Polynomial Computer Algebra and implementation of Wilf-Zeilberger's method

Lars Åström
Supervisor: Victor Ufnarovski

LUND UNIVERSITY
Department of Mathematics

MSc Thesis TFRT-TODO
ISSN TODO–TODO

Department of Mathematics
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

The purpose of the thesis is to get a better understanding of computer algebra in general, and polynomial computer algebra in particular. This is done by implementing a library for a polynomials and methods that are needed to be able to perform operations on the polynomials. Then this library is used to implement Wilf-Zeilberger's method, which is a method that can be used to prove certain combinatorial identities involving summation. The thesis consists mostly of three parts; theory, implementation and examples. In the theory section, all the theoretical results used in the project are presented. The implementation section then treats the difficulties arising when turning theory into practice, and focuses in particular on when theoretically easy methods and concepts become much more challenging in implementation. The program that is developed seems to work well, both on examples that were used throughout the project as testing and on validation examples that were found after all the implementation was done. This means that the program can solve and produce a paper proving that identities indeed are true. Therefore the thesis shows one example of how automated proofs can be generated, but mostly the thesis highlights the difficulties arising in computer algebra while implementing the specific example of Wilf-Zeilberger's method.

Abstrakt

Syftet med uppsatsen är att få en djupare och bättre förståelse för datoralgebra i allmänhet, och polynomisk datoralgebra i synnerhet. Detta görs genom att implementera ett bibliotek för polynom och metoder som behövs för att kunna utföra operationer på polynom. Därefter kommer detta bibliotek att användas för att implementera Wilf-Zeilbergers metod, som är en metod som kan användas för att bevisa vissa kombinatoriska identiteter som involverar summation. Uppsatsen består i huvudsak av tre delar; teori, implementation och exempel. I teoriavsnittet presenteras alla teoretiska resultat som används i projektet. Implementationsavsnittet behandlar därefter svårigheter som uppstår när teorin ska omvandlas till praktik och fokuserar framför allt på när teoretiskt enkla metoder och koncept blir mycket mer komplicerade och utmanande vid implementering. Programmet som har tagits fram verkar fungera väl, både på exempel som har använts genom hela projektet som testning och på valideringsexempel som hittades först efter att implementering var klar. Detta innebär att programmet löser problemen och producerar ett bevis för att identiteterna som ges faktiskt gäller. Därför visar uppsatsen ett exempel på hur automatiserade bevis kan genereras, men framför allt belyser uppsatsen svårigheterna som uppstår inom datoralgebra medan implementationen av det specifika exemplet Wilf-Zeilbergers metod görs.

Preface

The thesis has been written in Lund during September-December 2019, as a master's thesis in the program Engineering in Mathematics. The idea of the thesis came from my supervisor, professor Victor Ufnarovski, who presented ten different, all very interesting problem formulations for a master thesis, only a few days after I asked if he had any ideas that we could do as a master's thesis.

Firstly, I would like to thank my supervisor Victor Ufnarovski, who has not only been of great help during the thesis work, but also previously in my life as coach in the international mathematics olympiad. There I got help both to become better at mathematics, but also in other aspects of life with continuous positive feedback.

Secondly, I would like to thank my family and friends, who have supported me both while writing my thesis and previously in my studies and in my life.

Contents

1 Introduction

In the thesis computer algebra is investigated and a library for manipulating and working with polynomials is produced. Then this package is used to implement Wilf-Zeilberger's method, which is a method to prove certain combinatorial identities involving (infinite) summation. Although there already are toolboxes available in several different software languages that handles polynomials we will develope a new system, in order to experience implementation pitfalls and difficulties first hand.

Computer algebra started to be developed during the second half of the 20th century. Martinus Veltman won the Nobel prize in physics 1999 due to his work with developing a computer algebraic software for particle physics. The first working version dates back to the 1960s.[1] This is one example of an early computer algebraic system.

Computer algebraic systems are used to automize manipulation of mathematical expressions that are similar to traditional pen and paper calculations. The advantage of computer algebra is that the power of a computer is used, meaning that computations can be performed extremely fast and as computational power keeps improving the possibilities are neverending. This is the reason computer algebra is growing as a field, and why it is both important and interesting to investigate.

Before starting this thesis I did not know much about computer algebra. Although it is a big part of almost all tools that have been used during courses in my program at LTH (such as MATLAB and Maple), I have never given the details of how implementation is done much thought. Therefore this thesis has given me the opportunity to investigate the basics of computer algebra, while providing an open source implementation of Wilf-Zeilberger's method.

The goal of the thesis is to, by implementing a library for polynomial computer algebra, implement an automatic solver which can prove identities using Wilf-Zeilberger's method. Furthermore the goal is to get a deeper understanding of implementation difficulties in computer algebra by implementing everything from scratch, and the implementation should rather be simple and easy to understand than complicated but more efficient computationally.

2

Background

Computer algebra, also called symbolic computation or algebraic computation, is the study of algorithms and software for simplifying and manipulating mathematical expressions. Computer algebra has many applications, is a part of many different mathematical languages, such as Maple and Mathematica, and is used widely in many mathematical and engineering applications, such as physics and cryptography.[2][3]

In combinatorics binomial coefficients are constantly used for counting different things. Also the sum of binomial coefficients are often of interest in solving the theoretical questions in combinatorics and similar fields. Some combinatorial identities have beautiful and simple proofs using combinatorial arguments while others require long and complicated proofs. Therefore Wilf-Zeilberger's method comes in handy, to automize the proof procedure.

In the thesis Wilf-Zeilberger's method will be used and implemented, which is a method to prove the correctness of an (infinite) sum. One example where it can be used is to prove

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k} \binom{2k}{k} 4^{n-k} = \binom{2n}{n}. \tag{2.1}$$

Here we see that we have a finite sum, but it can also be seen as an infinite sum from $-\infty$ to ∞ too, if we define $\binom{n}{a} = 0$, when $a < 0$ or $a > n$.

The steps of the method will of course be shown in chapter 3, but the general idea is to change our addends in the sum to something that telescopes. After this is done, proving the identity gets reduced to prove the identity for a single n. As we see in equation 2.1 this is quite easy to prove for some n, for instance $n = 0$. Although it is not always easy to prove the identity for some n, it is usually the case.

Wilf-Zeilberger's method itself is not a very new method, it dates back to 1990.[4] Even implementations of the method in computer algebraic softwares is not very new, for instance an implementation of the method in Mathematica was published in 1994.[5] The contribution of this thesis is therefore not an implementation of something that has not been implemented before, but rather an open source version of it. Furthermore, as stated in chapter 1, a large part of the goal is to get a better understanding of computer algebra.

3 Theory

One main goal of the thesis is to end up with an implementation of Wilf-Zeilberger's method. In the process of this method polynomials are used in all the steps. Therefore a big part of the thesis is to understand the computer algebraic difficulties involved when implementing for instance polynomials. Therefore a large library with polynomial functions is implemented. The goal for this library is not to be efficient – rather keeping functions simple – as this is not needed for the implementation of Wilf-Zeilberger's method, and since that would be out of scope for the thesis. Furthermore, there are already existing libraries that are highly efficient for polynomic computer algebra. Still, implementing a library of polynomial functions from scratch is relevant in order to see and experience the difficulties first hand. Since this is also a goal of the implementation, we will try to throughout the theory and implementation chapters to point out difficulties in the implementations that occurred on the way. This might however give the impression that the implementation is very naïve, but this is since the problems that are considered in this thesis are not very demanding computationally and simple and understandable implementation is preferred.

In order to implement Wilf-Zeilberger's method we actually only need polynomials of two variables (and possibly some constant parameters). Even though this is the case we will implement a more general polynomial structure, in order to see what extra requirements that gives. Furthermore in order to ease how we show polynomials we will associate each polynomial with a variable, which means that for a polynomial we do not only store the coefficients but also a variable. This is not necessary and a bit inefficient, still we do this in order to ease the understanding.

3.1 Polynomials

In the thesis we are continuously discussing polynomials. We are always using polynomials with integer coefficients – which in some sense is generalized to include rational coefficients by looking at fractions of polynomials with integer coefficients.

Therefore, from now on whenever we talk about polynomials they are assumed to have integer coefficients as long as otherwise is not stated.

We will now have a discussion on what operations we want to be able to perform on the polynomials both theoreticly and partly implementation wise. Therefore there will not be a specific subchapter regarding polynomials later on in chapter 4. The structure of this section is that first, in subsection 3.1.1, we describe how a polynomial can be implemented and stored in the code – and this is how it was implemented in this thesis. Then in subsection 3.1.2 the simplest and most straight forward operations performed on polynomials are defined, while the most complicated operations – division and greatest common divisor – are left for subsection 3.1.3

3.1.1 Polynomial structure and definitions

A polynomial, say $p(n)$, can be seen as a list of coefficients $[a_0, a_1, \ldots, a_d]$. This represents the polynomial

$$p(n) = a_0 + a_1 n + \ldots + a_d n^d. \tag{3.1}$$

This is fairly simple if the polynomial only has one variable. As soon as we introduce polynomials with more than one variable – which is the case in all examples where Wilf-Zeilberger's method is used – it gets a bit more complicated. The way we will visualize polynomials of more than one variable is to try to keep the very same structure as in 3.1, namely we store a polynomial $p(n)$ as a list of coefficients $[a_0, a_1, \ldots, a_d]$, but instead of having a_i just representing integers they are instead polynomials. In order to show how this can be done, we consider the following example.

EXAMPLE 3.1 Lets look at the polynomial

$$\begin{aligned} p(k,m,n) =& 1 + 3n + 2n^2 - m - 2mn - 2mn^2 + 3m^2 + 3m^2n + 3m^2n^2 + \\ & 3k - 3kn + 2kn^2 + km - 2kmn - km^2 + 2k^2 - k^2n - k^2n^2 + \\ & 3k^2m - 3k^2mn - 2k^2mn^2 - 3k^2m^2 - k^2m^2n - 3k^2m^2n^2. \end{aligned}$$

This will be stored as

$$\begin{aligned} p(k,m,n) =& -((3k^2-3)m^2 + (2k^2+2)m + (k^2-2k-2))n^2 - \\ & ((k^2-3)m^2 + (3k^2+2k+2)m + (k^2+3k-3))n - \\ & ((3k^2+k-3)m^2 - (3k^2+k-1)m - (2k^2+3k+1)) \end{aligned}$$

Note that this can be written as

$$p(k,m,n) = p_2(k,m)n^2 + p_1(k,m)n + p_0(k,m),$$

where

$$\begin{aligned}
p_0(k,m) &= -(3k^2+k-3)m^2-(3k^2+k-1)m-(2k^2+3k+1)\\
p_1(k,m) &= -(k^2-3)m^2+(3k^2+2k+2)m+(k^2+3k-3)\\
p_2(k,m) &= -(3k^2-3)m^2+(2k^2+2)m+(k^2-2k-2).
\end{aligned}$$

Now since our definition of polynomials is recursive, each of $p_0(k,m), p_1(k,m), p_2(k,m)$ is written as $q_2(k)m^2+q_1(k)m+q_0(k)$ for polynomials q_0,q_1,q_2. For instance p_1 has

$$p_1(k,m) = q_2(k)m^2+q_1(k)m+q_0(k),$$

where

$$\begin{aligned}
q_0(k) &= k^2+3k-3\\
q_1(k) &= 3k^2+2k+2\\
q_2(k) &= -k^2+3
\end{aligned}$$

Lastly we have the polynomials q_0,q_1,q_2 which have integers as coefficients. □

As we saw in example 3.1 we define all polynomials as in equation 3.1 where the coefficients a_i are polynomials or integers, where it is only integers in the base case.

Note that in this case we wrote the polynomial with n ”outmost” in the representation of p. There was no specific reason for this, and the choice is not very important when only looking at one polynomial at a time. When we are using several polynomials at the same time, for instance if we want to add two polynomials, the structure is crucial. Therefore we will define some concepts regarding the representation of a polynomial.

DEFINITION 3.2 A polynomial p is said to have the variable setup $\mathbf{x} = (x_1,x_2,\ldots,x_m)$ if it is stored as

$$p(\mathbf{x}) = \sum_{i_1=0}^{d_1}\left(\sum_{i_2=0}^{d_2}\Big(\sum_{i_3=0}^{d_3}\cdots\big(\sum_{i_m=0}^{d_m} c_{\mathbf{i}}x_m^{i_m}\big)\ldots\Big)x_2^{i_2}\right)x_1^{i_1}, \tag{3.2}$$

where $c_{\mathbf{i}}$ is the coefficient of $x_1^{i_1}x_2^{i_2}\ldots x_m^{i_m}$ and $\mathbf{i} = (i_1,\ldots,i_m)$.

Note that the same polynomial can be expressed in various different ways, and we say that two variable setups, $\mathbf{x}$ and $\mathbf{y}$, are different if $|\mathbf{x}| \neq |\mathbf{y}|$ (where $|\mathbf{z}|$ denotes the length of the vector $\mathbf{z}$) or there exist an integer i such that $x_i \neq y_i$. □

REMARK If the polynomial p has variable setup $\mathbf{x} = (x_1,x_2,\ldots,x_m)$, then the coefficients of p have variable setup $\mathbf{x}' = (x_2,\ldots,x_m)$. □

REMARK Note that if the polynomial p has variable setup $\mathbf{x}$ such that $|\mathbf{x}| = 1$, then the coefficients of p are integers. $\square$

Here we can note that d_i is the degree of p with respect to x_i. Furthermore we note that we can express a polynomial p with the variable setup $\mathbf{x} = (x_1, \ldots, x_m)$ even though p does not have all the variables x_j in its expression. If p does not depend on x_j that will be seen in that $d_j = 0$. A polynomial p cannot, however, be expressed with the variable setup $\mathbf{x}$ if not all its variables are in $\mathbf{x}$. A last note is that we can convert a polynomial p with variable setup $\mathbf{x}$ to any variable setup $\mathbf{y}$ as long as $x_i \in \mathbf{y} \;\; \forall\, x_i \in \mathbf{x}$.

There are of course many operations that are needed when operating with polynomials; addition, negate, subtraction, multiplication, division, modulo, gcd (greatest common divisor), checking for equality, finding roots and evaluating at a point. In the implementation all of them are defined recursively. We will show how this is done for all the most straight forward operations here, while the more complicated operations – division, modulo and gcd – are described later in section 3.1.3.

First we define how we write that a polynomial is constructed:

DEFINITION 3.3 When we construct a polynomial p of a variable x then we write this as

$$p = polynomial(\mathbf{a}, "x"), \tag{3.3}$$

where $\mathbf{a} = (a_0, a_1, \ldots, a_d)$. This means that

$$p(x) = a_0 + a_1 x + \ldots + a_d x^d. \tag{3.4}$$

Note that we have not specified what a_i are; they can be either integers or polynomials – but if they are polynomials they have to have the same variable setup. Also note that once we have the polynomial p we will use the notation $p[i]$ to reference the i-th variable, namely a_i. If we reference $p[j]$ where $j > d$, then this is defined as $p[j] = 0 \;\; \forall\, j > d$. $\square$

We also define the degree of a polynomial:

DEFINITION 3.4 Let the polynomial p have variable setup $\mathbf{x} = (x_1, \ldots, x_m)$. Then we define the degree of p, denoted $deg(p)$, as the largest exponent of x_1 in the expression for p. Furthermore we define $deg(\mathbf{0}) = -\infty$, where the polynomial $\mathbf{0}$ is the constant polynomial identically equal to zero. $\square$

Using this definition of degree we see that the degree upon addition and multiplication behaves as described in the following example.

EXAMPLE 3.5 Let f and g be polynomials with degree d_f, d_g. Then we have that $deg(f+g) \leq max(d_f, d_g)$ and $deg(f \cdot g) = d_f + d_g$. □

3.1.2 Simple operations

For all the operations we will use two polynomials f and g. These are polynomials with the variable setup $\mathbf{x}$. We will now provide procedures that show how all operations that are needed. In this subsection we define the simplest ones, which division and GCD are left for subsection 3.1.3. Note that the procedures are assuming that the polynomials have the same variable setup, if this is not the case then the polynomials first are converted into a common variable setup.

