Computer Algebra
and
Algebraic Programming

Computer Algebra
and
Algebraic Programming

Robert Smith

Quad's Printing — St. Paul

Contents

Contents v

I The Algebraic Paradigm 1

1 The Symbolic and Algebraic Paradigm 3
1.1 Introduction 3

2 Structure of Algebraic Programs 7
2.1 Symbols, Variables, and Values 7
2.2 Expressions 7
2.3 Mappings and Constructions 8
2.4 Substitution Semantics 8

3 Values and Types 11
3.1 A Discourse on Types 11

II Computer Algebra Fundamentals 15

4 Relations, Identities, Forms 17
4.1 Relationships Between Values 17
4.2 Equality in Computer Algebra 18
4.3 Canonical Forms 19
4.4 Normal Forms 20

5 Arbitrary Precision Arithmetic 23
5.1 Introduction 23
5.2 Computer Representation 24
5.3 High-Precision Evaluation of Hypergeometric Series 25

IIIAbstract Algebra 29

6 Basic Structures and Divisibility 31

6.1 Algebraic Structures 31
6.2 Divisibility and Factorization 32

7 Polynomial, Rational, and Series Structures 37
7.1 Univariate Polynomial Domains 37

8 Field Extensions 39
8.1 Algebraic and Transcendental Extensions 40

IV Polynomials 43

9 Polynomial Algorithms 45
9.1 Sylvester Matrices and Resultants 45
9.2 Polynomial Roots and Algebraic Numbers 47

10 Algebra In the Reals 49
10.1 Introduction 49
10.2 Real Closed Fields 49

V Series 51

11 Hypergeometric Series 53
11.1 Hypergeometric Series and Functions 53
11.2 Series as Hypergeometric Functions 55
11.3 Discrete Operators and The Summation Problem 55
11.4 Gosper's Algorithm for Indefinite Summation 57
11.5 Zeilberger's Algorithm for Definite Summation 60

VI Integration 63

12 Risch Integration 65
12.1 Preliminaries 65
12.2 Risch Algorithm: Input 66

Appendix 71

A Function Tables 71
A.1 Elementary Functions 71
A.2 Gamma and Friends 73
A.3 Hypergeometric Identities 75
A.4 Miscellaneous 75

B Miscellaneous 77

B.1 The Gudermannian Function 77
B.2 The Γ-Function 79
B.3 The Lambert W-Function 82

Bibliography **83**

Index **85**

Part I

The Algebraic Paradigm

Chapter 1

The Symbolic and Algebraic Paradigm

1.1 Introduction

The Symbolic Paradigm in Languages

The term “symbolic” is thrown around quite often in the programming world. For example, in the very meaning of the name BASIC, “Beginners All-purpose Symbolic Instruction Code”. What does “symbolic” mean here anyway? BASIC allowed symbolic names of variables, and they could be referred to by their symbolic name, instead of, say, an address in memory. BASIC code is indeed “instruction code”, but it isn’t a native code a computer can immediately understand. Essentially every mainstream programming language is “symbolic” by the definition put forth by BASIC.

Another language brought another definition of the word “symbolic”—Lisp. Lisp introduced symbols as essentially first-class citizens. As such, they could be used as arguments, manipulated, generated, returned, etc. With symbols as first-class citizens and the uniform syntactic and semantic structure of programs, it was attractive to those doing symbolic mathematics, as opposed to numerical mathematics.

The contrast between symbolic and numerical mathematics illustrate what is meant by “symbolic”. Consider computing the derivative of a function $f\colon \mathbb{R} \to \mathbb{R}$ at $x = x_0$. There are essentially two ways of doing this.

Numerical We can take the definition of the derivative

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

and make approximations with very small values of δ. The following C program illustrates how one might do it.

```
float diff(float(*f)(float), float x0, float dx)
{
    float df;
    df = f(x0 + dx) - f(x0);
    return df/dx;
}
```

With this function, we can compute the derivative of `f` at `x0` with finite precision. With more sophistication, we could use a multiple-precision arithmetic library to get even more precision, possibly only limited by the amount of available memory.

Symbolic Instead of using the definition of the derivative in a straightforward way, we can instead view the derivative as a function defined algebraically. For brevity, we could define the derivative as a function D_x with the following properties:

- $D_x(k) = 0$
- $D_x[f(x) + g(x)] = D_x[f(x)] + D_x[g(x)]$
- $D_x(x) = 1$
- $D_x[f(x)g(x)] = D_x[f(x)]g(x) + f(x)D_x[g(x)]$

Since Lisp has first-class symbols, this could be quite simply done using only the basic language features.

```
(defun diff (E X)
  ;; Given an expression f(X) = E, compute f'.
  (cond ((numberp E) 0)                          ; k' = 0
        ((symbolp E) (if (eqp E X) 1 0))         ; x' = 1
        ((eqp 'PLUS (car E))                     ; (f + g)' = f' + g'
         (list 'PLUS (diff (cadr E) X)
                     (diff (caddr E) X)))
        ((eqp 'TIMES (car E))                    ; (fg)' = f'g + fg'
         (list 'PLUS (list 'TIMES
                           (diff (cadr E) X)
                           (caddr E))
                     (list 'TIMES
                           (diff (caddr E) X)
                           (cadr E))))
        (t (list 'diff E X))))
```

Here one can provide a symbolic expression, for example $x^2 - x$, which written in code is `(PLUS (TIMES X X) (TIMES -1 X))`, and pass it to `DIFF` with respect to `X` to obtain an S-expression equivalent to $(1 \cdot x + x \cdot 1) + (0 \cdot x + (-1) \cdot 1)$. After simplification, this leads to $2x - 1$, the correct result. Furthermore, one can subsitutute a value in for `X` to get the derivative at that value *exactly*, as opposed to a finite-precision approximation.

In summary, numerical computation deals with numbers and approximation of (typically) real-valued functions, while symbolic computation works by rewriting structures (lists in the case of Lisp) in a mathematically defined way.

Of course, in the Lisp symbolic differentiation example, it is easy to see why it is not the most ideal abstraction.

Pattern-Matching to Make Programs Simpler

In reality, the Lisp differentiation program is an *ad hoc* form of pattern matching. Pattern matching made a more dominating appearance in the ML family of languages. Typically, one has a function defined by a set of patterns and their respective form of reduction. There are a few caveats, however. First, patterns must usually be *unconditional*. This means that the pattern must be able to match an expression without any form of conditional checking. Usually this is "remedied" by introducing a conditional check in the reduced expression. For example

```
(* returns a complex number represented by (re, im) *)
fun geomean 0 y = (0, 0)
  | geomean x 0 = (0, 0)
  | geomean x y = if x>0 and y>0
                  then (sqrt x*y, 0)
                  else if   (x<0 and y>0) or (x>0 and y<0)
                       then (0, sqrt (abs x)*(abs y))
                       else (sqrt x*y, 0)
```

The conditions on `x` and `y` make even a C program easier to analyze and follow and sort of ruins the point and flow of pattern-matching. Ideally, some sort of conditions could be imposed in the pattern itself, as in the *ad hoc* Lisp example (checking if the argument is a number or symbol, etc.).

The second caveat is the lack of ability to clearly express *non-linear patterns*. A non-linear pattern, simply, is a pattern with two or more variables that are the same. In the last example, if we had `geomean x x` to denote "match the arguments of `geomean` iff they are equal", then the pattern would be non-linear. This is not possible to do in most pattern-matching languages. The work-around is imposing a condition such as in the following code.

```
(* non-linear pattern via conditions *)
fun add x y = if   x = y
              then 2*x
              else x + y
```

Term Rewriting and Reduction

An algebraic program consists of a set of rules defined by the programmer, with which the core computation engine, or **kernel**, can use to transform expressions. This notion of transformation of expressions is known as **term rewriting**.

Term rewriting done by a human is quite trivial. We have the ability to make decisions either concretely or based on intuition on what rules to apply to an expression and when. For a machine, however, it is a little more complicated.

The principle problems with term rewriting are (1) when to apply rules, (2) when to stop applying rules, and (3) which rules to apply.

Types and Domains

Obviously algebraic and symbolic computation correlate strongly to concepts in mathematics, especially abstract algebra. And mathematics is generally built upon the notion of *sets*, and operations are defined as mappings between the sets, even from a set to itself. It therefore makes sense for an algebraic language to have the ability to declare the domain in which a variable is and over which computations can take place.

Haskell, though not an algebraic language, covers this area well. It employs Milner type inference, and since the language is "purely functional", all operations are type safe and bound statically (typically at compile time).

Mathematica, which does indeed have algebraic capabilities, covers this area inadequately. Patterns in the language can include certain type restrictions. For example, `x_Integer` is a pattern that will only match integral quantities. This is dynamically checked, and no advantages can be taken from it from the perspective of computational time or memory. Moreover, this style of declaration is "shallow" and slightly inaccurate in fact. Using the variable `x_Rational` will *not* match an integral quantity. Mathematica, however, does allow assumptions to be made on variables. For example, `Assume[Element[x,Integers]]` will internally declare that `x` is or must be an integer. However, it is not known whether this is any different from the aforementioned dynamic checking.

Other computer algebra systems have included types or domains. Most notably, the Axiom computer algebra system is based entirely on types and domains. Like Haskell, every computation in Axiom is type safe and well defined. Axiom code is also compiled to efficient machine code, much of which is due to the ability to check types statically. Less notably, MuPAD has included domains and categories as first-class values. However, the language is interpreted and therefore one would assume little optimization could be gained from such declarations (however, type soundness and other advantages come with the ability to declare types).

A major drawback in having type systems in mathematical or algebraic languages is that types can become unwieldy. For example, it is not immediately clear without extra computations what the type of `5+2` might be. It could be seen as an integer, rational, real, polynomial, element of an additive ring, etc. Therefore, much care must be taken into the design of a type system when dealing with, especially, mathematics and symbolic quantities. There must be a balance between programmability and type consciousness.

Chapter 2

Structure of Algebraic Programs

2.1 Symbols, Variables, and Values

We begin by defining some terms. A **domain** is a mathematical set, usually consisting of related objects. Elements of a domain are called **values**. Domains correspond closely to the notion of a mathematical domain of a mathematical function.

A **symbol** is an identifier, possibly identifying itself, in which case the symbol is **nullary**. Essentially any kind of identifier is a symbol. For example, ‘x’, ‘$+$’, ‘385’, and ‘Δx’ are all symbols. A symbol that represents an unknown value of a particular domain is a **variable**. A symbol that represents a *single*, *particular* value in a domain is a **constant**. The term ‘variable’ is a sort of misnomer; a variable does not change what it represents. As such, variables are sometimes used to universally quantify a relationship—that is, declare truth for a statement for *every* value in a domain.

2.2 Expressions

Symbols are the words of an algebraic language, and expressions are the sentences. There is only a single way to construct expressions: to list them. LISP terms these as *symbolic expressions* or *S-expressions.* Actually, LISP specifies how such expressions are constructed—via cons pairs.

An algebraic language purposefully omits the necessity that a list must be recursively built up by nested pairs. Instead, expressions are constructed via two parts, the stem and the leaves. The **leaves** collectively is a sequence of expressions all of which are related by the **stem**, which specifies how the leaves are related to each other.

Although the leaves are often best represented with a list, they are not lists in the traditional sense. Suppose we denote a sequence of leaves a, b, and c with a stem f via $f\langle a, b, c\rangle$. Although syntactically the stem is written in a serial fashion, there is *no* implication that a is any way “before” b. The relationship between the leaves is purely established by the stem.

A sequence of leaves which is empty is called a *null sequence* and is represented simply by $\langle\rangle$.

Example 2.1 Suppose we have the expression $+\langle 3, 9, 5\rangle$ which represents the sum of 3, 9, and 5. There is no ordering on the elements and we can obtain a perfectly equivalent expression by transposing any of the leaves.

Now consider the expression $\mathrm{tuple}\langle 4, 9, 3\rangle$. If this represents a triple of integers 4, 9, and 3, then clearly transposing any two leaves would *not* give an equivalent expression.

Remark. In general, we will assume that the leaves are ordered and any other expression is not equal to it, unless otherwise stated or implied.

2.3 Mappings and Constructions

Mathematically, a *function* is an object which assigns an output for a set of defined inputs. We can also think of a function not as an object that allows one to travel from a given object to another, but as an object that *transforms* or *projects* an object into another realm entirely. Suppose we have a function ℓ which takes a number between 1 and 26, and outputs the corresponding letter in the English alphabet. So for example, $\ell(6) = \texttt{f}$. Then we can think of ℓ as a way to transform a given number into a letter. The *inverse* ℓ^{-1} reverses this operation, so we can give a letter and receive a number, as with $\ell^{-1}(\texttt{f}) = 6$.

An expression which acts as a transformation or function application is called a **mapping**, and the stem of that expression is called the **map**. We will intentionally defer the explanation of how precisely maps are defined until later.

A closely related concept to mappings are constructions. Constructions are essentially equivalent to mappings from a purely mathematical view, but from a programmatic standpoint, constructions are used to *construct*, not to *transform*. More specifically, **constructions** allow for the creation of new objects with special properties. The head of an expression which is a construction is a **constructor**.

Example 2.2 Let Cplx be a constructor on two leaves such that $\mathrm{Cplx}\langle a, b\rangle$ represents the complex number $a + b\mathrm{i}$. Then $\mathrm{Cplx}\langle 2, 3\rangle$ is a literal construction of the complex number and is an object itself.

Let List be a constructor on any number of leaves constructing an ordered list of expressions. We can represent the empty list nil as $\mathrm{nil} = \mathrm{List}\langle\rangle$.

2.4 Substitution Semantics

One of the general ideas of an algebraic language is the notion of substitution. First, let us re-think of the way we view expressions.

It should be obvious that the structure of an algebraic program is a tree. For example, we could represent the expression $\text{List}\langle x, f\langle x, y\rangle, 2\rangle$ as the tree in figure 2.1.

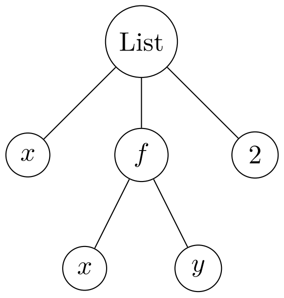

Figure 2.1

Notation 2.1 A **substitution** of x by y in the expression E is denoted $(x \mapsto y)\ E$.

Substitution is certainly nothing new; the lambda calculus is essentially based upon the idea of simple substitutions. Aside from a few technicalities (such as variable capture), we can restate a lambda abstraction $(\lambda x.E)\ y$ as $(x \mapsto y)\ E$.

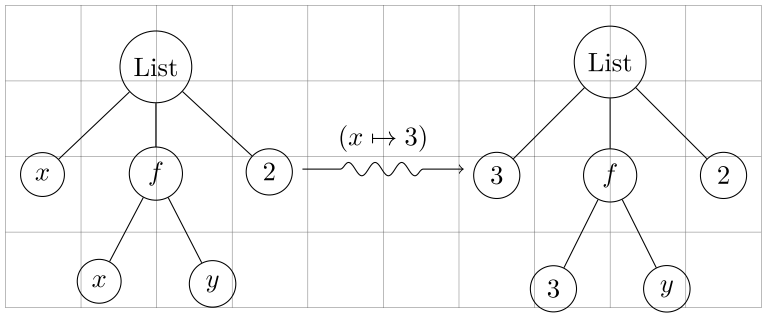

Figure 2.2

Chapter 3

Values and Types

3.1 A Discourse on Types

Informally, a type denotes a class of values that perhaps obey certain properties. In programming, there are many kinds of type systems. The C programming language, for example, uses statically bound types but is unsafe—there can be many implicit type casts which allow undeclared properties to be true at runtime. On the other hand, the language Haskell has a rich type system, that is statically determined and completely safe; no unspecified operations can occur at runtime. In the sequel, we will describe a type system that favors static binding and expressiveness.

Type Assertion and Base Types

Types are essentially equivalent to a domain. A slight distinction is made, however. A type can be thought of as the *name* of a set, and the domain is the *description* of the set. With such a distinction, one says, for example, "x is *of type* $\mathsf{Integer}$," and "x is *in the domain* $\mathbb{Z}$." Symbolically, one writes $x : \mathsf{Integer}$ and $x \in \mathbb{Z}$ respectively. As such, we adopt the following base[1] definition for a type.

