebraic model for symbolic computation [CHAR]. The implementation was not meant to form a comprehensive system, and so it does not perform extensive type checking of arguments or rigorous error checking.

Some difficulties were encountered in the way Maple manipulates and represents polynomials and rational functions. While correct, Maple's representation of rational functions differs from the "traditional" generating function representation found throughout much of the literature. For example, while it is natural to write the generating function of the Fibonacci numbers as

$$F(z) = \frac{z}{1 - z - z^2},$$

Maple treats the z^2 in the denominator as the dominant term and writes this as

$$F(z) = -\frac{z}{-1 + z + z^2},$$

forcing the z^2 term to have a positive coefficient. Maple's representation of logarithmic functions also differs from the more traditional generating function view. While the generating function of $\langle 1/k \rangle$ is often written as

$$G(z) = \ln\left(\frac{1}{1 - z}\right),$$

Maple writes it as

$$G(z) = -\ln(1-z).$$

Implementation of many of the procedures that manipulate generating functions required adopting the Maple viewpoint. While these representations are easily handled within a procedure, they are not an ideal form in which to present a result to a user.

Maple's automatic simplification rules also present some difficulties. For example, it is not possible to write the polynomial $2-z$ as $2(1-z/2)$. Maple simplifies this expression to $2-z$. Thus, in Maple we cannot write $1/(2-z)$ as

$$\frac{1}{2(1-z/2)}.$$

Rather, we must bring the constant out in front and write the expression as

$$\frac{1}{2}\,\frac{1}{1-z/2}.$$

We describe the procedures we implemented. They are categorized here as polynomial algorithms, rational generating function algorithms, summation algorithms, and harmonic number algorithms. The Maple source code is given in Appendix B, which also documents the arguments needed to use the functions. Appendix B presents examples of the use of each function.

8.1.1. Polynomials

Maple provides an extensive collection of functions for the manipulation of polynomials. These include versions of all of the polynomial functions in Table 4.1. The following functions were implemented to provide special capabilities that are useful when dealing with generating functions.

polynom/coeffs(). Returns a list of the nonzero coefficients of a polynomial ordered by the degree of their term. A corresponding list of the degree of each coefficient's term is returned via a parameter.

polynom/equate(). Determines if two polynomials can be equated by (2.8) and (2.9). The constants involved are returned via a parameter.

polynom/factor(). Factors a polynomial over the complex field. The Maple procedure factors only over the rationals.

polynom/hasprod(). Determines if a given polynomial is a factor of a product of polynomials. The product of polynomials can involve polynomials raised to powers given by parameters. Maple's 'gcd()' function cannot handle such expressions.

polynom/map(). Applies a function to each term in a polynomial.

polynom/terms(). Returns a list of the nonzero terms in a polynomial ordered by degree.

polynom/termscale(). Applies (2.15) to a sequence.

8.1.2. Rational Generating Functions

rational/encode(). Encodes rational generating functions.

rational/expand(). Expands rational generating functions.

rational/pfrac(). Performs partial fractions. The denominator is factored over the complex field. Maple's version factors over the rationals.

rational/recur(). Finds the homogeneous linear recurrence for a sequence encoded by a rational generating function.

rational/relate(). Relates sequences with common factors in the denominators of their generating functions. Their are options to relate sequences with common factors in their generating functions, to relate the characteristic sequence of a sequence to the sequence itself, and to determine the sequence encoded by a generating function raised to an integer power.

rational/simp(). Simplifies expressions involving consecutive terms from sequences in RAT_SEQ.

type/ratseq(). Determines if an expression is a valid member of $\mathcal{T}_{\mathbb{C}}$, the set of closed form expressions for RAT_SEQ.

8.1.3. Summation

simple_sum(). Finds an extended closed form solution for simple sums of sequences in RAT_SEQ.

convolve(). Finds an extended closed form expression for a convolution of sequences in RAT_SEQ.

hybrid_sum(). Implements the hybrid sum technique for sequences in RAT_SEQ. It returns an extended closed form solution.

8.1.4. Harmonic Numbers

harmonic/plog(). Finds an extended closed form expression of a sequence encoded by a multivariate poly-log generating function.

harmonic/simp(). Simplifies expressions involving harmonic numbers.

8.2. Capabilities

8.2.1. Summary

We summarize the capabilities of our generating function methods for summations and recurrences.

- *Simple sums and hybrid sums on RAT_SEQ.* Closed form solutions or extended closed form solutions involving the sequences in the summand can be found.

- *Convolutions on RAT_SEQ.* Generating function techniques can be used to evaluate convolutions involving any number of hybrid terms. Closed form solutions or extended closed form solutions involving the sequences in the summand can be found.

- *Definite binomial sums on RAT_SEQ.* Sums involving hybrid terms can be evaluated. Closed form solutions can be found.

- *Indefinite binomial sums on RAT_SEQ.* Binomial coefficients can involve symbolic parameters other than the index of summation. Closed form solutions or extended closed form solutions involving sequences from the summand can be found.

- *Harmonic sums.* Generating function methods can be applied to certain summations involving harmonic numbers. The result is either a solution to the sum or a transformation of the sum using summation by parts. Certain classes of rational summands can be solved.

- *Homogeneous linear recurrences with constant coefficients.* A closed form expression for sequences defined by a homogeneous linear recurrence or a set of sequences defined by a system of homogeneous linear recurrences with constant coefficients can be found.

- *Inhomogeneous linear recurrences with constant coefficients.* These recurrences or systems of these recurrences can be solved if the inhomogeneous part of the recurrence is in *RAT_SEQ*. A closed form expression or extended closed form expression involving terms from the inhomogeneous part can be found.

8.2.2. Comparison with Other Algorithms

Indefinite Summation

The three important algorithms for indefinite summation have been developed by Gosper, Moenck, and Karr, respectively [GOSPER 77, GOSPER 78, MOENCK, KARR]. We consider the capabilities and limitations of each.

Moenck's algorithm evaluates sums of polynomials or rational numbers. The summation of rational numbers can always be expressed using polygamma functions. These are transcendental numbers and only in certain cases can a rational expression be found for the solution. Certain polygamma functions can be mapped to harmonic numbers. Savio, Lamagna, and Liu have modified Moenck's algorithm to recognize such situations [SAVIO].

Gosper's algorithm can solve any indefinite summation when the solution s_n, has the property that s_n/s_{n+1} is rational function of n. There is no known way to tell for an arbitrary summand if s_n/s_{n+1} is rational without determining s_n. In this sense, the

summations that this method solves cannot be completely classified. Gosper's method will however, always solve sums whose summands are polynomials, exponentials, or have the form $k^m \alpha^k$.

Karr's algorithm is a decision procedure for evaluating first order difference equations. It determines whether or not a solution exists as a rational function over a field of symbols. If it does, it finds an expression in that field. It begins with the field of rational numbers of the symbol n. New symbols are added to the field by defining them as first order linear difference equation. For example, $n! = n(n-1)!$, $\alpha^n = \alpha\alpha^{n-1}$, and $H_n = H_{n-1} + 1/n$ are all valid extensions. Failure to find a solution does not mean that the algorithm cannot solve a particular difference equations. It only means that a solution does not exist within a given field of symbols.

Moenck's algorithm can solve the sum of any polynomial or rational function. However, the solution may involve polygamma functions. Thus, we cannot claim that generating function methods are better than any of them. We cannot completely classify the sums that Gosper's method and Karr's method can solve. However, generating function algorithms have capabilities that these algorithms cannot duplicate. We consider some differences in the capabilities of the various algorithms.

Simple sums and hybrid sums on RAT_SEQ can be evaluated using either Gosper's algorithm or Karr's algorithm. If a closed form can be found for the summand, we express it in sum of products form and apply the algorithm to each term. The result is a closed form expression for the sum. Moenck's algorithm cannot be used here since the terms that are being summed are not polynomials or rational functions. Gosper's algorithm cannot be used to find extended closed form expressions involving sequence names. While function names can be used in Karr's system if they can be defined as first order difference equations, many sequences in RAT_SEQ do not have such definitions.

With the modifications of Savio, Lamagna, and Liu for harmonic numbers, Moenck's algorithm can handle any of the harmonic sums that we were able to solve with generating function methods. These include sums of harmonics, sums of products between harmonics and polynomials, and sums of products between harmonics and integer reciprocals. In addition, Moenck's algorithm can solve sums involving the rational functions in the set $\mathcal{RF}$.

For particular values of m, the binomial coefficients $\langle \binom{m+n}{m} \rangle$ and $\langle \binom{m}{m} \rangle$ are polynomials. The algorithms for indefinite summation can treat them as such. When m is a parameter, we must treat these binomial coefficients as a symbolic name. Of the indefinite summation algorithms, Karr's is the only one that might be able to handle such sums. Since $\binom{m}{k} = (m-k-1)\binom{m}{k-1}/k$ and $\binom{k}{m} = k\binom{k-1}{m}/(k-m)$, we are able to define these sequences as first order difference equations for Karr's algorithm. However, for general m, the solutions to sums containing binomial coefficients often involve a sum of $m+1$ binomials. Karr's algorithm finds closed forms that are rational functions of the symbols in the extension field. Since m is a parameter, a sum of $m+1$ terms is not a rational function and so Karr's algorithm cannot find these solutions.

There are problems that the other methods can solve that generating functions cannot. For example, Karr's method and Gosper's method can solve $\sum_{k=1}^{n} k\, k!$. Since

the sequence $\langle k! \rangle$ does not have an ordinary generating function, our algorithms cannot solve this sum.

Convolutions

Generating functions appear to be the best known method for evaluating convolutions in RAT_SEQ. There are no other known algorithms that can generally solve such convolutions. Some particular convolutions can be evaluated by transforming them to a hybrid sum. For example,

$$\sum_{k=0}^{n} a_n 2^{n-k} = 2^n \sum_{k=0}^{n} \frac{a_k}{2^k}.$$

However, this is not easily done for many sequences in RAT_SEQ.

Definite Binomial Summations

Generating functions are the only known method for evaluating definite binomial sums on RAT_SEQ. Hypergeometric techniques can be used for some special cases. The Hypergeometric method of Hayden and Lamagna can handle many other definite sums involving binomial coefficients that generating functions cannot [HAYDEN 86]. For example, generating functions cannot solve sums involving ratios of binomial coefficients. Thus, hypergeometrics are the best apparent choice for definite sums involving binomial coefficients.

Linear Recurrences

Cohen and Katcoff implemented an interactive system for solving linear recurrences and systems of linear recurrences [COHEN]. They use generating functions to determine a mapping from the problem to the solution. The inhomogeneous part of a recurrence is mapped to a summation that is evaluated using table lookup. This imposes limitations on the problems that the method can solve and makes it difficult to solve recurrences involving special functions. Their system is also limited in the degree of the recurrences that it can solve.

Karr's summation method is based on first order linear difference equations. His algorithm can solve first order linear recurrences with constant coefficients. He is also able to solve first order linear recurrences with nonconstant coefficients. We did not attempt to solve such recurrences. However, summation factors may be used to convert many of these recurrences to summations which we can then attempt to evaluate using generating functions [GRAHAM].

8.3. Run-Time Complexity

We discuss considerations affecting the run time performance of our algorithms on three levels. First, we study their behavior as procedures written using Maple's builtin

functions as a set of primitive commands. Second, we consider the contribution of the underlying Maple system, including Maple procedures and data representation. Third, we examine how the expressions that we are manipulating contribute to the run time complexity.

8.3.1. Generating Function Algorithms

The run time of most of the algorithms that we implemented is dependent on the size and structure of the expressions being processed. The algorithms for encoding and expanding rational generating functions, evaluating hybrid sums, and evaluating convolutions are the most complex. We consider the factors affecting their run time performance.

rational/encode(). The implementation of this algorithm is different than that described in Algorithm 5.4. The 'expand()' procedure of Maple expands to terms of the form $\alpha\theta^n n^i$. The **rational/encode()** procedure uses Maple's 'collect()' function to gather the coefficients of each n^i. It iterates through these "coefficients" to the highest degree of n. For n^i, it computes the generating function of $\langle n^i \rangle$ from the generating function of $\langle n^{i-1} \rangle$. It uses this generating function, (2.8), and (2.11) to encode each term in the collected coefficient of n^i. The total execution time depends on the number of terms in the expanded expression and the highest degree of n. Both of these quantities are functions of the complexity of the input expression.

rational/expand(). The generating function is expanded by partial fractions. The number of terms in the result is equal to the degree of the denominator. The algorithm iterates through each of these terms in order to expand them. To expand $1/(1-z)^m$ for a particular m requires m iterations to compute $\binom{m+n}{m}$. The number of terms with $m > 1$ depends on the number of factors in the denominator with degree greater than one.

convolve(). The generating function of the solution is expanded using partial fractions. The resulting number of terms is equal to the degree of the denominator. The algorithm iterates through each term in the result and compares the term with the generating functions of each sequence in the summand in order to associate the term with at least one of the sequences. The total execution time is a function of the product of the number of terms in the partial fraction expansion and the number of factors in the summand. The terms associated with each sequence are then summed together and expanded. Execution time for this step depends on the number of factors in the summand.

hybrid_sum(). The algorithm iterates through each factor in the hybrid term and performs a partial fraction expansion at each iteration. The final partial fraction expansion results in a generating function that usually does not involve any of the input generating functions. This generating function is compared with each of

the input generating functions using **polynom/equate()**. If it cannot be related to any of the input generating functions, **rational/expand()** is used to find its closed form. The number of iterations in each part of the algorithm depends only on the number of factors in the summand.

8.3.2. Underlying Maple System

Many of Maple's procedures contribute considerably to the execution time of our routines. For example, Maple provides a procedure for performing partial fractions. In analyzing our algorithms, we count this as one operation. However, the algorithm for partial fractions is complex. If an approach such as Gaussian elimination is used, its run time is a cubic function of the degree of the denominator. Since many of our algorithms use partial fractions, the execution time of Maple's implementation of partial fractions has a considerable impact on their performance.

Many of Maple's procedures must traverse or manipulate the entire expression that is passed as an argument. The more complex the size and structure of an expression, the greater the execution time needed to manipulate it. Also, the algorithms Maple uses to build and access parse trees can affect the time needed to traverse an expression. Thus, the size and structure of the expressions being manipulated have a direct bearing on the run time performance of our algorithms. We cannot analyze this contribution without knowing the actual expressions involved and details concerning the internal implementation of Maple.

8.3.3. Expression Complexity

Expression complexity depends on the problem being solved. For summation, the factors in the summand and the size of the summand affect the complexity of the solution.

Consider the hybrid sums $\sum_{k=0}^{n} a^k F_k$ and $\sum_{k=0}^{n} 2^k F_k$. Conceptually the two sums are the same and the steps needed to solve them are identical. However, because a is a symbolic quantity while 2 is a numeric quantity, the manipulations involved with the first sum are more complex than those involved with the second. The solution to the first contains the quantity $a^2 + a - 1$. In the solution to the second, the constant 5, which is $a^2 + a - 1$ evaluated at 2, appears instead. In this case the factor a^k introduces more complexity into the solution that the factor 2^k.