All the procedures try to follow the same structure, which is quite recursive. All procedures have a base case, usually when the variable setup $\mathbf{x}$ has $|\mathbf{x}| = 1$. In all other cases recursive calls are made. A note to be made here is that recursive calls are not computationally heavy in theory but might take time if many are made, especially if the depth is large, in practice. In this thesis however, the depth is very small – usually less than 5 – since the depth comes from the number of variables in the polynomials. Therefore the recursion is not expected to cause any problems computationally.

Addition is as it often is a foundation for all other methods. As we will see later on it is called from most other methods at one point or another.

Algorithm 3.1 Addition

Input: Polynomials f and g, with variable setup $\mathbf{x}$
Output: Polynomial h such that $h = f + g$

```
procedure ADD($f, g$)
    $\mathbf{a} \leftarrow [\,]$
    for $i \leftarrow 0, \dots, max(deg(f), deg(g))$ do
        if $|\mathbf{x}| = 1$ then
            $\mathbf{a}[i] \leftarrow f[i] + g[i]$
        else
            $\mathbf{a}[i] \leftarrow$ ADD($f[i], g[i]$)
        end if
    end for
    return $polynomial(\mathbf{a}, \mathbf{x}[0])$
end procedure
```

REMARK Note that in all procedures we return a polynomial. We denote this as **return**$polynomial(\mathbf{a}, \mathbf{x}[0])$. Here $\mathbf{a}$ are the coefficients and $\mathbf{x}[0]$ is the variable that is associated with the polynomial that is returned. □

Negation is sometimes needed itself, but mostly this procedure is used when subtracting two polynomials. That means that this procedure is quite long, but on the other hand that makes subtract really short. One can choose to make subtract long and negate short instead, but we chose to do it this way.

Algorithm 3.2 Negatation

```
Input: Polynomial $f$ with variable setup $\mathbf{x}$
Output: Polynomial $h$ such that $h = -f$

 1: procedure NEGATE($f$)
 2:     $\mathbf{a} \leftarrow [\,]$
 3:     for $i \leftarrow 0, \dots, deg(f)$ do
 4:         if $|\mathbf{x}| = 1$ then
 5:             $\mathbf{a}[i] \leftarrow -f[i]$
 6:         else
 7:             $\mathbf{a}[i] \leftarrow$ NEGATE($f[i]$)
 8:         end if
 9:     end for
10:     return $polynomial(\mathbf{a}, \mathbf{x}[0])$
11: end procedure
```

Algorithm 3.3 Subtraction

```
Input: Polynomials $f$ and $g$, with variable setup $\mathbf{x}$
Output: Polynomial $h$ such that $h = f - g$

1: procedure SUBTRACT($f, g$)
2:     $g' \leftarrow$ NEGATE($g$)
3:     return ADD($f, g'$)
4: end procedure
```

The multiplication procedure that is implemented is the most simple and naïve version, which multiply polynomials in time complexity $\mathcal{O}(d_f \cdot d_g)$, where d_f, d_g are the degrees of f and g, respectively. Time complexity indicates that as polynomials change in degree the running time of performing multiplication is proportional to $d_f \cdot d_g$. There are much more efficient implementations of polynomial multiplication – for instance Fast Fourier Transform can give a time complexity of $\mathcal{O}(n \log n)$, where $n = max(d_f, d_g)$. As we in the application of polynomials in this thesis only treat polynomials of small degrees and in order to keep the implementation simple and understandable more efficient multiplication algorithms have not been implemented.

Algorithm 3.4 Multiplication

Input: Polynomials f and g, with variable setup $\mathbf{x}$
Output: Polynomial h such that $h = f \cdot g$

```
procedure MULTIPLY($f, g$)
    $\mathbf{a} \leftarrow [\,]$
    for $i \leftarrow 0, \dots, deg(f) + deg(g)$ do
        $\mathbf{a}[i] \leftarrow 0$
    end for
    for $i \leftarrow 0, \dots, deg(f)$ do
        for $j \leftarrow 0, \dots, deg(g)$ do
            if $|\mathbf{x}| = 1$ then
                $\mathbf{a}[i+j] \leftarrow \mathbf{a}[i+j] + f[i] \cdot g[j]$
            else
                $y_{i,j} \leftarrow$ MULTIPLY($f[i], g[j]$)
                $\mathbf{a}[i+j] \leftarrow$ ADD($\mathbf{a}[i+j], y_{i,j}$)
            end if
        end for
    end for
    return $polynomial(\mathbf{a}, \mathbf{x}[0])$
end procedure
```

Algorithm 3.5 Equality

Input: Polynomials f and g, with variable setup $\mathbf{x}$
Output: *True* if f and g are equal, otherwise *False*

```
procedure EQUALS($f, g$)
    if $deg(f) \neq deg(g)$ then
        return False
    end if
    for $i \leftarrow 0, \dots, deg(f)$ do
        if $|\mathbf{x}| = 1$ AND $f[i] \neq g[i]$ then
            return False
        else if $|\mathbf{x}| > 1$ AND not EQUALS($f[i], g[i]$) then
            return False
        end if
    end for
    return True
end procedure
```

Quite often polynomials are compared and we are interested in whether they are equal or not, hence an equality method was implemented. The reason this is needed is that polynomials are implemented as a class, thus using the normal ”==” operator will not work.

Algorithm 3.6 Evaluating at a point

```
    **Input:** Polynomials $f$, with variable setup $\mathbf{x}$, variable $v \in \mathbf{x}$ and value $y$
    **Output:** Polynomial $h$ that is $f$ evaluated at $v = y$

 1: procedure EVALUATE($f, v, y$)
 2:     if $\mathbf{x}[0] = v$ then
 3:         $g \leftarrow f[0]$
 4:         for $i \leftarrow 1, \ldots, deg(f)$ do
 5:             $h \leftarrow$ MULTIPLY($f[i], y^i$)
 6:             $g \leftarrow$ ADD($g, h$)
 7:         end for
 8:         return $g$
 9:     else
10:         $\mathbf{a} \leftarrow [\,]$
11:         for $i \leftarrow 0, \ldots, deg(f)$ do
12:             $\mathbf{a}[i] \leftarrow$ EVALUATE($f[i], v, y$)
13:         end for
14:         return $polynomial(\mathbf{a}, \mathbf{x}[0])$
15:     end if
16: end procedure
```

The next procedure is used to find roots of a polynomial. Here if the polynomial is only dependent on one variable, meaning $|\mathbf{x}| = 1$, then we solve this equation (we call this function *SolveEquation*). In the implementation of the program only solving equations completely is implemented up to degree 2, but the program also tries to insert values in the range $(-100, 100)$ and sees if any of these are roots. Here we note that this weak implementation to find roots might cause problems later on, which we will come back to in more detail in section 4.2.1. We do not include the procedure for *SolveEquation* in the report, since it is just solving a second degree equation and additionally trying a bunch of different small values.

Algorithm 3.7 Finding roots

```
Input: Polynomial $f$, with variable setup $\mathbf{x} = (x_1, \dots, x_m)$
Output: Integers $x'$ such that $x_m = x' \implies f = 0$

procedure ROOTS($f$)
    if $|\mathbf{x}| = 1$ then
        return SOLVEEQUATION($f$)
    else
        $\mathbf{a} \leftarrow$ ROOTS($f[0]$)
        for $i \leftarrow 1, \dots, deg(f)$ do
            $\mathbf{b} \leftarrow [\,]$
            for all $a' \in \mathbf{a}$ do
                if EVALUATE($f[i], x_m, a'$)=0 then
                    $\mathbf{b}[size(\mathbf{b})] \leftarrow a'$
                end if
            end for
            $\mathbf{a} \leftarrow \mathbf{b}$
        end for
        return $\mathbf{a}$
    end if
end procedure
```

3.1.3 Complicated operations – division and GCD

Now we come to the slightly more complicated operations that we need to perform on polynomials. First we define GCD for polynomials a bit more carefully, which we do in a few steps.

DEFINITION 3.6 A polynomial g is said to divide another polynomial a if there exists a polynomial q such that $a = g \cdot q$. We will use the notation $g|a$ to denote that g divides a. □

DEFINITION 3.7 A polynomial g is called a *common divisor* to two polynomials a and b if $g|a$ and $g|b$. □

DEFINITION 3.8 We say that a polynomial g is the *greatest common divisor* (GCD) to two polynomials if

1. g is a common divisor to a and b
2. For all common divisors g' to a and b we have that $g'|g$. □

In general performing division, modulo and gcd is fairly straight forward even on polynomials. Usually one might have the following structure on the implementation:

$$DIVIDE(a,b) \to (q,r) \text{ such that } a = b \cdot q + r, deg(r) < deg(b), \tag{3.5}$$

$$MODULO(a,b) \to r \text{ where } DIVIDE(a,b) = (q,r), \tag{3.6}$$

$$GCD(a,b) \to a \text{ if } b = 0 \text{ otherwise } GCD(b, MODULO(a,b)), \tag{3.7}$$

where $\to$ indicates what the function returns.

This implementation scheme is fairly straight forward, even though the divide function might be a bit messy. The problem we encounter in this thesis though, is that we cannot necessarily find polynomials q,r such that 3.5 is satisfied. The reason for this is that we are working with integer coefficients and not in a ring. In order to show the problems we encounter we will use a common example throughout this section to show what the problem is and how we resolve it.

EXAMPLE 3.9 Consider dividing $a(x) = x^2 + 2x + 1$ by $b(x) = 2x$. Assume we have q,r that fulfill 3.5. Then we need to have $deg(r) < deg(b) = deg(2x) = 1$. This means that $r(x) = c$, where c is an integer constant. This gives us

$$x^2 + 2x + 1 = (2x)q(x) + c. \tag{3.8}$$

Furthermore we have that $deg(\text{Left hand side}) = 2$, hence $deg(q) = 1$. Let $q(x) = c_0 + c_1 x$, then 3.8 becomes

$$x^2 + 2x + 1 = (2x)(c_1 x + c_0) + c = 2c_1 x^2 + 2c_0 x + c. \tag{3.9}$$

By looking at degree 2 we get $2c_1 = 1$, which of course does not have any solutions for integers c_1. $\square$

The way we solve this is to instead of giving two polynomials q,r when we divide a by b we also give a third parameter, f, which is a factor of the same variable setup as the coefficients in a and b. The condition that needs to be fulfilled is

$$DIVIDE(a,b) \to (q,r,f) \text{ such that } f \cdot a = b \cdot q + r, deg(r) < deg(b). \tag{3.10}$$

Now we will show that it actually is possible to find q,r,f such that 3.10 holds.

THEOREM 3.10 Let $a(x_1), b(x_1)$ be polynomials of the same variable setup $\mathbf{x} = (x_1, \ldots, x_m)$. Then it is possible to find polynomials q,r,f, where q,r have variable setup $\mathbf{x}$ and f has variable setup $\mathbf{x}' = (x_2, \ldots, x_m)$, such that

$$f \cdot a(x_1) = q(x_1)b(x_1) + r(x_1), \tag{3.11}$$

and $deg(r) < deg(b)$. $\square$

Proof Let $a = a_0 + a_1x_1 + \ldots + a_cx_1^c$ and $b = b_0 + b_1x_1 + \ldots + b_dx_1^d$, where c, d are the degrees of a and b, respectively. If $c < d$ then we can use $q = 0, r = a, f = 1$ and we have that 3.11 is fulfilled.

Now assume that $c \geq d$. We will now show that we can find a', q_0, f_0 such that

$$f_0a(x_1) = q_0(x_1)b(x_1) + a'(x_1), \tag{3.12}$$

and $deg(a') < deg(a)$.

This is obtained by choosing $f_0 = b_d, q_0 = a_cx_1^{c-d}$ and $a'(x_1) = b_da(x_1) - a_cx_1^{c-d}b(x_1)$. This gives us

$$\begin{aligned} a'(x_1) &= b_da(x_1) - a_cx_1^{c-d}b(x_1) = b_d(a_0 + \ldots + a_cx_1^c) - a_cx_1^{c-d}(b_0 + \ldots + b_dx_1^d) \\ &= b_d(a_0 + \ldots + a_{c-1}x_1^{c-1}) - a_cx_1^{c-d}(b_0 + \ldots + b_{d-1}x_1^{d-1}) \end{aligned} \tag{3.13}$$

Now we can clearly see that $deg(a') < c$, which means we have reduced the problem by (at least) 1 in the degree of a. We continue to do this until the degree of a is less than the degree of b, and then we collect the results. If we have that

$$DIVIDE(a', b) \rightarrow (q', r', f'), \tag{3.14}$$

then since a' is given by 3.13 we get

$$\begin{aligned} f' \cdot a' &= q' \cdot b + r' && \Longrightarrow \\ f'(b_da - a_cx_1^{c-d}b) &= q' \cdot b + r' && \Longrightarrow \\ (f'b_d)a &= (q' + a_cx_1^{c-d})b + r', \end{aligned} \tag{3.15}$$

and thus we get

$$DIVIDE(a, b) \rightarrow (q, r, f) = (q' + a_cx_1^{c-d}, r', f'b_d). \tag{3.16}$$

Now we have shown the theorem, and provided the method for dividing polynomials at once. □

DEFINITION 3.11 When we divide polynomial a by polynomial b (both having variable setup $\mathbf{x} = (x_1, \ldots, x_m)$) we are given three things:

- q, a polynomial with variable setup $\mathbf{x}$, which is called the ”quotient”,
- r, a polynomial with variable setup $\mathbf{x}$, which is called the ”remainder”,
- f, a polynomial with variable setup $(x_2, \ldots, x_m)$, which is called the ”factor”.

These polynomials fulfill equation 3.10. □

REMARK One important thing to note is that the definition of how we divide is not unique, if q, r, f are all multiplied by a polynomial s with variable setup $(x_2, \ldots, x_m)$ then equation 3.10 will still be fulfilled. □

Now we have defined division and will soon move over to calculating the greatest common divisor, but first we have a look at what happens to our example.

EXAMPLE 3.12 Consider dividing $a(x) = x^2 + 2x + 1$ by $b(x) = 2x$. Now we see that $q = x + 2, r = 2, f = 2$ gives us a solution where $0 = deg(r) < deg(b) = 1$. □

We will show a theorem which states how the greatest common divisor is computed, but first we need a little bit of notation.

DEFINITION 3.13 Let p be a polynomial with variable setup $\mathbf{x} = (x_1, \ldots, x_m)$. If we write p as $p(x_1) = p_0 + p_1 x_1 + \ldots p_d x_1^d$, then we denote the greatest common divisor of all the coefficients $p_i, 0 \leq i \leq d$ by $gcd_c(p)$. □

THEOREM 3.14 Let g denote the greatest common divisor of polynomials a and b. Then we can compute this by the following:

1. If $b = 0$, then $g = a$,
2. Otherwise g is given by

$$g = \frac{g'}{gcd_c(g')} \cdot gcd(gcd_c(a), gcd_c(b)), \tag{3.17}$$

   where g' is the greatest common divisor of b and r, where r is the remainder when a is divided by b, which means

$$DIVIDE(a, b) \rightarrow (q, r, f). \tag{3.18}$$

□

Before we prove theorem 3.14 we show the reasoning behind why we get this more complicated version of gcd in 3.17 instead of the much easier version in 3.7. We start by showing the example and see what happens if we naïvely using the new division combined with 3.7.

EXAMPLE 3.15 Consider computing the greatest common divisor of $a(x) = x^2 + 2x + 1$ and $b(x) = 2x$ using 3.7.

$$gcd(x^2 + 2x + 1, 2x) = gcd(2x, 2) = gcd(2, 0) = 2, \tag{3.19}$$

since

$$\begin{aligned} DIVIDE(x^2+2x+1,2x) &\to (q,r,f) = (x+2,2,2) \\ DIVIDE(2x,2) &\to (q,r,f) = (x,0,1). \end{aligned} \tag{3.20}$$

But now we see that we have the greatest common divisor being 2, but 2 does not divide $a(x)$. This highlights a problem with using 3.7 for gcd.