Definition 3.1 The set of *all* types is denoted $\mathcal{T}$. There exist a set of **base types** $\mathcal{T}_0 \subset \mathcal{T}$, and is defined as a finite set of symbols for each of which there exists a corresponding domain.

This definition suggests there exist a set of "predetermined" types for a type system. So, for example, in C, one might say that $\mathcal{T}_0 = \{\texttt{int}, \texttt{float}, \texttt{char}, \ldots\}$.

The entirety of $\mathcal{T}$ can now be defined inductively, which we will do in steps.

[1]N.B., this definition is not a complete definition!

Product Types

Product types enable one to form well-typed tuple-like data structures. For example, we can define n-tuples as the product of n types equipped with "accessor functions" to retrieve the value of any one of the n elements.

Definition 3.2 Given τ_1 and $\tau_2 \in \mathcal{T}$, the **product type** is the type $\tau_1 \times \tau_2 \in \mathcal{T}$ whose domain is the Cartesian product of the domains of τ_1 and τ_2. The **type product operator** '$\times$' is associative.

Functions

The nomenclature of the term "functional paradigm" is exactly a result of the existence of function types. As one might expect, a function that maps values from one type to another (possibly the same). One important aspect of a function is that (1) functions always have *one* argument and (2) functions always have *one* result. This seems contrary to most programming paradigms and perhaps even mathematics itself. But it is in fact not. Consider the following C function.

```c
float expt(float x, int n)
{
    int i;
    float res = x;
    for (i = 1; i < n; i++)
        res *= x;
    return res;
}
```

It seems as if `expt` takes two arguments, and that is indeed fair to say in the context of C programming. However, one can also see it as taking a *single* argument whose type is `float` $\times$ `int`. In other words, `expt` is a function that takes a *tuple* as an argument, and returns the float. We say the type of `expt` is `float` $\times$ `int` $\to$ `float`.

Formalizing this, we have the following definition.

Definition 3.3 Given τ_1 and $\tau_2 \in \mathcal{T}$, the type $\tau_1 \times \tau_2$ is a **function type**, written $\tau_1 \to \tau_2 \in \mathcal{T}$, if for some value $f : \tau_1 \to \tau_2$ and value $x : \tau_1$, there exists a function apply such that $\text{apply}(f, x)$ is of type τ_2. The **type function operator** '$\to$' associates to the right—that is $x \to y \to z \equiv x \to (y \to z)$.

Tuples

Definition 3.4 An n**-tuple** is a product of n types τ_1, τ_2, $\ldots$, τ_n with n corresponding **accessor functions** $f_1, \ldots, f_n$ such that given an n-tuple $X = (x_1, \ldots, x_n)$, $f_k(X) = x_k$ for $1 \le k \le n$.

Sum Types

Often it is desirable to construct a new type that is a union of already known types. This gives one the ability to allow functions to accept several specific types. This “or”-like type operation creates a *sum type*. However, this is insufficient. Consider a function `toNumber` : $\sigma \to \mathsf{Integer}$ where σ is the union of Real and String, and is defined such that if the argument is a string, then it returns the value of the function `stringToInt`, and if the argument is a real number, then it returns the floor of that argument. Clearly in the implementation of such a function, it must be necessary to dispatch to the right operations depending on the type. Therefore, it is necessary to declare **type tags**, which allow access to either of the data in the type summands. In other words, the union of types must be paired with identifiers to allow their query. In Standard ML, one writes:

```
datatype realstring = RealVal of real
                    | StrVal  of string
```

As such, if `5.4` is of type `real`, then `RealVal 5.4` is of type `realstring`. This allows the definition of `toNumber` to be defined like so (in Standard ML):

```
fun toNumber x =
  case x
   of RealVal r = floor r
    | StrVal s  = stringToInt s
```

We can now concretely define a sum type.

Definition 3.5 Given types τ_1 and $\tau_2 \in \mathcal{T}$, a **sum type** is the union $\tau_1 + \tau_2 \in \mathcal{T}$ such that there exist unique bijective functions $s_1 : \tau_1 \to \tau_1 + \tau_2$ and $s_2 : \tau_2 \to \tau_1 + \tau_2$.

Type Classes and Type Functions

Definition 3.6 A **type constructor** is a type function $\phi : \mathcal{T} \to \mathcal{T}$ such that for $\tau \in \mathcal{T}$, $\phi(\tau)$ is a new type.

Type constructors are especially useful for containers for data. Probably the most often used container in programming is the list. A **list** is defined by

$$\forall \alpha \in \mathcal{T}, \quad \mathsf{List}(\alpha) = \mathrm{Nil} + \mathrm{Cons}(\alpha \times \mathsf{List}(\alpha)).$$

This expresses the notion of a **recursive type**, as well as declaring a nullary as a constructor for a type.

Subtypes

We can equip the type algebra with the notion of subtypes.

Definition 3.7 Consider σ and $\tau \in \mathcal{T}$ with associated domains S and T respectively. Then τ is a **subtype** of σ iff $T \subset S$ and for all $f : \sigma \to \alpha$, there exists $f' : \tau \to \alpha$ such that for every $x : \tau$, $f(x) = f'(x)$. One writes $\tau :< \sigma$ to denote τ is a subtype of σ, or equivalently $\sigma :> \tau$ to denote σ is a **super type** or **suptype** of τ.

Parametric Types

Naturally, some objects need to be categorized in a more specific fashion than a simple word or symbol. For example, a "matrix" is not at all specific enough to define matrix addition or multiplication. There are certain constraints that must be imposed. For addition of matrices A and B, the dimensions of A and B must be equal. For multiplication, if A has dimensions $p_a \times q_a$ and B has dimensions $p_b \times q_b$, then they can be multiplied only if $q_a = p_b$. This suggests that types should be able to encode certain static properties of an object. Such types are *parametric*, and take a value as an argument producing a new type.

Definition 3.8 A **parametric type**[2] is the result of applying a type function $\phi : X \to \mathcal{T}$ where X is some domain to some value $x \in X$, giving $\phi(x) \in \mathcal{T}$.

Example 3.1 Let $\mathsf{Matrix}(p, q)$ be a matrix of real values of dimension $p \times q$. The matrix $\left(\begin{smallmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{smallmatrix}\right)$ would have type $\mathsf{Matrix}(2, 3)$. For natural numbers a, b, and c, we define the types of addition and multiplication in the following way:

$$\begin{aligned} + &: \mathsf{Matrix}(a, b) \times \mathsf{Matrix}(a, b) \to \mathsf{Matrix}(a, b) \\ \cdot &: \mathsf{Matrix}(a, b) \times \mathsf{Matrix}(b, c) \to \mathsf{Matrix}(a, c). \end{aligned}$$

Note that the repeated use of variables implies equality in each line—a matrix can only be multiplied of the number of columns, b, are equal in each matrix.

N1. Details on type equivalency, referential transparency, types at runtime, and type classes.

[2]These are sometimes called **dependent types** because the type of a value *depends* on the *value* of another.

Part II

Computer Algebra Fundamentals

Chapter 4

Relations, Identities, Forms

4.1 Relationships Between Values

Arguably, mathematics is all about relationships between different classes of items, especially the equivalence of items. Consider a smooth, egg-shaped stone, a rubber paddle ball, and a cube of billiard chalk. Pardoning physical imperfections, are any of these shapes equivalent? Of course, it depends on how we define *equivalent*. To a…

- …topologist, all of these shapes are the same, or more precisely, they are *homeomorphic*; if the shapes were made of modelling clay, any of the shapes can be molded into any other without tearing or puncturing the clay.
- …differential topologist, the stone and ball are equivalent to each other, but not to the cube of chalk. The stone and the ball are *diffeomorphic* — we can mold an egg-like shape into a sphere and back using "smooth" manipulations, but we can't get a sharp edge such as on the billiard chalk using smooth movements. Getting a sharp edge would require cutting or creasing.
- …geometer, all of these shapes are different. None of the shapes are either *similar* — equivalent up to rotation, translation, dilation, and reflection — or *congruent* — equivalent up to translation, rotation, and reflection.

Of course, there are many more fields of mathematics, and each has its peculiar versions of equality.

Equality is just one kind of relation. Simply, a relation between two objects is either true or false. Hence, '$\geq$' is a relation over integers — for all integers x and y, $x \geq y$ is either true, or it's not.

Definition 4.1 A **binary relation** ρ over a set S is defined by a set $R \subseteq S \times S$ such that for each $x, y \in S$, either $(x, y) \in R$ or $(x, y) \notin R$. Then $x \;\rho\; y$ is true[1]

[1]The statement "$x \;\rho\; y$ is true" is really a tautology, and hence one usually simply says "$x \;\rho\; y$" alone (e.g., one says "$2 \neq 3$", not "$2 \neq 3$ is true").

iff $(x, y) \in R$.

The notion of equality is clearly a relation. Provided we have some specific way to define equality, two objects are either equal, or they are not. We can define what it means for a relation to be categorized as a form of equality by adding a few simple requirements.

Definition 4.2 A binary relation $\sim$ over a set S is an **equivalence relation** if for all $a, b, c \in S$, we have the following properties:

Reflexivity $a \sim a$,

Symmetry $a \sim b$ implies $b \sim a$, and

Transitivity $a \sim b$ and $b \sim c$ imply $a \sim c$.

The set $\{x \in S \mid x \sim a\}$ is called the **equivalence class** of a, and is denoted $[a]$.

Definition 4.3 Given a predicate $p : X \to \mathbb{B}$ and elements $a, b \in X$, we say $a \equiv_p b$ if $p(a) \wedge p(b)$.

Theorem 4.1 Given a predicate $p : X \to \mathbb{B}$, $\equiv_p$ is an equivalence relation.

Proof. Since p maps each element of X to one of two values, p partitions X into two subsets $X_1 = \{x \in X \mid p(x)\}$ and $X_2 = X \setminus X_1$, we can pair the values which are equivalent into a set $E = X_1 \times X_1 = \{(a, b) \mid p(a) \wedge p(b)\}$. Therefore, $\equiv_p$ is a binary relation.

The fact that it is an equivalence relation follows simply from the definition of the Cartesian product. ■

4.2 Equality in Computer Algebra

In computer algebra and programming, we typically need to pay attention to three kinds of equality which derive from three different views of quantities: objects can be seen by their mathematical properties, their form, or their (computer) data[GCL92, p. 80].

Definition 4.4 The three principle equalities are

1. **object-level equality** (or **mathematical equality**) where elements of a domain D are considered distinct objects equal to only themselves,
2. **form-level equality** where objects are equal only if they are syntactically and structurally equal, and
3. **data-level equality** where objects are equal only if they admit the same objects in computer memory[2].

[2]This form of equality is less important in mathematics, and more important in concrete computer algorithms and data structures.

We will also refer to these kinds of equalities by **level-0**, **level-1**, and **level-2** equality respectively.

Notation 4.5 We write object-level equality of a and b via the standard equality sign: $a = b$. We write form-level equality via $a \bumpeq b$ (one bump for *level-1*). We write data-level equality as $a \Bumpeq b$ (two bumps for *level-2*).

Example 4.1 Consider $A := x^2 + 2x + 1$ and $B := (x + 1)^2$. At the object level, $A = B$, because they represent the same object in the domain $\mathbb{Z}[x]$. However, at the form level, they are syntactically and structurally different, so $a \neq b$. However, if we had a procedure `expand` which expands powers of polynomials and orders the monomials by decreasing degree, then $A \bumpeq \texttt{expand}(B)$.

Example 4.2 In Lisp, the code `(eq a b)` is essentially equivalent to $\texttt{a} \Bumpeq \texttt{b}$, since it (usually) does pointer-level equality[X3J94, pp. 5-46–47].

4.3 Canonical Forms

Often mathematical objects take several, sometimes infinitely many, different but equivalent forms. Most of us are familiar with this notion in grade school mathematics through fractions. The fractions 15/25, 6/10, and 3/5 are equivalent, and perhaps to make it easier for teachers to grade, we are told to put it in a form that is either exactly the same as the teacher's answer or not. For each equivalence class of fractions, we are taught a way to choose the "correct" fraction (i.e., one that the teacher can compare with simply by inspection); we are told to put the fraction "in lowest terms" so that for a/b, we write an equivalent form a'/b' where a' and b' are relatively prime.

The idea that we can pick a certain representative element from each equivalence class in order to allow us to easily check equivalence of two elements is the essence of *canonical forms.*

Definition 4.6 Let S be a set under the equivalence relation $\sim$. The **canonical form** of an element $x \in S$, denoted $\kappa(x)$, is an element of $[x]$ such that for all $y \in [x]$, $\kappa(y) = \kappa(x)$.

Remark. The definition of a canonical form implies that for each equivalence class, there's an element x in the equivalence class such that $x = \kappa(x)$. Moreover, we can conclude that $a \sim b$ iff $\kappa(a) = \kappa(b)$.

Example 4.3 The rational numbers are defined as the set of integer ordered pairs

(a, b) such that $b \neq 0$ obeying a few properties. We can define the canonical form of a rational number as the pair whose parts are coprime. We can define κ as a function removing common factors:

$$\kappa\left(\frac{a}{b}\right) := \frac{\operatorname{sgn}(ab)|a| \div \gcd(a,b)}{|b| \div \gcd(a,b)}.$$

In the most ideal situation, the canonical form of an object can be derived algorithmically in the same way as all other objects of some set, such as with rational numbers. That is, given a set of objects S under a binary relation $\sim$, if we view κ as a function from S to itself, κ should be the same function for each equivalence class. Such a function is called the S-**canonizing function**.

Example 4.4 The complex numbers are closed under the respective field operations and have a canonical form of $a + b\mathrm{i}$ for $a, b \in \mathbb{R}$. The $\mathbb{C}$-canonizing function is defined as follows. Let $w, z \in \mathbb{C}$.

$$\begin{aligned}
\kappa(w+z) &= (\operatorname{Re} w + \operatorname{Re} z) + (\operatorname{Im} w + \operatorname{Im} z)\mathrm{i} \\
\kappa(-z) &= (-\operatorname{Re} z) + (-\operatorname{Im} z)\mathrm{i} \\
\kappa(wz) &= (\operatorname{Re} w \operatorname{Re} z - \operatorname{Im} w \operatorname{Im} z) + (\operatorname{Re} w \operatorname{Im} z + \operatorname{Im} w \operatorname{Re} z)\mathrm{i} \\
\kappa(z^{-1}) &= \left[\frac{\operatorname{Re} z}{(\operatorname{Re} z)^2 + (\operatorname{Im} z)^2}\right] + \left[\frac{-\operatorname{Im} z}{(\operatorname{Re} z)^2 + (\operatorname{Im} z)^2}\right]\mathrm{i}
\end{aligned}$$

Note that this canonization function is actually recursive in nature, for to compute $\operatorname{Re} z$ and $\operatorname{Im} z$, one must compute its canonical form.

The importance of canonical forms is vast in the field of computer algebra. If a elements of an algebraic structure have a canonical form, then elements of this structure can be represented uniquely and hence equivalences, or **identities**, can be verified or proved. Quite simply, we have the following fact.

Canonical forms admit structural equality.

This is a central theme of computer algebra and this book. Often a problem reduces to finding a canonical form for a certain collection of elements.

4.4 Normal Forms

Canonical forms are restricted to sets equipped with some equivalence relation. However, many interesting or useful relations are not equivalence relations, like '$\geq$'. We can make an analogous notion to equivalence classes for such relations.

Suppose we want to compute first if $a/b = c/d$. By way of example example 4.3, we can do this easily—bring each fraction into lowest terms and check that $a/b \doteq c/d$. That is, check $\kappa(a/b) \doteq \kappa(c/d)$.

However, suppose we want to compute if $a/b \geq c/d$. Putting each fraction into canonical form is useless as '$\geq$' is not an equivalence relation. If we were to implement '$\geq$' in a programming language where we have only arithmetic operations, equality, and the sign function, we might do so by "normalizing" the items being compared and testing their sign. Specifically, $a \geq b$ iff $b - a \geq 0$. But $\operatorname{sgn} x = 1$ iff $x \geq 0$ (x is non-negative). So all in all, we can compute $x \geq y$ via computing $g(x, y) = 1$ for $g(x, y) := \operatorname{sgn}(x - y)$. Here, we say g is the *normalizing function.*

Definition 4.7 Given a relation $\rho : X^2 \to \mathbb{B}$, a function $\eta : X^2 \to Y$ is called the ρ-**normalizing function** if given $\forall x \in X, \eta(x, x)$ and $a, b \in X$,

$$\eta(a, b) = \eta_0 \iff a \; \rho \; b.$$

The value of $\eta(a, b)$ is called then ρ-**normal form** of a and b.