Size of the summand also affects complexity of the result. Consider a hybrid term with m factors. If the ith factor satisfies an order k_i homogeneous linear recurrence, then the solution to the sum may require $k_1 \cdots k_m$ terms that are products of the sequences in the hybrid term. This quantity grows exponentially for each $k_i \geq 2$. Thus, even though the hybrid sum algorithm appears to be linear in the number of factors in the summand, the solution size grows exponentially with the number of factors and the execution time for the Maple primitives and many of our primitives will also become exponential. We note that any algorithm for solving hybrid sums must experience this exponential growth of expression size since it will be finding the same solution.

Chapter 9

Future Work and Conclusion

Future Work

This work is one of the few to explore the algorithmic applications of generating functions. Many open questions and areas of research involving generating functions and the theory of summation remain to be addressed. We consider some of them here.

Other Generating Function Methods

This research examined the application of ordinary generating functions to several well defined classes of summations and recurrences. Our goal was to demonstrate their use as an algebraic model of computation. Many other generating function methods and applications have yet to be explored.

Exponential generating functions. Exponential generating functions have application to many classes of problems, including definite binomial sums and linear recurrences. Their applications need to be classified and algorithms for their manipulation developed.

Multivariate rational generating functions. It is still an open question as to whether or not there is a general method for expanding such generating functions. Development of such a method, or even a collection of methods, will allow the implementation of algorithms for solving multivariate linear recurrences with constant coefficients.

Asymptotics. Flajolet, Salvy, and Zimmermann implemented a method based on complex variable techniques for deriving asymptotic information about a sequence from its generating function [FLAJOLET]. It might be possible to apply similar methods to multivariate generating functions in order to extract asymptotic information from them. For example, if the generating function of a hybrid sum cannot be expanded to find a closed form solution for the sum, it would still be useful to find asymptotic information about the sum.

Recurrences with nonconstant coefficients. It is possible to use generating functions to solve many linear recurrences with nonconstant coefficients. The recurrences which can be solved should be classified and algorithms for their solution developed. Methods for solving some of these recurrences result in differential equations defining a generating function. Algorithms for solving many such recurrences will depend on the development of techniques for solving differential equations.

Probability and statistics. Moment generating functions are used in probability and statistics to study distributions of discrete random variables and to define their moments. Many moment generating functions can be found using techniques similar to those we employed to encode rational generating functions. Application of our algorithms to these problems can be explored.

Theory of Summation

We used generating functions to develop a theory of summation on RAT_SEQ. This allowed us to demonstrate the capabilities of our algorithms and to determine where to find solutions to the summations being solved.

Before algorithms are developed for solving other classes of summations, these classes should be defined and studied. If only a few sums in the class have a known solution, then there may be little point in developing algorithms for solving them. However, if many sums in the class can be evaluated, that knowledge is useful in designing algorithms and analyzing their capabilities.

Risch's method solves the problem of symbolic indefinite integration [RISCH]. At this point, there does not appear to be a counterpart for symbolic summation. However, extensions to existing algebraic models can be considered in an attempt to understand better symbolic summation.

Karr's algorithm is limited by the types of difference equations that it can solve and the field extensions that it allows. His method solves only first order difference equations. Extending it to solve order k difference equations would allow it to solve any linear recurrence. Field extensions in Karr's method are limited to those defined by first order difference equations. If field extensions could be defined by order k difference equations, then his method would be able to completely solve sums from RAT_SEQ. It would also solve problems involving sequences defined by linear recurrences with nonconstant coefficients.

Extensions to generating function methods may also be possible. Sequences besides those in RAT_SEQ, such as Stirling numbers and Euler numbers, have known generating functions. Summations involving these functions can be studied and the insight gained may help lead to generating function algorithms for their solution.

Discrete Math Tool

One purpose for developing symbolic algorithms that solve summations and recurrences is to build computational tools for discrete mathematics. The next logical step is to use

these algorithms to build a graphical system for discrete math. Such a system would provide an electronic worksheet that edits and manipulates sequences and expressions. It would offer the user some of the advantages that computers have over hand calculations including increased computational speed, increased accuracy, and elimination of tedious computations.

Routines for solving summations and recurrences could be built into the system. Other built in features could include tools for asymptotic analysis and implementations of commonly used techniques, such as summation by parts or summation factors. Such systems are being developed for other areas of discrete math. Skiena has used Mathematica to develop a system for combinatorics and graph theory [SKIENA].

Conclusion

An early goal of this research was to explore the use of generating functions as an algebraic model for symbolic computation. The intent was to bridge the gap between the theory of generating functions and its application. However, in the process of applying generating functions to solve summations and recurrences, we were led back to theoretical concerns. Two major issues had to be addressed. The first was to show that we knew where to find the solutions for the problems that we were attempting to solve. This was necessary to show that our algorithms would be able to expand the generating functions encoding the solutions. The second was to demonstrate the ability of our algorithms to completely solve particular classes of problems. To answer both questions we studied the algebraic properties of the rational generating functions and the sequences that they encode. As a result, we developed a formal theory of summation on RAT_SEQ.

This theory of summation on RAT_SEQ is an important contribution to summation. We now have a standard against which we can measure symbolic algorithms for summation. Generating function methods hold up against this standard. They can solve any sum or convolution on RAT_SEQ, including those involving hybrid terms. Not only can they be used to find closed form expressions, but they can also be used to find extended closed forms involving sequences in the summand. None of the other algorithms to date can make this claim.

We were also able to apply generating functions to sums involving certain expressions that are not in RAT_SEQ. These include definite and indefinite binomial sums on RAT_SEQ and several classes of harmonic sums. In addition, we can solve linear recurrences with constant coefficients and system of linear recurrences with constant coefficients using generating functions.

Generating functions cannot solve all problems in discrete math. However, they do provide a powerful tool with wide application to many problems. In addition, they provide an algebraic model of computation that is much easier to apply and understand than many of the other methods. They have proven themselves to be a good foundation for symbolic computation.

Appendix A

Partial Fraction Expansions

An important tool when dealing with generating functions is partial fraction expansions. Many of our proofs and derivations involving rational generating functions require the application of partial fractions. This appendix derives the partial fraction expansions of the following functions:

- $1/((1-az)(1-bz)^m)$, where $a \neq b$,
- $1/((1-az)^j(1-bz)^m)$, where $a \neq b$,
- $1/(p(z)(1-az))$, where $\mathrm{lterm}(p(z)) = 1$ and $1-az$ is not a factor of $p(z)$,
- $1/(p(z)(1-z))$, where $1-z$ may be a factor of $p(z)$,
- $1/(p(wz)q(z))$, where w is a variable independent of the polynomial $q(z)$.

A.1. Expansion of $1/((1-az)(1-bz)^m)$

The partial fraction expansion of

$$\frac{1}{(1-az)(1-bz)^m},$$

where $a \neq b$, is

$$\frac{1}{(1-az)(1-bz)^m} = \frac{a^m}{(a-b)^m(1-az)} + \sum_{k=0}^{m-1} \frac{b(-a)^k}{(b-a)^{k+1}(1-bz)^{m-k}}. \qquad \text{(A.1)}$$

The proof is by induction. The base case is

$$\frac{1}{(1-az)(1-bz)} = \frac{a}{a-b}\frac{1}{1-az} + \frac{b}{b-a}\frac{1}{1-bz}.$$

To prove the inductive step, assume that (A.1) is true for $m = m_0$ and prove it is true for $m = m_0 + 1$. By the inductive hypothesis we have that

$$\begin{aligned}
&\frac{1}{(1-az)(1-bz)^{m_0+1}} \\
&= \frac{1}{1-bz}\,\frac{1}{(1-az)(1-bz)^{m_0}} \\
&= \frac{1}{1-bz}\left(\frac{a^{m_0}}{(a-b)^{m_0}(1-az)} + \sum_{k=0}^{m_0-1}\frac{b(-a)^k}{(b-a)^{k+1}(1-bz)^{m_0-k}}\right) \\
&= \frac{1}{1-bz}\,\frac{a^{m_0}}{(a-b)^{m_0}(1-az)} + \sum_{k=0}^{m_0-1}\frac{b(-a)^k}{(b-a)^{k+1}(1-bz)^{m_0-k+1}}.
\end{aligned}$$

Performing partial fractions on the left hand term and absorbing the new term into the summation gives

$$\begin{aligned}
&\frac{1}{(1-az)(1-bz)^{m_0+1}} \\
&= \frac{a^{m_0}}{(a-b)^{m_0}}\left(\frac{a}{(a-b)(1-az)} + \frac{b}{(b-a)(1-bz)}\right) \\
&\quad + \sum_{k=0}^{m_0-1}\frac{b(-a)^k}{(b-a)^{k+1}(1-bz)^{m_0-k+1}} \\
&= \frac{a^{m_0+1}}{(a-b)^{m_0+1}(1-az)} + \frac{b(-1)^{m_0}a^{m_0}}{(b-a)^{m_0+1}(1-bz)} + \sum_{k=0}^{m_0-1}\frac{b(-a)^k}{(b-a)^{k+1}(1-bz)^{m_0+1-k}} \\
&= \frac{a^{m_0+1}}{(a-b)^{m_0+1}(1-az)} + \sum_{k=0}^{m_0}\frac{b(-a)^k}{(b-a)^{k+1}(1-bz)^{m_0+1-k}},
\end{aligned}$$

which proves (A.1).

A.2. Expansion of $1/((1-az)^j(1-bz)^m)$

The partial fraction expansion of

$$T_{j,m}(z) = \frac{1}{(1-az)^j(1-bz)^m},$$

where $a \neq b$, is

$$T_{j,m}(z) = a^m \sum_{k=0}^{j-1}\frac{(-b)^k\binom{m-1+k}{m-1}}{(a-b)^{m+k}(1-az)^{j-k}} + b^j\sum_{k=0}^{m-1}\frac{(-a)^k\binom{j-1+k}{j-1}}{(b-a)^{j+k}(1-bz)^{m-k}}. \qquad \text{(A.2)}$$

This result can be obtained as an extension of Problem 5.39 in [GRAHAM], which states that if $xy = ax + by$ then

$$x^m y^n = \sum_{k=1}^{m}\binom{m+n-1-k}{n-1}a^n b^{m-k}x^k + \sum_{k=1}^{n}\binom{m+n-1-k}{m-1}a^{n-k}b^m y^k.$$

We can also prove (A.2) by induction on j. From (A.1) the base case is

$$T_{1,m}(z) = \frac{1}{(1-az)(1-bz)^m} = \frac{a^m}{(a-b)^m(1-az)} + \sum_{k=0}^{m-1} \frac{b(-a)^k}{(b-a)^{k+1}(1-bz)^{m-k}}.$$

To prove the inductive step, assume that (A.2) is true for $j = j_0$ and prove it is true for $j = j_0 + 1$. By the inductive hypothesis we have that

$$\begin{aligned} & T_{j_0+1,m}(z) \\ &= \frac{1}{1-az} T_{j_0,m}(z) \\ &= \frac{1}{1-az} \left(a^m \sum_{k=0}^{j_0-1} \frac{(-b)^k \binom{m-1+k}{m-1}}{(a-b)^{m+k}(1-az)^{j_0-k}} + b^{j_0} \sum_{k=0}^{m-1} \frac{(-a)^k \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+k}(1-bz)^{m-k}} \right). \end{aligned}$$

Multiplying through and applying (A.1) gives

$$\begin{aligned} T_{j_0+1,m}(z) &= a^m \sum_{k=0}^{j_0-1} \frac{(-b)^k \binom{m-1+k}{m-1}}{(a-b)^{m+k}(1-az)^{j_0+1-k}} \\ &\quad + b^{j_0} \sum_{k=0}^{m-1} \frac{(-a)^k \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+k}} \frac{1}{(1-az)(1-bz)^{m-k}} \\ &= a^m \sum_{k=0}^{j_0-1} \frac{(-b)^k \binom{m-1+k}{m-1}}{(a-b)^{m+k}(1-az)^{j_0+1-k}} \\ &\quad + b^{j_0} \sum_{k=0}^{m-1} \frac{(-a)^k a^{m-k} \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+k}(a-b)^{m-k}(1-az)} \\ &\quad + b^{j_0} \sum_{k=0}^{m-1} \sum_{i=0}^{m-k-1} \frac{b(-a)^{k+i} \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+k+i+1}(1-bz)^{m-k-i}}. \end{aligned} \tag{A.3}$$

We will consider first the middle term in (A.3). Rearranging terms and evaluating the summation gives

$$\begin{aligned} b^{j_0} \sum_{k=0}^{m-1} \frac{(-a)^k a^{m-k} \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+k}(a-b)^{m-k}(1-az)} &= b^{j_0} \sum_{k=0}^{m-1} \frac{(-1)^{j_0+2k} a^m \binom{j_0-1+k}{j_0-1}}{(a-b)^{j_0+m}(1-az)} \\ &= \frac{(-b)^{j_0} a^m}{(a-b)^{j_0+m}(1-az)} \sum_{k=0}^{m-1} \binom{j_0-1+k}{j_0-1} \\ &= \frac{(-b)^{j_0} a^m}{(a-b)^{j_0+m}(1-az)} \binom{j_0+m-1}{m-1}. \end{aligned}$$

We will now evaluate the third term in (A.3). Shifting the limits of the inner summation, changing the order of summation, and evaluating the inner summation produces

$$b^{j_0} \sum_{k=0}^{m-1} \sum_{i=0}^{m-k-1} \frac{b(-a)^{k+i} \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+k+i+1}(1-bz)^{m-k-i}}$$

$$
\begin{aligned}
&= b^{j_0} \sum_{k=0}^{m-1} \sum_{i=k}^{m-1} \frac{b(-a)^i \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+i+1}(1-bz)^{m-i}} \\
&= b^{j_0} \sum_{i=0}^{m-1} \sum_{k=0}^{i} \frac{b(-a)^i \binom{j_0-1+k}{j_0-1}}{(b-a)^{j_0+i+1}(1-bz)^{m-i}} \\
&= b^{j_0+1} \sum_{i=0}^{m-1} \frac{(-a)^i}{(b-a)^{j_0+i+1}(1-bz)^{m-i}} \sum_{k=0}^{i} \binom{j_0-1+k}{j_0-1} \\
&= b^{j_0+1} \sum_{i=0}^{m-1} \frac{(-a)^i}{(b-a)^{j_0+i+1}(1-bz)^{m-i}} \binom{j_0+i}{j_0}.
\end{aligned}
$$

Substituting back into (A.3) and absorbing the middle term into the first sum gives

$$
\begin{aligned}
&T_{j_0+1,m}(z) \\
&\quad = a^m \sum_{k=0}^{j_0-1} \frac{(-b)^k \binom{m-1+k}{m-1}}{(a-b)^{m+k}(1-az)^{j_0+1-k}} + \frac{(-b)^{j_0} a^m \binom{m-1+j_0}{m-1}}{(a-b)^{j_0+m}(1-az)} \\
&\qquad + b^{j_0+1} \sum_{i=0}^{m-1} \frac{(-a)^i \binom{j_0+i}{j_0}}{(b-a)^{j_0+i+1}(1-bz)^{m-i}} \\
&\quad = a^m \sum_{k=0}^{j_0} \frac{(-b)^k \binom{m-1+k}{m-1}}{(a-b)^{m+k}(1-az)^{j_0+1-k}} + b^{j_0+1} \sum_{i=0}^{m-1} \frac{(-a)^i \binom{j_0+i}{j_0}}{(b-a)^{j_0+i+1}(1-bz)^{m-i}},
\end{aligned}
$$

which proves (A.2).