Now let us consider using 3.17 instead. This gives us (working backwards in steps)

$$\begin{aligned} gcd(2,0) &= 2 \\ gcd(2x,2) &= \frac{gcd(2,0)}{gcd_c(gcd(2,0))} \cdot gcd(gcd_c(2x), gcd_c(2)) = \frac{2}{2} \cdot gcd(2,2) = 2 \\ gcd(x^2+2x+1,2x) &= \frac{gcd(2x,2)}{gcd_c(gcd(2x,2))} \cdot gcd(gcd_c(x^2+2x+1), gcd_c(2x)) = \\ &= \frac{2}{2} \cdot gcd(1,2) = 1. \end{aligned} \tag{3.21}$$

Using 3.17 seems to work on this example. We need, however to prove it in general and to prove it regardless of which q,r,f are used for $DIVIDE(a,b)$. □

As we saw in example 3.15 if we try to naïvely use 3.7 to compute the gcd, but with the new definition of how we divide, then that results in a too large divisor since the factor f allows the divisor to be larger. This can be seen by looking at the formula for dividing a by b. We have that

$$f \cdot a(x_1) = b(x_1) \cdot q(x_1) + r(x_1). \tag{3.22}$$

Now the greatest common divisor of b and r will divide the left hand side of equation 3.22, $f \cdot a(x_1)$, but not necessarily $a(x_1)$. With this insight in mind, we will now prove theorem 3.14, but first we need a few lemmas.

LEMMA 3.16 Let a,b,d be polynomials such that $d|(a \cdot b)$. Then we can factor d into two polynomials d_a, d_b such that

1. $d = d_a \cdot d_b$
2. $d_a|a, d_b|b$ □

Proof Let $d_a = gcd(a,d)$. Then we get that $d_a|a, d_a|d \iff \exists\, a', d' : d = d_a d', a = d_a a'$ and $gcd(a',d') = 1$. We know that $d|(a \cdot b) \iff \exists\, x : a \cdot b = d \cdot x$. Now we replace a and d by $d_a a', d = d_a d'$ which gives us $a' \cdot b = d' \cdot x$, where $gcd(a',d') = 1$. Therefore $d'|b$, and thus we can choose $d_b = d'$. □

LEMMA 3.17 Let p be a polynomial $p(x_1) = p_0 + \ldots + p_d x_1^d$ with variable setup $\mathbf{x} = (x_1, \ldots, x_m)$ and f be a polynomial with variable setup $(x_2, \ldots, x_m)$ such that $f|p$. Then we have that $f|p_i \ \ \forall\, 0 \le i \le d$. $\square$

Proof We know that $f|p$, which is equivalent to $p = fp'$ for some polynomial p'. Let $p' = p'_0 + \ldots + p'_c x_1^c$. Now first of all, since f does not depend on x_1 we have that $deg(p) = deg(p') = d$. Furthermore we have that

$$\begin{aligned} p_0 + p_1 x_1 + \ldots + p_d x_1^d = p = fp = f(p'_0 + p'_1 x_1 + \ldots + p'_d x_1^d) = \\ = (fp'_0) + (fp'_1)x_1 + \ldots + (fp'_d)x_1^d \end{aligned} \tag{3.23}$$

Now by looking at the terms for each degree of x_1 in 3.23 we see that

$$p_i = fp'_i \iff f|p_i, \ \ \forall\, 0 \le i \le d. \tag{3.24}$$

$\square$

LEMMA 3.18 Let p and d be polynomials with variable setup $\mathbf{x} = (x_1, \ldots, x_m)$, $gcd_c(d) = 1$ and f be a polynomial with variable setup $(x_2, \ldots, x_m)$ such that $d|p$ and $f|p$. Then we have that $(d \cdot f)|p$. $\square$

Proof We know that $f|p$, hence we can write p as $p = fp'$. We know that $d|p = fp'$ and thus we can due to lemma 3.16 factor g into $d_f, d_{p'}$ such that $d = d_f \cdot d_{p'}$ and $d_f|f, d_{p'}|p$. But since f has variable setup $(x_2, \ldots, x_m)$ then d_f cannot depend on x_1 and thus can be expressed with variable setup $(x_2, \ldots, x_m)$. Furthermore $d_f|d$, which together with lemma 3.17 gives us that $d_f|d_i$ for all coefficients d_i of d. Therefore $d_f|gcd_c(d) = 1 \implies d_f = 1$. Thus $d = d_{p'}|p'$ and therefore $(d \cdot f)|p$ $\square$

Now we are ready to finish this section with the proof of theorem 3.14.

Proof Theorem 3.14 Recall that $gcd(b, r)$ is denoted by g'. We will show this theorem in two steps. First of all we want to prove that

$$g = \frac{g'}{gcd_c(g')} \cdot gcd(gcd_c(a), gcd_c(b)), \tag{3.25}$$

in fact is a divisor of a and b. After that we will show that

$$d|a, d|b \implies d\Big|\Big(\frac{g'}{gcd_c(g')} \cdot gcd(gcd_c(a), gcd_c(b))\Big). \tag{3.26}$$

We start with the first part. Since $g'|b$ and $g'|r$, g' divides the right hand side of 3.22. But the left hand side is equal to the left hand side, thus we know that $g'|fa$. This in turns implies that we (due to lemma 3.16) can factor g into two parts, g_f, g_a, such that $g' = g_a g_f$, $g_a|a$ and $g_f|f$. But now since f has variable setup $(x_2,\dots,x_m)$ then g_f cannot depend on x_1.

We will now show that $\frac{g'}{gcd_c(g')}|a$. We know that since g_f is of variable setup $(x_2,\dots,x_m)$ and $g_f|g'$, then by lemma 3.17 we get that $g_f|g'_i$ for all coefficients g'_i in g'. This means that $g_f|gcd_c(g')$. We also know that

$$\frac{g'}{gcd_c(g')} \cdot gcd_c(g') = g' = g_f \cdot g_a, \tag{3.27}$$

therefore we get that $\frac{g'}{gcd_c(g')}|g_a$, and thus $\frac{g'}{gcd_c(g')}|a$. Furthermore

$$gcd_c\Big(\frac{g'}{gcd_c(g')}\Big) = 1. \tag{3.28}$$

We also clearly see that $gcd(gcd_c(a),gcd_c(b))|gcd_c(a)$ and $gcd_c(a)|a$. Therefore $gcd(gcd_c(a),gcd_c(b))$ divides a. Now we use lemma 3.18 and conclude that with

$$\begin{aligned} p &= a, \\ d &= \frac{g'}{gcd_c(g')}, \\ f &= gcd(gcd_c(a),gcd_c(b)) \end{aligned} \tag{3.29}$$

plugged into lemma 3.18 we get that

$$\frac{g'}{gcd_c(g')} \cdot gcd(gcd_c(a),gcd_c(b))|a \tag{3.30}$$

Furthermore we see that

$$\frac{g'}{gcd_c(g')}\bigg|g', g'|b \implies \frac{g'}{gcd_c(g')}\bigg|b. \tag{3.31}$$

Therefore, again according to lemma 3.18 we get that

$$\frac{g'}{gcd_c(g')} \cdot gcd(gcd_c(a),gcd_c(b))|b. \tag{3.32}$$

This finished the first part of the proof. Now we need to show that

$$d|a, d|b \implies d\bigg|\Big(\frac{g'}{gcd_c(g')} \cdot gcd(gcd_c(a),gcd_c(b))\Big), \tag{3.33}$$

where d is a polynomial with variable setup $\mathbf{x} = (x_1,\dots,x_m)$.

First we note that any polynomial that divides a and b also divide r, which is clear by looking at equation 3.22. We now factor d into $d_1 \cdot d_2$, where $d_1 = \frac{d}{gcd_c(d)}, d_2 = gcd_c(d)$. We will now show that $d_1 | \frac{g'}{gcd_c(g')}$ and $d_2 | gcd(gcd_c(a), gcd_c(b))$.

We have that $d_1 | d, d | g' \implies d_1 \Big| \frac{g'}{gcd_c(g')} \cdot gcd_c(g')$. Now lemma 3.16 gives us that d_1 can be factored into d_{11}, d_{12} such that $d_{11} \Big| \frac{g'}{gcd_c(g')}$ and $d_{12} | gcd_c(g')$. But $d_{12} | gcd_c(g')$ means that d_{12} has variable setup $(x_2, \ldots, x_m)$ But this gives us that $d_{12} = 1$ since $d_1 = \frac{d}{gcd_c(d)}$ (otherwise we would have a larger common divisor for the coefficients of d than $gcd_c(d)$). Hence $d_1 = d_{11} \Big| \frac{g'}{gcd_c(g')}$.

Next we show that $d_2 | gcd(gcd_c(a), gcd_c(b))$. We know that $d_2 | d, d | a, d | b \implies d_2 | a, d_2 | b$. Furthermore d_2 has variable setup $(x_2, \ldots, x_m)$, therefore lemma 3.17 gives us that $d_2 | a_i$ for all coefficients a_i in a, and $d_2 | b_i$ for all coefficients b_i in b. This gives us that $d_2 | gcd_c(a), d_2 | gcd_c(b)$, therefore d_2 is a common divisor of $gcd_c(a), gcd_c(b)$, hence $d_2 | gcd(gcd_c(a), gcd_c(b))$.

This finished the proof for that the greatest common divisor of polynomials a, b is

$$g = \frac{g'}{gcd_c(g')} \cdot gcd(gcd_c(a), gcd_c(b)), \tag{3.34}$$

where $g' = gcd(b, r), DIVIDE(a, b) \rightarrow (q, r, f)$. $\square$

Although the procedures of division and gcd are partly already described in the theorems and proofs in this section we will finish this section by formalize the procedures of division, gcd and gcd$_C$, by providing the pseudocode.

Algorithm 3.8 Division

Input: Polynomial a and b, with variable setup $\mathbf{x} = (x_1, \dots, x_m)$
Output: Polynomials q, r with variable setup $\mathbf{x} = (x_1, \dots, x_m)$ and polynomial f with variable setup $\mathbf{x} = (x_2, \dots, x_m)$ such that $f \cdot a = q \cdot b + r$ and $deg(r) < deg(b)$

```
procedure DIVIDE($a, b$)
    if $deg(a) < deg(b)$ then
        return $(0, a, 1)$
    end if
    $f_0 \leftarrow \mathbf{b}[0]$
    $\mathbf{c} \leftarrow [\,]$
    $\mathbf{c}[deg(a) - deg(b)] \leftarrow a[deg(a)]$
    $q_0 \leftarrow polynomial(c, \mathbf{x}[0])$
    $r_0 \leftarrow$ SUBTRACT(MULTIPLY($f_0, a$),MULTIPLY($q_0, b$))
    $q_1, r_1, f_1 \leftarrow$ DIVIDE($r_0, b$)
    $q \leftarrow$ ADD(MULTIPLY($f_1, q_0$) ,$q_1$)
    $r \leftarrow r_1$
    $f \leftarrow$ MULTIPLY($f_0, f_1$)
    return $(q, r, f)$
end procedure
```

REMARK In the actual implementation of division a small extra step is added. In this step all coefficients in q, r as well as the polynomial f are divided by g where

$$g = gcd(gcd_c(q), gcd_c(r), f). \tag{3.35}$$

This step is not necessary, but ensures that q, r, f do not have unnecessarily large coefficients. The reason it is not included is that adding this step most likely adds more confusion than value. □

Now we have gcd and gcd_C left to write pseudocode for. Since all the code is very recursive here, it seems to be the case that we get infinite recursive calls. This will not be the case, however, since every time GCD is called either

1. the degree of some polynomial, or

2. the size of the variable setup

decreases. The fact that one of these are true is easily checked in all the procedures.

Algorithm 3.9 Greatest common divisor

Input: Polynomial a and b, with variable setup $\mathbf{x} = (x_1, \dots, x_m)$

Output: Polynomial g with variable setup $\mathbf{x} = (x_1, \dots, x_m)$ such that g is the greatest common divisor of a and b

```
procedure GCD($a, b$)
    if $b = 0$ then
        return $a$
    end if
    $q_0, r_0, f_0 \leftarrow$ DIVIDE($a, b$)
    $g' \leftarrow$ GCD($b, r_0$)
    $q', r', f' \leftarrow$ DIVIDE($g'$,GCD$_C$($g'$))
    return MULTIPLY($q'$,GCD(GCD$_C$($a$),GCD$_C$($b$)))
end procedure
```

Algorithm 3.10 Greatest common coefficient divisor

Input: Polynomial a with variable setup $\mathbf{x} = (x_1, \dots, x_m)$

Output: Polynomial g with variable setup $\mathbf{x} = (x_2, \dots, x_m)$ such that g is the greatest common divisor of all coeffiecients a_i in a

```
procedure GCD$_C$($a$)
    $g \leftarrow a[0]$
    for $i \leftarrow 1, \dots, deg(a)$ do
        $g \leftarrow$ GCD($g, a[i]$)
    end for
    return $g$
end procedure
```

These procedure descriptions conclude a quite theory heavy section about polynomials. Next will follow Wilf-Zeilberger's method, and will decribe how the method works.

3.2 Wilf-Zeilberger's method

Wilf-Zeilberger's method is a method to show an equality by using a certifying pair, or a proof certificate. The method can just be used to show an identity that is given, and cannot itself come up with a guess for what a sum would be. This means that the method can be used to solve problems like

EXAMPLE 3.19 Show that

$$\sum_{k=0}^{n}(-1)^k\binom{n}{k}\binom{2k}{k}4^{n-k}=\binom{2n}{n}$$

but not problems like

EXAMPLE 3.20 Find a general formula for

$$\sum_{k=0}^{n}k\binom{n}{k}.$$

Hence in order to use the method we need to have an equality to verify.

We will now give a general outline of how a proof using Wilf-Zeilberger's method works. The steps in the proof[6] are that assume we want to prove

$$\sum_{k=-\infty}^{\infty}A(n,k)=B(n). \tag{3.36}$$

Then we let

$$F(n,k)=\frac{A(n,k)}{B(n)} \tag{3.37}$$

Now equation 3.36 becomes

$$\sum_{k=-\infty}^{\infty}F(n,k)=1 \tag{3.38}$$

The way we want to prove this is by first proving

$$\sum_{k=-\infty}^{\infty}F(n+1,k)-F(n,k)=0 \tag{3.39}$$

and then calculate the left hand side of 3.38 for one n, usually but not always $n=0$. Then we get that

$$\sum_{k=-\infty}^{\infty}F(n+1,k)=\sum_{k=-\infty}^{\infty}F(n,k) \tag{3.40}$$

which in turns gives us that

$$\sum_{k=-\infty}^{\infty}F(n,k)=c, \tag{3.41}$$

where c is a constant, for all values of n and thus equal to the result of our previous calculations. The trick we use to prove equation 3.39 is to find another function $G(n,k)$ which satisfies:

i)

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k) \tag{3.42}$$

ii)

$$\lim_{k\to\pm\infty} G(n,k) = 0, \ \forall\, n \tag{3.43}$$

If these two conditions are fulfilled, then equation 3.39 is fulfilled since by replacing $F(n+1,k) - F(n,k)$ with $G(n,k+1) - G(n,k)$ we get a telescopic sum:

$$\begin{aligned}\sum_{k=-\infty}^{\infty} F(n+1,k) - F(n,k) = \sum_{k=-\infty}^{\infty} G(n,k+1) - G(n,k) =&\\ = \lim_{M\to\infty} \sum_{k=-M}^{M} G(n,k+1) - G(n,k) =&\\ = \lim_{M\to\infty} \left[G(n,M+1) - G(n,-M)\right] = 0.&\end{aligned}$$

Sometimes the method looks slightly different, for instance if the identity only holds for some n.