Remark. The symbol η is derived from the term *equalizer*, which refers to the equalizer of two functions $\operatorname{Eq}(f, g) := \{x \in X \mid f(x) = g(x)\}$. The equalizer is used to construct the *difference kernel* $\ker(f - g)$.

Example 4.5 Even for equivalence relations, sometimes a normal form can be more useful than a canonical form. For example, to determine if $a = b$ given two expressions a and b, we might develop a canonizing function κ and show $\kappa(a) \doteq \kappa(b)$. If this is too difficult or costly, we can use $\eta(x, y) = x - y$ so $\eta_0 = 0$, and this is our $=$-normalizing function. As such, $a = b$ reduces to determining if $a - b$ is zero, which is known as the *zero-equivalence problem.*

N2. Perhaps reference a later section

Exercises

1. [$\star$] Show another possible $\mathbb{C}$-canonizing function, different from example 4.4.

2. [$\star$] Prove that for $a, b \in \mathbb{Z}$,

$$\kappa\left(\frac{a}{b}\right) := \frac{a \div \gcd(a,b)}{b \div \gcd(a,b)}$$

is indeed a $\mathbb{Q}$-canonizing function.

Chapter 5

Arbitrary Precision Arithmetic

5.1 Introduction

When one writes programs, one is typically restricted to using numerical values in a fixed range, usually related to the width of the CPU's registers. Computers whose processor works with 32-bits typically can only work with integers in $[0, 2^{32} - 1]$, or if a bit[1] is used to denote the sign, $[-2^{31}, 2^{31} - 1]$.

Floating-point numbers, or *floats*, allow for higher precision. Instead of representing a number directly by its base-2 representation (perhaps with a sign bit), one instead represents a number by the sign, a set of significant digits called the **mantissa** or **significand**, and then a set of digits representing the **exponent**. Suppose we let the sign be s, mantissa be m, and the exponent be e. Then a number represented by these would be $sm2^e$. Still, we can only represent a finite amount of numbers in a finite amount of bits. Figure 5.1 shows the floating-point representation mandated by the IEEE 754 floating-point standard [IEE85]. With this representation, we have a minimum[2] value of -1.2×10^{-38} and a maximum of 3.4×10^{38}.

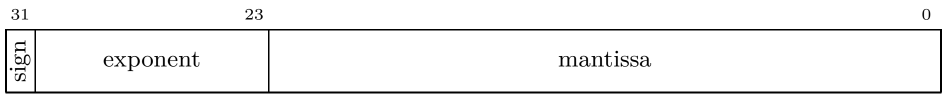

Figure 5.1. Bit-fields of an IEEE 754 floating-point number.

There is an obvious drawback: these numbers don't represent any more information than integers within the finite range do. At the cost of removing precision, we allow a wider range of numerical values. This is perfectly suitable

[1] Or one's complement or two's complement or some other scheme.

[2] This is slightly false, for we can have so-called *subnormalized* numbers which divide the gap between the smallest representable *normal* number and zero. However, these numbers do not allow minimizing the exponent (e.g., right shift the mantissa by 1 and decrease the exponent by 1), a property by which normal numbers have to abide.

for applications that either need approximate results (such as sketching a plot of a function), or very quick results (such as in modern graphics where rendering speed is more important than rendering accuracy).

Some applications require more precision, however, that is outside of the range of "native" CPU arithmetic. There are principally two solutions for this: (a) write new routines to handle the use of two or more integers which combined really represent a single large integer or, more commonly, floating-point number (*double* or *quadruple* precision which can have double or quadruple the precision of a given float), or (b) use arbitrary precision arithmetic.

Arbitrary precision arithmetic, also called **multiple precision arithmetic** or **bignum** ("big number") arithmetic, uses a computer's memory and/or disk to store integers or floats at any precision. Of course, there is still a limitation; the size of the number is limited by the space available. However, this usually isn't an issue, for a single gigabyte of memory is sufficient to store an integer with billions of decimal digits.

Arbitrary precision arithmetic is especially important in computer algebra, where precision is of utmost importance, and where numbers can become large very easily. A common example of so called "numerical blowup" is in computations with rational numbers. Suppose we want to compute

$$\frac{3555}{15131}+\frac{3835}{3444319}$$

precisely. Then by obtaining common denominators, we have

$$\frac{3555\cdot 3444319+3835\cdot 15131}{15131\cdot 3444319}=\frac{12302581430}{5209890789},$$

with the numerator and denominator already coprime. Such a result typically cannot be stored in an implementation of rational numbers whose numerators and denominators are stored in machine integers. More deceivingly, suppose we wish to compute

$$\frac{5}{151}+\frac{35}{449}+\frac{45209}{203397}.$$

After like denominators, we receive

$$\frac{4596704401}{13790113203}$$

which simplifies to 1/3. Often times, we never see the intermediate computations, and so a simple output does *not* imply simple intermediate computations as shown. As such, the lack of arbitrary precision arithmetic can severly limit the ability to do even simple fraction arithmetic.

5.2 Computer Representation

There are many ways to represent bignums. By far the most common way is by radix form. **Radix form** stores a positive integer x as a sequence of integers

$X = (X_i)_{i=0}^{n-1}$, called the **limbs**, such that $0 \le X_i < B$ for some **radix** or **base** $B \in \mathbb{Z}$. In this representation,

$$x = \sum_{i=0}^{n-1} X_i B^i.$$

The radix is usually chosen as large as possible, so long as a processor can manipulate the limbs atomically. It is possible to have a radix as large as the maximum representable integer, but this is usually not desirable, for multiplying two limbs which are as large as $\sqrt{B}$ or larger will cause numerical overflow if done atomically[3].

It is also beneficial to consider whether or not the radix should be a power of 2 or a power of 10. If it is a power of 2, arithmetic is often faster, since processors often contain specialized instructions for bit manipulations. However, when the radix is a power of 10, it is in immediate form for printing; simply printing all X_i in reverse order is sufficient to show x in its entirety in decimal.

If it is not necessary to print numbers often or have them in a "palatable form", then a power-of-two form is more advantageous. If one needs to print, then converting into decimal is necessary.

First, consider converting from a base-B representation where $B > 10$. This means each limb can represent up to $\log_{10} B$ digits in decimal.

Algorithm 5.1: `ToSmallerBase`(X, B')

```
Input  : sequence $X = (X_i)_{i=0}^{n-1}$ in base $B$
         a target base $B'$
Output : sequence $Y = (Y_i)_{i=0}$ in base $B'$, congruent to $X$
Require  $B > B'$
:
$Y := X$
$i := 0$
while $i < n \vee X_i \ge B'$ do
    $Y_{i+1} := Y_{i+1} + \lfloor Y_i / B' \rfloor$
    $Y_i := Y_i \,\%\, B'$
    $i := i + 1$
return $Y$
```

5.3 High-Precision Evaluation of Hypergeometric Series

The Chudnovsky brothers describe in [CC93] a method for computing hypergeometric series quickly, giving initial attributions to Gosper. Refer to definition 11.2 for the definition of the hypergeometric series.

[3]It is possible to multiply naïvely, but the but the number of atomic operations necessary is far more.

The idea is to use an lower-triangular matrix recurrence suggested by Gosper,

$$\begin{pmatrix} a_n & 0 \\ b_n & c_n \end{pmatrix} = \begin{pmatrix} a_{n-1} & 0 \\ b_{n-1} & c_{n-1} \end{pmatrix} \begin{pmatrix} A(n) & 0 \\ B(n) & C(n) \end{pmatrix}, \tag{5.1}$$

which can be recurred to

$$\begin{pmatrix} a_n & 0 \\ b_n & c_n \end{pmatrix} = \begin{pmatrix} A(0) & 0 \\ B(0) & C(0) \end{pmatrix} \begin{pmatrix} A(1) & 0 \\ B(1) & C(1) \end{pmatrix} \cdots \begin{pmatrix} A(n-1) & 0 \\ B(n-1) & C(n-1) \end{pmatrix}, \tag{5.2}$$

where c_n is numerator of the nth *coefficient* of the a series, b_n is the numerator of the nth *partial sum*, and a_n is the denominator of the nth partial sum. The entries $A(k)$, $B(k)$, and $C(k)$ are polynomials in $\mathbb{Z}[k]$. We present them below.

If we wish to compute the nth partial sum of

$$\sum_{k\geq 0} \frac{(p_k)^{\uparrow k}}{(q_k)^{\uparrow k}} \frac{z^k}{k!}, \tag{5.3}$$

we compute the matrix product in (5.2) with

$$\begin{aligned} A(0) &= q_0 & \qquad A(k) &= kq_k \\ B(0) &= 1 & B(k) &= z^k \\ C(0) &= p_0 & C(k) &= p_k. \end{aligned}$$

This will give us a matrix $\left(\begin{smallmatrix} a & 0 \\ b & c \end{smallmatrix}\right)$ and the nth partial sum will be b/a. Using the fact that $(ab)^{\uparrow k} = a^{\uparrow k} b^{\uparrow k}$, we may transform into a general hypergeometric series as in definition 11.2.

However, it is inefficient to compute the matrix products in a serial manner; it is better to use a divide-and-conquer method and evaluate the products in pairs, splitting off groups of factors recursively. By doing this, the factors of the multiplications are kept around the same size, which is much more efficient than otherwise. This method of computing rational series is known as **binary splitting**, and is presented in algorithm 5.2.

Algorithm 5.2: `BinarySplitHypergeometric`(P, Q, z, N)

```
Input  : an integer polynomial function $P(k)$
         an integer polynomial function $Q(k)$
         a value $z \in \mathbb{C}$
         number of terms $N$
Output : a matrix $\left(\begin{smallmatrix} a & 0 \\ b & c \end{smallmatrix}\right)$ such that $b/a = \sum_{k=0}^{N-1} \frac{[P(k)]^{\uparrow k}}{[Q(k)]^{\uparrow k}} \frac{z^k}{k!}$
Require  $N > 1$
:
$M_0 := \left(\begin{smallmatrix} 1 & 0 \\ 1 & P(0) \end{smallmatrix}\right)$
for $k := 1$ to $N - 1$ do
    $M_k := \left(\begin{smallmatrix} kQ(k-1) & 0 \\ z^k & P(k) \end{smallmatrix}\right)$
repeat
    for $k := 0$ to $\lfloor N/2 \rfloor - 1$ do
        $M_k := M_{2k} \cdot M_{2k+1}$
    if $N$ is odd then $M_{(N-1)/2} := M_{N-1}$
    $N := \lceil N/2 \rceil$
until $N \leq 1$
return $M_1$
```

Part III

Abstract Algebra

Chapter 6

Basic Structures and Divisibility

6.1 Algebraic Structures

Definition 6.1 A **group** $(G; \oplus)$ is a set $G \neq \varnothing$ closed under the a binary operation $\oplus$ satisfying the following axioms.

A1. (Associativity) $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ for all $a, b, c \in G$.

A2. (Identity) There is an element, called the **identity element**, $\iota \in G$ such that $\iota \oplus a = a \oplus \iota = a$ for all $a \in G$.

A3. (Inverses) For all $a \in G$, there is an element $a^{-1} \in G$ such that $a \oplus a^{-1} = a^{-1} \oplus a = \iota$, and is called the **inverse** of a.

An **abelian** or **commutative group** is one that satisfies an additional axiom.

A4. (Commutativity) $a \oplus b = b \oplus a$ for all $a, b \in G$.

Definition 6.2 A **ring** $(R; \oplus, \otimes)$ is a set $R \neq \varnothing$ closed under two binary operations $\oplus$ and $\otimes$ such that $(R; \oplus)$ is an abelian group (A1–A4), $\otimes$ is associative and has an identity (A1 and A2), and which satisfies the following axiom:

A5. (Distributivity) $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ for all $a, b, c \in R$.

A **commutative ring** is a ring in which $\otimes$ is commutative (A4 holds for $\otimes$). An **integral domain** is a commutative ring which satisfies the axiom:

A6. (Cancellation) $a \otimes b = a \otimes c \implies b = c$ for all $a, b, c \in R$ and $a \neq 0$.

Axiom A6 can be reformulated into an equaivalent, though meaningful, one.

A6'. (No Zero Divisors) $a \otimes b = 0 \implies a = 0 \vee b = 0$ for all $a, b \in R$.

Rings usually denote the identity element with respect to $+$ as 0, the identity element of $\cdot$ as 1, and the inverse of a with respect to $+$ as $-a$.

Definition 6.3 A **field** $(F; \oplus, \otimes)$ is a set F having two binary operations $\oplus$ and $\otimes$ such that $(F; \oplus)$ is an abelian group, $(F \setminus \{0\}; \otimes)$ is an abelian group, and $\otimes$ is distributive over $\oplus$.

Remark. We can equivalently say a field is a commutative ring in which every non-zero element has a multiplicative inverse.

Theorem 6.1 Every finite integral domain is a field.

Proof. Let D be a finite integral domain and let $a \in D$ be non-zero. Define $f : D \to D$ by $f(x) = ax$.

Suppose $f(x) = f(y)$ for some $x, y \in D$. Then $ax = ay \implies a(x - y) = 0$. But since $a \neq 0$ and the cancellation property, $x - y = 0$. Therefore, $x = y$ proving f is injective. And since D is finite, by the pidgeonhole principle, f must also be surjective. As such, there exists a $b \in D$ such that $f(b) = 1$, so $ab = 1$ and therefore a has an inverse and so D is a field. ■

Structure	Notation	Axioms		
		$\oplus$	$\otimes$	$\otimes$ over $\oplus$
Group	$(G; \oplus)$	1–3		
Abelian Group	$(G; \oplus)$	1–4		
Ring	$(R; \oplus, \otimes)$	1–4	1, 2	5
Commutative Ring	$(R; \oplus, \otimes)$	1–4	1, 2, 4	5
Integral Domain	$(D; \oplus, \otimes)$	1–4	1, 2, 4	5, 6
Field	$(F; \oplus, \otimes)$	1–4	1–4	5, 6

Table 6.1. Summary of algebraic structures.

6.2 Divisibility and Factorization

Division is of course possible in a field. In general, division is not possible in an integral domain, but the concept of factorizing into primes familiar to us with $\mathbb{Z}$ can be generalized to other integral domains. In this section, let $(D; +, \cdot)$ be an integral domain, and let $ab \equiv a \cdot b$ as in standard mathematical notation.

Divisibility

Definition 6.4 For $a, b \in D$, a is called a **divisor** of b iff $b = ax$ for some $x \in D$, and we say a **divides** b, usually denoted $a \mid b$. Equivalently, we say b is a **multiple** of a.

Definition 6.5 For $a, b \in D$, an element $c \in D$ is called a **greatest common divisor** or **GCD** of a and b if $c \mid a$ and $c \mid b$ and c is a multiple of every other element that divides both a and b. On the other hand, c is called a **least common multiple** or **LCM** if $a \mid c$ and $b \mid c$.

We say "a GCD" and not "the GCD" because when the GCD of $a, b \in D$ exists, it is not (but almost) unique. A familiar application of GCDs is to reduce rational numbers to "lowest terms".

Definition 6.6 Two elements $a, b \in D$ are **associates** if $a \mid b$ and $b \mid a$.

Definition 6.7 An element $u \in D$ is called a **unit** (or **invertible**) if u has a multiplicative inverse $u^{-1} \in D$.

Example 6.1 In the integral domain $\mathbb{Z}$, (a) the units are 1 and -1, (b) 6 is a GCD of 18 and 30, (c) -6 is also a GCD of 18 and 31, and (d) 6 and -6 are associates.

Theorem 6.2 For any integral domain, two elements a and b are associates iff $au = b$ for some unit u.

Proof. First, $au = b$ implies $a \mid b$. If u is a unit, then u^{-1} exists. By the cancellation axiom, $auu^{-1} = bu^{-1} \implies a = bu^{-1}$, therefore $b \mid a$. It follows from definition 6.6 that a and b are associates. ■

Theorem 6.3 For any integral domain, if c is a GCD of a and b, then so is any associate $d = cu$. Conversely, if c and d are GCDs of a and b, then d must be an associate of c.