A.3. Expansion of $1/(p(z)(1-az))$

The partial fraction expansion of

$$\frac{1}{p(z)\,(1-az)},$$

where

$$p(z) = a_0 + a_1 z + a_2 z^2 + \cdots + a_d z^d$$

and $1 - az$ is not a factor of $p(z)$, is given by

$$\frac{1}{p(z)\,(1-az)} = \frac{\sum_{j=0}^{d-1} \sum_{k=0}^{j} a^j a_{d+k-j} z^k}{a^d p(1/a) p(z)} + \frac{1}{p(1/a)(1-az)}. \tag{A.4}$$

We can factor $p(z)$ over the complex numbers as

$$p(z) = a_d (z - r_1)(z - r_2) \cdots (z - r_{j-1})(z - r_m).$$

Let

$$
\begin{aligned}
p_m(z) &= b_{m,0} + b_{m,1} z + b_{m,2} z^2 + \cdots + b_{m,m} z^m \\
&= b_{m,m} (z - r_1)(z - r_2) \cdots (z - r_{j-1})(z - r_m),
\end{aligned}
$$

where $b_{m,m} = a_d$ and $1 \le m \le d$. Thus, $p(z) = p_d(z)$ and $p_{m+1}(z) = (z - r_{m+1})p_m(z)$. Relating coefficients of z in $p_m(z)$ and $p_{m+1}(z)$ we find that

$$b_{m+1,i} = \begin{cases} b_{m,0}r_{m+1}, & \text{if } i = 0; \\ b_{m,i-1} - r_{m+1}b_{m,i}, & \text{if } 1 \le i \le m; \\ b_{m,m}, & \text{if } i = m+1. \end{cases} \tag{A.5}$$

We will prove by induction on m that

$$\frac{1}{p_m(z)\,(1-az)} = \frac{\sum_{j=0}^{m-1}\sum_{k=0}^{j} a^j b_{m+k-j} z^k}{a^m p_m(1/a) p_m(z)} + \frac{1}{p_m(1/a)(1-az)}. \tag{A.6}$$

The base case is

$$\begin{aligned} \frac{1}{p_1(z)(1-az)} &= \frac{1}{(b_{1,0}+b_{1,1}z)(1-az)} \\ &= \frac{b_{1,1}}{(ab_{1,0}+b_{1,1})(b_{1,0}+b_{1,1}z)} + \frac{a}{(ab_{1,0}+b_{1,1})(1-az)} \\ &= \frac{b_{1,1}}{ap_1(1/a)p_1(z)} + \frac{1}{p_1(1/a)\,(1-az)}. \end{aligned}$$

To prove the inductive step, assume that (A.4) is true for $m = m_0$ and prove it is true for $m = m_0 + 1$. By the inductive hypothesis we have that

$$\begin{aligned} &\frac{1}{p_{m_0+1}(z)\,(1-az)} \\ &= \frac{1}{(z - r_{m_0+1})\,p_{m_0}(z)\,(1-az)} \\ &= \frac{1}{z - r_{m_0+1}} \left(\frac{\sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k}{a^{m_0} p_{m_0}(1/a) p_{m_0}(z)} + \frac{1}{p_{m_0}(1/a)(1-az)} \right). \end{aligned}$$

Distributing the product, recognizing $p_{m_0+1}(z)$, and applying partial fractions gives

$$\begin{aligned} &\frac{1}{p_{m_0+1}(z)\,(1-az)} \\ &= \frac{\sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k}{(z - r_{m_0+1}) a^{m_0} p_{m_0}(1/a) p_{m_0}(z)} + \frac{1}{(z - r_{m_0+1}) p_{m_0}(1/a)(1-az)} \\ &= \frac{\sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k}{a^{m_0} p_{m_0}(1/a) p_{m_0+1}(z)} + \frac{1}{p_{m_0}(1/a)\,(1 - ar_{m_0+1})\,(z - r_{m_0+1})} \\ &\quad + \frac{a}{p_{m_0}(1/a)\,(1 - ar_{m_0+1})\,(1-az)}. \end{aligned}$$

We can put the first two terms over a common denominator, rearrange the third term, and recognize $p_{m_0+1}(1/a)$:

$$\frac{1}{p_{m_0+1}(z)\,(1-az)} = \frac{\sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k (1 - ar_{m_0+1}) - a^{m_0} p_{m_0}(z)}{a^{m_0} p_{m_0}(1/a)\,(1 - ar_{m_0+1}) p_{m_0+1}(z)}$$

$$
\begin{aligned}
&+ \frac{a}{a p_{m_0}(1/a)(1/a - r_{m_0+1})(1-az)} \\
&= \frac{\sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k (1 - a r_{m_0+1}) - a^{m_0} p_{m_0}(z)}{a^{m_0+1} p_{m_0+1}(1/a) p_{m_0+1}(z)} \\
&\quad + \frac{1}{p_{m_0+1}(1/a)(1-az)}.
\end{aligned}
$$

The second term is in the desired form. We can use the distributive law to rearrange the numerator of the first term to find

$$
\begin{aligned}
&\sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k (1 - a r_{m_0+1}) - a^{m_0} p_{m_0}(z) \\
&= \sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k - \sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^{j+1} r_{m_0+1} b_{m_0,m_0+k-j} z^k - a^{m_0} p_{m_0}(z) \\
&= \sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k - \sum_{j=1}^{m_0}\sum_{k=0}^{j-1} a^j r_{m_0+1} b_{m_0,m_0+k-j+1} z^k - a^{m_0} \sum_{k=0}^{m_0} b_{m_0,k} z^k \\
&= \sum_{j=0}^{0}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k + \sum_{j=1}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k \\
&\quad - \sum_{j=1}^{m_0-1}\sum_{k=0}^{j-1} a^j r_{m_0+1} b_{m_0,m_0+k-j+1} z^k - \sum_{j=m_0}^{m_0}\sum_{k=0}^{j-1} a^j r_{m_0+1} b_{m_0,m_0+k-j+1} z^k \\
&\quad - a^{m_0} \sum_{k=0}^{m_0} b_{m_0,k} z^k. \qquad \text{(A.7)}
\end{aligned}
$$

We now apply (A.5) to derive the coefficients of $p_{m_0+1}(z)$ from $p_{m_0}(z)$. The first term in (A.7) is a double sum having simply $k = j = 0$. Its value is $b_{m_0,m_0} = b_{m_0+1,m_0+1}$, which allows us to write the double sum as

$$
\sum_{j=0}^{0}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k = \sum_{j=0}^{0}\sum_{k=0}^{j} a^j b_{m_0+1,m_0+1+k-j} z^k.
$$

The second and third terms in (A.7) combine with each other,

$$
\begin{aligned}
&\sum_{j=1}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k - \sum_{j=1}^{m_0-1}\sum_{k=0}^{j-1} a^j r_{m_0+1} b_{m_0,m_0+k-j+1} z^k \\
&= \sum_{j=1}^{m_0-1} a^j \left(\sum_{k=0}^{j-1} (b_{m_0,m_0+k-j} - r_{m_0+1} b_{m_0,m_0+k-j+1}) z^k + \sum_{k=j}^{j} b_{m_0,m_0+k-j} z^k \right) \\
&= \sum_{j=1}^{m_0-1} a^j \left(\sum_{k=0}^{j-1} b_{m_0+1,m_0+1+k-j} z^k + \sum_{k=j}^{j} b_{m_0+1,m_0+1+k-j} z^k \right) \\
&= \sum_{j=1}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0+1,m_0+1+k-j} z^k.
\end{aligned}
$$

Finally, we can combine the fourth and fifth terms in (A.7),

$$\begin{aligned}
&\sum_{j=m_0}^{m_0}\sum_{k=0}^{j-1} a^j r_{m_0+1} b_{m_0,m_0+k-j+1} z^k - a^{m_0}\sum_{k=0}^{m_0} b_{m_0,k} z^k \\
&= \sum_{j=m_0}^{m_0}\sum_{k=0}^{j-1} a^j r_{m_0+1} b_{m_0,m_0+k-j+1} z^k - \sum_{j=m_0}^{m_0} a^j \sum_{k=0}^{j} b_{m_0,m_0+k-j} z^k \\
&= \sum_{j=m_0}^{m_0} a^j \left(\sum_{k=0}^{j-1} (r_{m_0+1} b_{m_0,m_0+k-j+1} - b_{m_0,m_0+k-j}) z^k - \sum_{k=j}^{j} b_{m_0,m_0+k-j} z^k \right) \\
&= \sum_{j=m_0}^{m_0} a^j \left(\sum_{k=0}^{j-1} b_{m_0+1,m_0+1+k-j} z^k + \sum_{k=j}^{j} b_{m_0+1,m_0+1+k-j} z^k \right) \\
&= \sum_{j=m_0}^{m_0}\sum_{k=0}^{j} a^j b_{m_0+1,m_0+1+k-j} z^k .
\end{aligned}$$

Substituting back into (A.7), we find that

$$\begin{aligned}
&\sum_{j=0}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0,m_0+k-j} z^k (1 - a r_{m_0+1}) - a^{m_0} p_{m_0}(z) \\
&= \sum_{j=0}^{0}\sum_{k=0}^{j} a^j b_{m_0+1,m_0+1+k-j} z^k + \sum_{j=1}^{m_0-1}\sum_{k=0}^{j} a^j b_{m_0+1,m_0+1+k-j} z^k \\
&\quad + \sum_{j=m_0}^{m_0}\sum_{k=0}^{j} a^j b_{m_0+1,m_0+1+k-j} z^k \\
&= \sum_{j=0}^{m_0}\sum_{k=0}^{j} a^j b_{m_0+1,m_0+1+k-j} z^k .
\end{aligned}$$

Thus,

$$\frac{1}{p_{m_0+1}(z)(1-az)} = \frac{\sum_{j=1}^{m_0+1}\sum_{k=0}^{j} a^j b_{m_0+1+k-j} z^k}{a^{m_0+1} p_{m_0+1}(1/a) p_{m_0+1}(z)} + \frac{1}{p_{m_0+1}(1/a)(1-az)},$$

which proves (A.6). Letting $m = d$ we find that (A.4) is true. We note that this partial fraction expansion is valid even if $p(z)$ has z as a factor. In that case, some of the roots r_i are zero but the proof is not dependent on any of the roots being nonzero.

A.4. Expansion of $1/(p(z)(1-z))$

The partial fraction expansion depends on whether or not $p(z)$ has $1 - z$ as a factor. We consider both cases.

We first consider the case where $1 - z$ is not a factor of $p(z)$. Let

$$p(z) = p_0 + p_1 z + \cdots + p_j z^j$$

and

$$b(z) = 1 + b_1 z + \cdots + b_j z^j,$$

where $b_i = p_i/p_0$. Thus $p(z) = p_0 b(z)$. By (A.4) we find that

$$\begin{aligned} \frac{1}{p(z)(1-z)} &= \frac{1}{p_0 b(z)(1-z)} \\ &= \frac{1}{p_0 b(1)(1-z)} + \frac{\sum_{i=0}^{j-1}\sum_{k=0}^{i} b_{j+k-i} z^k}{p_0 b(1) b(z)}. \end{aligned}$$

Multiply the second term by p_0/p_0 and recognize that $p(z) = p_0 b(z)$ and $p_i = p_0 b_i$ to find that

$$\begin{aligned} \frac{1}{p(z)(1-z)} &= \frac{1}{p_0 b(1)(1-z)} + \frac{p_0 \sum_{i=0}^{j-1}\sum_{k=0}^{i} b_{j+k-i} z^k}{p_0^2 b(1)\, b(z)} \\ &= \frac{1}{p(1)(1-z)} + \frac{\sum_{i=0}^{j-1}\sum_{k=0}^{i} p_{j+k-i} z^k}{p(1)\, p(z)}. \end{aligned} \tag{A.8}$$

We now consider the case where $p(z)$ has $1 - z$ as a factor. Let

$$p(z) = (1-z)^m q(z),$$

where

$$q(z) = q_0 + q_1 z + \cdots + q_j z^j.$$

Factoring out $1/(1-z)^m$, applying (A.8), and recognizing $p(z)$ we find that

$$\begin{aligned} \frac{1}{p(z)(1-z)} &= \frac{1}{(1-z)^m (1-z)\, q(z)} \\ &= \frac{1}{(1-z)^m} \left(\frac{1}{q(1)(1-z)} + \frac{\sum_{i=0}^{j-1}\sum_{k=0}^{i} q_{j+k-i} z^k}{q(1)\, q(z)} \right) \\ &= \frac{1}{q(1)(1-z)^{m+1}} + \frac{\sum_{i=0}^{j-1}\sum_{k=0}^{i} q_{j+k-i} z^k}{q(1)\, p(z)}. \end{aligned} \tag{A.9}$$

A.5. Expansion of $1/(p(wz)q(z))$

The partial fraction expansion of

$$\frac{1}{p(wz)q(z)},$$

where w is a variable independent of the polynomial $p(z)$, has the form

$$\frac{1}{p(wz)q(z)} = \frac{r(w,z)}{t(w)p(wz)} + \frac{s(w,z)}{t(w)q(z)}, \tag{A.10}$$

where $r(w)$, $s(w,z)$, and $t(w)$ are polynomials.