If we summarize the method we have the following steps:

1. Start with equation on the form
$$\sum_{k=-\infty}^{\infty} A(n,k) = B(n).$$
2. Let
$$F(n,k) = \frac{A(n,k)}{B(n)}.$$
3. Find $G(n,k)$ such that equations 3.42 and 3.43 are fulfilled.
4. Show that $\sum_{k=-\infty}^{\infty} F(n',k) = 1$ for some n'.

The crucial step here is how to find $G(n,k)$, where Gosper's algorithm is used. This means that an identity can only be proved by Wilf-Zeilberger's method if the conditions in Gosper's algorithm are fulfilled, which we will come back to in section 3.3.

We can formalize the concepts in the method like this.[6]

DEFINITION 3.21 A pair of functions (F,G) that satisfy equations 3.42 and 3.43 are said to "certify" an identity like 3.38 (or is simply called a "certifying pair"). We can also speak of a "proof certificate" $R(n,k)$ of an identity. That is a function such that $G(n,k) = R(n,k)F(n,k-1)$ gives that (F,G) is a certifying pair to the identity. □

Let us show an short example of how the method can work.

EXAMPLE 3.22 Show that

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

SOLUTION We will use the method and do all the steps mentioned above.

1. We have $A(n,k) = \binom{n}{k}$ and $B(n) = 2^n$.
2. Let
$$F(n,k) = \frac{A(n,k)}{B(n)} = \frac{\binom{n}{k}}{2^n}.$$
3. With
$$G(n,k) = -\frac{\binom{n}{k-1}}{2^{n+1}},$$
we have that
$$\begin{aligned}F(n+1,k) - F(n,k) &= \frac{\binom{n+1}{k}}{2^{n+1}} - \frac{\binom{n}{k}}{2^n} = \\ &= \frac{1}{2^{n+1}}\binom{n}{k}\left(\frac{n+1}{n+1-k} - 2\right) = \\ &= \frac{1}{2^{n+1}}\binom{n}{k}\left(\frac{k}{n+1-k} - 1\right) = \\ &= \frac{1}{2^{n+1}}\left(\binom{n}{k-1} - \binom{n}{k}\right) = \\ &= G(n,k+1) - G(n,k)\end{aligned}$$
Furthermore, we have that $G(n,k) = 0$ when $k < 0$ and $k > n$.
4. Lastly we need to show that
$$\sum_{k=0}^{n} \frac{\binom{n}{k}}{2^n} = 1$$

for some n. If we choose $n = 0$ we see that this equality holds. Therefore the equality holds for all n. □

Now that we have seen an example, hopefully the methodology is clear. What still is not clear is how to find the function $G(n,k)$. This is in general a hard task to do by hand, but is certainly managable by using a computer since there are algorithms – for instance Gosper's algorithm – that can derive $G(n,k)$ in certain cases.

3.3 Gosper's algorithm

Gosper's algorithm is an algorithm that can be used to find a sum when certain conditions are fulfilled for the sum.[7] The setting of the problem that is solved using the algorithm is that we want to find S_k where

$$\sum_{k=1}^{n} a_k = S_n - S_0. \tag{3.44}$$

This is the same thing as finding S_n such that

$$a_k = S_k - S_{k-1}. \tag{3.45}$$

The algorithm provides those S_k such that

$$\frac{S_k}{S_{k-1}} = \text{rational function of } k. \tag{3.46}$$

This is the same thing as that

$$\frac{a_k}{a_{k-1}} = \text{rational function of } k. \tag{3.47}$$

Therefore Wilf-Zeilberger's method only works if

$$\frac{a_k}{a_{k-1}} = \frac{F(n+1,k) - F(n,k)}{F(n+1,k-1) - F(n,k-1)} = \text{rational function of } k. \tag{3.48}$$

Note that all the time when we write x_a in this section, this denotes a polynomial x evaluated in a, or $x(a)$. For instance S_k denotes that the polynomial $S(k)$. We are only using this notation to make it easier to read. Note that this means that x is a function of the variable a, but does not mean that it is a one variable function of a – it can be a function of several variables but we are just interested in the variable a while the others are viewed as constants. This will actually be the case amost all the time, since $a_k = F(n+1,k) - F(n,k)$ (where F comes from Wilf-Zeilberger's method) will be used most of the time. Also here, note that $F(n,k)$ only denotes that F is a function of n and k, but it might be of more variables as well.

In order to make the thesis more easily readable and for completeness, we will provide a quite thorough derivation of the algorithm – even though it is perfectly described in the original paper by Gosper (1978). This will mostly focus on the steps of the algorithm, but also include the proofs of claims that are made during the steps. Firstly we will describe the main steps and thereafter we will provide the proofs.

First of all we need to show the connection between the formulation of Wilf-Zeilberger's method and how Gosper's algorithm takes part in it. Gosper's algorithm will solve the third step in Wilf-Zeilberger's method, namely to find a $G(n,k)$ such that conditions 3.42 and 3.43 will be fulfilled. The first condition is that

$$F(n+1,k)-F(n,k)=G(n,k+1)-G(n,k). \tag{3.49}$$

By looking at n as a constant we define $a_k = F(n+1,k)-F(n,k)$. Thereafter we use Gosper's algorithm to find a S_k such that

$$a_k = S_k - S_{k-1}. \tag{3.50}$$

Once we have that, we let $G(n,k) = S_{k-1}$ and we have obtained our function $G(n,k)$.

The steps of Gosper's algorithm are:

1. Finding polynomials p_k, q_k, r_k such that
$$\frac{a_k}{a_{k-1}} = \frac{p_k}{p_{k-1}}\frac{q_k}{r_k} \tag{3.51}$$
and $\gcd(q_k, r_{k+j}) = 1, \ \forall\ j \geq 0.$
2. Finding polynomial f_k such that
$$p_k = q_{k+1}f_k - r_k f_{k-1}. \tag{3.52}$$
3. Let
$$S_k = \frac{q_{k+1}}{p_k} f_k a_k. \tag{3.53}$$

The reason this algorithm will provide a solution is that with S_k given by 3.53 we have

$$S_k - S_{k-1} = \frac{q_{k+1}}{p_k} f_k a_k - \frac{q_k}{p_{k-1}} f_{k-1} a_{k-1}. \tag{3.54}$$

By factoring out $\frac{a_k}{p_k}$ in 3.54, and using 3.51 followed by 3.52 gives us:

$$\begin{aligned} S_k - S_{k-1} &= \frac{a_k}{p_k}\Big(q_{k+1}f_k - \frac{q_k}{p_{k-1}}f_{k-1}p_k\frac{a_{k-1}}{a_k}\Big) = \\ &= \frac{a_k}{p_k}\Big(q_{k+1}f_k - \frac{q_k}{p_{k-1}}f_{k-1}p_k\frac{p_{k-1}}{p_k}\frac{r_k}{q_k}\Big) = \\ &= \frac{a_k}{p_k}\Big(q_{k+1}f_k - r_k f_{k-1}\Big) = \frac{a_k}{p_k}p_k = a_k, \end{aligned} \tag{3.55}$$

which is exactly what we wanted to be true for S_k.

Now, we have quite a few details to derive in each step – both how the step is performed more precisely and furthermore explaining why all the polynomials actually need to be polynomials and not rational polynomials instead.

3.3.1 How to get polynomials p,q,r in equation 3.51

First of all, we need to prove that it is possible to find polynomials p,q,r such that 3.51 is fulfilled. Since $\frac{S_k}{S_{k-1}}$ is a rational function of k, then

$$\frac{a_k}{a_{k-1}} = \frac{S_k - S_{k-1}}{S_{k-1} - S_{k-2}} = \frac{\frac{S_k}{S_{k-1}} - 1}{1 - \frac{S_{k-2}}{S_{k-1}}} \tag{3.56}$$

is a rational function of k as well. Therefore we will be able to find polynomials such that 3.51 is fulfilled. Left is to show how to find polynomials such that $gcd(q_k, r_{k+j}) = 1 \;\; \forall\, j \geq 1$ as well.

We do this in a series of steps. First we let $p_k = 1$ and q_k, r_k be the numerator and denominator of $\frac{a_k}{a_{k-1}}$, respectively. Then if $gcd(q_k, r_{k+j}) = 1, \;\; \forall\, j \geq 0$ we are done. Otherwise we replace the polynomials by

$$\begin{aligned} q'_k &\leftarrow \frac{q_k}{g_k}, \\ r'_k &\leftarrow \frac{r_k}{g_{k-j}}, \\ p'_k &\leftarrow p_k g_k g_{k-1} \cdots g_{k-j+1}, \end{aligned} \tag{3.57}$$

where $g_k = gcd(q_k, r_{k+j})$ for the smallest $j \geq 0$ such that $gcd(q_k, r_{k+j}) \neq 1$. Then we can see that we still have that

$$\frac{a_k}{a_{k-1}} = \frac{p'_k}{p'_{k-1}}\frac{q'_k}{r'_k} \tag{3.58}$$

and the degrees of the polynomials q_k, r_k are smaller. We then iterate this procedure as long as there exists a $j \geq 0$ such that $gcd(q_k, r_{k+j}) > 1$. When this procedure finishes, we have obtained q_k, r_k, p_k such that 3.51 is fulfilled and $gcd(q_k, r_{k+j}) = 1 \;\; \forall\, j \geq 0$.

3.3.2 How to get polynomial f in equation 3.52

The proof provided here is just due to Gosper's paper and mostly very similar. First of all, we need to prove that f is a polynomial and not a rational polynomial.

THEOREM 3.23 There exists a polynomial f_k such that equation 3.52 is fulfilled.□

Proof Assume that $f_k = \frac{c_k}{d_k}$ where $gcd(c_k, d_k) = 1$. Then by plugging this into 3.52 and multiplying by $d_k d_{k-1}$ gives us:

$$d_k d_{k-1} p_k = c_k d_{k-1} q_{k+1} - d_k c_{k-1} r_k. \tag{3.59}$$

Let j be the largest integer such that

$$gcd(d_k, d_{k+j}) = g_k \neq 1 \tag{3.60}$$

Since $g_k | d_{k+j}$ we get that

$$gcd(d_{k-1}, d_{k+j}) = 1 = gcd(d_{k-1}, g_{k+j}). \tag{3.61}$$

By just shifting k by $-j-1$ in 3.60 we get that

$$gcd(d_{k-j-1}, d_{k-1}) = g_{k-j-1} \neq 1 \tag{3.62}$$

By shifting k by j in 3.61 we get that

$$gcd(d_{k-j-1}, d_k) = 1 = gcd(g_{k-j-1}, d_k), \tag{3.63}$$

again since $g_{k-j-1} | d_{k-j-1}$.

Now we consider equation 3.59 upon dividing by first g_k and then g_{k-j-1}. Clearly $g_k | d_k d_{k-1} p_k$, since $g_k | d_k$. This means that g_k divides the right hand side of equation 3.59 as well:

$$g_k | c_k d_{k-1} q_{k+1} - d_k c_{k-1} r_k. \tag{3.64}$$

Furthermore $g_k | d_k c_{k-1} r_k$ since $g_k | d_k$, which gives us that

$$g_k | c_k d_{k-1} q_{k+1}. \tag{3.65}$$

By equation 3.61 we get $gcd(d_{k-1}, g_k) = 1$ and since $g_k | d_k$ and $gcd(c_k, d_k) = 1$ we get $gcd(c_k, g_k) = 1$. This means that

$$g_k | q_{k+1} \implies g_{k-1} | q_k. \tag{3.66}$$

Similarly $g_{k-j-1} | d_k d_{k-1} p_k$, since $g_{k-j-1} | d_{k-1}$. This means that

$$g_{k-j-1} | c_k d_{k-1} q_{k+1} - d_k c_{k-1} r_k. \tag{3.67}$$

Furthermore $g_{k-j-1}|c_k d_{k-1} q_{k+1}$ since $g_{k-j-1}|d_{k-1}$, which gives us that

$$g_k|d_k c_{k-1} r_k. \tag{3.68}$$

By equation 3.63 we get $gcd(d_k, g_{k-j-1}) = 1$ and by $gcd(c_{k-1}, d_{k-1}) = 1$ together with $g_{k-j-1}|d_{k-1}$ we get $gcd(c_{k-1}, g_{k-j-1}) = 1$. This means that

$$g_{k-j-1}|r_k \implies g_{k-1}|r_{k+j}. \tag{3.69}$$

But now we have g_{k-1} divides both r_{k+j} where $j \geq 0$ and q_k. From the first step of Gosper's algorithm we know that $gcd(r_{k+j}, q_k) = 1, \ \forall \ j \geq 0$ meaning that $g_k = 1$. This means that for all $j \geq 0$ we have that

$$gcd(d_k, d_{k+j}) = 1. \tag{3.70}$$

By putting $j = 0$ we get that

$$d_k = gcd(d_k, d_k) = 1. \tag{3.71}$$

Hence $d_k = 1$ for all k and thus f_k is a polynomial. □

Now, let us derive how to get the polynomial f_k such that 3.52 is fulfilled. It is possible to get a bound for the degree of f_k with respect to k. This is derived in the paper by Gosper (1978) and for the interested reader the details can be found there. The results is that

$$deg(f_k) \leq deg(p_k) - max\big(deg(q_{k+1} + r_k), deg(q_{k+1} - r_k)\big) + 1. \tag{3.72}$$

In some cases the degree can be further limited (without the extra $+1$) but we are mostly interested in that we have a bound – not the extra added (small) constant.

We will now firstly show that there is a way to get the polynomial f_k given that the degree of f_k is less than N for all variables in f_k. This is done by assigning

$$f_k = \sum_{\substack{\mathbf{i} \in (0,1,\ldots,N)^{m-1} \\ i_1 \in (0,1,\ldots,N)}} \left(a_{\mathbf{i},i_1} k^{i_1} \prod_{j=2}^{m} v_j^{i_j} \right), \tag{3.73}$$

where m is the number of variables in f_k, the variables of f_k are named v_j for $1 \leq j \leq m$ where $v_1 = k$, $a_{\mathbf{i}}$ are the coefficients and $\mathbf{i} = (i_2, i_3, \ldots, i_m)$ goes through all combinations of degrees in the range $0 \leq deg(v_j) = i_j \leq N$ for all the different variables. The number i_1 is the exponent of k and also goes through all the values for in the range $0 \leq i_1 \leq N$. This exponent is treaded slightly different just because it is the exponent of k.