Proof. If c is a GCD of a and b, then it divides every multiple of numbers which also divide a and b. If we let $c = du^{-1}$, then clearly du^{-1} divides a and b as well, and as such, d divides a and b and d is also a multiple of c.

For the converse, assume d must not be an associate of c. Then either $c \nmid d$ or $d \nmid c$. But if this is true, then either c or d is not a GCD because one is not a multiple of all other common divisors of a and b, which is contrary to the definition. ■

It is customary to impose restrictions to make the GCD a unique number. Since associates are related by an equivalence relation, integral domains can be decomposed into **associate classes**. For example, the associate classes of $\mathbb{Z}$ are $\{0\}$, $\{-1, 1\}$, $\{-2, 2\}$, and so on. For a particular integral domain, we can choose a single element from each associate class to represent it canonically, and define it to be **unit normal**[1]. For $\mathbb{Z}$, we define the non-negative elements to be unit normal elements. In any field F, every nonzero element is an associate of every other nonzero element (and also, every nonzero element is a unit). In this case, we define 0 and 1 to be unit normal.

Definition 6.8 In any integral domain D for which unit normal elements are defined, an element $c \in D$ is called the **unit normal GCD** of $a, b \in D$, denoted $c = \gcd(a, b)$, if c is a GCD of a and b and c is unit normal.

[1]Confusingly, a "unit normal" is not always a "unit". The term comes from the fact that we "normalize" the GCD by choice of a unit.

The unit normal GCD of two elements $a, b \in D$ is unique. If unit normals are appropriately defined, the following properties can hold:

1. 0 is unit normal.
2. 1 is the unit normal element for the associate class of unit normals.
3. If $a, b \in D$ are unit normal, then their product ab is also unit normal in D.

Henceforth, we will mean "the unit normal GCD" when we mention "the GCD".

Definition 6.9 Let D be an integral domain in which unit normals have been defined. The **normal part** of $a \in D$, denoted $\nu(a)$, is defined to be the unit normal representative of the associate class containing a. The **unit part** of $a \in D \setminus \{0\}$, denoted $\upsilon(a)$, is the unique unit in D such that $a = \upsilon(a)\nu(a)$. We define for convenience $\upsilon(0) = 1$, since clearly, $\nu(0) = 0$.

Example 6.2 In the integral domain $\mathbb{Z}$, $\nu(a) = |a|$ and $\upsilon(a) = \operatorname{sgn}(a)$, where the sign of an integer is defined by

$$\operatorname{sgn}(a) = \begin{cases} -1 & \text{if } a < 0 \\ 1 & \text{if } a \geq 0. \end{cases} \tag{6.1}$$

The LCM of two elements $a, b \in D$, when it exists, can be made unique similarly.

Theorem 6.4 The LCM of $a, b \in D$ exists iff $\gcd(a, b)$ exists, and equals $ab/\gcd(a, b)$.

Proof. First, the GCD and LCM of $(0, 0)$ is 0 by definition. Since $\gcd(a, b) \mid a$ and $\gcd(a, b) \mid b$, clearly $\gcd(a, b) \mid ab$ as well. Moreover, ab is a multiple of a and b. This can be reduced by removing common factors of a and b which gives the least number divisible by both a and b. Removing the factors is done by dividing by the GCD, giving the LCM $ab/\gcd(a, b)$ for non-zero a and b. ■

We define the unique **unit normal LCM** of $a, b \in D$ denoted $\operatorname{lcm}(a, b)$ by

$$\operatorname{lcm}(a, b) = \frac{\nu(ab)}{\gcd(a, b)}. \tag{6.2}$$

As with the GCD, we will henceforth refer to the "unit normal LCM" as simply "the LCM".

Unique Factorization Domains

Definition 6.10 An element $p \in D \setminus \{0\}$ is called **prime** or **irreducible** if (a) p is not a unit, and (b) whenever $p = ab$, either a or b is a unit.

Definition 6.11 Two elements $a, b \in D$ are called **relatively prime** if $\gcd(a, b) = 1$.

Definition 6.12 An integral domain D is called a **unique factorization domain** or **UFD** if for all $a \in D \setminus \{0\}$, either a is a unit, or a can be expressed as a finite product of primes, but not both, such that this factorization is unique up to associates and reordering.

Corollary If $a = p_1 \cdots p_n$ and $a = q_1 \cdots q_m$ are two prime factorizations of a, then $n = m$ and there exists a reordering of the q_j values such that p_i is an associate of q_i for $1 \le i \le n$.

Definition 6.13 Let D be a UFD in which unit normal elements have been defined. Then for $a \in D$, then a prime factorization of the form

$$a = \upsilon(a) p_1^{e_1} p_1^{e_2} \cdots p_n^{e_n} \tag{6.3}$$

is called a **unit normal factorization** if p_i are unit normal primes, $e_i > 0$, and $p_i \neq p_j$ whenever $i = j$ for $0 \le i \le n$ and $0 \le j \le n$.

These definitions admit the property that if $p \mid ab$, then $p \mid a$ or $p \mid b$. In other words, if p divides ab, then p or an associate thereof must appear in the factorization of a or b.

Henceforth, assume unit normal elements satisfy the three properties 6.2.

Theorem 6.5 If D is a UFD and if $x, y \in D$ are not both zero, then $\gcd(x, y)$ exists and is unique.

Proof. The uniqueness has been established already. In order to show existence, first suppose that $x, y \neq 0$ and let their unique unit normal factorization be $x = \upsilon(x) p_1^{a_1} \cdots p_n^{a_n}$ and $y = \upsilon(y) q_1^{b_1} \cdots q_m^{b_m}$. Let $\{r_1, \ldots, r_\ell\} = \{p_1, \ldots, p_n\} \cup \{q_1, \ldots, q_m\}$. Then the factorizations of x and y may be written

$$x = \upsilon(x) \prod_{i=1}^{\ell} r_i^{\alpha_i} \qquad \text{and} \qquad y = \upsilon(y) \prod_{i=1}^{\ell} r_i^{\beta_i} \tag{6.4}$$

probably with $\alpha_i, \beta_i = 0$ for "some" i. Obviously,

$$d = \prod_{i=1}^{\ell} r_i^{\min\{\alpha_i, \beta_i\}} \tag{6.5}$$

is $\gcd(x, y)$. If x or y is zero but not both, assume without loss of generality that $x \neq 0$ and $y = 0$. If x has a unique unit normal factorization $\upsilon(x)p_1^{a_1} \cdots p_n^{a_n}$, then

$$d = \prod_{i=1}^{n} p_i^{a_i} \tag{6.6}$$

is $\gcd(x, y)$. ■

Euclidean Domains

Definition 6.14 A **Euclidean domain** is an integral domain D equipped with a valuation $v : D \setminus \{0\} \to \mathbb{N}$ such that

Property 1. for all $a, b \in D \setminus \{0_D\}$, $v(ab) \geq v(a)$ and

Property 2. for all $a, b \in D$ with $b \neq 0_D$, there exist $q, r \in D$ such that $a = bq + r$ where $r = 0_D$ or $v(r) < v(b)$.

Property 2 is known as the *division property*, q is known as the **quotient**, and r the remainder.

Remark. Every Euclidean domain is a UFD.

The set of integers is a Euclidean domain. However, quotients and remainders are not always unique.

Theorem 6.6 Let D be a Euclidean domain and $a, b \in D \backslash 0_D$. If $g = \gcd(a, b)$, then there exist elements s and t such that $g = sa + tb$.

Chapter 7

Polynomial, Rational, and Series Structures

7.1 Univariate Polynomial Domains

Chapter 8

Field Extensions

Often times it is advantageous to extend a field with new elements. One reason it is done is to make a field closed under some operation. For example, if we consider the real numbers together with the square root operation, the field of reals is not closed with respect to square root. However, we can append an element to the reals in order to make this field closed, namely $\mathrm{i} := \sqrt{-1}$.

Definition 8.1 Let F and E be fields such that $F \subset E$. F is said to be a **sub-field** of E, or equivalently E is an **extension field** of F, and $E : F$ is a **field extension**. Given $E : F$ and some $S \subset E$, the smallest sub-field of E that contains F and S is denoted as $F(S)$, and one says $F(S)$ is generated by the **adjunction** of S to F. If A is an extension field of E but is a sub-field of E, then A is an **intermediate field** of the field extension $E : F$.

Intuitively, this means that $F(S)$ is a field F extended with the elements S forming a new field. Often times, one will extend a field F by a single element s, called a **primitive element**, is denoted by $F(s)$ as opposed to $F(\{s\})$. For example, the field of rational numbers extended with the quadratic irrational $\sqrt{2}$ is denoted $\mathbb{Q}(\sqrt{2})$.

Field extensions can be seen from a linear algebra point-of-view. Let $E : F$ be a field extension. Then E can be seen as a vector space over F. For example, $\mathbb{C} : \mathbb{R}$ is a field extension, and can be seen as the set of pairs (x, y) representing $x + y\mathrm{i}$. In other words, a vector space whose scalars are the real numbers generates the complex numbers. We can say then that $\mathbb{C}$ has a *basis* of $\{1, \mathrm{i}\}$ because all elements are a linear combination of reals and elements of the basis. This motivates the definition of the "size" of an extension.

Definition 8.2 Given a field extension $E : F$, the **degree** of $E : F$, denoted $|E : F|$, is the dimension of E when seen as a vector space over F.

From this definition, one sees that $|\mathbb{C} : \mathbb{R}| = 2$ and this extension can be written as an adjuction of i to $\mathbb{R}$, hence $\mathbb{C} = \mathbb{R}(\mathrm{i})$. Furthermore, $|\mathbb{R} : \mathbb{Q}| = 2^{\aleph_0}$ because there is no finite extension to the rationals that generates the reals.

Lastly, the field extension $\mathbb{Q}(\sqrt{3}, \sqrt{5})$ has a degree of 4 because $\{1, \sqrt{3}, \sqrt{5}, \sqrt{15}\}$ is the basis of the vector space. In other words, $|\mathbb{Q}(\sqrt{3}, \sqrt{5}) : \mathbb{Q}| = 4$.

Theorem 8.1 If we have the field extension $E : F$ and an intermediate field A, then $|E : F| = |E : A| \cdot |A : F|$.

We may use this theorem[1] to find the size of intermediate fields. By this theorem and that $|\mathbb{Q}(\sqrt{3}, \sqrt{5}) : \mathbb{Q}| = 4$, we may conclude that $|\mathbb{Q}(\sqrt{3}, \sqrt{5}) : \mathbb{Q}(\sqrt{5})| = 2$. As another example, we can conclude that there are no (non-trivial) intermediate fields of $\mathbb{C} : \mathbb{R}$ because, supposing there was a field F, $|\mathbb{C} : F| \cdot |F : \mathbb{R}| = 2$ and the only integral solutions are $|\mathbb{C} : F| = 1$ and $|F : \mathbb{R}| = 2$ or $|\mathbb{C} : F| = 2$ and $|F : \mathbb{R}| = 1$.

8.1 Algebraic and Transcendental Extensions

Algebraic numbers must owe their definition to geometry. Suppose we have a right triangle with legs of length 2 and 3. We know from the Pythagorean theorem that $2^2 + 3^2 = c^2$ for a hypotenuse of length c. If we only have knowledge of integers, all we know is that $c^2 = 13$. We might suppose that c is some ratio of two integers, but we can prove this isn't the case using basic number theory. Since c must not be integer or rational, we might put it into a new class of numbers, the *algebraic* numbers.

Rewrite $r^2 = s$ as $r^2 - s = 0$. We define $r = \sqrt{s}$ as the positive solution to $r^2 - s = 0$ for $s \in \mathbb{Z}$, or **root** or **zero** of $r^2 - s$. This generalization of $c^2 = 13$ (or $c^2 - 13 = 0$) allows a whole new class of numbers, most of which are non-integral. We can generalize one step by allowing any power of r, not just 2. But we can do better.

We can widen the breadth of this new class of numbers by allowing the left-hand side of $r^2 - s = 0$ to be *any* integer polynomial[2]. As such, we define the **real algebraic numbers** as the set of all roots to all integer polynomials. Note that for an integer k, the polynomial $x - k$ shows us that all integers are real-algebraic, and for integers a and nonzero b, the polynomial $ax - b$ shows that all rationals are real-algebraic.

From this definition of real algebraics, we make one more generalization in the spirit of abstract algebra. Since we are constructing numbers simply as the roots to a polynomial, we could generalize the domain of the coefficients to an arbitrary field.

[1] This theorem can be seen as an analogy to Lagrange's theorem stating that the order of every subgroup H of a group G divides the order of G itself. More descriptively, if $|G : H|$ is the number of left-cosets of H in G, then $|G| = |G : H| \cdot |H|$.

[2] Integer polynomials are sufficient to cover rational polynomials. Suppose we have a rational polynomial whose coefficients c_i for $0 \leq i < n$ are $c_i = a_i/b_i$. Let $g = \gcd(b_0, b_1, \ldots, b_n)$. Then given $\sum_{i=0}^{n} c_i = s$, we have $g \sum_{i=0}^{n} c_i = gs$. With $c_i' :- gc_i$, where c_i' is obviously integral, we have an integer polynomial whose coefficients are c_i' equated to a rational (or more specifically, integer if s is integral).

Definition 8.3 Let $E : F$ be a field extension and let $a \in E$. Then a is **algebraic** over F if there is a polynomial function f over F such that $f(a) = 0$. If there is no f such that $f(a) = 0$, then a is called **transcendental** over F. If all a in E are transcendental, then E is a **transcendental extension**. If all are algebraic, then it is an **algebraic extension**.

Part IV

Polynomials

Chapter 9

Polynomial Algorithms

9.1 Sylvester Matrices and Resultants

Definition 9.1 Let $A(x), B(x) \in F[x]$ be non-zero polynomials in some field F with $A(x) = \sum_{k=0}^{m} a_k x^k$ and $B(x) = \sum_{k=0}^{n} b_k x^k$. The **Sylvester matrix** of A and B, denoted $S(A, B)$ is an $(m+n) \times (m+n)$ matrix defined in the following way:

- the first row has the coefficients of $A(x)$ in descending order, with the last $n-1$ entries as 0;
- the second row is equivalent to the first row, but the entries are shifted one column to the right, with the last 0 "wrapping" to the first entry;
- the next $n-2$ rows are done in the same way;
- the $(n+1)$th row has the coefficients of $B(x)$ in descending order, with the last $m-1$ entries as 0;
- repeat the shifting process for the last $m-1$ rows.

Example 9.1 Consider the polynomials $p(x) = 5x^2 - 3x + 1$ and $q(x) = 9x^3 + 2x + 7$. The Sylvester matrix of p and q is

$$S(p,q) = \begin{pmatrix} 5 & -3 & 1 & 0 & 0 \\ 0 & 5 & -3 & 1 & 0 \\ 0 & 0 & 5 & -3 & 1 \\ 9 & 0 & 2 & 7 & 0 \\ 0 & 9 & 0 & 2 & 7 \end{pmatrix}.$$

Definition 9.2 The **resultant** of two polynomials $p(x), q(x) \in F[x]$ is the deteminant of the Sylvester matrix of p and q and is written $\operatorname{res}_x(p(x), q(x))$, or $\operatorname{res}(p(x), q(x))$ when the variable is unambiguous. Also, $\operatorname{res}(0, q(x)) := 0$ and $\operatorname{res}(a, b) := 1$ for $a, b \in F$.

We prove an important criterion as a corollary to the following lemma.