The proof is by induction. If $p(wz)$ is a polynomial of degree m in wz and $q(z)$ is of degree n, then we can factor $p(wz)$ over the complex numbers as

$$p(wz) = p_m(wz - r_1)\cdots(wz - r_m)$$

and express $q(z)$ as

$$q(z) = q_0 + q_1 z + \cdots + q_n z^n.$$

Let

$$\begin{aligned} p_i(wz) &= p_m(wz - r_1)\cdots(wz - r_i), \\ T_0(w,z) &= \frac{1}{p_m q(z)}, \quad \text{and} \\ T_i(w,z) &= \frac{T_{i-1}(w,z)}{wz - r_i}. \end{aligned}$$

Then $1/(p(wz)q(z)) = T_m(w,z)$. We will use induction on m to prove that $T_m(w,z)$ has the form

$$T_m(w,z) = \frac{r_m(w,z)}{t_m(w)p_m(wz)} + \frac{s_m(w,z)}{t_m(w)q(z)}. \tag{A.11}$$

The base case is

$$T_1(w,z) = \frac{1}{p_m(wz - r_1)q(z)} = \frac{1}{-p_m r_1(1 - wz/r_1)q(z)}.$$

By (A.4) this is

$$T_1(w,z) = \frac{1}{-p_m r_1 q(r_1/w)(1 - wz/r_1)} + \frac{\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_1}\right)^j q_{n+k-j} z^k}{-p_m r_1 \left(\frac{w}{r_1}\right)^n q(r_1/w)q(z)}.$$

We find that

$$\begin{aligned} q\left(\frac{r_1}{w}\right) &= q_0 + q_1\left(\frac{r_1}{w}\right) + q_2\left(\frac{r_1}{w}\right)^2 + \cdots + q_n\left(\frac{r_1}{w}\right)^n \\ &= \frac{1}{w^n}\left(q_0 w^n + q_1 r_1 w^{n-1} + q_2 r_1^2 w^{n-2} + \cdots q_n r_1^n\right). \end{aligned}$$

Substituting and rearranging gives

$$\begin{aligned} T_1(w,z) &= \frac{w^n}{p_m(wz - r_1)\left(q_0 w^n + q_1 r_1 w^{n-1} + q_2 r_1^2 w^{n-2} + \cdots + q_n r_1^n\right)} \\ &\quad + \frac{\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_1}\right)^j q_{n+k-j} z^k}{-p_m r_1 \left(\frac{1}{r_1}\right)^n \left(q_0 w^n + q_1 r_1 w^{n-1} + q_2 r_1^2 w^{n-2} + \cdots + q_n r_1^n\right) q(z)} \\ &= \frac{w^n}{p_1(z)\left(q_0 w^n + q_1 r_1 w^{n-1} + q_2 r_1^2 w^{n-2} + \cdots + q_n r_1^n\right)} \\ &\quad + \frac{-\frac{r_1^{n-1}}{p_m}\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_1}\right)^j q_{n+k-j} z^k}{\left(q_0 w^n + q_1 r_1 w^{n-1} + q_2 r_1^2 w^{n-2} + \cdots + q_n r_1^n\right) q(z)}. \end{aligned}$$

Letting

$$\begin{aligned} r_1(w,z) &= w^n, \\ s_1(w) &= -\frac{r_1^{n-1}}{p_m}\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_1}\right)^j q_{n+k-j}z^k, \quad \text{and} \\ t_1(w) &= q_0w^n + q_1r_1w^{n-1} + q_2r_1^2w^{n-1} + \cdots q_nr_1^n, \end{aligned}$$

we find that

$$T_1(w,z) = \frac{r_1(w,z)}{t_1(w)p_1(wz)} + \frac{s_1(w,z)}{t_1(w)q(z)}.$$

To prove the inductive step, assume that (A.11) is true for $m = m_0$ and prove it is true for $m = m_0 + 1$. By the induction hypothesis we have

$$\begin{aligned} & T_{m_0+1}(w,z) \\ &= \frac{T_{m_0}(w,z)}{wz - r_{m_0+1}} \\ &= \frac{1}{wz - r_{m_0+1}}\left(\frac{r_{m_0}(w,z)}{t_{m_0}(w)p_{m_0}(wz)} + \frac{s_{m_0}(w,z)}{t_{m_0}(w)q(z)}\right) \\ &= \frac{r_{m_0}(w,z)}{(wz - r_{m_0+1})\,t_{m_0}(w)p_{m_0}(wz)} + \frac{s_{m_0}(w,z)}{(wz - r_{m_0+1})\,t_{m_0}(w)q(z)} \\ &= \frac{r_{m_0}(w,z)}{(wz - r_{m_0+1})\,t_{m_0}(w)p_{m_0}(wz)} + \frac{-s_{m_0}(w,z)}{t_{m_0}(w)}\,\frac{1}{r_{m_0+1}\,(1 - wz/r_{m_0+1})\,q(z)}. \end{aligned}$$

We now want to expand $1/((1 - wz/r_{m_0+1})q(z))$ using partial fractions with respect to z. Applying (A.4) and rearranging gives

$$\begin{aligned} & \frac{1}{(1 - wz/r_{m_0+1})q(z)} \\ &= \frac{1}{q(r_{m_0+1}/w)(1 - wz/r_{m_0+1})} + \frac{\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_{m_0+1}}\right)^j q_{n+k-j}z^k}{\left(\frac{w}{r_{m_0+1}}\right)^n q(r_{m_0+1}/w)q(z)} \\ &= \frac{w^n}{(1 - wz/r_{m_0+1})Q_{m_0+1}(w)} + \frac{\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_{m_0+1}}\right)^j q_{n+k-j}z^k}{\left(\frac{1}{r_1}\right)^n Q_{m_0+1}(w)q(z)}, \end{aligned}$$

where $Q_{m_0+1}(w) = w^n q(r_{m_0+1}/w) = q_0w^n + q_1r_{m_0+1}w^{n-1} + q_2r_{m_0+1}^2w^{n-2} + \cdots + q_nr_{m_0+1}^n$. Substituting,

$$\begin{aligned} & T_{m_0+1}(w,z) \\ &= \frac{r_{m_0}(w,z)}{(wz - r_{m_0+1})\,t_{m_0}(w)p_{m_0}(wz)} + \frac{-s_{m_0}(w,z)w^n}{t_{m_0}(w)r_{m_0+1}(1 - wz/r_{m_0+1})Q_{m_0+1}(w)} \\ &\quad + \frac{-s_{m_0}(w,z)\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_{m_0+1}}\right)^j q_{n+k-j}z^k}{t_{m_0}(w)r_{m_0+1}\left(\frac{1}{r_{m_0+1}}\right)^n Q_{m_0+1}(w)q(z)} \end{aligned}$$

$$= \frac{r_{m_0}(w,z)}{(wz - r_{m_0+1})\,t_{m_0}(w)p_{m_0}(wz)} + \frac{s_{m_0}(w,z)w^n}{t_{m_0}(w)(wz - r_{m_0+1})Q_{m_0+1}(w)} + \frac{-s_{m_0}(w,z)r_{m_0+1}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_{m_0+1}}\right)^j q_{n+k-j}z^k}{t_{m_0}(w)Q_{m_0+1}(w)q(z)}.$$

We can put the first two terms over a common denominator and recognize $p_{m_0+1}(wz)$ to get

$$\begin{aligned} T_{m_0+1}(w,z) &= \frac{r_{m_0}(w,z)Q_{m_0+1}(w) + s_{m_0}(w,z)w^n p_{m_0}(wz)}{t_{m_0}(w)(wz - r_{m_0+1})p_{m_0}(wz)Q_{m_0+1}(w)} \\ &\quad + \frac{-s_{m_0}(w,z)r_{m_0+1}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_{m_0+1}}\right)^j q_{n+k-j}z^k}{t_{m_0}(w)Q_{m_0+1}(w)q(z)} \\ &= \frac{r_{m_0}(w,z)Q_{m_0+1}(w) + s_{m_0}(w,z)w^n p_{m_0}(wz)}{t_{m_0}(w)p_{m_0+1}(wz)Q_{m_0+1}(w)} \\ &\quad + \frac{-s_{m_0}(w,z)r_{m_0+1}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_{m_0+1}}\right)^j q_{n+k-j}z^k}{t_{m_0}(w)Q_{m_0+1}(w)q(z)}. \end{aligned}$$

Letting

$$\begin{aligned} r_{m_0+1}(w,z) &= r_{m_0}(w,z)Q_{m_0+1}(w) + s_{m_0}(w,z)w^n p_{m_0}(wz), \\ s_{m_0+1}(w,z) &= -s_{m_0}(w,z)r_{m_0+1}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{j}\left(\frac{w}{r_{m_0+1}}\right)^j q_{n+k-j}z^k, \quad \text{and} \\ t_{m_0+1}(w) &= t_{m_0}(w)Q_{m_0+1}(w), \end{aligned}$$

we find that

$$T_{m_0+1}(w,z) = \frac{r_{m_0+1}(w,z)}{t_{m_0+1}(w)p_{m_0+1}(wz)} + \frac{s_{m_0+1}(w,z)}{t_{m_0+1}(w)q(z)}.$$

Appendix B

MAPLE Source Code

Source code is presented for the procedures that were implemented. Following each procedure are examples of its use. Where Maple provides a similar function, their differences are illustrated.

B.1. Polynomial Procedures

B.1.1. Package polynom

Source Code

```
############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
############################################################
#
#
# defines table for polynom package
#

polynom := table([

    `polynom/coeffs` = `readlib(`polynom/coeffs`,
                        ``.my__lib__name.`/polynom/coeffs.m`)`,
    `polynom/equate` = `readlib(`polynom/equate`,
                        ``.my__lib__name.`/polynom/equate.m`)`,
    `polynom/factor` = `readlib(`polynom/factor`,
                        ``.my__lib__name.`/polynom/factor.m`)`,
    `polynom/hasprod` = `readlib(`polynom/hasprod`,
                        ``.my__lib__name.`/polynom/hasprod.m`)`,
    `polynom/map`    = `readlib(`polynom/map`,
                        ``.my__lib__name.`/polynom/map.m`)`,
    `polynom/terms`  = `readlib(`polynom/terms`,
                        ``.my__lib__name.`/polynom/terms.m`)`,
    `polynom/termscale` = `readlib(`polynom/termscale`,
                        ``.my__lib__name.`/polynom/termscale.m`)`
  ]):
```

```
save polynom, ``.my__lib__name.`/polynom.m`;
```

Example

```
> with(polynom);
[polynom/coeffs, polynom/equate, polynom/factor, polynom/hasprod,

    polynom/map, polynom/terms, polynom/termscale]
```

B.1.2. Procedure polynom/coeffs

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# like coeffs, except orders by degree
#
# pz    a polynomial of z
# z     polynomial variable
# r     unnasssigned name for returning list of powers of z
#       that correspond to coefficients returned by the procedure
#

`polynom/coeffs` := proc(pz, z, r)
  local c,    # expression sequence of coefficients
        t;    # expression sequence of powers of z

  if not type(r, name) then
    ERROR(`third argument`, r, `not a name`)
  fi;

  c := NULL;
  t := NULL;

  collect(expand(pz), z);

#
# process terms in order of degree
#

  while " <> 0 do
    t:=t, z^ldegree(", z);
    c:=c, tcoeff("", z);
    """ - tcoeff("""", z)* z^ldegree("""", z)
  od;

#
# return expression sequence of coefficients
#
```

```
  r := t;
  c
end:

save 'polynom/coeffs', ''.my__lib__name.'/polynom/coeffs.m';
```

Example

```
> p:=w*x^2+w^2*x;
```

$$p := w\,x^2 + w^2\,x$$

```
> 'polynom/coeffs'(p, w, 't');
```

$$x^2, x$$

```
> t;
```

$$w, w^2$$

```
> coeffs(p, w, 't');
```

$$x, x^2$$

```
> t;
```

$$w^2, w$$

```
> 'polynom/coeffs'(p, x, 't');
```

$$w^2, w$$

```
> t;
```

$$x, x^2$$

```
> coeffs(p, x, 't');
```

$$w^2, w$$

```
> t;
```

$$x, x^2$$

B.1.3. Procedure polynom/equate

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
```

```
# Returns p(z) = a * t(b*z)
#
# ty        polynomial in y
# y         polynomial variable
# pz        polynomial in z
# z         polynomial variable
# result    name
#           if function is true, assigned list [a,b]
#

`polynom/equate` := proc(ty, y, pz, z, result)
  local t_lt, p_lt,       # trailing coeffs of pz, ty
        tty, ppz,         # ty/tcoeff(ty),  pz/tcoeff(pz)
        t_c, p_c,         # polynomial coefficients
        t_t, p_t,         # corresponding powers of y, z
        i,                # loop counter
        dd,               # expression sequence of degree
                          # of elements in t_t

        ratio;            # prospective value of b

  if not type(result, name) then
    ERROR(`fifth argument must be type name`)
  fi;

  collect(expand(ty), y);
  t_lt := tcoeff(", y);

  collect(expand(pz), z);
  p_lt := tcoeff(", z);

  tty := collect(expand(ty/t_lt), y, distributed, normal);
  ppz := collect(expand(pz/p_lt), z, distributed, normal);

  t_c := `polynom/coeffs`( tty, y, 't_t');
  p_c := `polynom/coeffs`( ppz, z, 'p_t');

#
# determine if polynomials have same number of terms
#

  if nops([t_c]) <> nops([p_c]) then
    result := NULL;
    RETURN(false)
  fi;

#
# determine if polynomial terms have same powers
#

  true;
  for i to nops([t_c]) while " do
    degree(t_t[i], y) = degree(p_t[i], z)
  od;
```

```
  if not " then
    result := NULL;
    RETURN(false)
  fi;

#
#
#

  dd := op(map( proc(t,v) degree(t,v) end, [t_t], y) );

#
# determine if found b and can relate coefficients
#

  if type(dd[2],odd) then
    ratio := (p_c[2]/t_c[2])^(1/dd[2]);
    if ppz - subs(y=ratio*z, tty) = 0 then
      result:=[p_lt/t_lt, ratio];
      RETURN(true)
    fi
  else
    ratio := (p_c[2]/t_c[2])^(1/dd[2]);
    if (ppz - subs(y=ratio*z, tty) = 0) then
      result:=[p_lt/t_lt, ratio];
      RETURN(true)
    elif (ppz - subs(y=-ratio*z, tty) = 0) then
      result:=[p_lt/t_lt, -ratio];
      RETURN(true)
    fi

  fi;
  result := NULL;
  false
end:

save 'polynom/equate', ``.my__lib__name.`/polynom/equate.m`;
```

Example

```
> 'polynom/equate'(1+z+z^2, z, 1-2*w+4*w^2, w, 'c');
                                        true

> c;
                                     [1, -2]

> 'polynom/equate'(1+z+z^2, z, 1+2*w+4*w^2, w, 'c');
                                        true

> c;
                                      [1, 2]

> 'polynom/equate'(1+z+z^2, z, 1-3*z+4*z^2, z, 'c');
```

```
                                        false

> 'polynom/equate'(1+z+z^2, z, 1-z, z, 'c');
                                        false

> 'polynom/equate'(1-y, y, 3-6*w, w, 'c');
                                         true

> c;
                                      [3, 2]
```

B.1.4. Procedure polynom/factor

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# factors a polynomial over the complex field
# does not fail gracefully if it cannot find the roots
#
# pz      polynomial in z
# z       polynomial variable
#

'polynom/factor' := proc(pz, z)

  local i, roots, c;

  collect(expand(pz), z);
  c := lcoeff(", z);
  collect(expand(pz/c), z);

  roots := solve(" = 0, z);

  roots := map( proc(r,z) z-r end, ["], z);
  1;
  for i in roots do
    "*i
  od;

  c * "
end:

save 'polynom/factor', ''.my__lib__name.'/polynom/factor.m';
```

Example

```
> factor(1-z-z^2);
                                          2
                                    1 - z - z
```

```
> `polynom/factor`(1-z-z^2, z);
                                    1/2                    1/2
                  - (z + 1/2 - 1/2 5   ) (z + 1/2 + 1/2 5   )

> expand(");
                                          2
                                    1 - z - z
```

B.1.5. Procedure polynom/hasprod

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# determines if polynomial expression p has t as a factor
# p can have factors of the form n(z)^m, where m is a symbol
#
# p  expression built from product of polynomials
#    and polynomials to a power
# t  a polynomial
#

`polynom/hasprod` := proc(p, t)
  local i;

  `polynom/hasprod/gcd` := 'readlib( ''`polynom/hasprod/gcd`'',
                           `` . my__lib__name . `/polynom/hasprod/gcd.m`)';

#
# if type is product check each factor
#

  if type(p, `*`) then
    false;
    for i to nops(p) while not " do
      not type( `polynom/hasprod/gcd`( op(i,p), t), constant)
    od
  else
    not type( `polynom/hasprod/gcd`( p, t), constant)
  fi

end:

save `polynom/hasprod`, ``.my__lib__name.`/polynom/hasprod.m`;

#
# determine if t is a factor of p
# p is either a polynomial or a polynomial ^ power
#
```

```
'polynom/hasprod/gcd' := proc(p,t)

  if type(p, '^') then
    gcd( op(1,p), t)
  else
    gcd( p, t)
  fi
end:

save 'polynom/hasprod/gcd', ''.my__lib__name.'/polynom/hasprod/gcd.m';
```