Then from this we can derive what the polynomial f_{k-1} is. First we write the polynomial f_{k-1} on the same form as in 3.73 but with other coefficients:

$$f_{k-1} = \sum_{\mathbf{i}\in(0,1,\dots,N)^m} \left(b_{\mathbf{i},i_1} k^{i_1} \prod_{j=2}^{m} v_j^{i_j} \right). \tag{3.74}$$

Now we will derive the expression for $b_{\mathbf{i},i_1}$ in terms of $a_{\mathbf{i},i_1}$. If we replace k by $k-1$ in 3.73 we get

$$\begin{aligned} f_{k-1} &= \sum_{\substack{\mathbf{i}\in(0,1,\dots,N)^{m-1}\\ i_1\in(0,1,\dots,N)}} \left(a_{\mathbf{i},i_1}(k-1)^{i_1} \prod_{j=2}^{m} v_j^{i_j} \right) = \\ &= \sum_{\substack{\mathbf{i}\in(0,1,\dots,N)^{m-1}\\ i_1\in(0,1,\dots,N)}} \left(a_{\mathbf{i},i_1} \left(\sum_{u=0}^{i_1} \binom{i_1}{u} k^u (-1)^{i_1-u} \right) \prod_{j=2}^{m} v_j^{i_j} \right) = \\ &= \sum_{\substack{\mathbf{i}\in(0,1,\dots,N)^{m-1}\\ i_1\in(0,1,\dots,N)}} \left(\sum_{u=0}^{i_1} a_{\mathbf{i},i_1} \binom{i_1}{u} (-1)^{i_1-u} k^u \right) \prod_{j=2}^{m} v_j^{i_j} \end{aligned} \tag{3.75}$$

Now we can see that the term $k^{i'_1} \prod_{j=2}^{m} v_j^{i_j}$ from 3.74 (i'_1 denotes a specific i_1) appears in 3.75 in several places – namely when $i_1 \in \{i'_1, i'_1+1, \dots, N\}$. This means that we get

$$b_{\mathbf{i},i_1} = \sum_{s=i_1}^{N} a_{\mathbf{i},s} \binom{s}{i_1} (-1)^{s-i_1} \tag{3.76}$$

Now we have all the background we need in order to set up the system of equations and then solve for $a_{\mathbf{i},i_1}$. This is done by noticing that the left and right hand sides of the equation

$$p_k = q_{k+1} f_k - r_k f_{k-1} \tag{3.77}$$

In order to ease the formulation of the equation system, let $\hat{\mathbf{i}} = (\mathbf{i}, i_1)$, let $\ll$ denote that a vector is pointwise smaller than or equal to e.g. $(1,2,3) \ll (2,3,3)$ and let $\mathbf{i}-\mathbf{j}$ denote the pointwise difference between $\mathbf{i}$ and $\mathbf{j}$, e.g. $(2,3,3)-(1,2,3) = (1,1,0)$. Finally we also denote the coefficients in front of $k^{i_1} \prod_{j=2}^{N} v_j^{i_j}$ in the polynomials p_k, q_{k+1}, r_k by $r_{\hat{\mathbf{i}}}, q_{\hat{\mathbf{i}}}, r_{\hat{\mathbf{i}}}$, respectively. This gives us that

$$p_{\hat{\mathbf{i}}} = \sum_{\substack{\hat{\mathbf{j}}\\ \hat{\mathbf{j}}\ll\hat{\mathbf{i}}\\ \hat{\mathbf{j}}'=\hat{\mathbf{i}}-\hat{\mathbf{j}}}} a_{\hat{\mathbf{j}}} q_{\hat{\mathbf{j}}'} - \sum_{\substack{\hat{\mathbf{j}}\\ \hat{\mathbf{j}}\ll\hat{\mathbf{i}}\\ \hat{\mathbf{j}}'=\hat{\mathbf{i}}-\hat{\mathbf{j}}}} b_{\hat{\mathbf{j}}} r_{\hat{\mathbf{j}}'}, \tag{3.78}$$

where $b_{\hat{\mathbf{j}}}$ is given by 3.76. Note that at this point in the algorithm all the numbers $p_{\hat{\mathbf{i}}}, q_{\hat{\mathbf{i}}}, r_{\hat{\mathbf{i}}}$ are simply constants. This means that we can formulate a system of equations

$$Ma = P, \tag{3.79}$$

where M is the matrix that is given by the right hand side of 3.78, a consists of all coefficients $a_{\hat{\mathbf{j}}}$ and P is the left hand side of 3.78. Note that each row in this system of equations corresponds to a different $\hat{\mathbf{i}}$. Here we go through all $\hat{\mathbf{i}}$ that are relevant – meaning all $\hat{\mathbf{i}}$ with degree smaller than what either of the sides of equation 3.78 can have. This means that we have an overdetermined system of equations. Furthermore we demand that all the coefficients of f_k have to be integers. This means that it is not certain that we will get a solution, but in case we do, then we have found f_k. If we do not get a solution that can be because of two different reasons:

i) We chose a too small maximal degree N for the polynomial f_k.

ii) There does not exist a solution, hence the identity cannot be shown by Wilf-Zeilberger's method.

In the paper by Gosper (1978) the first reason for not finding a f_k does not exists, since the paper proves an upper limit on the degree of f_k with respect to k. This cannot be done for other variables, for instance with $p_k = q_{k+1} = r_k = 1$ any polynomial f_k on the form $f_k = n^c + k$ satisfies 3.77.

In the system 3.79, M can be constructed to have dimension $L \times (N^m)$, where $L = max(deg(p_k) + 1, max(deg(q_{k+1}), deg(r_k)) + N + 1)$. Here $deg(g)$ denotes the largest degree in any variable of the polynomial g. The reason for this limit is that the row with highest degree for any variable can be chosen to be L, since the degree of either side of the equation 3.78 is at most L.

The time complexity for this system of equations is $\mathcal{O}(L \cdot N^{2m})$, since we perform the following algorithm:

Algorithm 3.11 Gaussian Elimination

Input: Matrix M, vector P
Output: Vector x such that $Mx = P$

```
procedure GaussianElimination($M, P$)
    $r \leftarrow$ Number of rows in M
    $c \leftarrow$ Number of columns in M
    for $row \leftarrow 1, \ldots, c$ do
        $i \leftarrow$ first row with row number $\geq row$ and $M[i][row] \neq 0$
        if No such $i$ exists then
            Continue loop
        end if
        $swap(M[row], M[i])$
        $swap(P[row], P[i])$
        for $row_2 \leftarrow row + 1, \ldots, L$ do
            $g \leftarrow gcd(M[row][c], M[row_2][c])$
            $M[row_2] \leftarrow M[row_2] \cdot (M[row][c]/g) - M[row] \cdot (M[row_2][c]/g)$
            $P[row_2] \leftarrow P[row_2] \cdot (M[row][c]/g) - P[row] \cdot (M[row_2][c]/g)$
        end for
    end for
    for $row \leftarrow c + 1, \ldots, r$ do
        if $P[row] \neq 0$ then
            return $None$
        end if
    end for
    $x \leftarrow [P[c]/M[c][c]]$
    for $row \leftarrow c - 1, \ldots, 1$ do
        $s \leftarrow 0$
        for $row_2 \leftarrow row + 1, \ldots, c$ do
            $s \leftarrow s + M[row][row_2] \cdot P[row_2]$
        end for
        if $M[row][row] = 0$ then
            if $s = P[row]$ then
                $x \leftarrow [0] + x$
            else
                return 0
            end if
        else
            $x \leftarrow [(P[row] - s)/M[row][row]] + x$
        end if
    end for
    return $x$
end procedure
```

REMARK In the code we multiply a row vector by an integer. This means that we perform elementwise multiplication with this integer. □

REMARK Each time we divide in the algorithm we need to make sure that the resulting integer is the exact division, otherwise we have gotten an error and return *None*. □

Here we can clearly see that we have two loops, which are of size $c = N^m$ and $r = L$, respectively. The reason for the last N^m-factor is that when we perform row 9–14 we do this elementwise, hence once for each column.

This means that the time complexity is exponential in the number of variables. In general exponential time complexities are not desired at all – since they grow very fast. In this thesis however we are working with identities that just involve a few (less than 5) different variables. Therefore this time complexity is acceptable. This means that given that given the limit on m we have a polynomial time complexity for obtaining f_k.

4 Implementation

The implementation of the project was done in python. In general classes containing logic for different objects were implemented, for instance classes for polynomials, factorials, powers and so on were implemented. All classes were implemented from scratch and we want to stress that this makes some of the code redundant – surely it is implemented by others before – and inefficient – no highly efficient methods are used, but only the most simple versions. The reason the classes were implemented from scratch anyways was to more deeply understand the problems arising within computer algebra, and during the process it became clear that even methods that are very simple in theory can cause several hours of implementation and many lines of code.

The implementation of Wilf-Zeilberger's method can be divided into mainly three parts: parsing, testing and methods. Parsing is when we read a string and convert this into classes or logic that is used in the rest of the code. This is mainly to ease testing of the methods that are implemented but also for running the main program. Testing is when we test the methods and see that they behave as intended. Methods is the ”real” code for the thesis, which is when we implement the logic that is described in chapter 3. In total the code produced in the thesis consists of about 2200 lines of code out of which about 50% are the methods and classes used to implement the theory from chapter 3, 20% is parsing and 30% is testing. The testing is of course not necessary to keep for the sake of having functioning code, but is kept anyways in order to show that the code has been tested.

The code consists of eight files that do different parts of the method. These are:

File name	What does the file include?
Polynomial	A class for polynomials with all methods that are needed.
Factor	Classes for factorials and powers (integer to the power of polynomials)
Expressions	Classes for storing expressions of factors, addends and fractional expressions
Main	Main function that takes a parser and an input string as input and produces a LaTeX-file with a part proof where the user only needs to do a few steps (which are indicated in the LaTeX-file)
Gosper	Functions for running all steps of Gosper's algorithm and returning the solution
Get f	Function for solving step 2 of Gosper's algorithm
Gaussian elimination	Function for solving a system of linear equations by using Gaussian elimination, which is used by Get f
Parsers	Parser from a LaTeXequation to the format that is used in the rest of the program

The dependencies of the files can be drawn as follows where an arrow from *A* to *B* means that *B* depends on *A*:

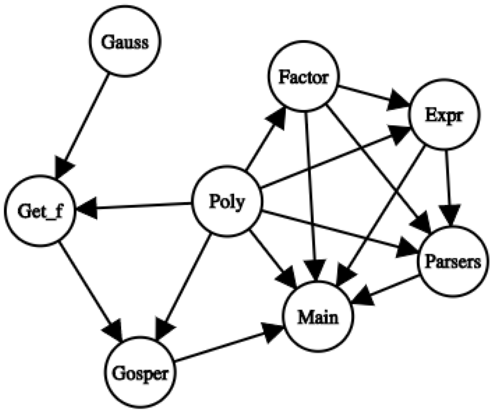


Figure 4.1 Dependency graph of code. Expr and Poly is short for Expressions and Polynomial, respectively.

Now that we have shown the general structure of the program, we will describe each of the important and interesting parts of the implementation. Firstly we will describe the structure used for parsing, thereafter we will discuss implementation of the methods needed for Wilf-Zeilberger's method. After that we will describe how a proof is written automatically and lastly we will describe how testing has been done.

4.1 Parsing

In the project several parsers are used. The parsers that have been implemented are shift-reduce parsers, which are a type of bottom-up parsers. In this type of parser a stack is used and a string is parsed character by character. When a character is read first the stack is reduced and then the character is pushed to the stack. That the stack is reduced means that if we read for instance a "+", then first all multiplications on the top of the stack are evaluated and after that we push the "+". For more details and further explanation of a shift-reduce parser, see [8].

In the code there are four parsers, which all read a string of a specific format and return either an object that the string represents or a string on another format. We are now going to describe the parsers with what they have as input and output, but first we need to describe an internal format that is used in the program to represent binomial coefficients, factorials and powers of integers. This format will in the rest of the report be referred to as *the internal format* These three will be represented as follows:

What?	**Represented as:**	**Example:**
Factorial	$F[x]$, where x is a polynomial	$(m+n)!$ is written as $F[n+m]$
Binomial coefficient	$B[n,k]$ where n,k are polynomials	$\binom{n+1}{k-1}$ is written as $B[n+1,k-1]$
Power of integer	$P[a,n]$ where a is an integer and n is a polynomial	2^{n+m} is written as $P[2,n+m]$

The parsers are described by the following table:

Parser	Input format	Output format
Polynomial parser	String with a polynomial, for instance ”n^2+3km”	Polynomial that the input string represents
Expression parser	String with an expression on the internal format, for instance ”(n!+k*m)/(n^2+B[n,k])”	Expression that the input string represents
LaTeXparser	String in LaTeXformat of an identity, for instance ”\sum_{k=0}^n \binom{n}{k}=2^n”	F, a_k and the numerator and denominator of $\frac{a_k}{a_{k-1}}$ as the first step in the method after which Gosper’s algorithm can be run
Internal format to LaTeXparser	String on the internal format	String on LaTeXformat, which is used when generating the proof file

4.2 Gosper’s algorithm

Although the whole algorithm is described in section 3.3, we will again go through the algorithm but from the perspective of implementation, and trying to point out difficulties along the way. There are some parts where the theory is straight forward, but the details in the implementation are quite difficult. In this section we try to describe the choices that were made during the work of the thesis, and also point out advantages and disadvantages with these choices.

4.2.1 Finding p,q,r

In section 3.3.1 an algorithm is described for how to find polynomials p, q, r such that

$$\frac{a_k}{a_{k-1}} = \frac{p_k}{p_{k-1}} \frac{q_k}{r_k}, \tag{4.1}$$

such that $gcd(q_k, r_{k+j}) = 1 \ \ \forall \ j \geq 0$. The algorithm can be described as follows:

Algorithm 4.1 Get p, q, r

```
Input: Rational polynomial $b = \frac{b_n}{b_d} = \frac{a_k}{a_{k-1}}$
Output: Polynomials $p, q, r$ such that 4.1 holds and $gcd(q_k, r_{k+j}) = 1 \quad \forall\ j \geq 0$

procedure GETPQR($b_n, b_d$)
    $r_k \leftarrow b_d$
    $q_k \leftarrow b_n$
    $p_k \leftarrow 1$
    while $\exists\ j \geq 0 : gcd(q_k, r_{k+j}) \neq 1$ do
        $j \leftarrow j$ such that $gcd(q_k, r_{k+j}) \neq 1$
        $g_k \leftarrow gcd(q_k, r_{k+j})$
        $q_k \leftarrow \frac{q_k}{g_k}$
        $r_k \leftarrow \frac{r_k}{g_{k-j}}$
        $p_k \leftarrow p_k g_k g_{k-1} \cdots g_{k-j+1}$
    end while
    return $p_k, q_k, r_k$
end procedure
```

This algorithm is in theory fairly simple – find a common factor and get rid of it – but finding $gcd(q_k, r_{k+j})$ for all j and see if this can be a common factor is tricky implementationwise. In the thesis this is implemented in the following fashion:

Algorithm 4.2 Find $GCD(q_k, r_{k+j}) \quad \forall\ j \geq 0$

```
Input: Polynomials $q_k, r_k$
Output: Polynomial $g_k = gcd(q_k, r_{k+j}) \neq 1$ for some $j \geq 0$

procedure GCDWITHSHIFT($q_k, r_k$)
    $g_{j,0} \leftarrow r_k$ evaluated at $k = k + j$
    $g_{j,1} \leftarrow \mathrm{GCD}(q_k, g_{j,0})$
    $i \leftarrow 1$
    while $g_{j,i} \neq 1$ do
        if $\exists\ j' \geq 0 : g_{j,i}(j') = 0$ then
            return $j'$
        end if
        $i \leftarrow i + 1$
        $g_{j,i} \leftarrow \mathrm{GCD}(g_{j,i-2}, g_{j,i-1})$
    end while
    return $None$
end procedure
```

REMARK Note that the procedure cannot always find a $g_k \neq 1$ such that $g_k = GCD(q_k, r_{k+j})$ for some integer $j \geq 0$. If it cannot find such g_k then the procedure returns *None*, which can be seen as an error in finding the gcd. □

Here we perform the check in the if-statement on line 6 by calling procedure 3.7 and then checking if there is any root that is positive. Since this is the crucial step of this algorithm both the advantages and disadvantages come from the Roots-algorithm. In the implementation of Roots, only up to second degree polynomials are solved completely and after that values in the interval $(-100, 100)$ are tested. This means that we cannot be sure that we find the roots. In general when we can use Wilf-Zeilberger's method, however, all roots are fairly small – almost always less than 100. Therefore even though the implementation in theory is limiting it is very likely not to cause any problems in practice.

4.2.2 Finding f

The implementation in order to find the polynomial f in Gosper's algorithm is performed just as described in section 3.3.2. The goal of that is to find f such that

$$p_k = q_{k+1} f_k - r_k f_{k-1}. \tag{4.2}$$

First the matrix M and the vector P are constructed. M comes from the right hand side of 4.2 after assigning f as stated in equation 3.73. P is the vector that represent the coefficients on the left hand side. After the construction, Gaussian elimination is performed and then hopefully an answer will be given.

In case no answer has been found one can try to increase the maximal degree of f. It is not certain however that a solution can be given by this method since possibly the degree of the smallest degree solution for f can be arbitrarily large with respect to the variables that are not k. If a solution is given, however, then we have performed the whole Gosper's algorithm.