Lemma 9.1 Let $A(x), B(x) \in F[x]$ be polynomials of positive degrees m and n respectively. Then there exist polynomials $U(x), V(x) \in F[x]$ with $\deg U < n$ and $\deg V < m$ such that

$$A(x)U(x) + B(x)V(x) = \operatorname{res}(A(x), B(x)). \tag{9.1}$$

Proof. For $A(x)$ and $B(x)$ with coefficients $(a_i)_{0\le i\le m}$ and $(b_i)_{0\le i\le n}$, we construct $m+n$ simultaneous equations

$$\begin{aligned}
a_m x^{m+n-1} + a_{m-1}x^{m+n-2} + \cdots + a_0 x^{n-1} &= x^{n-1}A(x)\\
a_m x^{m+n-2} + \cdots + a_1 x^{n-1} + a_0 x^{n-2} &= x^{n-2}A(x)\\
\vdots &= \vdots\\
a_m x^m + a_{m-1}x^{m-1} + \cdots + a_0 &= A(x)\\
b_n x^{m+n-1} + b_{n-1}x^{m+n-2} + \cdots + b_0 x^{m-1} &= x^{m-1}B(x)\\
b_n x^{m+n-2} + \cdots + b_1 x^{m-1} + b_0 x^{m-2} &= x^{m-2}B(x)\\
\vdots &= \vdots\\
b_n x^n + b_{n-1}x^{n-1} + \cdots + b_0 &= B(x)
\end{aligned}$$

which can be written as a matrix equation as

$$M\begin{pmatrix} x^{m+n-1}\\ x^{m+n-2}\\ \\ \vdots\\ x\\ 1\end{pmatrix} = \begin{pmatrix} x^{n-1}A(x)\\ x^{n-2}A(x)\\ \\ \vdots\\ xB(x)\\ B(x)\end{pmatrix}.$$

Using Cramer's rule to solve for the last entry, 1, gives

$$\det[S(A,B) \mid R] = \det M, \text{ with } R = \begin{pmatrix} x^{n-1}A(x)\\ \vdots\\ A(x)\\ x^{m-1}B(x)\\ \vdots\\ B(x)\end{pmatrix}, \tag{9.2}$$

where $S(A,B)$ is the Sylvester matrix of A and B and $X \mid Y$ is matrix X augmented by Y. Expanding the left-hand side of (9.2) by minors about the column R proves the lemma. ■

Corollary (Sylvester's criterion) The polynomials $A(x)$ and $B(x) \in F[x]$ have a non-trivial (i.e., non-constant) common factor iff $\operatorname{res}(A,B) = 0$.

Proof. If $\operatorname{res}(A,B) \neq 0$ then lemma 9.1 implies that any factor that divides both A and B must also divide the resultant of A and B. Since the resultant is constant, the only divisors of A and B must be polynomials of degree 0, and therefore there are no non-trivial divisors.

Conversely, assume $\operatorname{res}(A,B) = 0$. Then (9.1) becomes

$$A(x)U(x) = -B(x)V(x).$$

If there are no non-trivial factors dividing both A and B, then $B(x) \mid U(x)$. However, this is not possible since $\deg U < \deg B$ by lemma 9.1. ■

9.2 Polynomial Roots and Algebraic Numbers

As described in §8.1, real algebraic numbers are defined as the set of roots to all integer polynomials. By the fundamental theorem of algebra, we can factor $p(x) \in \mathbb{Z}[x]$ into binomials $x - r_i$ where $i = \deg p(x)$, and some constant factor $C \in \mathbb{Z}$. Therefore, every algebraic number can be encoded by the coefficients of their representative polynomial, along with an integer i. However, such a representation is not unique. Consider $\sqrt{2}$. Ordering roots from least to greatest, we could represent $\sqrt{2}$ with $(x^2-2, 1)$ or $(x^4+6x^3+7x^2-12x-18, 4)$. In general, given $p \in \mathbb{Z}[x]$ and any $q \in \mathbb{Z}[x]$, and supposing (p, n) represents some algebraic number, then (pq, n') will also represent the same number for some $n' \leq \deg pq$. This motivates the definition of a minimal polynomial.

Definition 9.3 A **minimal polynomial** of a real algebraic α is a polynomial $p(x) \in \mathbb{Z}[x]$ of the smallest degree such that $p(\alpha) = 0$. More generally, given a field extension $E : F$ with $\alpha \in E$ and $p(x) \in F[x]$, the minimal polynomial of α is the monic polynomial of least degree such that $p(\alpha) = 0$.

Remark. Note that it is no problem to define a minimal polynomial as being monic. If the leading term of a polynomial is not 1, then we can divide through to get a polynomial in a rational field which is monic.

This also motivates a theorem.

Theorem 9.2 If $p(x) \in F[x]$ is a minimal polynomial of α, then $p(x)$ is irreducible over $F[x]$, and if a polynomial $q \neq p$ has the property that $q(\alpha) = 0$, then $p(x) \mid q(x)$.

Chapter 10

Algebra In the Reals

10.1 Introduction

Topics include

- real closed fields and Sturm theory,
- algebraic numbers,
- Tarski sentences,
- semialgebraic geometry.

10.2 Real Closed Fields

Definition 10.1 An **ordered field** F is a commutative field with a subset of **positive elements** P such that

1. $0 \notin P$,
2. if $a \in F$ then $a \in P$, $-a \in P$, or $a = 0$,
3. P is closed under addition and multiplication.

If the field F in question has an ordering $>$, then we can say an element a is *positive* if $a > 0$, and if $-a > 0$, then a is *negative*. Clearly this allows us to construct $P = \{a \in F \mid a > 0\}$, but also a set of **negative elements** $N = \{a \in F \mid -a > 0\}$. As such, $F = P \cup N \cup \{0\}$.

If a field F along with P is defined, we can construct an ordering by defining $x > y$ if $x - y \in P$. This is a **strict linear ordering**, an ordering which satisfies the following properties:

- $a > b$ implies $\forall x \in F$, $a + x > b + x$,
- $a > b$ implies $\forall x \in P$, $ax > bx$,
- and $a > b$ for $a, b \in P$ implies $b^{-1} > a^{-1}$.

N3. write on intervals

Definition 10.2 Given an ordered field F, the **absolute value** of $x \in F$ is defined as

$$|x| := \begin{cases} x & \text{if } x \geq 0, \\ -x & \text{if } x < 0. \end{cases}$$

Definition 10.3 Given an ordered field F, the **sign** of $x \in F$ is defined as

$$\operatorname{sgn} x := \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0. \end{cases}$$

Easily we see that given an ordered field F and $x \in F$, we have

- $x = \operatorname{sgn}(x)|x|$ and $|x| = \operatorname{sgn}(x)x$, and
- $\operatorname{sgn}(ab) = \operatorname{sgn}(a)\operatorname{sgn}(b)$.

Part V

Series

Chapter 11

Hypergeometric Series

11.1 Hypergeometric Series and Functions

Hypergeometric series are motivated understandably by geometric series. Recall that a **geometric series** is a power series $\sum_{k\geq 0} C_k x^k$ where C_{k+1}/C_k is constant. This motivates a generalization by loosening the restriction to be constant. In particular, we define the following.

Definition 11.1 The sequence $(C_k)_{k\geq 0}$ is **hypergeometric** if $\rho = C_{k+1}/C_k$ for some $\rho \in \mathbb{Q}(k)$.

From this, we define the hypergeometric series in the same way as the geometric series.

Definition 11.2 The **generalized hypergeometric series** is a power series $\sum_{k\geq 0} C_k z^k$ such that $(C_k)_{k\geq 0}$ is hypergeometric and $C_0 = 1$.

Since ρ is a ratio of two polynomials P and Q, by the fundamental theorem of algebra, we can express ρ as a product of binomials. Therefore,

$$\rho(k) = \frac{(k+a_1)\cdots(k+a_p)}{(k+b_1)\cdots(k+b_q)(k+1)} \tag{11.1}$$

for $a_1, \ldots, a_p, b_1, \ldots, b_q \in \mathbb{C}$. The $k+1$ factor in the denominator is present due to Gauss [Gau23] and generally tradition[1], and presents no problem, for if Q does not have a factor $k+1$, then let $P' = P(k)(k+1)$ so that $\rho = P/Q = P'/[Q(k+1)]$.

Definition 11.2 and (11.1) suggest that a hypergeometric series is completely determined by parameters $(a_i)_{1\leq i\leq p}$ and $(b_j)_{1\leq j\leq q}$. This is true, and we can moreover express the series more explicitly. First though, we define an operator that will prove useful.

[1] According to [PWZ96, p. 35] and [Wei10].

Definition 11.3 The **rising product**[2] is defined by $a^{\uparrow n} := a(a+1)\cdots(a+n-1)$ for $n \in \mathbb{Z}$.

Because $C_0 = 1$ and $C_{k+1}/C_k = \rho(k)$, we have $C_1/C_0 = C_1 = \rho(0)$. Therefore, by (11.1),

$$C_1 = \frac{a_1 \cdots a_p}{b_1 \cdots b_q}. \tag{11.2}$$

Repeating this, we receive

$$C_2 = \frac{a_1(a_1+1)\cdots a_p(a_p+1)}{b_1(b_1+1)\cdots b_q(b_q+1)} \cdot \frac{1}{2} \tag{11.3}$$

$$C_3 = \frac{a_1(a_1+1)(a_1+2)\cdots a_p(a_p+1)(a_p+2)}{b_1(b_1+1)(b_1+2)\cdots b_q(b_q+1)(b_q+2)} \cdot \frac{1}{2\cdot 3} \tag{11.4}$$

and by induction,

$$\begin{aligned} C_n &= \frac{a_1^{\uparrow n}\cdots a_p^{\uparrow n}}{b_1^{\uparrow n}\cdots b_q^{\uparrow n}} \cdot \frac{1}{n!} \\ &= \frac{\prod_{i=1}^{p}(a_i)^{\uparrow n}}{\prod_{j=1}^{q}(b_j)^{\uparrow n}} \cdot \frac{1}{n!}. \end{aligned} \tag{11.5}$$

Putting C_n into sum in definition 11.2, we have

$$\sum_{k\geq 0} \frac{\prod_{i=1}^{p}(a_i)^{\uparrow k}}{\prod_{j=1}^{q}(b_j)^{\uparrow k}} \cdot \frac{x^k}{k!}. \tag{11.6}$$

Parameterizing a_i and b_j motivates the definition of the hypergeometric function.

Definition 11.4 The **generalized hypergeometric function** $\mathrm{F}_q^p : \mathbb{C} \to \mathbb{C}$ is defined by the hypergeometric series

$$\mathrm{F}_q^p \left[\begin{matrix} a_1, \ldots, a_p \\ b_1, \ldots, b_q \end{matrix} \middle| \, z\right] := \sum_{k=0}^{\infty} \frac{\prod_{i=1}^{p}(a_i)^{\uparrow k}}{\prod_{j=1}^{q}(b_j)^{\uparrow k}} \cdot \frac{z^k}{k!} \tag{11.7}$$

for the $a_i, b_j, z \in \mathbb{C}$. The function with arguments is also written $\mathrm{F}_q^p(a_1, \ldots, a_p; b_1, \ldots, b_q; z)$.

Convergence Conditions

Until now, the convergence of the hypergeometric series has been neglected. Refer to definition 11.4. If a_i is a non-positive integer for some $i = i_0$, then for all $k \geq 0$, $(a_{i_0})^{\uparrow k} = 0$ and therefore the numerator is 0 for all $i \geq i_0$ and the sum is finite (and therefore converges). Similarly, if any b_i is a non-positive integer, then the series is undefined because the denominator would be 0.

Using the ratio test, we can determine the radius of convergence for other cases. There are essentially three cases.

[2]Sometimes called the **rising factorial**, or **Pochhammer symbol** if written $(a)_n$.

Case I. If $p = q+1$, then $\rho(k) \to 1$ which implies the radius of convergence is 1 (F_q^p converges absolutely for $|z| < 1$ and diverges for $|z| > 1$. See below about $|z| = 1$.)

Case II. If $p < q + 1$, then $\rho(k) \to 0$ which implies that the radius of convergence is ∞ (F_q^p converges absolutely for all z).

Case III. If $p > q + 1$, then $\rho(k) \to \infty$ which implies the radius of convergence is 0 and F_q^p diverges for $z \neq 0$.

Detemining convergence of F_q^p when $p = q + 1$ is difficult when z is on the unit circle. Note the following criterion.

Theorem 11.1 Let $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_{n-1})$ be positive sequences. Then $\mathrm{F}_{n-1}^n(\mathbf{a}; \mathbf{b}; z)$ converges absolutely at $z = 1$ if

$$\sum_{i=1}^{n-1} \operatorname{Re} b_i > \sum_{i=1}^{n} \operatorname{Re} a_i. \qquad (11.8)$$

11.2 Series as Hypergeometric Functions

Many series can be converted into hypergeometric functions. This is done by writing the summand of the series in an equivalent way with rising products.

Lemma 11.2 Every binomial in k can be represented as a function of rising products in k. Specifically,

$$ak + b = \frac{b(1 + b/a)^{\uparrow k}}{(b/a)^{\uparrow k}}. \qquad (11.9)$$

Proof. By inspection, one can see that $(x + 1)^{\uparrow k} = x^{\uparrow k+1}/x$ and $x^{\uparrow m}/x^{\uparrow n} = (x + n)^{\uparrow m-n}$ (for $m > n$). So

$$\begin{aligned}\frac{b(1 + b/a)^{\uparrow k}}{(b/a)^{\uparrow k}} &= \frac{b(b/a)^{\uparrow k+1}}{(b/a)(b/a)^{\uparrow k}} \\ &= a(k + b/a)^{\uparrow k+1-k} \\ &= a(k + b/a) \\ &= ak + b.\end{aligned}$$
■

11.3 Discrete Operators and The Summation Problem

Given a function g, define the **forward difference** as $\Delta g(x) := g(x + 1) - g(x)$. In the case that g depends on several variables, define the **forward partial difference** as $\Delta_x g(x, y) := g(x+1, y) - g(x, y)$. The forward difference is a **discrete operator**, namely because it works in a non-dense[3] domain of integers. Moreover, the forward difference is a "discrete analog" to the continuous derivative, as shown in figure 11.1.

$$\boxed{\begin{array}{rclrcl} \mathrm{d}c &=& 0 & \Delta c &=& 0 \\ \mathrm{d}(af+bg) &=& a\mathrm{d}f + b\mathrm{d}g & \Delta(af+bg) &=& a\Delta f + b\Delta g \\ \mathrm{d}(fg) &=& g\mathrm{d}f + f\mathrm{d}g & \Delta(fg) &=& g\Delta f + f\Delta g + \Delta f\Delta g \\ \mathrm{d}\left(\frac{f}{g}\right) &=& \frac{g\mathrm{d}f - f\mathrm{d}g}{g^2} & \Delta\left(\frac{f}{g}\right) &=& \frac{g\Delta f - f\Delta g}{(g+\Delta g)g} \end{array}}$$

Figure 11.1. Relationship between the continuous and discrete derivative.

Since the forward difference is a discrete analog to the derivative, it motivates questioning the existence of a discrete analog to the antiderivative, i.e., integral. In other words, given a function f, does there exist a function g such that $f = \Delta g$, and if so, what is it?

We begin to "solve" for g by **telescoping**, which is adding consecutive terms so that cancellation occurs. So, by telescoping Δg, we have

$$\begin{aligned} &\Delta g(n) + \Delta g(n-1) + \cdots + \Delta g(1) \\ &= \quad [g(n) - g(n-1)] + [g(n-1) - g(n-2)] + \cdots + [g(1) - g(0)] \\ &= \quad g(n) - g(0). \end{aligned}$$

Quite simply, we have solved for $g(n)$ upto the additional constant term. After writing this as an indefinite sum, we get[4]

$$\sum_{k=1}^{n} \Delta g(k) = g(n) - g(0), \tag{11.10}$$

or equivalently

$$\sum_{k=1}^{n} f(k) = g(n) - g(0). \tag{11.11}$$

This is **indefinite summation** and finding g is called the **indefinite summation problem**, and finding g is equivalent to computing the left-hand side

[3]A set S with an ordering relation $<$ is **dense** if for *every* $a, b \in S$ and $a < b$, then there exists a $c \in S$ such that $a < c < b$.

[4]Note that this is the discrete analog to the fundamental theorem of calculus: $\int_0^x \frac{\mathrm{d}}{\mathrm{d}t} f(t)\,\mathrm{d}t = f(x) - f(0)$.

of (11.11) in "closed form". Indefinite summation is sometimes called the **antidifference** Δ^{-1}, specifically defined by

$$\Delta^{-1} f(n) := \sum_{k=1}^{n} f(k). \tag{11.12}$$

Obviously, $(\Delta^{-1} \circ \Delta) f(n) = f(n) + c$, for a constant[5] c, and $(\Delta \circ \Delta^{-1}) f(n) = f(n)$. As such, it is apparent the difference and antidifference are inverses (up to a constant), and the antidifference is the discrete analog of antidifferentiation or indefinite integration.