Example

```
> p := (1-w*z)*(1-z-z^2);
                                                    2
                          p := (1 - w z) (1 - z - z )

> 'polynom/hasprod'(p, 1-w*z);
                                      true

> 'polynom/hasprod'(p, 1-z-z^2);
                                      true

> 'polynom/hasprod'(p, 1-w);
                                     false

> 'polynom/hasprod'(p, 1-w-w^2);
                                     false

> p := (1-w*z)*(1-z)^m;
                                                   m
                             p := (1 - w z) (1 - z)

> 'polynom/hasprod'(p, 1-w*z);
                                      true

> 'polynom/hasprod'(p, 1-z);
                                      true

> 'polynom/hasprod'(p, 1-w);
                                     false
```

B.1.6. Procedure polynom/map

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# maps fn to each term of pz
#
```

```
# fn   MAPLE procedure
# pz   polynomial in z
# z    polynomial variable
#
# each term in pz passed to fn()
# optional arguments to `polynom/map`() passed to
# fn() as arguments
#

`polynom/map` := proc(fn, pz, z)
  local i, t;    # terms of pz

  t := `polynom/terms`(pz, z);

  0;
  if nargs > 3 then
    for i to nops([t]) do
      " + fn(op(i,[t]), args[4..nargs])
    od
  else
    for i to nops([t]) do
      " + fn(op(i,[t]))
    od
  fi

end:

save `polynom/map`, ``.my__lib__name.`/polynom/map.m`;
```

Example

```
> `polynom/map`( proc(t,z,c) subs(z=c*z, t) end, 1+w+w^2, w, w, 2);
                                        2
                                1 + 2 w + 4 w
```

B.1.7. Procedure polynom/terms

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# returns sequence of terms from pz in order of degree
#
# pz  polynomial in z
# z   polynomial variable
#

`polynom/terms` := proc(pz, z)
  local i, c, d;
```

```
  c := NULL;    # build expression sequence of terms

  collect(expand(pz), z, distributed, normal);

#
# pick off terms by degree
#

  while " <> 0 do
    d := ldegree(", z);
    c := c, tcoeff("", z)*z^d;
    """ - tcoeff("""", z)*z^d
  od;

  c
end:

save 'polynom/terms', ''.my__lib__name.'/polynom/terms.m';
```

Example

```
> 'polynom/terms'(a+c*z^2+b*z, z);
                                          2
                                 a, b z, c z
```

B.1.8. Procedure polynom/termscale

Source Code

```
############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
############################################################
#
#
# Shifts and scales fn to determine sequence encoded
# by the product of pz and the generating function of fn
#
# pz  polynomial in z
# z   polynomial variable
# fn  expression for term in sequence
#     ie  n*2^n or F(n) or n*C(n)
# n   sequence index used in fn
#

'polynom/termscale' := proc( pz, z, fn, n )
  local t, c, i;

  c := 'polynom/coeffs'(pz, z, 't');

  0;
  for i to nops([c]) do
    " + op(i,[c]) * subs( n = n - degree( op(i,[t]) ), fn)
  od
```

```
end:

save `polynom/termscale`, ``.my__lib__name.`/polynom/termscale.m`;
```

Example

```
> `polynom/termscale`(1-z+z^2, z, F(n), n);
                          F(n) - F(n - 1) + F(n - 2)

> `polynom/termscale`(1-z-z^2, z, n^2, n);
                            2          2          2
                           n  - (n - 1)  - (n - 2)

> `polynom/termscale`(1-2*w+3*w^2, w, n*2^n*F(n), n);
        n                   (n - 1)                         (n - 2)
     n 2  F(n) - 2 (n - 1) 2        F(n - 1) + 3 (n - 2) 2        F(n - 2)
```

B.2. Rational Generating Functions

B.2.1. Package rational

Source Code

```
###############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
###############################################################
#
rational := table([

    `rational/encode` = 'readlib(''rational/encode'',
                        ``.my__lib__name.`/rational/encode.m`)',
    `rational/expand` = 'readlib(''rational/expand'',
                        ``.my__lib__name.`/rational/expand.m`)',
    `rational/pfrac`  = 'readlib(''rational/pfrac'',
                        ``.my__lib__name.`/rational/pfrac.m`)',
    `rational/relate` = 'readlib(''rational/relate'',
                        ``.my__lib__name.`/rational/relate.m`)',
    `rational/recur`  = 'readlib(''rational/recur'',
                        ``.my__lib__name.`/rational/recur.m`)',
    `rational/simp`   = 'readlib(''rational/simp'',
                        ``.my__lib__name.`/rational/simp.m`)',
    `type/ratseq`     = 'readlib(''type/ratseq'',
                        ``.my__lib__name.`/type/ratseq.m`)'
  ]):

save rational, ``.my__lib__name.`/rational.m`;
```

Example

```
> with(rational);
      [rational/encode, rational/expand, rational/pfrac, rational/recur,

          rational/relate, rational/simp, type/ratseq]
```

B.2.2. Procedure rational/encode

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# encodes fn as a rational generating function
# fn must satisfy type(fn, ratseq, n)
# where type ratseq is defined by type/ratseq
#
# fn    an expression in n of type ratseq
# n     index variable
# z     variable for generating function
#
# optional arguments:
# n = c     index of first nonzero term, ie n=5
# [n_i=fn_i, ...]  list of terms not defined by fn
#           n_i is integer index, fn_i is value of sequence at n_i
#

`rational/encode` := proc(fn, n, z)
  local i,
        ep,         # expanded polynomial form of fn
        gz,         # generating function of n^i, built iteratively
        Fz,         # builds generating function of <fn>
        start,      # special case -- index first nonzero term
        l;          # special case -- list of terms not defined by fn

  `rational/encode/expterms` := `readlib(``rational/encode/expterms``,
                        `` . my__lib__name. `/rational/encode/expterms.m`)`;

#
# get optional arguments
#

  start := 0;
  l := [];

  for i from 4 to nargs do
    args[i];
    if type(", `list`) then
      l := "
    else
      start := rhs(")
    fi
  od;

#
# shift to handle any index for first nonzero term
#
```

```
  subs(n=n+start, fn);

#
# expand fn, collect as a polynomial in n
# each coefficient will be sum of terms of form
# a*b^(c*n+d)
#

  expand(");
  if type(",`+`) then
    map(simplify,")
  else
    simplify(")
  fi;

  ep := collect(", n);

#
# loop through coefficients of n^i
# gz is generating function of <n^i>
# for each coefficient, loop through terms and encode
# generating function of a*b^(c*n+d)n^i using gz
#

  gz := 1/(1-z);
  Fz := 0;
  for i from 0 to frontend( degree, [ep,n]) - 1 do
    frontend(coeff, [ep, n, i]);
    if " <> 0 then
      Fz := Fz + `rational/encode/expterms`( ",  n, gz, z)
    fi;
    gz := z * diff(gz,z)
  od;

  Fz := Fz + `rational/encode/expterms`( frontend( lcoeff, [ep, n]),  n, gz,
z);

#
# shift back generating function to handle any first nonzero term
#

  Fz*z^start;

#
# correct for terms not defined by fn
#

  for i in 1 do
    " + simplify( rhs(i) - subs(n=lhs(i), fn) ) * z^lhs(i)
  od;
  "
end:

save `rational/encode`, ``.my__lib__name.`/rational/encode.m`;
```

```
#
# encode coefficients of each term in polynomial ep
#

`rational/encode/expterms` := proc(fn, n, gz, z)
  local t, c, e;

  if type(fn, `+`) then
    map(`rational/encode/expterms`, fn, n, gz, z)
  else

#
# parse fn so that a*b^(c*n+d) is (a*b^d)*(b^c)^n
# and encode it
#

    if type(fn, `*`) then
      c := 1;
      e := 1;
      for t in fn do
        if type(t, `^`) then
          c := c * op(1,t)^coeff(op(2,t), n, 0);
          e := e * op(1,t)^coeff(op(2,t), n, 1)
        else
          c := c * t
        fi
      od
    elif type(fn, `^`) then
      c := op(1, fn)^coeff(op(2, fn), n, 0);
      e := op(1, fn)^coeff(op(2, fn), n, 1)
    else
      c := fn;
      e := 1
    fi;
    c * subs(z = e*z, gz)

  fi
end:

save `rational/encode/expterms`,
``.my__lib__name.`/rational/encode/expterms.m`;
```

Example

```
> `rational/encode`(a*n^2+b*n+c, n, z);
```

$$\frac{c}{1-z} + \frac{b\,z}{(1-z)^2} + a\,z\left(\frac{1}{(1-z)^2} + \frac{2\,z}{(1-z)^3}\right)$$

```
> normal(");
                                  2            2            2
                    c - 2 c z + c z  + b z - b z  + a z + a z
                  - ---------------------------------------
                                          3
                                 (- 1 + z)

> collect(numer("), z)/denom(");
                                 2
                     (- c + b - a) z  + (2 c - b - a) z - c
                     ------------------------------------
                                          3
                                 (- 1 + z)

> 'rational/encode'( (n*2^n+3^n)*(n-2)*5^n, n, z);
                        1                z                z
                  - 2 -------- - 20 ----------- + 15 -----------
                      1 - 15 z               2                2
                                     (1 - 10 z)       (1 - 15 z)

                              /     1                 z     \
                           + 10 z |----------- + 20 ----------|
                              |           2               3|
                              \(1 - 10 z)       (1 - 10 z) /

> normal(");
                                      2         3          4
                    - 2 + 95 z - 1350 z  + 4250 z  + 22500 z
                  - ---------------------------------------
                                      2              3
                           (- 1 + 15 z)   (- 1 + 10 z)
```

B.2.3. Procedure `rational/expand`

Source Code

```
############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
############################################################
#
#
# Expands rational generating function Fz
#
# Fz  rational function in z
# z   generating function variable
# n   index variable for sequence
#

'rational/expand' := proc(Fz, z, n)
  local num,     # numerator
        den,     # denominator
        i,       #
        d_c,     # low order coeff of denom
        d_d;     # low degree of denom
```

```
  `rational/expand/extract` := 'readlib(''`rational/expand/extract`'',
                               ``.my__lib__name.`/rational/expand/extract.m`)';

  if not type(z,name) then
    ERROR(`argument`, z, `not a name`)
  elif not type(n,name) then
    ERROR(`argument`, n, `not a name`)
  elif not type(Fz,ratpoly,z) then
    ERROR(`argument`, Fz, `not rational`)
  fi;

  if not has(Fz,z) then
    Fz * `delta`(n,0)
  elif type(Fz,`+`) then
    term := map( `rational/expand`, Fz, z, n)
  else
    normal(Fz);
    num := numer(");
    den := denom("");

    d_c := tcoeff(den, z);
    d_d := ldegree(den,z);
    den := den/d_c/z^d_d;

    if degree(den,z) = 0 then

#
#      handle finite rational generating functions
#

      `polynom/map`(proc(t,z,n,d) lcoeff(t,z)*`delta`(n,degree(t,z)-d) end,
                    num, z, z, n, d_d);
      "/d_c
    else

#
#     apply partial fractions, expand each term
#

      `rational/pfrac`(1/den, z);
      if type(", `+`) then
        map(`rational/expand/extract`, ", z, n)
      else
        `rational/expand/extract`(", z, n)
      fi;

#
#  shift and scale for d_c, d_d
#

      subs(n=n+d_d, ")/d_c;
      `polynom/termscale`(num, z, ", n)
    fi
```

```
  fi
end:

save `rational/expand`, ``.my__lib__name.`/rational/expand.m`;

`rational/expand/extract` := proc( Fz, z, n )
  local m, i, c, num, den, pz;

  normal(Fz);
  num := numer(");
  den := denom("");

#
# factor out any constants from den
#

  if type(",`*`) then
    c := map(proc(t,z) if not has(t,z) then t else 1 fi end, ", z);
    den := den/"
  else
    c := 1
  fi;

#
# den is a polynomial now, of form (a-b*z)^(m+1)
# map 1/den to Choose(m+n, n) * b^n / a^(m+n+1)
#
  if type(den,`^`) then
    m := op(2,den) - 1;
    pz := op(1,den)
  else
    m:= 0;
    pz := den;
  fi;

#
# compute the binomial coefficient
#

  product( (n+'i')/'i', 'i'=1..m );

  " *  (-coeff(pz,z,1) / coeff(pz,z,0) ) ^ n  / coeff(pz,z,0)^(m+1);

#
# use num to shift and scale
# multiply in any constant
#

  expand(numer(Fz));
  `polynom/termscale`(", z, "", n);
  "/c
end:

save `rational/expand/extract`,
```

```
''.my__lib__name.'/rational/expand/extract.m';
```

Example

```
> 'rational/expand'(z/(1-z-z^2), z, n);
                 /       1    \(n - 1)      /       1    \(n - 1)
                 |- 2 --------|             |- 2 --------|
                 |         1/2|             |         1/2|
                 \    1 - 5   /             \    1 + 5   /
               - 2 ---------------------- + 2 ----------------------
                              1/2   1/2                    1/2   1/2
                         (1 - 5   ) 5                 (1 + 5   ) 5

> 'rational/expand'(1/(2-3*z)^4, z, n);
                                                                   n
                  1/16 (1 + n) (1/2 n + 1) (1/3 n + 1) (3/2)

> normal(");
                                                                 n
                      1/96 (1 + n) (n + 2) (n + 3) (3/2)
```

B.2.4. Procedure rational/pfrac

Source Code

```
############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
############################################################
#
#
# Performs partial fractions over the complex field
#
# Fz   nontrivial rational generating function
# z    generating function variable
#
# NOTE : no error checking of failure in 'polynom/factor'()
#
'rational/pfrac' := proc(Fz, z)
  local temp, den, d_poly, d_power, new_gf, i, num, res, c;

  if nargs < 2 then
    ERROR('Insufficient arguments')
  elif not type(z, name) then
    ERROR('Argument ' . z . ' not a name')
  elif not type(Fz, ratpoly, z) or type(Fz, polynom, z) then
    ERROR('Argument ' . Fz . ' not rat poly')
  fi;

#
# partial fractions over the rationals
#

  new_gf := convert(normal(Fz), parfrac, z);
```

```
#
#  process each term,
#  if can be factored, do so and apply partial fractions again
#

  if type(new_gf, `+`) then
    res := 0;
    for i from 1 to nops(new_gf) do
      temp := op(i,new_gf);
      num := numer(temp);
      den := denom(temp);
      if type(den,`*`) then
        c:= map(proc(f,v) if has(f,v) then 1 else f fi end, den, z)
      else
        c:=1
      fi;
      den := den/c;
      if type(den, `^`) then
        d_poly := op(1,den);
        d_power := op(2,den)
      else
        d_poly := den;
        d_power := 1
      fi;
      if degree(d_poly, z)>1 then
        res := res
               + convert(num/c/(`polynom/factor`(d_poly, z))^d_power,
                         parfrac, z, true)
      else
        res := res + temp
      fi
    od
  else
    num := numer(new_gf);
    den := denom(new_gf);
    if type(den,`*`) then
      c:= map(proc(f,v) if has(f,v) then 1 else f fi end, den, z)
    else
      c:=1
    fi;
    den := den/c;
    if type(den, `^`) then
      d_poly := op(1,den);
      d_power := op(2,den)
    else
      d_poly := den;
      d_power := 1
    fi;
    if degree(d_poly, z)>1 then
      res := convert(num/c/(`polynom/factor`(d_poly, z))^d_power,
                     parfrac, z, true)
    else
      res := new_gf
```