4.3 Proof generation

The proof generation is a method which needs an input string and a parser. Then the method produces a proof, written in LaTeXcode, for that the input string in fact is true. Right now the parser that has been developed in the thesis can only handle a few different types of operators in the input string – binomial coefficients, factorials, integer powers and polynomials can be handled, but no other operators. The reason for this is that the parser mostly is an example of how Wilf-Zeilberger's method can be used, and is not meant to be a full solution. Instead users can write their own parsers and input strings which convert the string into four things: F, a_k and the numerator and denominator of the fraction $\frac{a_k}{a_{k-1}}$. When the user has provided this parser and input string, then the main-function does the rest of the job in producing the proof in LaTeXformat, which proves the identity.

The reasons for this design choice (to only provide a simple parser as an example) are a few: firstly this gives a flexibility for a user and works well in an open source setting, secondly it would be at least as time demanding as writing the rest of the method to write a more general parser for more kinds of identity parsing – and still some applications would probably be left out anyways.

The general methodology used in the proof generation is the following:

i) Get a parser and an input string as inputs.

ii) Parse the string, which gives us F, a_k and the polynomials that are the numerator and denominator of $\frac{a_k}{a_{k-1}}$.

iii) Call Gosper's algorithm which gives a $G(n,k)$ such that $G(n,k+1) - G(n,k) = a_k = F(n+1,k) - F(n,k)$. This step is the main contribution from the program, and is usually the by far hardest part of the proof.

iv) Write the proof in LaTeXformat.

v) Highlight parts of the proof that need to be verified or partly proved by the user. This includes proving that $\lim_{k\to\pm\infty} G(n,k) = 0 \;\; \forall\, n$ and may include verifying that $G(n,k+1) - G(n,k) = F(n+1,k) - F(n,k)$. This should be guaranteed by the solver, but might be worth verifying.

4.4 Testing

In general the testing has tried to test each method individually and then first after each methods works on its own it has been tested all together. The testing on each individual method has been performed in different ways for different methods, but the common thing for all of the methods is that the testing included many edge cases as well as other cases. Edge cases are usually when some parameters (for instance the degree of a polynomial or the number of variables in the polynomial) is very small or large.

The tests that were performed on methods all together are usually from one of the test examples provided in chapter 5. Then the implementation and testing was iterated until all identities worked. This testing was also performed in steps, for instance in order to test the methods in Gosper's algorithm, some initial steps were done by hand and then the relevant part was tested. Then, when all individual methods and groups of methods worked as expected, everything was put together and the final result was given.

5 Results

Throughout the thesis some identities have been used for testing the code, these will be called training examples. All identities that have been used in this thesis have been collected from [9], and [10]. When choosing the training examples many identities were considered and we tried to find a good variety of identities, both well known and not as well known ones as well as using identities that look quite different. The identities used as training examples are the following:

1. $\sum_{k=0}^{n} \binom{n}{k} = 2^n$
2. $\sum_{k=0}^{n} (-1)^k \binom{n}{k} \binom{2k}{k} 4^{n-k} = \binom{2n}{n}$
3. $\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$
4. $\sum_{k=0}^{n} 2^k \binom{n}{k} = 3^n$
5. $\sum_{k=0}^{n} k \binom{n}{k} = n2^{n-1}$
6. $\sum_{k=2}^{n} \frac{1}{k(k-1)} = 1 - \frac{1}{n}$
7. $\sum_{k=0}^{n} \binom{k}{c} = \binom{n+1}{c+1}$
8. $\sum_{k=0}^{n} \binom{r+k}{k} = \binom{r+n+1}{n}$
9. $\sum_{k=0}^{n} \binom{m-k}{n-k} = \binom{m+1}{n}$
10. $\sum_{k=n}^{\infty} \frac{1}{\binom{k}{n}} = \frac{n}{n-1}$

Some of these identities are quite trivial to show with combinatorial arguments, but others are not as easy. For instance identity 1 comes from counting in how many ways we can choose objects out of *n* different objects in two different ways, while identity 2 is not very easy to prove by providing a combinatorial argument. When these identities are shown, a LaTeXfile is produced. The proofs of all identities are provided in chapter 7. As we can see the identities 7 and 8 cannot be shown, the reason for this is that the "get f" method cannot give a solution, since f cannot be a polynomial. This is because the degree of f depends on c and r, respectively, and therefore Wilf-Zeilberger's method does not work well on the problem.

After the program worked for all the training examples the program was validated on other identities (called validation examples), in the hopes that the program works on identities it has not been developed to work on as well. Also here we tried to use validation identities that are different from each other, but now we also prioritized identities that are not very simple to solve with combinatorial arguments. The tests that were used are:

11. $\sum_{k=0}^{n} 3^k \binom{n}{k} = 4^n$
12. $\sum_{k=0}^{n} 4^k \binom{n}{k} = 5^n$
13. $\sum_{k=0}^{n} \binom{n}{2k} = 2^{n-1}$
14. $\sum_{k=1}^{\infty} \frac{1}{k\binom{k+n}{k}} = \frac{1}{n}$
15. $\sum_{k=0}^{\infty} 2^{-k} \binom{n+k}{k} = 2^{n+1}$
16. $\sum_{k=0}^{n} \frac{\binom{n}{k}}{\binom{2n-1}{k}} = 2$
17. $\sum_{k=0}^{n} k \frac{\binom{n}{k}}{\binom{2n-1}{k}} = 2\frac{n}{n+1}$
18. $\sum_{k=1}^{2n-1} (-1)^{k-1} \frac{k}{\binom{2n}{k}} = \frac{n}{n+1}$
19. $\sum_{k=0}^{n} \binom{2n+1}{k} = 2^{2n}$
20. $\sum_{k=0}^{\frac{n}{2}} (-1)^k \binom{n-k}{k} \frac{2^{2n-2k}}{n-k} = \frac{2^{n+1}}{n}$

As we can see in chapter 7 the program manages to solve most of the validation examples. The program solves identities 11, 12, 14, 15, 19 and 20 perfectly. For identities 16 and 17 the program manages to get a G which works in Wilf-Zeilberger's method but has not been able to verify that the first condition for a certifying pair is fulfilled. This has been done by hand afterwards though, which means that a user could do that. For identity 13 and 18 the program does not manage to prove the identity, this is since it cannot find a f that works. The reason for this is that the difference in degrees cannot be resolved on the right and left side when getting f. Therefore Wilf-Zeilberger's method cannot be used to solve these identities.

6

Discussion and conclusions

In the thesis Wilf-Zeilberger's method has been implemented with a library of polynomials as a foundation. During the work on the thesis it has become more and more obvious how even the simplest operations in theory might be quite tricky to implement with computer algebra. Furthermore it became brutally clear how details in program design could help or harm the rest of the dependent parts of the program. This required more thought before implementation, and also better code in general.

The program seems to work fine but has some obvious limitations. First of all only a very small part of all proofs can be shown by Wilf-Zeilberger's method. Also we saw in chapter 5 two seemingly similar identities, for instance identities 7 and 9, can have two very different outcomes when trying to use Wilf-Zeilberger's method – one is solvable and one is not! This makes it harder to trust the method to solve a problem that is given, since it seems to be quite hard to identifying if a problem can be solved by Wilf-Zeilberger's method or not.

Secondly the method needs to have an identity to prove. Wilf-Zeilberger's method itself cannot come up with a guess for what a sum is, but only prove the correctness or incorrectness of an identity once it is already guessed.

Thirdly with the current state of the implementation it is not obvious which part of the method that fails. Usually it is the part where we try to find polynomial f such that $p_k = q_{k+1}f_k - r_k f_{k-1}$, but it can also be other problems that make the program fail. Also when trying to find polynomial f, it is not certain what causes the problem. It can be both that the degree of the assigned polynomial is too small, and that the problem in fact is not solvable for some reasons.

Lastly, the Wilf-Zeilberger's method solver that has been developed leaves a few parts of the formal proof for the user to show. This is not optimal, however showing these parts automatically as well would demand even more implementation and was out of scope for this thesis.

Still, even though there are clear problems with both the Wilf-Zeilberger's method itself and the implementation that was chosen, we would argue that the thesis has value and contributes to the general knowledge about infinite sumation. If one has an identity it can quickly be checked if it is possible for the Wilf-Zeilberger's method solver to prove it. If it can, then one quickly has a proof and otherwise other methods have to be used. Even though it is not directly obvious what goes wrong when the solver fails to prove an identity, the implementation is split up in many different methods, which makes is fairly easy to get a sense of what goes wrong.

From the examples that were tested we can conclude that the automatic solver manages to solve most examples, even on the validation examples. The automatic solver showed eight out of ten identities, and afterwards it has been verified by hand that the remaining two were not solvable using Wilf-Zeilberger's method. We saw that out of the ten validation examples eight were shown and the others turned out to not be suited for Wilf-Zeilberger's method, which was verified by hand afterwards. Hence the automatic solver proves all identities that are possible to prove. It should be noted that the sample size of both the training and validation examples is very small. The reason for the results to still be valid is that the work in this thesis is more theoretical and the theoretical results have been proved in chapter 3. Therefore the examples are just used for testing the implementation, and since the results on training and validation examples are similar we can conclude that the implementation seems to be at least somewhat robust.

The problem that Wilf-Zeilberger's method cannot guess what a sum equals is a problem for the whole method. Finding ways to guess what a sum equals is also a very interesting area that would be fun to explore more, but is out of scope for this thesis.

Bibliography

[1] Veltman, M.J.G. and Williams, D.N. (1991). 'Schoonschip '91.'

[2] Lopez, R. *Computer Algebra Systems.* Maplesoft, viewed 26 November 2019. `https://www.maplesoft.com/ns/maple/cas/computer-algebra-systems-math-education.aspx`.

[3] Güyer, T. (2008). 'Computer Algebra Systems as the Mathematics Teaching Tool.' *World Applied Sciences Journal* 3 (1): 132-139, 2008.

[4] Wilf, H.S., and Zeilberger, D. (1990). 'Rational functions certify combinatorial identities.' *J. Amer. Math. Soc. 3* (1990), pp. 147–158.

[5] Paule, P. and Schorn, M. (1994). 'A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities.' *J. Symbolic Computation* (1995) 20, pp. 673–698.

[6] Wilf, H.S. (1994). *Generating functionology.* Academic Press, Inc. Philadelphia, Pennsylvania.

[7] Gosper, R.W. Jr. (1978). *Decision procedure for indefinite hypergeometric summation.* Proc. Natl. Acad. Sci. USA. Vol. 75, No. 1, pp. 40–42, January 1978. Palo Alto, California.

[8] Johnson, M. (2007). *Handout 08 Bottom-up Parsing.* Lecture notes Stanford University CS143. Delivered 2 July 2007. `https://web.archive.org/web/20160305041504/http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/08-Bottom-Up-Parsing.pdf`

[9] Gould, H.W. (1972). *Combinatorial Identities.* Morgamtown Printing and Binding Co. West Virginia.

[10] Tesler (2017). *Chapter 3.3, 4.1, 4.3. Binomial Coefficient Identities.* Lecture notes University of California San Diego Math 184A. Delivered winter 2019. `http://www.math.ucsd.edu/~gptesler/184a/slides/184a_ch4slides_19-handout.pdf`

7 Attachments

7.1 Code

All code is available on `https://github.com/LarsAstrom/Wilf-Zeilbergers-Method/tree/master/WZmethod`.

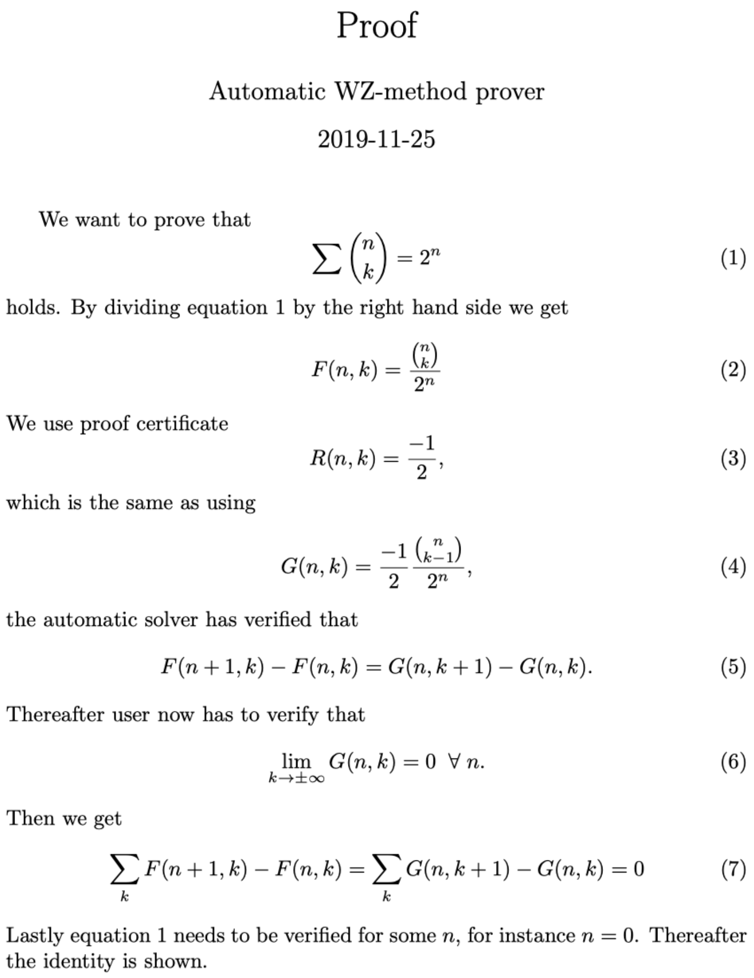
7.2 Proofs of identities

We want to remind the reader that in Wilf-Zeilberger's method we find a polynomial $F(n,k)$. Then we want to find another polynomial G such that (F,G) is a certifying pair of polynomials, meaning that

1. $F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k)$, and
2. $\lim_{k\to\pm\infty} G(n,k) = 0, \quad \forall\ n.$

We can also express this by finding a proof certificate $R(n,k)$ such that

$$G(n,k) = R(n,k)F(n,k-1).$$



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \binom{n}{k} = 2^n \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\binom{n}{k}}{2^n} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-1}{2}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-1}{2}\frac{\binom{n}{k-1}}{2^n}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

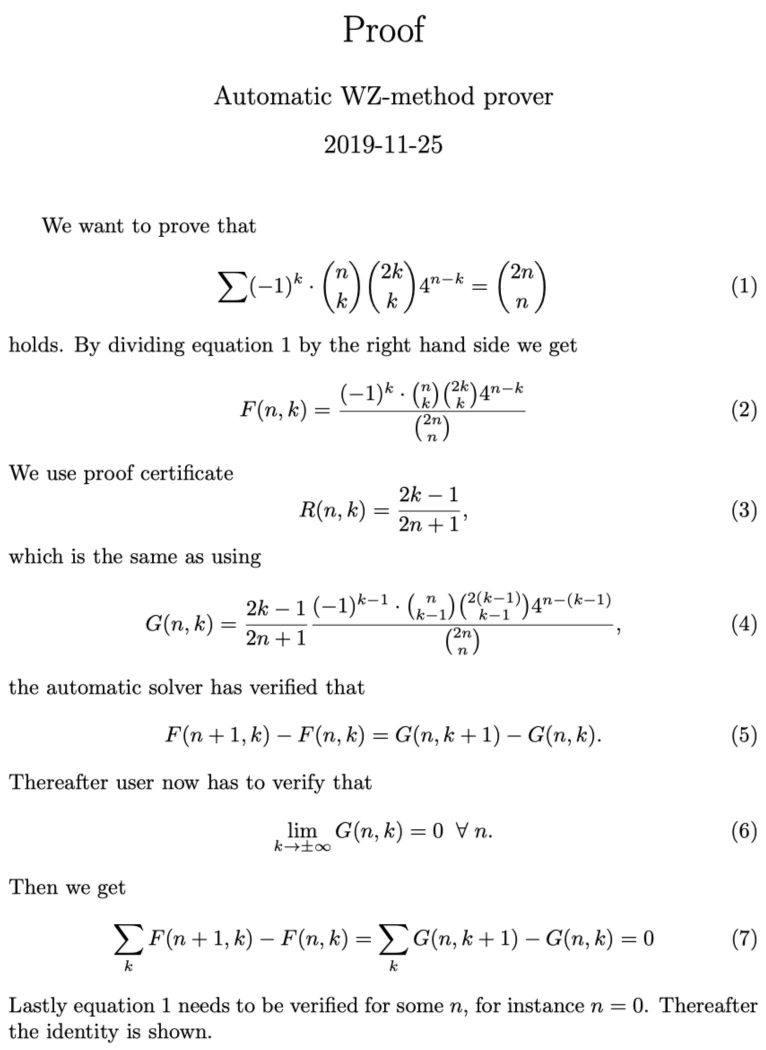
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.1 Proof of identity 1.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum(-1)^k \cdot \binom{n}{k}\binom{2k}{k}4^{n-k} = \binom{2n}{n} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{(-1)^k \cdot \binom{n}{k}\binom{2k}{k}4^{n-k}}{\binom{2n}{n}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{2k-1}{2n+1}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{2k-1}{2n+1}\frac{(-1)^{k-1} \cdot \binom{n}{k-1}\binom{2(k-1)}{k-1}4^{n-(k-1)}}{\binom{2n}{n}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