11.4 Gosper's Algorithm for Indefinite Summation

In 1978, Bill Gosper made his celebrated algorithm now called **Gosper's algorithm** for solving the problem of indefinite summation for a certain class of functions, viz. those sums whose summand is hypergeometric. If a hypergeometric solution exists, Gosper's algorithm will return it. Otherwise, the sum will have been proven impossible to evaluate in closed form.

Let

$$a(k) :- \Delta S_{k-1} = S_k - S_{k-1} \tag{11.13}$$

where S_k is hypergeometric. Given $a(k)$, we wish to find S_k, or equivalently, evaluating the left-hand side of

$$\sum_{k=1}^{n} a(k) = S_n - S_0. \tag{11.14}$$

Since S_k / S_{k-1} is a rational function in k, then

$$\frac{a(k)}{a(k-1)} = \frac{S_k - S_{k-1}}{S_{k-1} - S_{k-2}} = \frac{\frac{S_k}{S_{k-1}} - 1}{1 - \frac{S_{k-2}}{S_{k-1}}} \tag{11.15}$$

must also be rational in k, except when $a(k) = 0$. Express this as

$$\frac{a(k)}{a(k-1)} = \frac{P(k)}{P(k-1)} \cdot \frac{Q(k)}{R(k)} \tag{11.16}$$

with polynomials P, Q, and R in k under the condition

$$\gcd[Q(k), R(k+i)] = 1 \tag{11.17}$$

for all integers $i \geq 0$.

Lemma 11.3 It is always possible to put a rational function in the form of (11.16) under the condition (11.17).

[5]It will be advantageous to see c in the same way one views the constant of integration, since such a perspective allows one to generalize the notion of the forward difference to more arbitrary differences, such as $\Delta^k f(n) = f(n+k) - f(n)$.

Proof. Suppose that $\gcd[Q(k), R(k+i)] = \phi(k)$. We can eliminate this by computing

$$\begin{aligned} Q'(k) &:- \frac{Q(k)}{\phi(k)} \\ R'(k) &:- \frac{R(k)}{\phi(k-i)} \\ P'(k) &:- P(k) \prod_{j=0}^{i-1} \phi(k-j), \end{aligned} \tag{11.18}$$

which of course does not affect the term ratio $a(k)/a(k-1)$. The values of i for which such common factors exist can be computed via the resultant of p and q (see §9.1). ■

Now, we write

$$S_k = \frac{Q(k+1)}{P(k)} f(k) a(k) \tag{11.19}$$

where $f(k)$ shall be found. By (11.13),

$$\begin{aligned} f(k) &= \frac{P(k)}{Q(k+1)} \cdot \frac{S_k}{S_k - S_{k-1}} \\ &= \frac{P(k)}{Q(k+1)} \cdot \frac{1}{1 - \frac{S_{k-1}}{S_k}}. \end{aligned} \tag{11.20}$$

Therefore, $f(k)$ is rational iff $S(k)/S(k-1)$ is. Putting (11.19) into (11.13), one has

$$a(k) = \frac{Q(k+1)}{P(k)} f(k) a(k) - \frac{Q(k)}{P(k-1)} f(k-1) a(k-1). \tag{11.21}$$

Multiplying this by $P(k)/a(k)$ gives

$$P(k) = Q(k+1) f(k) - Q(k) \frac{P(k)}{P(k-1)} f(k-1) \frac{a(k-1)}{a(k)}. \tag{11.22}$$

And by using (11.16), we get a function equation in f:

$$P(k) = Q(k+1) f(k) - R(k) f(k-1). \tag{11.23}$$

Now we prove an important result.

Theorem 11.4 If $S_k/S_{k-1} \in \mathbb{Q}(k)$, then $f(k) \in \mathbb{Z}[k]$.

Proof. We know already that $f(k)$ is rational when $S(k)/S(k-1)$ is rational, so write

$$f(k) = \frac{\alpha(k)}{\beta(k)} \tag{11.24}$$

where $\beta(k)$ is a polynomial whose degree is positive and

$$\gcd[\alpha(k), \beta(k)] = \gcd[\alpha(k-1), \beta(k-1)] = 1. \tag{11.25}$$

Then (11.23) can be rewritten as

$$\beta(k)\beta(k-1)P(k) = \alpha(k)\beta(k-1)Q(k+1) - \alpha(k-1)\beta(k)R(k). \tag{11.26}$$

Now, let i be defined by

$$i :- \max_{x\in\mathbb{Z}} \left\{ \gcd[\beta(k), \beta(k+x)] = \phi(k) \neq 1 \right\}. \tag{11.27}$$

Of course, i exists and i is non-negative. Since i is the maximum of such values and $\phi(k) \mid \beta(k+i)$,

$$\gcd[\beta(k-1), \beta(k+i)] = \gcd[\beta(k-1), \phi(k)] = 1. \tag{11.28}$$

Applying the shift operator[6] E_{-i-1} to both sides of (11.27), we have

$$\gcd[\beta(k-i-1), \beta(k)] = \phi(k-i-1) \neq 1. \tag{11.29}$$

Similarly applying the shift operator E_{-j} to the left-hand side of (11.28) gives

$$\gcd[\beta(k-i-1), \beta(k)] = \gcd[\phi(k-i-1), \beta(k)] = 1, \tag{11.30}$$

noting that $\phi(k-i-1) \mid \beta(k-i-1)$.

Now, suppose we divide (11.26) by $\phi(k)$ and $\phi(k-i-1)$. By (11.27), we know $\phi(k) \mid \beta(k)$. Moreover, by (11.28) and (11.25), we know $\phi(k) \nmid \beta(k-1)$ and $\phi(k) \nmid \alpha(k)$. As such, in (11.26), $\phi(k) \mid Q(k+1)$ which implies $\phi(k-1) \mid Q(k)$. In a similar fashion, by (11.29), $\phi(k-i-1) \mid \beta(k-1)$. But then in that case, by (11.30) and (11.25), $\phi(k-i-1) \nmid \beta(k)$ and $\phi(k-i-1) \nmid \alpha(k-1)$. As such, in (11.26), $\phi(k-i-1) \mid R(k)$ and therefore $\phi(k-1) \mid R(k+i)$.

Thus, we can conclude i is a non-negative integer such that $\gcd[Q(k), R(k+i)] = \phi(k-1)$, contradicting (11.17), and therefore $\beta(k)$ is constant. ■

All that is left to do is find an $f(k)$ as a solution to (11.23) given $P(k)$, $Q(k)$, and $R(k)$. We can do this by establishing an upper bound δ on $\deg_k f(k)$.

First, rewrite (11.23) as

$$P(k) = \tfrac{1}{2}[Q(k+1)-R(k)][f(k)+f(k-1)] + \tfrac{1}{2}[Q(k+1)+R(k)][f(k)-f(k-1)] \tag{11.31}$$

to see that $\deg_k[f(k)+f(k-1)] = \deg_k[f(k)-f(k-1)]+1$ for any non-zero $f \in \mathbb{Z}[k]$ provided $\deg 0 = -1$.

Case I. The first possibility is

$$\deg_k[Q(k+1)+R(k)] \leq \deg_k[Q(k+1)-R(k)] = \lambda. \tag{11.32}$$

[6] Similar to the forward difference, the **shift operator** E_n is defined by $\mathsf{E}_n f(k) = f(k+n)$.

Via (11.31), we can approximate $f(n)$ and $f(n-1)$ by

$$\alpha(\delta)k^{\delta} + \mathrm{O}(k^{\delta-1}), \tag{11.33}$$

N4. Need to write a chapter on approximation, asymptotics, and big-O.

where O is "big-O" notation. Therefore,

$$P(k) = C\alpha(\delta)k^{\delta+\lambda} + \mathrm{O}(k^{\delta+\lambda-1}) \tag{11.34}$$

for some constant $C \neq 0$. Since the degrees of the left- and right-hand sides must be identical,

$$\delta = \deg_k f(k) = -\lambda + \deg_k P(n). \tag{11.35}$$

Case II. The second possibility is

$$\deg_k[Q(k+1) - R(k)] < \deg_k[Q(k+1) + R(k)] = \lambda. \tag{11.36}$$

Again estimating, we have

$$f(k) = \alpha(\delta)k^{\delta} + \alpha(\delta-1)k^{\delta-1} + \mathrm{O}(k^{\delta-2}) \tag{11.37}$$

and putting

$$f(k-1) = \alpha(\delta)k^{\delta} + (\alpha_{\delta-1} - \delta\alpha_{\delta})k^{\delta-1} + \mathrm{O}(k^{\delta-2}) \tag{11.38}$$

into (11.31) gives

$$P(k) = (C' + C\delta)\alpha_{\delta}k^{\delta+\lambda-1} + \mathrm{O}(k^{\delta+\lambda-2}), \tag{11.39}$$

for constants C and C'. Define $\delta_0 :- -C'/C$, the root of the leading term of the right-hand side. Then the maximal δ such that $\alpha_k \neq 0$ is

$$\delta = \begin{cases} \max\{\delta_0, 1-\lambda+\deg_k P(k)\} & \text{if } \delta_0 \in \mathbb{Z}, \\ 1-\lambda+\deg P(k) & \text{otherwise.} \end{cases} \tag{11.40}$$

In both cases, if $\delta < 0$, then a hypergeometric S_k certainly does not exist. Otherwise, we can put $f(k)$ as a δth degree polynomial with $\delta+1$ unknown coefficients in (11.23), and equate coefficients, giving us a linear system. If the system is insoluble, then a hypergeometric S_k does not exist. Otherwise, solving the system $f(k)$, which can be substituted in (11.19) to obtain S_k.

11.5 Zeilberger's Algorithm for Definite Summation

Doron Zeilberger extended Gosper's algorithm in a non-trivial fashion to work with sums of the form

$$\sum_{k=1}^{n} g(n,k) = A(n),$$

Algorithm 11.1: `GosperSum`(t_n)

```
Input  : A hypergeometric term $t_n$
Output : A hypergeometric term $z_n$ satisfying $z_{n+1} - z_n = t_n$ or FAIL.
$r(n) := t_{n+1}/t_n$
/* Write $r(n) = \frac{a(n)}{b(n)} \frac{c(n+1)}{c(n)}$ such that $\gcd[a(n), b(n+k)] = 1$ for all $k \geq 0$ */
begin
    Let $r(n) := K\frac{f(n)}{g(n)}$ where $f(n)$ and $g(n)$ are monic and relatively
    prime, and $K$ is constant.
    $R(k) := \mathrm{res}_n(f(n), g(n+k))$
    Let $S = \{k_1, \dots, k_N\}$ be the set of non-negative integer zeros of $R(k)$
    ($N \geq 0$ and $0 \leq k_1 < \cdots < k_N$).
    $p_0(n) := f(n)$; $q_0(n) := g(n)$
    for $1 \leq j \leq N$ do
        $s_j(n) := \gcd(p_{j-1}(n), q_{j-1}(n + k_j))$
        $p_j(n) := p_{j-1}(n)/s_j(n)$
        $q_j(n) := q_{j-1}(n)/s_j(n - k_j)$
    $a(n) := Kp_N(n)$
    $b(n) := q_N(n)$
    $c(n) := \prod_{i=1}^{N} \prod_{j=1}^{k_i} s_i(n - j)$
/* Find $x(n) \neq 0$ such that $a(n)x(n+1) - b(n-1)x(n) = c(n)$ if one exists,
   otherwise return FAIL */
begin
    if $\deg a(n) \neq \deg b(n) \vee \mathrm{lcoeff}\, a(n) \neq \mathrm{lcoeff}\, b(n)$ then
        $D := \{\deg c(n) - \max\{\deg a(n), \deg b(n)\}\}$
    else
        $A := [n^{k-1}]a(n)$
        $B := [n^{k-1}]b(n-1)$
        $D := \{1 + \deg c(n) - \deg a(n), (B - A)/\,\mathrm{lcoeff}\, a(n)\}$
    $D := D \cap \mathbb{N}$
    if $D = \varnothing$ then
        Print "No non-zero polynomial solution."
        return FAIL
    else
        $d := \max D$
    Using the method of undetermined coefficients, find a non-zero
    polynomial solution $x(n)$ of $a(n)x(n+1) - b(n-1)x(n) = c(n)$
    degree $d$ or less.
    if no $x(n)$ exists then
        Print "No non-zero polynomial solution."
        return FAIL
return $\frac{b(n-1)x(n)}{c(n)} t_n$
```

called an **indefinite sum**. Dividing both sides of the indefinite sum by $A(n)$ and letting $F(n,k) := g(n,k)/A(n)$ gives

$$\sum_{k=1}^{n} F(n,k) = 1, \tag{11.41}$$

which implies that

$$\sum_{k=1}^{n} \Delta_n F(n,k) = 0, \tag{11.42}$$

In Zeilberger's algorithm, we try to write $F(n+1,k) - F(n,k)$ as a discrete difference ΔS_k. Suppose

$$\Delta_n F(n,k) = \Delta_k G(n,k). \tag{11.43}$$

We can find G via Gosper's algorithm via

$$\sum_{k=-M}^{N} \Delta_n F(n,k) = G(n, M+1) - G(n, -N). \tag{11.44}$$

If we assume

$$\lim_{k\to\pm\infty} G(n,k) = 0, \tag{11.45}$$

then $\sum_{k=1}^{n} F(n,k)$ is constant.

Therefore, to prove an identity of the form (11.42), we find a G that satisfies (11.43) and (11.45), and then show it holds for a value of n.

Example 11.1 Consider

$$\sum_{k=0}^{n} \frac{(-1)^k(-n)^{\uparrow k}}{k!\,2^n}.$$

Obviously,

$$F(n,k) = \frac{(-1)^k(-n)^{\uparrow k}}{k!\,2^n}$$

and one can determine that

$$G(n,k) = \frac{(-1)^k(-n)^{\uparrow k-1}}{(k-1)!\,2^{n+1}}$$

satisfies (11.42) and (11.45). Letting $n = 0$, we have

$$\begin{aligned}\sum_{k=0}^{0} \frac{(-1)^k(-0)^{\uparrow k}}{k!\,2^0} &= \frac{1\cdot 0^{\uparrow 0}}{1\cdot 1}\\ &= 1,\end{aligned}$$

verifying the identity.

Part VI

Integration

Chapter 12

Risch Integration

12.1 Preliminaries

Differential Fields

Definition 12.1 A **differential field** is a field $(F;+,\cdot)$ equipped with the **differential operator** $\mathcal{D}\colon\ F \to F$ with the following properties:

1. $\mathcal{D}(f+g) = \mathcal{D}\,f + \mathcal{D}\,g$,
2. $\mathcal{D}(f \cdot g) = (\mathcal{D}\,f)\cdot g + f \cdot \mathcal{D}\,g$

Generally we will assume that $x \in F$ and $\mathcal{D}\,x = 1$.

Definition 12.2 A **constant field** is a differential field K such that $\mathcal{D}\,f = 0$ for all $f \in K$.

Definition 12.3 Let K be a constant field and $F = K(\theta_1, \ldots, \theta_n)$ be a differentiable field. Furthermore let $F_k = K(\theta_1, \ldots, \theta_k)$. Then F is an **elementary field** if $\mathcal{D}\,\theta_1 = 1$ and for $2 \leq k \leq n$, one of the following holds:

1. θ_k is algebraic over F_{k-1},
2. $\mathcal{D}\,\theta_k = (\mathcal{D}\,\eta)/\eta$ for some $\eta \in F_{k-1}$, or
3. $\mathcal{D}\,\theta_k = (\mathcal{D}\,\eta)\theta_k$ for some $\eta \in F_{k-1}$.

A **purely transcendental field** is an algebraic field in which θ_k is transcendental over F_k for $1 \leq k \leq n$.

Transcendental Functions

Definition 12.4 An **elemenetary function** is a function $x \mapsto f(x)$ such that $f(x)$ is built up from constants, x, $+$, $-$, $\cdot$, $/$, exp, and ln.

The purpose of elementary functions is two-fold: (1) to provide a consistent way to view a variety of functions as consituent functions and (2) to take advantage of the property that if f is in an elementary field F, then $\int f$ is also in F possibly extended with constant algebraics or logarithms.