```
    fi
  fi;

  res
end:

save 'rational/pfrac', ''.my__lib__name.'/rational/pfrac.m';
```

Example

```
> 'rational/pfrac'(z/(1-z-z^2), z);
                                1/2                       1/2
                          - 1 + 5                    1 + 5
                  - ---------------------- - ----------------------
                       1/2             1/2        1/2             1/2
                    5     (2 z + 1 - 5    )    5     (2 z + 1 + 5    )

> convert(z/(1-z-z^2), parfrac, z);
                                      z
                                - ------------
                                             2
                                  - 1 + z + z
```

B.2.5. Procedure type/ratseq

Source Code

```
############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
############################################################
#
#
# used with MAPLE's type function to determine if fn is
# a valid closed form expression for RAT_SEQ
#
# fn   expression in n
# n    index variable
#

'type/ratseq' := proc(fn, n)

  local i, expr;

  expr := fn;

  if type(expr, polynom, n) then
    RETURN(true)
  fi;

#
# parse expr
# continue until find invalid form or everything checks out
#
```

```
  if type(expr, '^') then
    if not has( op(1,expr), n) then
      op(2,expr);
      if type(", polynom, n) and degree(", n)<=1 then
        RETURN(true)
      fi
    fi;
    RETURN(false)
  fi;

  if type(expr, '+') then
    true;
    for i to nops(expr) while " do
      'type/ratseq'(op(i,expr), n)
    od;
    RETURN(")
  fi;

  if type(expr, '*') then
    true;
    for i to nops(expr) while " do
      'type/ratseq'(op(i,expr), n)
    od;
    RETURN(")
  fi;

  false;

end:

save 'type/ratseq', ''.my__lib__name.'/type/ratseq.m':
```

Example

```
> type(a*n^2+b*n+c, ratseq, n);
                                        true

> type((n*2^n+3^n)*(n-1)*5^n, ratseq, n);
                                        true

> type(n^(-1), ratseq, n);
                                       false

> 'rational/expand'(z/(1-z-z^2), z, n);
                  /       1    \(n - 1)      /       1    \(n - 1)
                  |- 2 --------|             |- 2 --------|
                  |          1/2|             |          1/2|
                  \    1 - 5   /             \    1 + 5   /
              - 2 ---------------------- + 2 ----------------------
                            1/2   1/2                   1/2   1/2
                       (1 - 5   ) 5                (1 + 5   ) 5

> type(", ratseq, n);
```

```
true
```

B.2.6. Procedure rational/recur

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# Determines homogeneous linear recurrence with
# constant coefficients that defines sequence encoded by Fz
#
# Fz  nontrivial rational function
# z   generating function variable
# F   name for sequence
# n   index variable
#

`rational/recur` := proc(Fz, z, F, n)
  local c, d, deg, recur, i, fn, bterms, res, nn, dd, shift;

  if not type(z, name) then
    ERROR(`second argument`, z, `not a name`)
  fi;

  if not type( normal(Fz), ratpoly, z) then
    ERROR(`first argument`, Fz, `not a rational function`)
  fi;

  if not type(n, name) then
    ERROR(`fourth argument`, n, `is not a name`)
  fi;

  if type(F, name) then
    fn := F(n)
  elif type(F, function) then
    fn := op(0,F)(n)
  else
    ERROR(`third argument`, F, `is not a function name`)
  fi;

#
# build recurrence from characteristic generating function
#

  denom(normal(Fz));
  simplify("/tcoeff(", z)/z^ldegree(", z));
  c := `polynom/coeffs`(", z, 'd');

  deg := degree( d[ nops([d]) ], z);

  recur := 0;
```

```
  for i from 2 to nops([d]) do
    recur := recur - c[i] * subs(n=n-degree(d[i], z), fn)
  od;
  recur := fn = recur;

#
# find boundary condition
#

  normal(Fz);
  nn := numer(");
  dd := denom("");

#
# determine index of first term
#

  shift := ldegree(nn, z);
  nn := expand(nn/z^ldegree(nn, z) );
  shift := shift - ldegree(dd, z);
  dd := expand(dd/z^ldegree(dd, z) );

#
# determine number of terms needed
#

  if degree(nn, z) < degree(dd, z) then
    bterms := degree(dd, z)
  else
    bterms := degree(nn, z)+1
  fi;

#
# use differentiation method to find values
#

  res := [];
  nn/dd;
  1;
  for i from 0 to bterms-1 do
    res := [ op(res), subs(n=i+shift, fn)= subs(z=0, "") / " ];
    normal(diff("""", z));
    """"*(i+1)
  od;

  [recur, op(res)]

end:

save `rational/recur`, ``.my__lib__name.`/rational/recur.m`;
```

Example

```
> `rational/recur`(z/(1-z-z^2), z, F(n), n);
bytes used=1001928, alloc=417792, time=3.116
                [F(n) = F(n - 1) + F(n - 2), F(1) = 1, F(2) = 1]

> `rational/recur`(1/(1-2*z), z, t(n), n);
                          [t(n) = 2 t(n - 1), t(0) = 1]

> `rational/recur`(z/(1-z)^2, z, s(n), n);
               [s(n) = 2 s(n - 1) - s(n - 2), s(1) = 1, s(2) = 2]

> `rational/recur`((z+z^2)/(1-z)^3, z, r(n), n);
   [r(n) = 3 r(n - 1) - 3 r(n - 2) + r(n - 3), r(1) = 1, r(2) = 4, r(3) = 9]

> `rational/recur`((1+z+z^2+z^3)/(1-2*z+z^2), z, t(n), n);
    [t(n) = 2 t(n - 1) - t(n - 2), t(0) = 1, t(1) = 3, t(2) = 6, t(3) = 10]
```

B.2.7. Procedure `rational/relate`

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# `rational/relate`( option, F(z), z, F(n), n, G(y), y)
#
#  Relates sequence encoded by G(y) to F(n), the sequence
#  encoded by F(z), when option is one of the following:
#
# common       denom(G(y)), denom(F(z)) have a common factor
#              includes case where denom(F(z)) is factor of denom(G(y))
# contains     denom(G(y)) is a factor of denom(F(z))
#              common will do this case
#              common calls contains to relate the common factor
# super        denom of G(y) contains only powers of factors of denom(F(z))
#
# `rational/relate`( power, F(z), z, F(n), n, pwr)
#
#  Relates sequence encoded by F(z)^pwr to F(n),
#  the sequence encoded by F(z)
#
# `rational/relate`( denom, F(z), z, F(n), n)
#
# find sequence represented by 1/denom(F(z))
# can be easily mapped to characteristic generating function
#

`rational/relate` := proc(fn_opt, Fz, z, Fn, n)

  `rational/relate/common` := `readlib( ``rational/relate/common``,
                              ``.my__lib__name.`/rational/relate/common.m`)`;
```

```
  `rational/relate/contains` := `readlib( ``rational/relate/contains``,
                              ``.my__lib__name.`/rational/relate/contains.m`)`;
  `rational/relate/super` := `readlib( ``rational/relate/super``,
                                 ``.my__lib__name.`/rational/relate/super.m`)`;
  `rational/relate/denom` := `readlib( ``rational/relate/denom``,
                                 ``.my__lib__name.`/rational/relate/denom.m`)`;
  `rational/relate/power` := `readlib( ``rational/relate/power``,
                                 ``.my__lib__name.`/rational/relate/power.m`)`;

  if not type( `rational/relate/` . fn_opt, procedure) then
    ERROR(`invalid relate option`, fn_opt)
  fi;

  if nargs > 5 then
    `rational/relate/` . fn_opt(Fz, z, Fn, n, args[6..nargs] )
  else
    `rational/relate/` . fn_opt(Fz, z, Fn, n)
  fi
end:

save `rational/relate`, `` . my__lib__name . `/rational/relate.m`;

#
#  common
#

`rational/relate/common` := proc(Fz, z, fn, n, Gy, y)
  local Gz, com_fact, term;

  `rational/relate/contains` := `readlib( ``rational/relate/contains``,
                              ``.my__lib__name.`/rational/relate/contains.m`)`;

  Gz := subs(y=z, Gy);

  # find sequence encoded by common factor in denom's

  com_fact := gcd(denom(normal(Fz)), denom(normal(Gz)) );
  term := `rational/relate/contains`(Fz, z, fn, n, 1/com_fact, z);

  # isolate factors in denom of Gy that are not common with denom Fz
  # expand to find sequence that they encode
  # termscale for numerators

  convert(normal(Gz), parfrac, z);
  map( proc(fz, pz, zz)
              if degree(gcd(denom(fz), pz), zz) = degree(denom(fz), zz)
                then fz else 0 fi end,  ", com_fact, z );

  `polynom/termscale`( normal("*com_fact), z, term, n);
  " + `rational/expand`( """ - "", z, n);
  if type(fn, function) then
    `rational/simp`(", Fz, z, fn, n)
  else
    "
```

```
  fi

end:

save `rational/relate/common`,
       `` . my__lib__name . `/rational/relate/common.m`;

#
# contains
#

`rational/relate/contains`  := proc(Fz, z, fn, n, Gy, y)
  local term, Gz;

  `rational/relate/denom` := `readlib( ``rational/relate/denom``,
                              ``.my__lib__name.`/rational/relate/denom.m`)`;

  # find sequence encode by 1/denom(Fz)

  term := `rational/relate/denom`(Fz, z, fn, n);

  # relate sequence encoded by 1/denom(Gy) to term

  Gz := subs(y=z, Gy);
  denom(normal(Fz))/ denom(normal(Gz)) ;
  simplify(");

  `polynom/termscale`(", z, term, n);

  # termscale for numerator of Gy

  `polynom/termscale`(numer(normal(Gz)), z, ", n) ;
  if type(fn, function) then
    `rational/simp`(", Fz, z, fn, n)
  else
    "
  fi

end:

save `rational/relate/contains`,
       `` . my__lib__name . `/rational/relate/contains.m`;

#
# super
#

`rational/relate/super` := proc(Fz, z, fn, n, Gy, y)
  local Gz, dd, f, i, new_gf;

  `rational/relate/denom` := `readlib( ``rational/relate/denom``,
                              ``.my__lib__name.`/rational/relate/denom.m`)`;

  Gz := subs(y=z, Gy);
```

```
  dd := denom(normal(Gz));

  # iterate until denominator has denom(Gy) as a factor

  f := fn*z^n;
  new_gf := Fz;
  i := 0;
  while degree(gcd( denom(new_gf), dd), z) <> degree(dd, z) do
    f := diff(f, z);
    new_gf := normal(diff(new_gf, z));
    i := i+1
  od;
  f:=subs(n=n+i, f);
  f := `rational/relate/denom`( new_gf, z, subs(z=1, f), n);

  # find polynomial to multiply 1/denom(new_gf) by
  # in order to get Gy
  # termscale sequence encoded  1/denom(new_gf) to find
  # sequence encoded by Gy

  expand( simplify(Gz * denom(new_gf) ));
  `polynom/termscale`(", z, f, n);
  if type(fn, function) then
    `rational/simp`(", Fz, z, fn, n)
  else
    "
  fi

end:

save `rational/relate/super`,
       `` . my__lib__name . `/rational/relate/super.m`;

#
# denom
#

`rational/relate/denom` := proc(Fz, z, fn, n)
  local nn, dd, pn, pd, temp, cc, i;

  normal(Fz);
  nn := numer(");
  dd := denom("");

  if ldegree(nn, z) = degree(nn, z) then
    RETURN( subs(n = n + degree(nn, z), fn) / lcoeff(nn, z) )
  fi;

  # find 1/(nn*dd) = A/nn + B/dd
  # where pn = A/nn and pd = B/dd

  temp:=convert(1/nn/dd, parfrac, z);
  pn := map(proc(f,p,v) if has(gcd(denom(f),p),v) then f else 0 fi end,
```

```
            ", nn, z);
  pd := map(proc(f,p,v) if has(gcd(denom(f),p),v) then f else 0 fi end,
            "", dd, z);
  pn := normal(pn);
  pd := normal(pd);

  # find sequence encoded by 1/dd

  simplify( pd * nn / Fz );
  collect(", z);
  cc := coeffs( ", z, 'temp' );
  0;
  for i to nops([cc]) do
   " + cc[i] * subs( n = n - degree(temp[i], z), fn )
  od;

  if type(fn, function) then
    `rational/simp`(", Fz, z, fn, n)
  else
    "
  fi

end:

save `rational/relate/denom`,
       `` . my__lib__name . `/rational/relate/denom.m`;

#
# power
#

`rational/relate/power` := proc(Fz, z, fn, n, pwr)

  local nn, dd, t, i, dnum, dden, target;

  `rational/relate/denom` := 'readlib( ``rational/relate/denom``,
                              ``.my__lib__name.`/rational/relate/denom.m`)';

  normal(Fz);

  nn := numer(");
  dd := denom("");

  t := `rational/relate/denom`("""", z, fn, n);

  t := t * z^n;
  new_gf := 1/dd;

  # iterate to find desired power

  target := dd ^ pwr;
  i := 0;
  while degree(gcd( denom(new_gf), target), z) <> degree(target, z) do
    t := diff(t, z);
```

```
    new_gf := normal(diff(new_gf, z));
    i := i+1
  od;
  t := subs(n=n+i, t);

  # find sequence encoded by denom, then termscale for numerator

  t := `rational/relate/denom`( new_gf, z, subs(z=1, t), n);

  expand(nn^pwr* denom(new_gf) / target );
  `polynom/termscale`(", z, t, n);

  if type(fn, function) then
    `rational/simp`(", Fz, z, fn, n)
  else
    "
  fi
end:

save `rational/relate/power`,
       `` . my__lib__name . `/rational/relate/power.m`;
```

Example

```
> f := 1/(1-a*z)/(1-b*z);
```

$$f := \frac{1}{(1 - a\,z)\,(1 - b\,z)}$$

```
> g := 1/(1-a*z);
```

$$g := \frac{1}{1 - a\,z}$$

```
> `rational/relate`(common, f, z, s(n), n, g, z);
```

$$s(n) - b\,s(n - 1)$$

```
> `rational/relate`(common, g, z, t(n), n, f, z);
```

$$\frac{a\,t(n)}{-b + a} + \frac{b^{n}\,b}{b - a}$$

```
> `rational/relate`(contains, f, z, s(n), n, g, z);
```

$$s(n) - b\,s(n - 1)$$

```
> g := 1/(1-b*z)/(1-c*z);
```

$$g := \frac{1}{(1 - b\,z)\,(1 - c\,z)}$$

```
> 'rational/relate'(common, f, z, s(n), n, g, z);
                                                          n
                          b s(n)     b a s(n - 1)    c c
                          ------- - ------------ + -----
                          - c + b      - c + b       c - b

> 'rational/relate'(denom, z*(1+z)/(1-z)^3, z, s(n), n);
                    - 17/8 s(n) + 5/2 s(n - 1) - 7/8 s(n - 2)

> 'rational/relate'(power, z/(1-z-z^2), z, F(n), n, 2);
                       (1/5 n - 1/5) F(n) + 2/5 F(n - 1) n

> 'rational/relate'(power, z/(1-z-z^2), z, F(n), n, 3);
                                          2
                 (- 3/50 n - 1/25 + 1/10 n ) F(n) - 3/25 F(n - 1) n