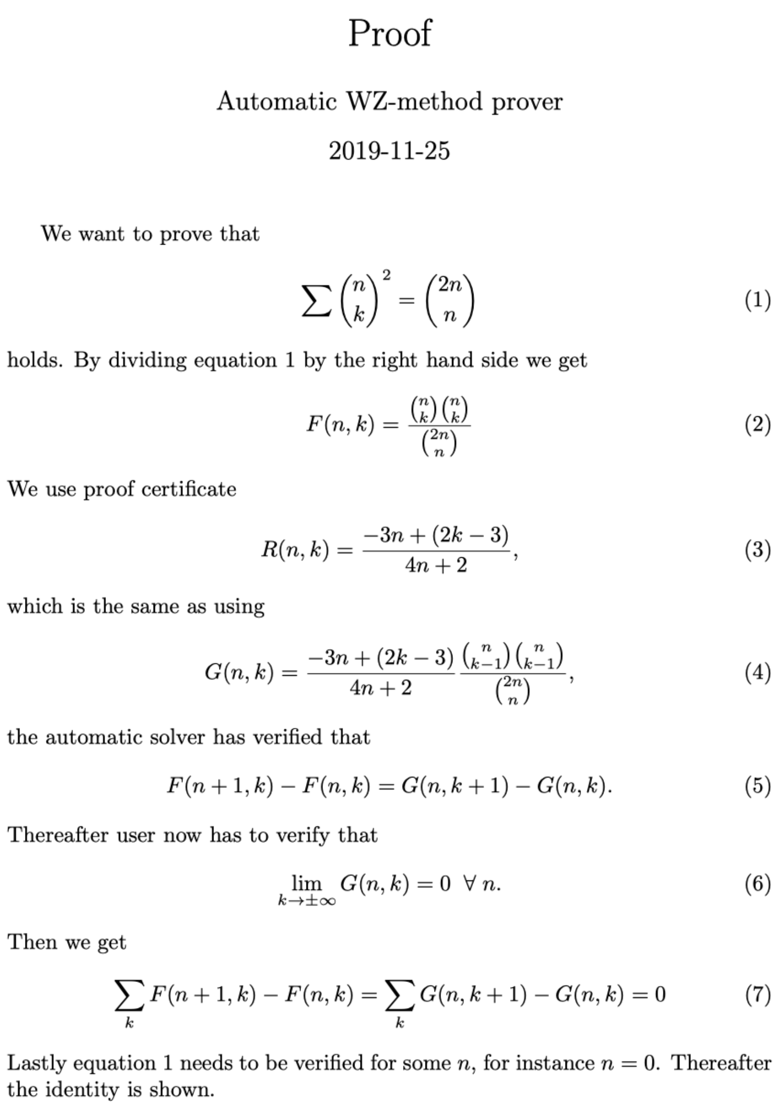
$$\lim_{k\to\pm\infty} G(n,k) = 0 \;\; \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.2 Proof of identity 2.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \binom{n}{k}^2 = \binom{2n}{n} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\binom{n}{k}\binom{n}{k}}{\binom{2n}{n}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-3n + (2k-3)}{4n+2}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-3n + (2k-3)}{4n+2} \frac{\binom{n}{k-1}\binom{n}{k-1}}{\binom{2n}{n}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

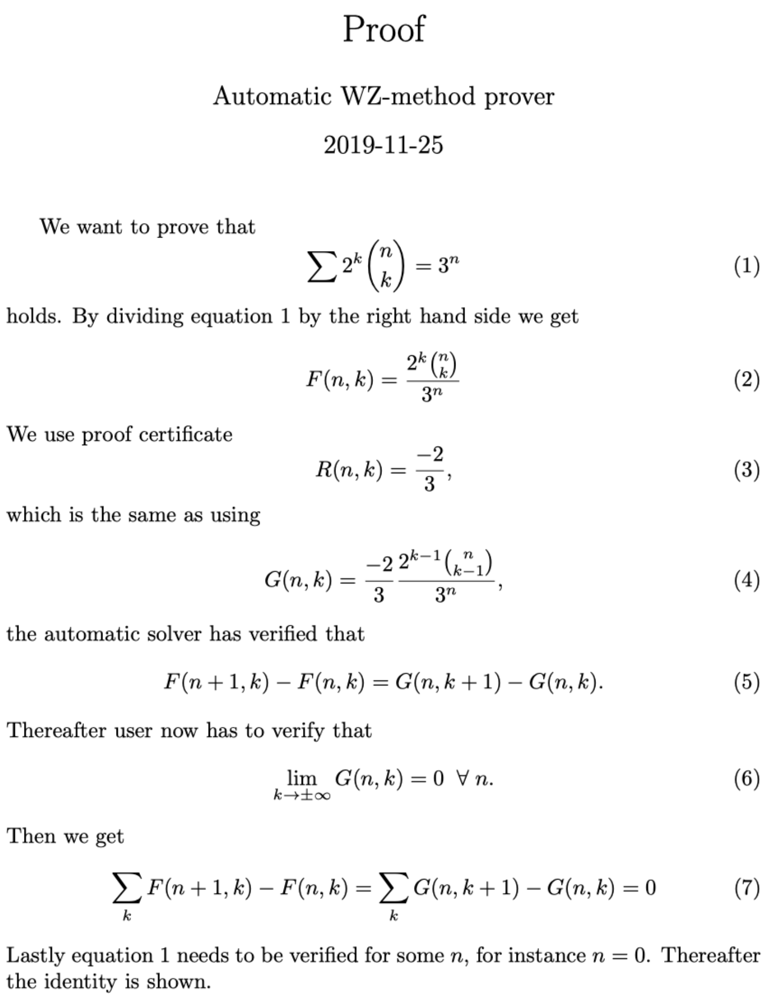
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.3 Proof of identity 3.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum 2^k \binom{n}{k} = 3^n \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{2^k \binom{n}{k}}{3^n} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-2}{3}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-2}{3} \frac{2^{k-1} \binom{n}{k-1}}{3^n}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

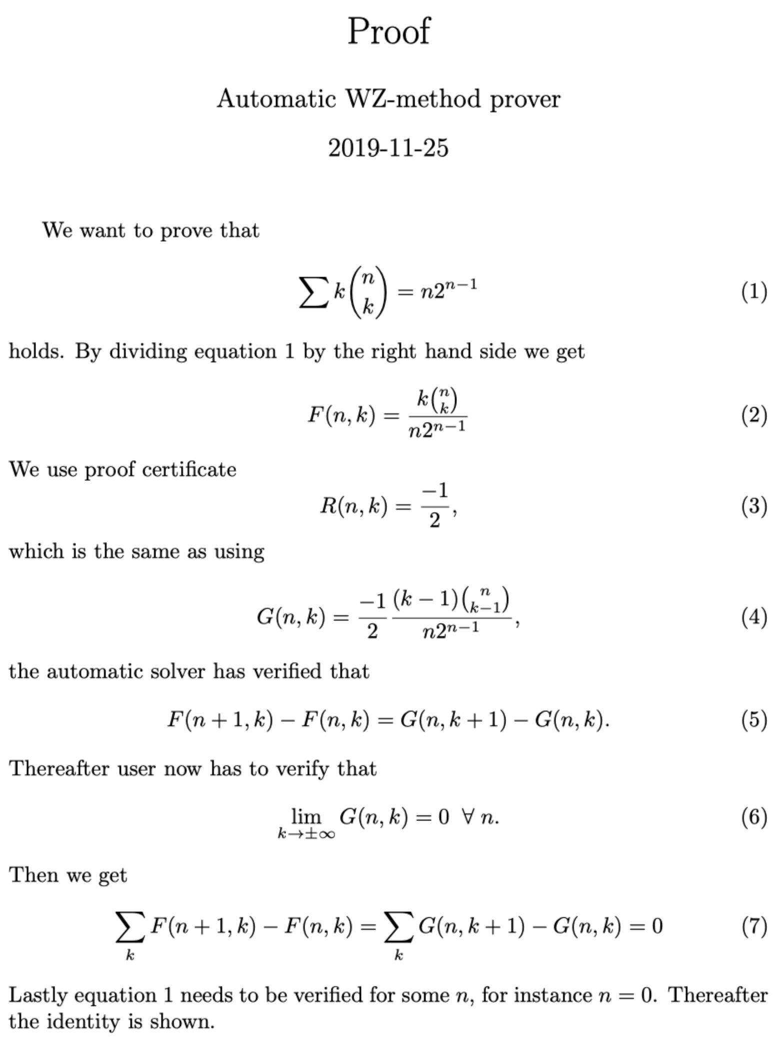
$$\lim_{k \to \pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.4 Proof of identity 4.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum k\binom{n}{k} = n2^{n-1} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{k\binom{n}{k}}{n2^{n-1}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-1}{2}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-1}{2}\frac{(k-1)\binom{n}{k-1}}{n2^{n-1}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

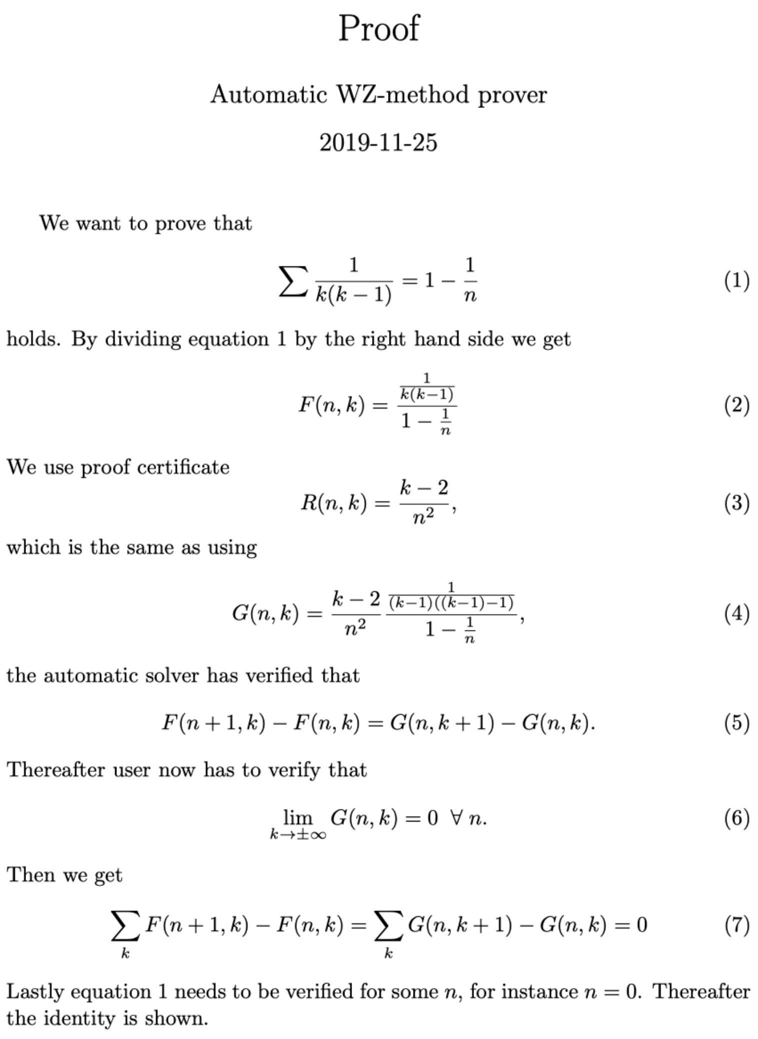
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.5 Proof of identity 5.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \frac{1}{k(k-1)} = 1 - \frac{1}{n} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\frac{1}{k(k-1)}}{1-\frac{1}{n}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{k-2}{n^2}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{k-2}{n^2}\frac{\frac{1}{(k-1)((k-1)-1)}}{1-\frac{1}{n}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

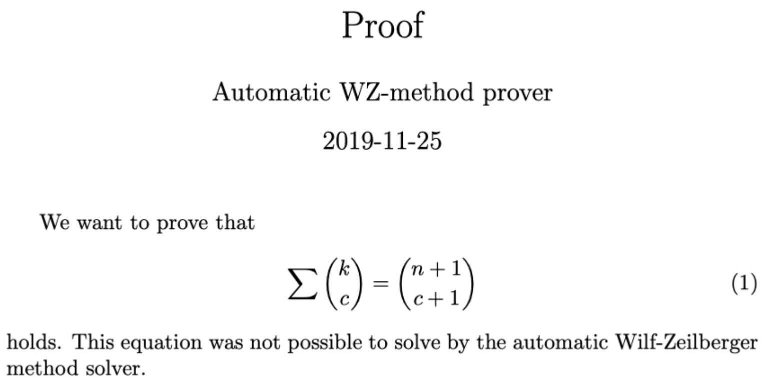
$$\lim_{k\to\pm\infty} G(n,k) = 0 \;\; \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.6 Proof of identity 6.



Proof

Automatic WZ-method prover

2019-11-25

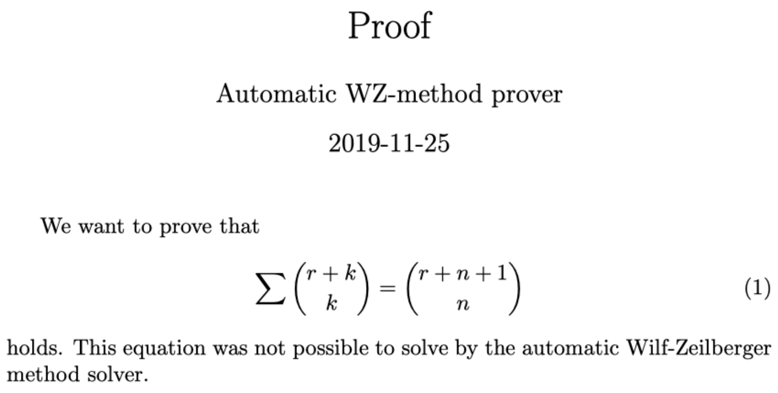
We want to prove that

$$\sum \binom{k}{c} = \binom{n+1}{c+1} \tag{1}$$

holds. This equation was not possible to solve by the automatic Wilf-Zeilberger method solver.

Figure 7.7 Proof of identity 7.

In both proof 7 and 8 we have integer variables, *c* and *r* respectively. These cause problems in step 2 in Gosper's algorithm, since the polynomial *f* would have a degree dependent on *c* and *r*, respectively. This cannot be handled by the automatic solver.