12.2 Risch Algorithm: Input

The **Risch algorithm** is a procedure to determine if, for some elementary function f, if there is an elementary function g such that $f = \mathcal{D}\, g$. This is known as **indefinite integration** as from elementary calculus. Since by the fundamental theorem of calculus[1], we may speak of an inverse of differentiation $\mathcal{D}^{-1}$ making the goal now to find $g = \mathcal{D}^{-1} f$, more commonly written $g = \int f$.

Decomposing the Input

The first part of the algorithm deals with decomposing the input into the "core" elementary functions and then determining the field in which the integrand lies.

Example 12.1 Let $f = \mathrm{e}^x + \mathrm{e}^{-x}$. We start from the inside and move out.

$$\begin{aligned} \mathrm{e}^x + \mathrm{e}^{-x} &= \mathrm{e}^{\theta_1} + \mathrm{e}^{-\theta_1} & \theta_1 &:= x \\ &= \theta_2 + \mathrm{e}^{-\theta_1} & \theta_2 &:= \mathrm{e}^{\theta_1} \\ &= \theta_2 + \theta_3 & \theta_3 &:= \mathrm{e}^{-\theta_1} \end{aligned}$$

We can relate these quantities by seeing that $\theta_2 = \theta_3^{-1}$. So, $f = \theta_2 + \theta_2^{-1} \in \mathbb{Q}(\theta_1, \theta_2)$.

Analyzing the Transcendence of the Input

Theorem 12.1 (Risch Structure Theorem) Let $F = K(\theta_1, \ldots, \theta_n)$ be a differential field and let $\eta \in F$. Let $E = \{n \mid \theta_n = \mathrm{e}^{\eta_n}\}$ and $L = \{n \mid \theta_n = \ln \eta_n\}$. Assume that $|E| + |L| = n - 1$ (because $\theta_1 = x$). Then

1. e^η is algebraic over F iff there exists a $c \in K$ and rationals r_i such that

$$\eta = c + \sum_{i \in E} r_i \eta_i + \sum_{i \in L} r_i \theta_i, \text{ or} \tag{12.1}$$

2. $\ln \eta$ is algebraic over F iff there exists a $c \in K$ and rationals r_i such that

$$\eta = c \prod_{i \in E} \theta_i^{r_i} \cdot \prod_{i \in L} \eta_i^{r_i} \iff \frac{\mathcal{D}\,\eta}{\eta} = \sum_{i \in E} r_i (\mathcal{D}\,\eta_i) + \sum_{i \in L} r_i \frac{\mathcal{D}\,\eta_i}{\eta_i}. \tag{12.2}$$

The Risch structure theorem is useful for determining if an extension e^η or $\ln \eta$ is transcendental in F; (12.1) and (12.2) can be solved simultaneously to find c and r_i or prove they do not exist.

[1]Recall that it states that derivatives and integrals are essentially opposite functions.

Normalization and Reduction of Differential Fields

Originally, Risch's proposition deemed it necessary to solve the **Risch differential equation** $y' + fy = g$ which requires tedious calculations of degree bounds and solving large systems of simultaneous equations. Davenport [Dav86] proposed a way to modify the fields in order to avoid this necessity.

Suppose we want to extend the differential field $F = K(\theta_1, \ldots, \theta_n)$ with the exponential e^{η}. If η depends on some ϕ and $\eta \in K(\theta_1, \ldots, \theta_n, \phi)$, then let the Laurent series expansion of η with respect to ϕ be

$$\eta = \cdots + \eta_{-1}\phi^{-1} + \eta_0 + \eta_1\phi + \cdots = \sum_{k\in\mathbb{Z}} \eta_k \phi^k \tag{12.3}$$

for coefficients ϕ_k. Then e^{η} is **reduced** if $\eta_1 \notin \mathbb{Q}$ or ϕ is not logarithmic. If this is not true, then $\eta_1\phi = \rho \ln \zeta$ and $\zeta^{\rho}\mathrm{e}^{\eta - \eta_1\phi}$ should be input instead of e^{η}.

Appendix

Appendix A

Function Tables

A.1 Elementary Functions

Basic Arithmetic Functions

$$\mathrm{e}^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \qquad \ln x = \int_1^x \frac{\mathrm{d}t}{t} \tag{A.1}$$

$$x^y = \mathrm{e}^{y \ln x} \qquad \sqrt[n]{x} = \mathrm{e}^{\frac{1}{n} \ln x} \tag{A.2}$$

Trigonometric Functions

Let $k \in \mathbb{Z}$.

$$\sin x = \frac{\mathrm{e}^{\mathrm{i}x} - \mathrm{e}^{-\mathrm{i}x}}{2\mathrm{i}} \qquad \sin^{-1} y = 2k\pi - \mathrm{i}\ln\left(\mathrm{i}y \pm \sqrt{1-y^2}\right) \tag{A.3}$$

$$\cos x = \frac{\mathrm{e}^{\mathrm{i}x} + \mathrm{e}^{-\mathrm{i}x}}{2} \qquad \cos^{-1} y = 2k\pi - \mathrm{i}\ln\left(y \pm \sqrt{y^2-1}\right) \tag{A.4}$$

$$\tan x = \frac{\mathrm{e}^{\mathrm{i}x} - \mathrm{e}^{-\mathrm{i}x}}{\mathrm{i}(\mathrm{e}^{\mathrm{i}x} + \mathrm{e}^{-\mathrm{i}x})} \qquad \tan^{-1} y = k\pi - \frac{1}{2}\mathrm{i}\ln\left(\frac{\mathrm{i}-y}{\mathrm{i}+y}\right) \qquad y \neq \pm\mathrm{i} \tag{A.5}$$

$$\sec x = \frac{2}{\mathrm{e}^{\mathrm{i}x} + \mathrm{e}^{-\mathrm{i}x}} \qquad \sec^{-1} y = 2k\pi - \mathrm{i}\ln\left(\frac{1 \pm \sqrt{1-y^2}}{y}\right) \qquad y \neq 0 \tag{A.6}$$

$$\csc x = \frac{2\mathrm{i}}{\mathrm{e}^{\mathrm{i}x} - \mathrm{e}^{-\mathrm{i}x}} \qquad \csc^{-1} y = 2k\pi - \mathrm{i}\ln\left(\frac{\mathrm{i} \pm \sqrt{y^2-1}}{y}\right) \qquad y \neq 0 \tag{A.7}$$

$$\cot x = \frac{\mathrm{i}(\mathrm{e}^{\mathrm{i}x} + \mathrm{e}^{-\mathrm{i}x})}{\mathrm{e}^{\mathrm{i}x} - \mathrm{e}^{-\mathrm{i}x}} \qquad \cot^{-1} y = k\pi - \frac{1}{2}\mathrm{i}\ln\left(\frac{y+\mathrm{i}}{y-\mathrm{i}}\right) \qquad y \neq \pm\mathrm{i} \tag{A.8}$$

Hyperbolic Functions

Let $k \in \mathbb{Z}$.

$$\sinh x = \frac{\mathrm{e}^x - \mathrm{e}^{-x}}{2} \quad \sinh^{-1} y = 2k\pi\mathrm{i} + \ln\left(y \pm \sqrt{y^2+1}\right) \tag{A.9}$$

$$\cosh x = \frac{\mathrm{e}^x + \mathrm{e}^{-x}}{2} \quad \cosh^{-1} y = 2k\pi\mathrm{i} + \ln\left(y \pm \sqrt{y^2-1}\right) \tag{A.10}$$

$$\tanh x = \frac{\mathrm{e}^x - \mathrm{e}^{-x}}{\mathrm{e}^x + \mathrm{e}^{-x}} \quad \tanh^{-1} y = k\pi\mathrm{i} + \frac{1}{2}\ln\left(\frac{1+y}{1-y}\right) \qquad y \neq \pm 1 \tag{A.11}$$

$$\operatorname{sech} x = \frac{2}{\mathrm{e}^x + \mathrm{e}^{-x}} \quad \operatorname{sech}^{-1} y = 2k\pi\mathrm{i} + \ln\left(\frac{1 \pm \sqrt{1-y^2}}{y}\right) \qquad y \neq 0 \tag{A.12}$$

$$\operatorname{csch} x = \frac{2}{\mathrm{e}^x - \mathrm{e}^{-x}} \quad \operatorname{csch}^{-1} y = 2k\pi\mathrm{i} + \ln\left(\frac{1 \pm \sqrt{1+y^2}}{y}\right) \qquad y \neq 0 \tag{A.13}$$

$$\coth x = \frac{\mathrm{e}^x + \mathrm{e}^{-x}}{\mathrm{e}^x - \mathrm{e}^{-x}} \quad \coth^{-1} y = k\pi\mathrm{i} + \frac{1}{2}\ln\left(\frac{y+1}{y-1}\right) \qquad y \neq \pm 1 \tag{A.14}$$

A.2 Gamma and Friends

Rising Power

$$(x+1)^{\uparrow k} = \frac{x^{\uparrow k+1}}{x} \tag{A.15}$$

$$\frac{x^{\uparrow m}}{x^{\uparrow n}} = (x+n)^{\uparrow m-n} \tag{A.16}$$

$$1^{\uparrow k} = k! \tag{A.17}$$

$$(-m)^{\uparrow k} = 0 \qquad \text{for } k > m \tag{A.18}$$

Factorial, Double Factorial, and Gamma

$$k! = \Gamma(k+1) = k\Gamma(k) \tag{A.19}$$

$$(2k)! = 2^{2k}(1/2)^{\uparrow k}k! \tag{A.20}$$

$$(z-1)!! = \frac{z!}{z!!} \tag{A.21}$$

$$(2z)!! = 2^z z! \tag{A.22}$$

$$\Gamma(2z) = \frac{2^{2z-1}}{\sqrt{\pi}}\Gamma(z)\Gamma(z+1/2) \tag{A.23}$$

$$\Gamma(z+k) = \Gamma(z)z^{\uparrow k} \qquad \text{for } k \in \mathbb{Z} \tag{A.24}$$

$$\Gamma(z-k) = \frac{(-1)^k\Gamma(z)}{(1-z)^{\uparrow k}} \tag{A.25}$$

$$\Gamma(z+1/2) = \frac{\sqrt{\pi}}{2^z}(2z-1)!! \qquad \text{for non-negative } z \in \mathbb{Z} \tag{A.26}$$

$$\frac{(2z)!}{z!^2} = \frac{2^{2z}}{\sqrt{\pi}}\frac{\Gamma(z+1/2)}{\Gamma(z+1)} \tag{A.27}$$

Binomial Coefficients and Sums

$$\binom{2n}{2k} = \frac{(1/2-n)^{\uparrow k}(-n)^{\uparrow k}}{(1/2)^{\uparrow k}k!} \tag{A.28}$$

$$\sum_{k=0}^{m}(-1)^k\binom{m}{k}\frac{\Gamma(k+b)}{\Gamma(k+a)} = \frac{\Gamma(b)\Gamma(m+a-b)}{\Gamma(m+a)\Gamma(a-b)} \tag{A.29}$$

A.3 Hypergeometric Identities

$$\mathrm{F}^2_1\left[\left.\begin{matrix}a,b\\c\end{matrix}\right|1\right]=\frac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)} \tag{A.30}$$

$$\mathrm{F}^2_1\left[\left.\begin{matrix}a,b\\c\end{matrix}\right|z\right]=(1-z)^{c-a-b}\mathrm{F}^2_1\left[\left.\begin{matrix}c-a,c-b\\c\end{matrix}\right|z\right] \tag{A.31}$$

$$\mathrm{F}^2_1\left[\left.\begin{matrix}a,1-a\\c\end{matrix}\right|\frac{1}{2}\right]=\frac{\Gamma[(c+1)/2]\Gamma(c/2)}{\Gamma[(1-a+c)/2]\Gamma[(a+c)/2]} \tag{A.32}$$

$$\begin{aligned}\mathrm{F}^2_1\left[\left.\begin{matrix}a,b\\c\end{matrix}\right|z\right]&=\frac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)}\mathrm{F}^2_1\left[\left.\begin{matrix}a,b\\a+b-c+1\end{matrix}\right|1-z\right]\\&\quad+(1-z)^{c-a-b}\frac{\Gamma(c)\Gamma(a+b-c)}{\Gamma(a)\Gamma(b)}\mathrm{F}^2_1\left[\left.\begin{matrix}c-a,c-b\\c-a-b+1\end{matrix}\right|1-z\right]\end{aligned} \tag{A.33}$$

$$\begin{aligned}\mathrm{F}^p_q\left[\left.\begin{matrix}a_1,\ldots,a_{p-1},1\\b_1,\ldots,b_{q-1},n\end{matrix}\right|z\right]&=\frac{n-1}{z}\cdot\frac{\prod_{j=1}^{q-1}(b_j-1)}{\prod_{j=1}^{p-1}(a_j-1)}\\&\quad\times\mathrm{F}^p_1\left[\left.\begin{matrix}a_1-1,\ldots,a_{p-1}-1,1\\b_1-1,\ldots,b_{q-1}-1,n-1\end{matrix}\right|z-1\right]\end{aligned} \tag{A.34}$$

$$\mathrm{F}^2_1\left[\left.\begin{matrix}a,b\\1+b-a\end{matrix}\right|-1\right]=\frac{\sqrt{\pi}}{2^b}\frac{\Gamma(1+b-a)}{\Gamma(1+b/2-a)\Gamma[(1+b)/2]}\qquad\text{for } 1+b-a\notin\mathbb{Z}^- \tag{A.35}$$

A.4 Miscellaneous

$$\sum_{k=n+1}^{\infty}\frac{n}{k^2-n^2}=\sum_{k=1}^{2n}\frac{1}{2k} \tag{A.36}$$

Appendix B

Miscellaneous

B.1 The Gudermannian Function

The **Gudermannian function** is defined as

$$\begin{aligned}\operatorname{gd} x &:= \int_0^x \frac{\mathrm{d}t}{\cosh t}\\ &= \sin^{-1}(\tanh x)\\ &= \tan^{-1}(\sinh x)\\ &= 2\tan^{-1}\left(\tan\frac{x}{2}\right)\\ &= 2\tan^{-1}\mathrm{e}^x - \frac{\pi}{2}.\end{aligned}$$

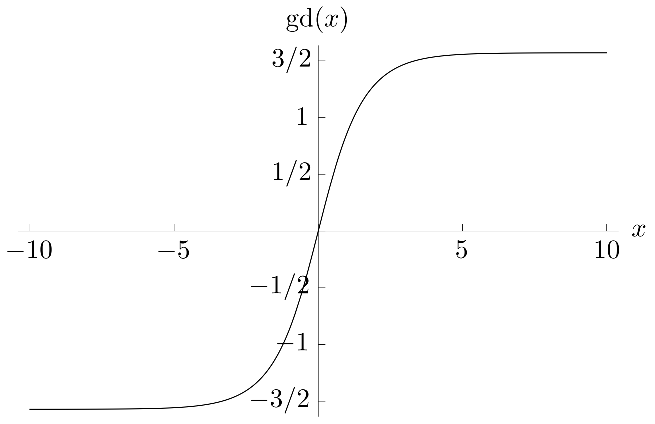

Figure B.1. Plot of the Gudermannian function $\operatorname{gd}(x)$.

The **inverse Gudermannian function** is defined for $x \in (-\pi/2, \pi/2)$ as

$$\begin{aligned}
\operatorname{gd}^{-1} x &:= \int_0^x \frac{\mathrm{d}t}{\cos t} \\
&= \ln\left|\frac{1+\sin x}{\cos x}\right| \\
&= \frac{1}{2}\ln\left|\frac{1+\sin x}{1-\sin x}\right| \\
&= \ln|\tan x + \sec x| \\
&= \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right| \\
&= \tan^{-1}(\sin x) \\
&= \sinh^{-1}(\tan x).
\end{aligned}$$

The Gudermannian function relates the circular trigonometric functions with the hyperbolic functions without using complex numbers. The following identities hold for the function:

$$\begin{aligned}
\sin(\operatorname{gd} x) &= \tanh x & \csc(\operatorname{gd} x) &= \coth x \\
\cos(\operatorname{gd} x) &= \operatorname{sech} x & \sec(\operatorname{gd} x) &= \cosh x \\
\tan(\operatorname{gd} x) &= \sinh x & \cot(\operatorname{gd} x) &= \operatorname{csch} x \\
\tan\frac{\operatorname{gd} x}{2} &= \tanh\frac{x}{2}.
\end{aligned}$$

The derivative of the function is

$$\frac{\mathrm{d}}{\mathrm{d}x}\operatorname{gd} x = \operatorname{sech} x \quad \text{and} \quad \frac{\mathrm{d}}{\mathrm{d}x}\operatorname{gd}^{-1} x = \sec x,$$

and the integral

$$\int \operatorname{gd} x \,\mathrm{d}x = -\frac{\pi}{2}x + \mathrm{i}[\operatorname{Li}_2(-\mathrm{i}\mathrm{e}^x) - \operatorname{Li}_2(\mathrm{i}\mathrm{e}^x)]$$

where $\operatorname{Li}_2 x = \sum_{k=1}^{\infty} x^k/k^2$ is the **dilogarithm**.