> f:=1/(1-2*z)/(1-3*z);
                                      1
                           f := --------------------
                                (1 - 2 z) (1 - 3 z)

> g:=1/(1-2*z)^3/(1-3*z);
                                        1
                          g := --------------------
                                              3
                               (1 - 2 z)  (1 - 3 z)

> 'rational/relate'(super, f, z, s(n), n, g, z);
                       2                          2
                 (1 - n  - 9 n) s(n) + (24 + 3 n  + 27 n) s(n - 1)
```

B.2.8. Procedure rational/simp

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# Simplify expressions involving named sequences from RAT_SEQ
#
# expr  expression to simplify
# Fz    generating function of sequence
# z     generating function variable
# F     sequence name
# n     index variable
#
# For order deg recurrence, simplfies to F(n-deg+1), ..., F(n)
# An optional parameter can specify a base, so that
# simplifies to F(base-deg+1), ..., F(base)
#
# A MAPLE procedure can be passed as a optional parameter
# Coefficient of F(base-i) is collected when done and the
```

```
# procedure is applied to each coefficient
#

`rational/simp` := proc(expr, Fz, z, F, n)
  local c, d, deg,
        recur, back_recur,    # recurrences
        i, base, s_proc;

  `rational/simp/walk` := `readlib(``rational/simp/walk``,
                          `` . my__lib__name . `/rational/simp/walk.m` )`;

#
# get optional parameters
#

  base := n;
  s_proc := NULL;
  for i from 6 to nargs do
    if type(args[i], procedure) then
      s_proc := args[i]
    else
      base := args[i]
    fi
  od;

#
# build recurrence equations for simplification
#

  denom(normal(Fz));
  expand(");
  simplify("/tcoeff(", z)/z^ldegree(", z));
  c := `polynom/coeffs`(", z, 'd');

  deg := degree( d[ nops([d]) ], z);

  recur := 0;
  back_recur := 0;

  for i from 2 to nops([d]) do
    recur := recur - c[i] * subs(n=n-degree(d[i], z), F);
  od;
  recur := F = recur;

  for i from 1 to nops([d]) -1 do
    back_recur := back_recur
                  - c[i]/c[ nops([d]) ]
                     * subs(n=n+deg-degree(d[i], z), F);
  od;
  back_recur := F = back_recur;

#
# Walk the expression, and simplify any terms not in
# F(base-deg+1), ..., F(base)
```

```
#
  `rational/simp/walk`(expr, op(0,F), n, recur, back_recur, deg, base);
  expand(");

#
# Collect coefficients of F(base-i)
#

  subs(n = base-'i', F)$'i'=0..(deg-1);
  if s_proc = NULL then
    collect("", ["], distributed)
  else
    collect("", ["], distributed, s_proc)
  fi
end:

save `rational/simp`, ``. my__lib__name . `/rational/simp.m`;

#
# Walk expression, find occurences of F()
#

`rational/simp/walk` := proc(expr, F, n, recur, back_recur, deg, base)

  `rational/simp/sub` := 'readlib(' `rational/simp/sub`',
                              `` . my__lib__name . `/rational/simp/sub.m` )';

  if not has(expr, F) then
    RETURN(expr)
  fi;

  if type(expr,function) and op(0,expr) = F then
    RETURN(`rational/simp/sub`(expr, n, recur, back_recur, deg, base));
  fi;

  map( proc(x, a1, a2, a3, a4, a5, a6)
                    `rational/simp/walk`(x,a1,a2,a3,a4,a5,a6) end,
       expr, F, n, recur, back_recur, deg, base)
end:

save `rational/simp/walk`, ``. my__lib__name . `/rational/simp/walk.m`;

#
# Apply recurrence to simplify occurences not in
# F(base-deg+1), ..., F(base)
#

`rational/simp/sub` := proc(f, n, recur, back_recur, deg, base)
  local c, i;

  c := op(f) - base;
  if type(c, integer) then
    if c > 0 then
```

```
      f;
      for i from c by -1 to 1 do
        subs( subs( n = base + i, recur), ");
        simplify(")
      od;
    elif c<= -deg then
      f;
      for i from c to -deg do
        subs( subs(n = base + i, back_recur), ");
        simplify(")
      od
    else
      f
    fi
  else
    f
  fi
end:

save `rational/simp/sub`, ``. my__lib__name . `/rational/simp/sub.m`;
```

Example

```
> `rational/simp`(F(n+1)+F(n)+F(n-1), 1/(1-z-z^2), z, F(n), n);
                                2 F(n) + 2 F(n - 1)

> `rational/simp`(", 1/(1-z-z^2), z, F(n), n, n+1);
                                  2 F(n + 1)

> `rational/simp`(", 1/(1-z-z^2), z, F(n), n, n-1);
                              4 F(n - 1) + 2 F(n - 2)
```

B.3. Summation Procedures

B.3.1. Procedure simple_sum

Source Code

```
############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
############################################################
#
#
# Evaluates the indefinite sum of Fn
#
# Fz  Generating function of summand
# z   Generating function variable
# Fn  Summand - ratseq expression or sequence name
# n   Index for solution sequence
#

simple_sum := proc(Fz, z, Fn, n)
  local m, term, nn, dd;
```

```
  normal(Fz);
  nn := numer(");
  dd := denom("");

  # find factors of (1-z) in denom of Fz

  m := degree( gcd( (1-z)^degree(dd, z), dd), z);

  # this must be a polynomial summand

  if m = degree(dd, z) then
    if type(Fn, function) then
      'rational/relate'( super, Fz, z, Fn, n, Fz/(1-z), z);
      'rational/simp'(", Fz, z, Fn, n)
    else
      'rational/expand'( Fz/(1-z), z, n)
    fi;
    RETURN(")
  fi;

  # non - polynomial, apply pfrac

  dd := expand( simplify( dd/(1-z)^m ) );

  convert(1/dd/(1-z), parfrac, z, true);

  # expand resulting generating functions

  if degree( gcd( denom(op(1,")), 1-z), z) > 0 then
    term := 'rational/expand'( nn*op(1,")/(1-z)^m, z, n)
            + 'polynom/termscale'( simplify( op(2,")*dd), z, Fn, n);

  else
    term := 'rational/expand'( nn*op(2,")/(1-z)^m, z, n)
            + 'polynom/termscale'( simplify( op(1,")*dd), z, Fn, n);
  fi;

  if type(Fn, function) then
    term := 'rational/simp'(term, Fz, z, Fn, n)
  fi;

  term
end;
```

Example

```
> simple_sum(z/(1-z-z^2), z, F(n), n);
                              - 1 + 2 F(n) + F(n - 1)

> 'rational/simp'(", z/(1-z-z^2), z, F(n), n, n+2);
                                   - 1 + F(n + 2)

> simple_sum(z*(1+z)/(1-z)^3, z, n^2, n);
```

```
          n (1/2 n + 1/2) (1/3 n + 2/3) + 1/2 (n - 1) n (1/3 n + 1/3)

> normal(");
                                    3         2
                                 1/3 n  + 1/2 n  + 1/6 n

> factor(");
                                 1/6 n (n + 1) (2 n + 1)

> simple_sum(z/(1-z)^2, z, n, n);
                                      n (1/2 n + 1/2)

> normal(");
                                       1/2 n (n + 1)
```

B.3.2. Procedure hybrid_sum

Source Code

```
############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
############################################################
#
#
# Evaluates hybrid sums and convolutions
#
# Conv      convolution gf -- for hybrid sum use 1/(1-z)
#           must be a diagonal gf of the product of variables
#           used in Gfs
# Gfs       list of gfs and sequences for the hybrid term
#           each item in list is a list [gf, z, fn]
#           where gf is generating function,
#                 z is generating function variable
#                 fn is sequence encoded by gf
#           each z must be distinct
# element   index for solution
#
#
# Optional args
#
#   cterms = [list of convolution terms]
#      list is same format as Gfs except variables need
#      not be distinct
#      used to pass list of factors defing the gf Conv
#   file = name
#   file
#      write intermediate results to file
#      if no name given write to terminal
#   rfile = name
#   rfile
#      write result to file
#      if no name given write to terminal
#   flags = [list]
#      list of flags to control intermediate results that are output
```

```
#       choices are
#         P  results of partial fractions
#         T  term resulting from expanding each generating function
#         W  print data used to expand final generating function
#         R  print final result
#

hybrid_sum := proc(Conv, Gfs, element)
  local term, temp, v, i, var, gf, F, cnst, c, conv, gfs,
        conv_terms, output, out_file, res_out, res_file, out_options;


#
# process optional args
#
  conv_terms := [];
  output := false;
  res_out := false;
  out_options := ['P', 'R'];

  gfs := Gfs;
  conv := Conv;

  for i from 4 to nargs do
    if type(args[i], '=') then
      args[i];
      op(1,");
      if " = 'cterms' then
        conv_terms := op(2,"")
      elif " = 'file' then
        out_file := op(2,"");
        output := true
      elif " = 'rfile' then
        res_file := op(2,"");
        res_out := true
      elif " = 'flags' then
        out_options := op(2,"")
      fi
    else
      args[i];
      if " = 'file' then
        output := true;
        out_file := 'terminal'
      elif " = 'rfile' then
        res_out := true;
        res_file := 'terminal'
      fi
    fi
  od;

#
# determine product of all gf variables
#
  v := 1;
```

```
  for i to nops(gfs) do
    v := v * op(2,op(i,gfs))
  od;

  if output then
    writeto(out_file)
  fi;

#
# process all but last gf
#

  while nops(gfs) > 1 do

    # get generating function and sequence it encodes

    op(1,gfs);
    gf := op(1,");
    var := op(2,"");
    F := op(3,"");

    # do partial fractions

    temp := convert(conv / denom(gf), parfrac, var, true);

    if output and has(out_options, 'P') then
      print('Partial fractions', temp)
    fi;

    # discard term that expands to 0

    temp := map(proc(f,v) if 'polynom/hasprod'(denom(f),v)
                            then 0 else f fi end,
                temp, denom(conv));
    temp := normal(temp);
    v:= v/var;

    # remaining term is a product
    # extract factors that involve the denom of the gf

    c:= 1/ map(proc(f,v) if 'polynom/hasprod'(f,v) then f else 1 fi end,
                          denom(temp), denom(gf) );
    temp := numer(temp) /
              map(proc(f,v) if 'polynom/hasprod'(f,v) then 1 else f fi end,
                          denom(temp), denom(gf) );

    #
    #  temp is product of factors not involving denom of gf
    #  the denom of temp is new convolution gf
    #  the num of temp is polynomial of v and var
    #  use it to termscale sequence F encoded by gf
    #

    cnst := denom(gf) / denom(temp);
```

```
  term := 'polynom/termscale'(numer(temp), var, F, element);

  # simplify if F is a function

  if type(F, function) then
    normal(term);
    term := 'rational/simp'(numer(''), gf, var, F, element)/denom('')
  fi;

  if output and has(out_options, 'T') then
    print('Term', term)
  fi;

  conv := term*c*cnst;
  gfs := [op(2..nops(gfs),gfs)]
od;

#
# one gf left
#

op(1,gfs);
gf := op(1,'');
var := op(2,"");
F := op(3,"""");

#
# apply partial fractions, isolate terms that make up denom of gf
#

temp := convert(conv / denom(gf), parfrac, var);

if output and has(out_options, 'P') then
  print('Partial fractions', temp)
fi;

c := map(proc(t,f,v) if has(gcd(denom(t),f),v) then 0 else t fi end,
               temp, denom(gf), var);
temp := normal(temp - c);

if type(c, '+') then
  c := map(proc(f,g) f*g end, c, numer(gf))
else
  c := c * numer(gf);
fi;

cnst := simplify(denom(gf) / denom(temp) );
'polynom/termscale'(numer(temp), var, F, element);
if type(F, function) then
  normal('');
  'rational/simp'(numer(''), gf, var, F, element)/denom('')
else
  ''
fi;
```

```
  term := " * cnst;

  if output and has(out_options, 'T') then
    print(`Term`, term)
  fi;

  #
  #  expand other terms in resulting gf
  #  envoke wrap up in attempt to relate it to
  #  input gf's or convolution gf's
  #

  if type(c, `+`) then
    term :=  term + map(wrap_up,c,var,[op(Gfs), op(conv_terms)],element,
                        output and has(out_options, 'W') )
  else
    term :=  term + wrap_up(c,var,[op(Gfs), op(conv_terms)],element,
                            output and has(out_options, 'W') )
  fi;

  if output and (has(out_options, 'R') or has(out_options, 'T')) then
    print(`Result`, term)
  fi;

  if res_out then
    writeto(res_file);
    print(`Result`, term)
  fi;
  writeto('terminal');

  term
end:

#
# uses `polynom/equate` to determine if gf can be related
# to any of the gf's in Gfs using the scale rule or the
# power rule, otherwise expand gf
#

wrap_up := proc(gf, var, Gfs, element, output)
  local i, g, v, f, res, t, temp;

  for i to nops(Gfs) do
    op(i,Gfs);
    g := op(1,");
    v := op(2,"");
    f := op(3,""");

    if `polynom/equate`(denom(normal(g)), v, denom(normal(gf)), var, 'res')
     then
      t := `rational/relate`('denom', g, v, f, element);
      t:= t/op(1,res) * op(2,res)^element;
      temp := `polynom/termscale`( numer(normal(gf)), var, t, element);
```

```
      if output then
        print(`Wrap-up`, gf, `expands to`, temp)
      fi;

      RETURN(temp)

    fi
  od;

  if output then
    print(`Wrap-up`, gf, `does not expand`)
  fi;

  ''Rat_expand''(gf,var,element)
end:
```

Example

```
> c4;
                                      1
                                   ---------
                                   1 - x y z

> gf4;
                 1          n        y                    z
          [[-------, x, 2 ], [--------, y, n], [----------, z, F(n)]]
            1 - 2 x                    2                   2
                                 (1 - y)           1 - z - z

> hybrid_sum(c4, gf4, n);
              n      n                        n      n
  1/5 (- 2 2  + 6 2  n) F(n) + 1/5 (- 2 2  + 2 2  n) F(n - 1)

                      n      n     (n - 1)       (n - 1)
       - 1/10 (- 2 2  + 2 2  n) 2         (1/4)

              n        (n - 1)
       + 1/5 2  n (1/2)

> map(simplify, ");
          (n + 1)      n                    (n + 1)     (n + 1)
 1/5 (- 2         + 6 2  n) F(n) + 1/5 (- 2         + 2         n) F(n - 1)

       + 2/5

> map(factor, ");
               (n + 1)      n                 (n + 1)
      1/5 (- 2         + 6 2  n) F(n) + 1/5 2         (n - 1) F(n - 1) + 2/5


>
> #
> # sum(i+j+k=n, i*2^j*k*F(k) )
> #
```

```
> c19;
                                    w z
                                ----------------
                                          2
                               (1 - w z)  (1 - 2 w z)

> gf19;
                          w                    z
                    [[--------, w, n], [----------, z, F(n)]]
                             2                    2
                       (1 - w)            1 - z - z

> hybrid_sum(c19, gf19, n);
                                                                    (n - 1)
       (- 13 n - 13) F(n) + (- 8 n - 8) F(n - 1) - (- 8 n - 40) 2

                    (n - 1)
            - 8 n 2        + 4 n - 12 + (n - 5) n - 2 n (1/2 n + 1/2)

> map(simplify, ");
                                                                (n - 1)
   (- 13 n - 13) F(n) + (- 8 n - 8) F(n - 1) - (- 8 n - 40) 2

                n
        - 4 n 2  + 4 n - 12 + (n - 5) n - n (n + 1)

> map(factor, ");
                                                          (n - 1)         n
  - 13 (n + 1) F(n) - 8 (n + 1) F(n - 1) + 8 (n + 5) 2         - 4 n 2

     + 4 n - 12 + (n - 5) n - n (n + 1)

> frontend( collect, [", [F(n), F(n-1), 2^n], distributed, simplify]);
                                                                    n
      - 13 F(n) n - 13 F(n) - 8 F(n - 1) n - 8 F(n - 1) + 20 2  - 2 n - 12
```

B.3.3. Procedure convolve

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# Extended closed form expressions for convolutions on RAT_SEQ
#
# Gfs  list of gfs, each item is list [gf, p, fn]
#      where gf is the gf in variable var
#            p is the power of that factor
#              (number of times fn is factor in the convolution)
#            fn function encoded by gf
#
# var     generating function variable
```

```
# element index for solution
#
# optional arg
#   debug   turns on debug messages
#           useful trace of execution
#

convolve := proc(Gfs, var, element)
  local i, j, items, gf_num, gf_den, fn, gf, deg, f, prod_den, prod_num,
        list_sub, list_sup, gf_sub, gf_sup, d, term, debug_it;


  debug_it := false;
  if nargs > 3 then
    if args[4]='debug' then
      debug_it := true
    fi
  fi;

  items := nops(Gfs);
  gf_num := array(1..items);
  gf_den := array(1..items);
  fn := array(1..items);
  gf_sub := array(1..items);
  gf_sup := array(1..items);
  prod_num := 1;
  prod_den := 1;

  for i to items do

    # get gf

    op(i, Gfs);
    gf := op(1,");
    deg := op(2,"");
    f := op(3,""");

    # build gf of solution
    # build lists of numer, denom for later use

    normal(gf^deg);
    gf_num[i] := numer(");
    prod_num := prod_num * ";
    gf_den[i] := denom(""");
    prod_den := prod_den * ";

    # store sequence encoded by gf

    fn[i] := `rational/relate`(power, gf, var, f, element, deg);

    gf_sub[i] := 0;
    gf_sup[i] := 0
  od;
```

```
#
# Do a very large partial fractions expansion
#

prod_den := convert(1/prod_den, parfrac, var);

if not type(", '+') then

  # if single term, find expansion

  op(1, Gfs);
  gf := op(1,");
  deg := op(2,"");
  f := op(3,"""");

  'rational/relate'(super, gf, var, f, element,
                              normal(prod_num*prod_den), var);

  normal(");
  if type(f, function) then
    'rational/simp'(numer("), gf, var, f, element, normal)
      / factor(denom("))
  else
    simplify(");
    if type(f, polynom, element) then
      normal(");
      collect(numer("), element) / factor(denom("))
    else
      factor(");
    fi
  fi;
  RETURN(")
fi;

#
# Relate each term in parfrac expansion to an input gf
# Sum up terms in arrays gf_sub[] and gf_sup[]
#

for i to nops( prod_den ) do
  gf := op(i, prod_den);
  list_sub := [];
  list_sup := [];

  d := denom(gf);
  for j to items do
    deg := degree( gcd(d, gf_den[j]), var);
    if deg > 0 then
      if deg = degree(d, var) then
        # d is factor of denom
        list_sub := [op(list_sub), j]
      else
        # d has common factor with denom
        # BUT has higher degree that the factor in denom
```

```
      list_sup := [op(list_sup), j]
    fi
  fi
od;

if debug_it then
  print(i, list_sub, list_sup)
fi;

#
# add d to entries in gf_sub[] or gf_sup[]
# corresponding to the gf with which d has common factors
#

if nops(list_sub) > 0 then
  for j to nops(list_sub) do
    gf_sub[op(j,list_sub)] := gf_sub[op(j,list_sub)] + gf/nops(list_sub)
  od
else
  for j to nops(list_sup) do
    gf_sup[op(j,list_sup)] := gf_sup[op(j,list_sup)] + gf/nops(list_sup)
  od
fi
od;

if debug_it then
  for i to items do
    print (gf_sub[i], gf_sup[i])
  od
fi;

#
# Expand the gf's stored in gf_sub[] and gf_sup[]
# by relating them to the sequences encoded by the input gf's
#

term := 0;
for i to items do
  op(i, Gfs);
  gf := op(1,");
  deg := op(2,"");
  f := op(3,""");

  normal(gf_sub[i]*prod_num*gf_den[i]/gf_num[i]);
  if gf_sup[i] <> 0 then
    # have factors in denom with higher powers than in gf
    `rational/relate`(super, gf, var, f, element,
                      normal(gf_sup[i]*prod_num + " ), var)
  else
    # have polynomial times gf
    `polynom/termscale`(", var, fn[i], element)
  fi;
  normal(");
  if type(f, function) then
```

```
      `rational/simp`(numer("), gf, var, f, element, normal)
          / factor(denom("))
    else
      simplify(");
      if type(f, polynom, element) then
        normal(");
        collect(numer("), element) / factor(denom("))
      else
        factor(")
      fi
    fi;
    term := term + ";

  od;

  #
  # Return the result
  #

  term

end:
```

Example

```
> read data;
                      z                          z                    1          n
       gf := [[----------, 1, F(n)], [--------, 1, n], [-------, 1, a ]]
                      2                          2           1 - a z
              1 - z - z                 (1 - z)

> convolve(", z, n);
        (- 5 a - 3) F(n) + (- 3 a - 2) F(n - 1)    (- 1 + a) n + 3 a - 2
        ---------------------------------------- + --------------------
                                  2                                  2
                        - a - 1 + a                        (- 1 + a)

                           (2 + n)
                          a
              + -------------------------
                                2             2
                (- 1 + a)  (- a - 1 + a )

> #
> # convolve F(i) F(j) k for i+j+k=n
> #
> [[z/(1-z-z^2), 2, F(n)], [z/(1-z)^2, 1, n]];
                          z                          z
                  [[----------, 2, F(n)], [--------, 1, n]]
                             2                          2
                    1 - z - z                 (1 - z)

> convolve(", z, n);
              1/5 (7 n - 32) F(n) + 1/5 (4 n - 20) F(n - 1) + n + 4
```

B.4. Harmonic Number Procedures

B.4.1. Procedure harmonic/plog

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
# Expands the multivariate poly-log generating function
#
#            1
#  ------------------------   prod(i=1..c, ln(1/(1-z.i))
#  (1-prod(i=1..c, z.i))^m
#
# c  number of generating fucntion variables
# m  degree of convolution
# n  index variable for solution
#

`harmonic/plog` := proc(c,m,n)
  local fn, sn, i, C;

  # procedure to compute a binomial coefficient
  C:= proc(n,k) local i; product((n+1-i)/i, i=1..k) end;

  if c=1 then
    # have a harmonic number
    fn := 'H'(m,n)
  elif m=1 then
    # have 1/k
    sn := `harmonic/plog`(c-1, m, n);
    fn := ( (n+1)*subs(n=n+1, sn) - C(c+n-1, c-1) ) / (c-1);
  else
    # have 1/k^m
    sn := n* `harmonic/plog`(c-1, m, n);
    fn := sn - `harmonic/plog`(c-1, m-1, n);
    fn := subs(n=n+1, fn)/(c-1)
  fi;

  expand(fn);

  collect(", ['H'(m+1-'i', n+c-1) $ 'i'=1..m]);
  map(factor,")
end:
```

Example

```
> `harmonic/plog`(1, 1, n);
                                        H(1, n)

> `harmonic/plog`(1, 2, n);
```

```
                                    H(2, n)

> `harmonic/plog`(2, 1, n);
                          (n + 1) H(1, n + 1) - n - 1

> `harmonic/plog`(3, 1, n);
                                                                  2
            1/2 (n + 2) (n + 1) H(1, n + 2) - 3/2 - 3/4 n  - 9/4 n

> #
> # the sum of H(m,n)
> #
> `harmonic/plog`(2, m, n);
                 H(m, n + 1) n + H(m, n + 1) - H(m - 1, n + 1)

> `harmonic/plog`(3, 3, n);
     1/2 (n + 2) (n + 1) H(3, n + 2) - 1/2 (3 + 2 n) H(2, n + 2)

          + 1/2 H(1, n + 2)
```

B.4.2. Procedure harmonic/simp

Source Code

```
#############################################################
# (C) Copyright 1991 by Robert Andrews Ravenscroft, Jr. #
#############################################################
#
#
#  Simplifies expression involving harmonic numbers
#  Expects ALL harmonic numbers to be expressed as H(m,n)
#  where, H(m,n) = sum(i=1..n, 1/i^m),
#  Thus, H(n) should be written as H(1,n)
#
#
#  expr   expression to simplify
#  n      index to simplify all Harmonic numbers relative to
#

`harmonic/simp` := proc(expr, n)
  local v, h_list, m_list, m, i, res;

  # find all harmonic numbers in expression
  # h_list is actually a set to eliminate multiple occurances
  h_list := `harmonic/simp/walk`(expr);

  res := expr;
  m_list := [];
  # for each harmonic, express it relative to H(m,n)
  for i in h_list do
    v := op(2,i) - n;
    if type(v,integer) then
      m := op(1,i);
      if not member(m, m_list) then
```

```
        m_list := [op(m_list), m]
      fi;
      if v < 0 then
        0;
        for j to -v do
          " + 1/(n+1-j)^m
        od;
        res := subs(i=H(m,n)-", res)
      elif v > 0 then
        0;
        for j to v do
          " + 1/(n+j)^m
        od;
        res := subs(i=H(m,n)+", res)
      fi

    fi
  od;

  map( proc(m,n) 'H'(m,n) end, m_list, n);
  collect(res, ", distributed, simplify)
end;

`harmonic/simp/walk` := proc(expr)
  local i;

  if not has(expr, 'H') then
    {}
  elif type(expr, function) and op(0,expr)='H' then
    {expr}
  else
    {};
    for i to nops(expr) do
      { op(''), op( `harmonic/simp/walk`(op(i,expr)) ) }
    od
  fi
end;
```

Example

```
> `harmonic/simp`(H(1,n), n+1);
                                          1
                              H(1, n + 1) - -----
                                        n + 1

> `harmonic/simp`(H(1,n), n-1);
                               H(1, n - 1) + 1/n

> `harmonic/simp`('', n);
                                    H(1, n)

> `harmonic/plog`(3, 3, n);
    1/2 (n + 2) (n + 1) H(3, n + 2) - 1/2 (3 + 2 n) H(2, n + 2)
```

```
        + 1/2 H(1, n + 2)

> `harmonic/simp`(", n);
        1/2 H(1, n) + 1/2 (n + 2) (n + 1) H(3, n) + (- 3/2 - n) H(2, n)
```

Bibliography

[BRONSTEIN] Manuel Bronstein, "Integration of Elementary Functions," Ph. D. Thesis, Department of Mathematics, Univeristy of California, Berkeley, 1987.

[BURTON] David M. Burton, *Introduction to Modern Abstract Algebra*, Addison-Wesley, 1967.

[CHAR] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, and Stephen M. Watt, *Maple Reference Manual*, Fifth Edition, Waterloo Maple Publishing, 1990.

[COHEN] Jacques Cohen and Joel Katcoff, "Symbolic Solution of Finite-Difference Equations," *ACM Transactions on Mathematical Software*, Vol. 3, No. 3, pp. 261–271, 1977.

[DESAI] Rajesh Desai, "Analysis and Implementation of Karr's Algorithms," Master's Thesis, Department of Computer Science, University of Rhode Island, 1988.

[FLAJOLET] Phillipe Flajolet, Bruno Salvy and Paul Zimmermann, "Lambda-Upsilon-Omega: An Assistant Algorithms Analyzer," *Proceedings of the 6th International Conference on Applied Algebra, Algebraic Algorithms and Error Correcting Code*, Rome, July 1988.

[GRAHAM] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics*, Addison-Wesley, 1989.

[GOSPER 77] R. William Gosper, Jr., "Indefinite Hypergeometric Sums in MACSYMA," *Proceedings of the MACSYMA User's Conference*, Berkeley, CA., pp. 237–251, 1977.

[GOSPER 78] R. William Gosper, Jr., "Decision Procedure for Indefinite Hypergeometric Summation," *Proceedings of the National Academy of Sciences USA*, Vol. 75, No. 1, pp. 40–42, 1978.

[GREENE] Daniel H. Greene and Donald E. Knuth, *Mathematics for the Analysis of Algorithms*, Second Edition, Birkhäuser, 1982.

[HAYDEN 86] Michael B. Hayden and Edmund A. Lamagna, "Summation of Binomial Coefficients Using Hypergeometric Functions," *Proceedings of the ACM Symposium on Symbolic and Algebraic Computation*, B. W. Char (ed.), ACM Press, pp. 77–81, 1986.

[HAYDEN 88] Michael B. Hayden and Edmund A. Lamagna, "Gosper's Summation Algorithm and Some Applications," unpublished, 1988.

[KARR] M. Karr, "Summation in Finite Terms," *Journal of the ACM*, Vol. 28, No. 2, pp. 305-350, 1981.

[KNUTH] Donald E. Knuth, *The Art of Computer Programming*, three vols., Addison-Wesley, 1968, 1969, 1972.

[LIU] C. L. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.

[MOENCK] Robert Moenck, "On Computing Closed Forms for Summation," *Proceedings of the MACSYMA User's Conference*, Berkeley, CA, pp. 225–236, 1977.

[PURDOM] Paul Walton Purdom, Jr. and Cynthia A. Brown, *The Analysis of Algorithms*, Holt, Rinehart and Winston, 1985.

[RAVENSCROFT] Robert A. Ravenscroft, Jr. and Edmund A. Lamagna, "Symbolic Summation with Generating Functions," *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, ACM Press, 1989.

[RISCH] R. H. Risch, "The Problem of Integration in Finite Terms," *Trans A. M. S.*, Vol. 139, pp. 167–189, 1969.

[RUSSELL 79] David L. Russell, "Sums of Recurrences of Terms from Linear Recurrence Sequences," *Discrete Mathematics*, Vol. 28, No. 1, pp. 65–79, 1979.

[RUSSELL 81] David L. Russell, "Summation of Second-Order Recuurence Terms and Their Squares," *Fibonacci Quarterly*, Vol. 19, No. 4, pp. 336–340, 1981.

[RUSSELL 82] David L. Russell, "Notes on Sums of Products of Generalized Fibonacci Numbers," *Fibonacci Quarterly*, Vol. 20, No. 2, pp. 114–117, 1982.

[SAVIO] Dominic Y. Savio, Edmund A. Lamagna and Shing-Min Liu, "Summation of Harmonic Numbers," *Proceedings of the Computers & Mathematics 1989 Conference*, E. Kaltofin and S. M. Watt (ed.), Springer-Verlag, 1989.

[SKIENA] Steven S. Skiena, *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Addison-Wesley, 1990.

[SPIEGEL] Murray R. Spiegel, *Complex Variables*, Schaum's Outline Series, McGraw-Hill, 1964.

[TRAGER] B. M. Trager, "On the Integration of Algebraic Functions," Ph. D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1985.

[WILF] Herbert S. Wilf, *generatingfunctionology*, Academic Press, Inc., 1990.