The Gudermannian function is related to its own inverse in a non-trivial way: $\operatorname{gd} \mathrm{i}x = \mathrm{i}\operatorname{gd}^{-1} x$. It is related to the exponential function via

$$\begin{aligned}
\mathrm{e}^x &= \sec(\operatorname{gd} x) + \tan(\operatorname{gd} x) \\
&= \tan(\tfrac{\pi}{4} + \tfrac{1}{2}\operatorname{gd} x) \\
&= \frac{1+\sin(\operatorname{gd} x)}{\cos(\operatorname{gd} x)}.
\end{aligned}$$

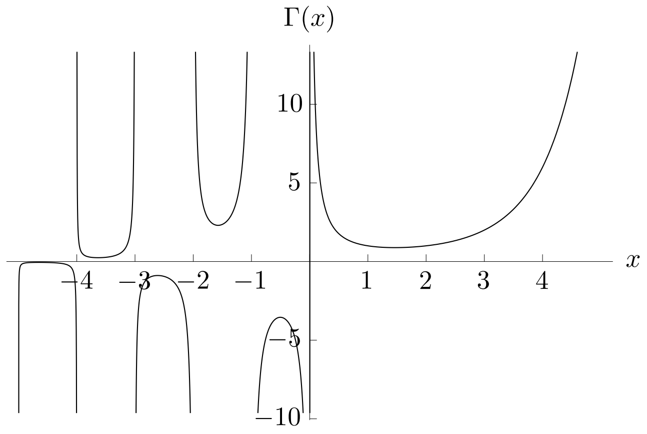

Figure B.2. Plot of the Γ-function.

B.2 The Γ-Function

Infinite Limit

The first definition, due to Euler, of $\Gamma\colon \mathbb{C} \to \mathbb{C}$ is

$$\Gamma(z) := \lim_{n\to\infty} \frac{1 \cdot 2 \cdots n}{z(z+1)\cdots(z+n)} n^z. \tag{B.1}$$

Substituting z with $z+1$ we have

$$\begin{aligned}
\Gamma(z+1) &= \lim_{n\to\infty} \frac{1 \cdot 2 \cdots n}{(z+1)(z+2)\cdots(z+n)(z+n+1)} n^{z+1} \\
&= \lim_{n\to\infty} \frac{nz}{z+n+1} \cdot \frac{1 \cdot 2 \cdots n}{z(z+1)(z+2)\cdots(z+n)} n^z \\
&= z \lim_{n\to\infty} \frac{n}{z+n+1} \lim_{n\to\infty} \frac{1 \cdot 2 \cdots n}{z(z+1)(z+2)\cdots(z+n)} n^z \\
&= z\Gamma(z) \qquad (\text{B.2})
\end{aligned}$$

This is one of the basic identities of the Γ-function, and it is a *difference* equation. It has been shown that the Γ-function of of a class of functions that does not satisfy any *differential* equations with rational coefficients.

Lastly, we show the Γ-function is equivalent to the factorial function (up to shifts). Compute Γ(1):

$$\begin{aligned}
\Gamma(1) &= \lim_{n\to\infty} \frac{1 \cdot 2 \cdots n}{1 \cdot 2 \cdots n(n+1)} n \\
&= \lim_{n\to\infty} \frac{n}{n+1} \\
&= 1
\end{aligned}$$

Using this with (B.2), we have

$$\begin{aligned}
\Gamma(2) &= 1 \cdot \Gamma(1) = 1 \\
\Gamma(3) &= 2 \cdot \Gamma(2) = 2 \\
&\vdots \\
\Gamma(n) &= (n-1)\cdots 2 \cdot 1 = (n-1)!, \qquad n \in \mathbb{N}.
\end{aligned}$$

Definite integral

Also due to Euler, we have

$$\Gamma(z) := \int_0^\infty t^{z-1} \mathrm{e}^{-t}\, \mathrm{d}t, \qquad \operatorname{Re} z > 0. \tag{B.3}$$

We restrict z to avoid divergence of the integral. Many variations of (B.3) appear, such as

$$\Gamma(z) = 2\int_0^\infty t^{2z-1} \mathrm{e}^{t^2}\, \mathrm{d}t, \qquad \operatorname{Re} z > 0 \tag{B.4}$$

and

$$\Gamma(z) = \int_0^1 \left(\ln \frac{1}{t}\right)^{z-1} \mathrm{d}t, \qquad \operatorname{Re} z > 0. \tag{B.5}$$

We see immediately that (B.4) is the Gauss error function, and can conclude

$$\Gamma(\tfrac{1}{2}) = \sqrt{\pi}.$$

Theorem B.1 The limit definition and definite integral definition are equivalent.

Proof. To show that the integral and limit definitions are equivalent, consider the functions

$$f_n(z) = \int_0^n \left(1 - \frac{t}{n}\right)^n t^{z-1}\, \mathrm{d}t, \qquad \operatorname{Re} z > 0 \tag{B.6}$$

for integral $n > 0$. Since $\lim_{n\to\infty}(1 - t/n)^n \equiv \mathrm{e}^{-t}$ by definition, we conclude that $\lim_{n\to\infty} f_n(z) = \Gamma(z)$ by (B.3).

Rewrite (B.6) with $s = t/n$ for convenience:

$$f_n(z)n^{-z} = \int_0^1 (1-s)^n s^{z-1}\, \mathrm{d}s. \tag{B.7}$$

Using integration by parts, we have

$$f_n(z)n^{-z} = \left[(1-s)^n \frac{s^z}{z}\right]_{s=0}^{s=1} + \frac{n}{z}\int_0^1 (1-s)^{n-1} s^z\, \mathrm{d} = \frac{n}{z}\int_0^1 (1-s)^{n-1} s^z\, \mathrm{d}s. \tag{B.8}$$

Repeating this a total of n times gives

$$\begin{aligned} f_n(z)n^{-z} &= \frac{n(n-1)\cdots 21}{z(z+1)\cdots(z+n)} \int_0^1 s^{z+n-1}\,\mathrm{d}s \\ &= \frac{1\cdot 2\cdots n}{z(z+1)\cdots(z+n)} \\ f_n(z) &= \frac{1\cdot 2\cdots n}{z(z+1)\cdots(z+n)} n^z. \end{aligned}$$

Letting $n \to \infty$ gives us (B.1) identically, showing the definitions are equal. ■

Infinite product

A third way of defining the Γ-function is via an infinite product due to Weierstrass:

$$\frac{1}{\Gamma(z)} := z\mathrm{e}^{\gamma z} \prod_{k=1}^{\infty} \left(1 + \frac{z}{k}\right) \mathrm{e}^{-z/k} \tag{B.9}$$

where $\gamma = 0.577216\ldots$ is the Euler-Mascheroni constant.

Theorem B.2 The infinite product definition is equivalent to the limit definition.

Proof. Rewrite (B.1) as

$$\Gamma(z) = \lim_{n\to\infty} \frac{1}{z} \prod_{k=1}^{n} \left(1 + \frac{z}{k}\right)^{-1} n^z. \tag{B.10}$$

Reciprocating and using $n^{-z} = \mathrm{e}^{-z\ln n}$, we have

$$\frac{1}{\Gamma(z)} = z \lim_{n\to\infty} \mathrm{e}^{-z\ln n} \prod_{k=1}^{n} \left(1 + \frac{z}{k}\right). \tag{B.11}$$

Recall the **nth harmonic number** is $H_n := \sum_{j=1}^{n} j^{-1}$. Multiplying and dividing by

$$\mathrm{e}^{H_n z} = \prod_{k=1}^{n} \left(1 + \frac{z}{k}\right)$$

gives

$$\frac{1}{\Gamma(z)} = z \lim_{n\to\infty} \mathrm{e}^{(H_n - \ln n)z} \prod_{k=1}^{n} \left(1 + \frac{z}{k}\right) \mathrm{e}^{-z/k}.$$

By definition,

$$\gamma = \lim_{n\to\infty} (H_n - \ln n),$$

and thus we arrive at (B.9). ■

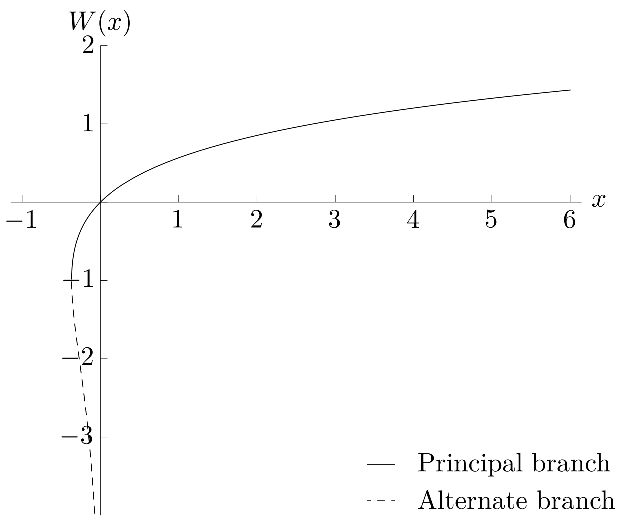

Figure B.3. Plot of the Lambert W-function.

B.3 The Lambert W-Function

The **Lambert W-function**, after Johann Heinrich Lambert, is the inverse of the complex map $z \mapsto z\mathrm{e}^z$. The function $W(z)$ therefore satisfies the functional relation

$$z = W(z)\mathrm{e}^{W(z)}. \tag{B.12}$$

If $z \in \mathbb{R}$ and we consider only the real values of W, then W has is defined (one one branch) over $[-1/\mathrm{e}, \infty)$, where $W(-1/\mathrm{e})$ is the minimum. This is the principal branch, and is monotonically increasing. The other branch, which we will denote W_{-1}, is defined over $[-1/\mathrm{e}, 0)$ with $\lim_{x\to 0} W_{-1}(x) = -\infty$.

$$\frac{\mathrm{d}W}{\mathrm{d}z} = \frac{W(z)}{z[1 + W(z)]} \qquad \text{for } z \neq -1/\mathrm{e} \tag{B.13}$$

$$\int W(x)\,\mathrm{d}x = x\left[W(x) + \frac{1}{W(x)} - 1\right] + C \tag{B.14}$$

Bibliography

[AS64] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth Dover printing, tenth GPO printing edition, 1964.

[CC93] D. V. Chudnovsky and G. V. Chudnovsky. Hypergeometric and modular function identities, and new rational approximations and continued fraction expansions of classical constants and functions. *Contemporary Mathematics*, 143:117–162, 1993. 5.3

[Dav86] J. H. Davenport. The Risch differential equation problem. *SIAM J. Comput.*, 15(4):903–918, 1986. 12.2

[Gau23] Carl Friedrich Gauss. *Werke*, chapter Disquisitiones Generales Circa Seriem Infinitam $1 + \frac{\alpha\beta}{1.\gamma}x + \frac{\alpha(\alpha+1)\beta(\beta+1)}{1.2.\gamma(\gamma+1)}xx + \frac{\alpha(\alpha+1)(\alpha+2)\beta(\beta+1)(\beta+2)}{1.2.3.\gamma(\gamma+1)(\gamma+2)}x^3 +$etc., pages 125–126. Köngliche Gesellschaft der Wissenschaft, 1823. 11.1

[GCL92] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for computer algebra*. Kluwer Academic Publishers, 1992. 4.2

[Gos78] Ralph William Gosper, Jr. Decision Procedure for Indefinite Hypergeometric Summation. *Proceedings of the National Academy of Science*, 75:40–42, January 1978.

[IEE85] IEEE. IEEE standard for binary floating-point arithmetic. *ACM SIGPLAN Notices*, 22(2):9–25, February 1985. 5.1

[Koe98] Wolfram Koepf. *Hypergeometric Summation: An Algorithmic Approach to Summation and Special Function Identities*. Vieweg Verlag, 1998.

[PWZ96] M. Petkovšek, Herbert S. Wilf, and Doron Zeilberger. $A = B$. A. K. Peters, Ltd., 1996. 1

[Ris69] Robert H. Risch. The problem of integration in finite terms. *Transactions of the American Mathematical Society*, 139:167–189, 1969.

[Wei10] Eric W. Weisstein. Generalized Hypergeometric Function — from Wolfram MathWorld, February 2010. 1

[X3J94] ANSI Subcommittee X3J13. Programming language—Common Lisp. Technical Report ANSI INCITS 226-1994 (R2004), American National Standards Institute, Inc., 1994. 4.2

Index

Symbols

n-tuple . . . 12
nth harmonic number . . . 81

A

abelian . . . 31
absolute value . . . 50
accessor functions . . . 12
adjunction . . . 39
algebraic . . . 41
algebraic extension . . . 41
antidifference . . . 56
arbitrary precision arithmetic . . . 24
associate classes . . . 33
associates . . . 33

B

base . . . 25
base types . . . 11
bignum . . . 24
binary relation . . . 17
binary splitting . . . 26

C

canonical form . . . 19
canonizing function . . . 20
commutative group . . . 31
commutative ring . . . 31
constant . . . 7
constant field . . . 65
constructions . . . 8
constructor . . . 8

D

data-level equality . . . 18
degree . . . 39
dense . . . 55
dependent types . . . 14
differential field . . . 65
differential operator . . . 65
dilogarithm . . . 78
discrete operator . . . 55
divides . . . 32
divisor . . . 32
domain . . . 7

E

elemenetary function . . . 65
elementary field . . . 65
equivalence class . . . 18
equivalence relation . . . 18
Euclidean domain . . . 36
exponent . . . 23
extension field . . . 39

F

field . . . 32
field extension . . . 39
form-level equality . . . 18
forward difference . . . 55
forward partial difference . . . 55
function type . . . 12

G

GCD . . . 32

generalized hypergeometric function 54
generalized hypergeometric series 53
geometric series 53
Gosper's algorithm 57
greatest common divisor 32
group 31
Gudermannian function 77

H
hypergeometric 53

I
identities 20
identity element 31
indefinite integration 66
indefinite sum 62
indefinite summation 56
indefinite summation problem . . 56
integral domain 31
intermediate field 39
inverse 31
inverse Gudermannian function . 78
invertible 33
irreducible 35

K
kernel 5

L
Lambert W-function 82
LCM 32
least common multiple 32
leaves 7
level-0 19
level-1 19
level-2 19
limb 25
list 13

M
mantissa 23
map 8
mapping 8
mathematical equality 18
minimal polynomial 47
multiple 32
multiple precision arithmetic . . . 24

N
negative elements 49
normal form 21
normal part 34
normalizing function 21
nullary 7

O
object-level equality 18
ordered field 49

P
parametric type 14
Pochhammer symbol 53
positive elements 49
prime 35
primitive element 39
product type 12
purely transcendental field 65

Q
quotient 36

R
radix 25
radix form 24
real algebraic numbers 40
recursive type 13
reduced 67
relatively prime 35
resultant 45
ring 31
Risch algorithm 66

Risch differential equation 67
rising factorial 53
rising product 53
root 40

S

shift operator 59
sign 50
significand 23
stem 7
strict linear ordering 49
sub-field 39
substitution 9
subtype 13
sum type 13
super type 13
suptype 13
Sylvester matrix 45
symbol 7

T

telescoping 56
term rewriting 5
transcendental 41
transcendental extension 41
type constructor 13
type function operator 12
type product operator 12
type tags 13

U

UFD 35
unique factorization domain 35
unit 33
unit normal 33
unit normal factorization 35
unit normal GCD 33
unit normal LCM 34
unit part 34

V

values 7
variable 7

Z

zero 40