Proof

Automatic WZ-method prover

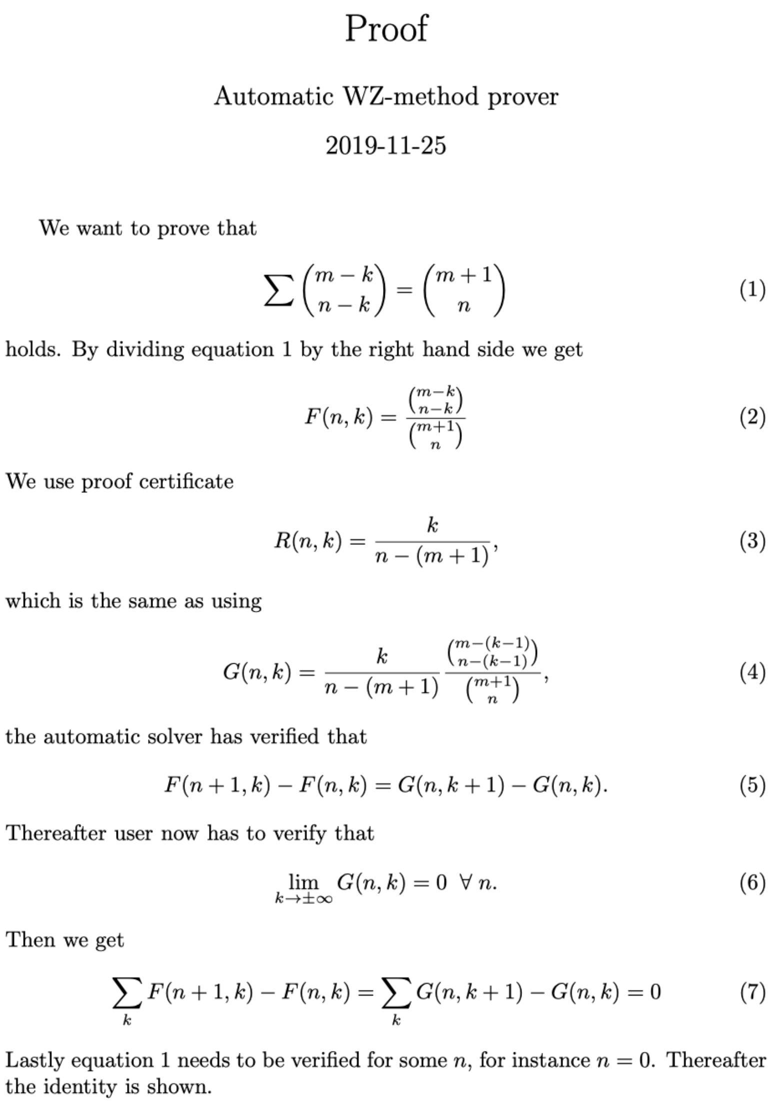
2019-11-25

We want to prove that

$$\sum \binom{r+k}{k} = \binom{r+n+1}{n} \tag{1}$$

holds. This equation was not possible to solve by the automatic Wilf-Zeilberger method solver.

Figure 7.8 Proof of identity 8.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \binom{m-k}{n-k} = \binom{m+1}{n} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\binom{m-k}{n-k}}{\binom{m+1}{n}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{k}{n-(m+1)}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{k}{n-(m+1)} \frac{\binom{m-(k-1)}{n-(k-1)}}{\binom{m+1}{n}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

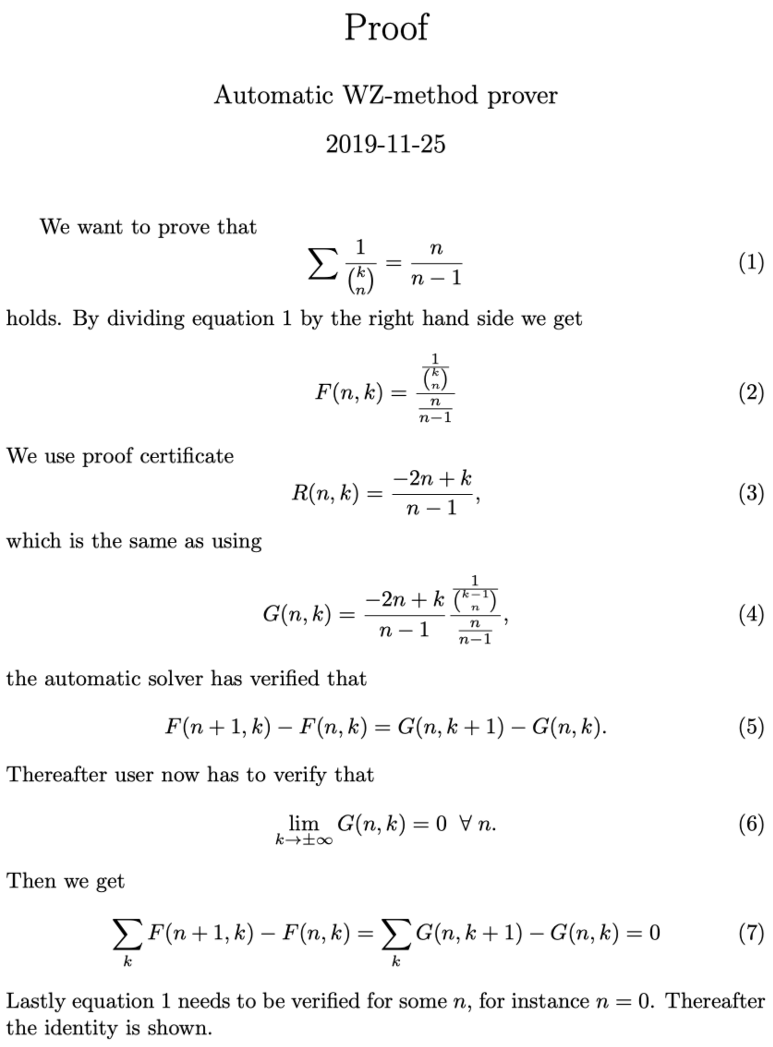
$$\lim_{k \to \pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.9 Proof of identity 9.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \frac{1}{\binom{k}{n}} = \frac{n}{n-1} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\frac{1}{\binom{k}{n}}}{\frac{n}{n-1}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-2n+k}{n-1}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-2n+k}{n-1}\frac{\frac{1}{\binom{k-1}{n}}}{\frac{n}{n-1}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

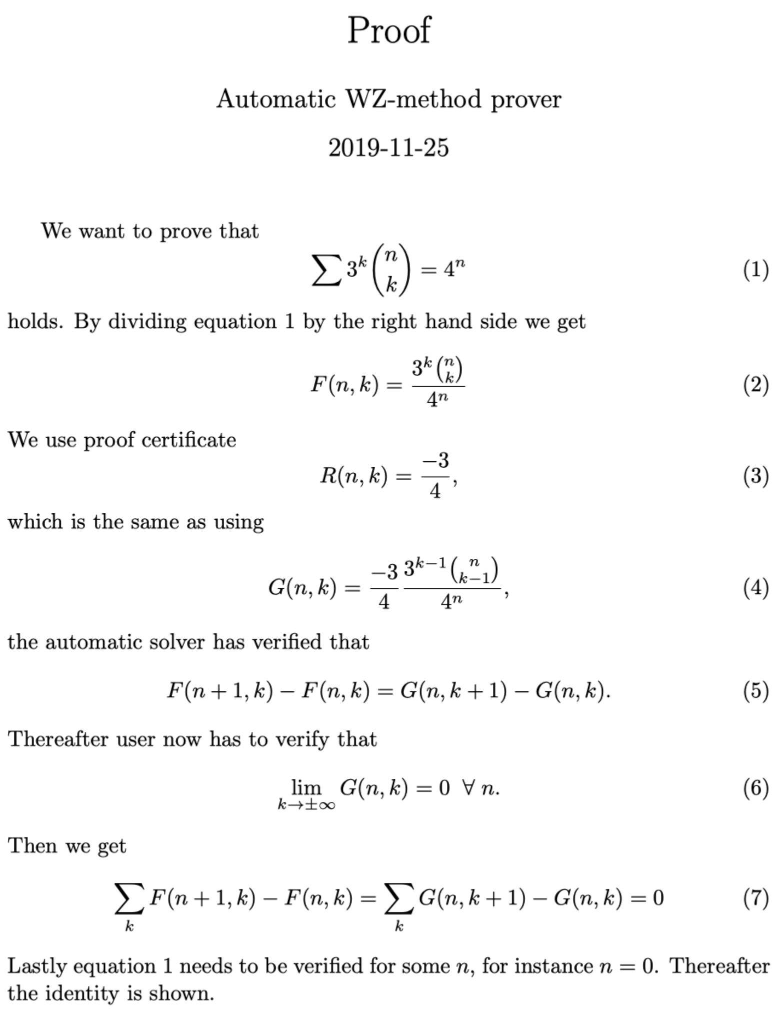
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.10 Proof of identity 10.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum 3^k \binom{n}{k} = 4^n \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{3^k \binom{n}{k}}{4^n} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-3}{4}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-3}{4} \frac{3^{k-1}\binom{n}{k-1}}{4^n}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

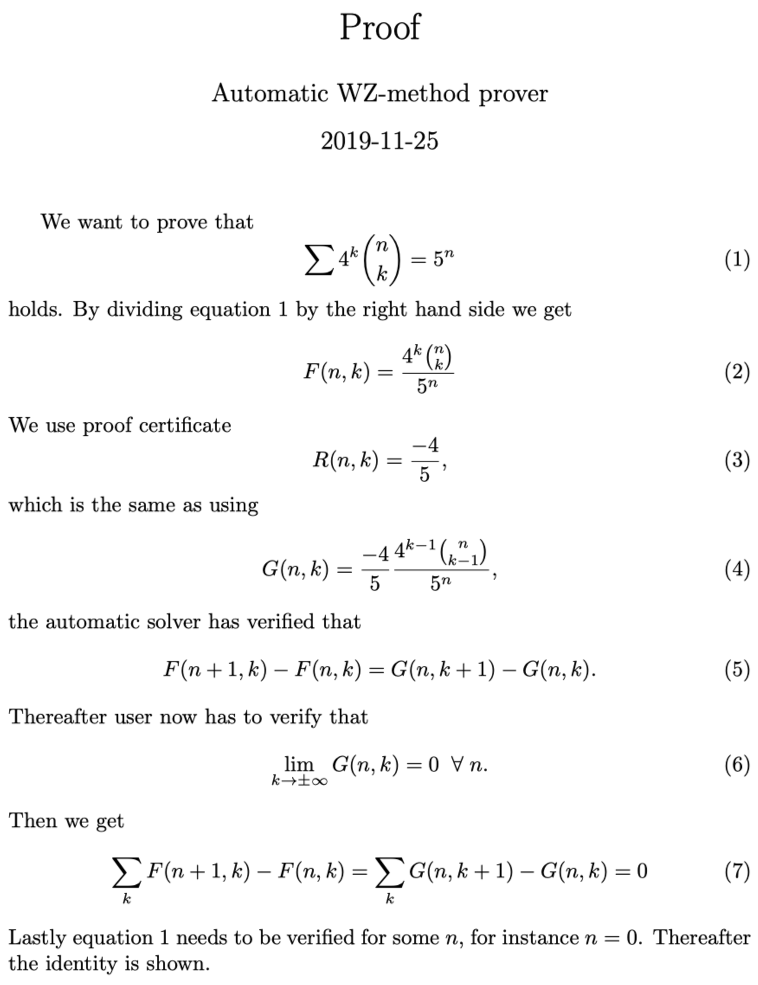
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.11 Validation of program. Proof of identity 11.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum 4^k \binom{n}{k} = 5^n \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{4^k \binom{n}{k}}{5^n} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-4}{5}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-4}{5} \frac{4^{k-1} \binom{n}{k-1}}{5^n}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

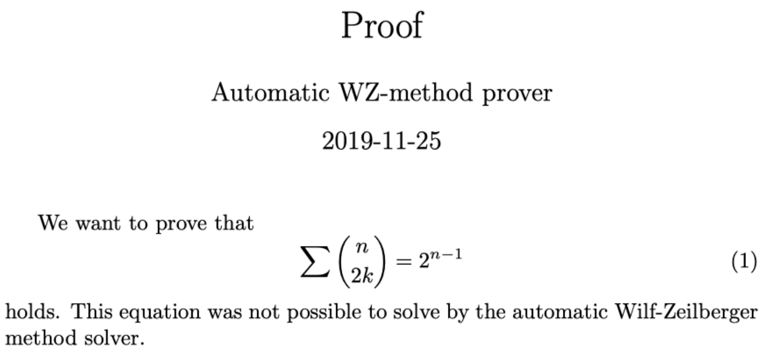
$$\lim_{k \to \pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.12 Validation of program. Proof of identity 12.


Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \binom{n}{2k} = 2^{n-1} \tag{1}$$

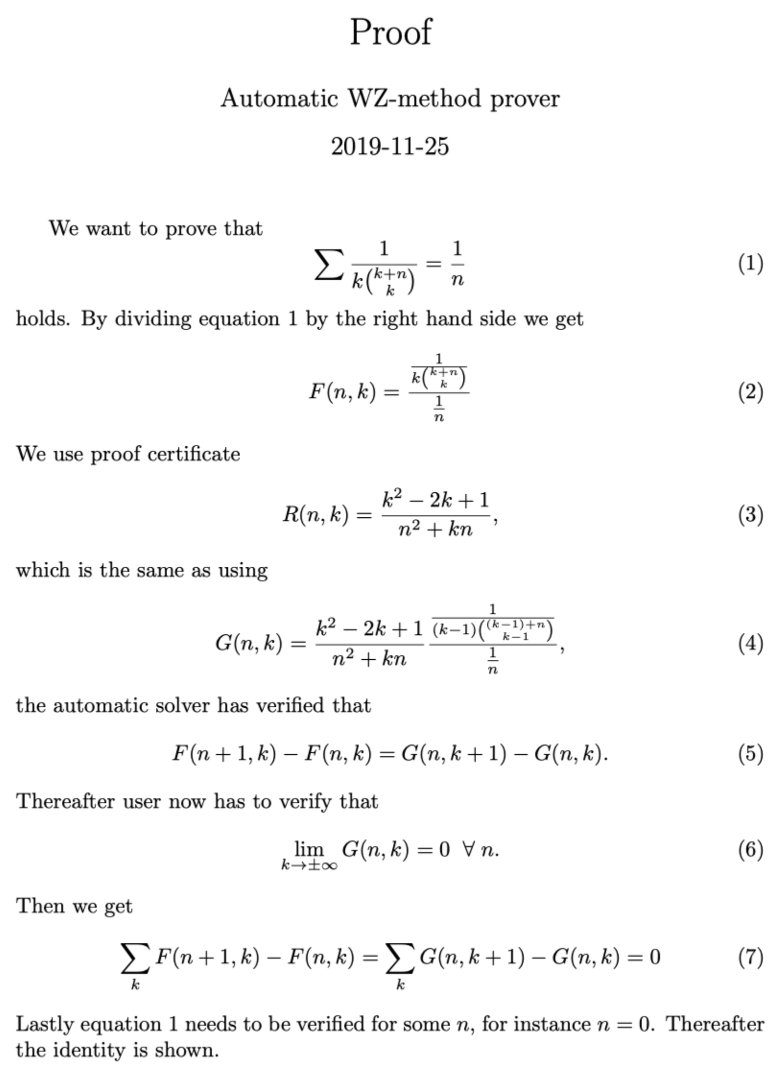
holds. This equation was not possible to solve by the automatic Wilf-Zeilberger method solver.

Figure 7.13 Validation of program. Proof of identity 13.

The automatic prover cannot find a polynomial f in step 2 of Gosper's algorithm. This is since the degree (with respect to n) of the left and right hand side of the equation

$$p_k = q_{k+1} f_k - r_k f_{k-1} \tag{7.1}$$

cannot be the same.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \frac{1}{k\binom{k+n}{k}} = \frac{1}{n} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\frac{1}{k\binom{k+n}{k}}}{\frac{1}{n}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{k^2 - 2k + 1}{n^2 + kn}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{k^2 - 2k + 1}{n^2 + kn} \frac{\frac{1}{(k-1)\binom{(k-1)+n}{k-1}}}{\frac{1}{n}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

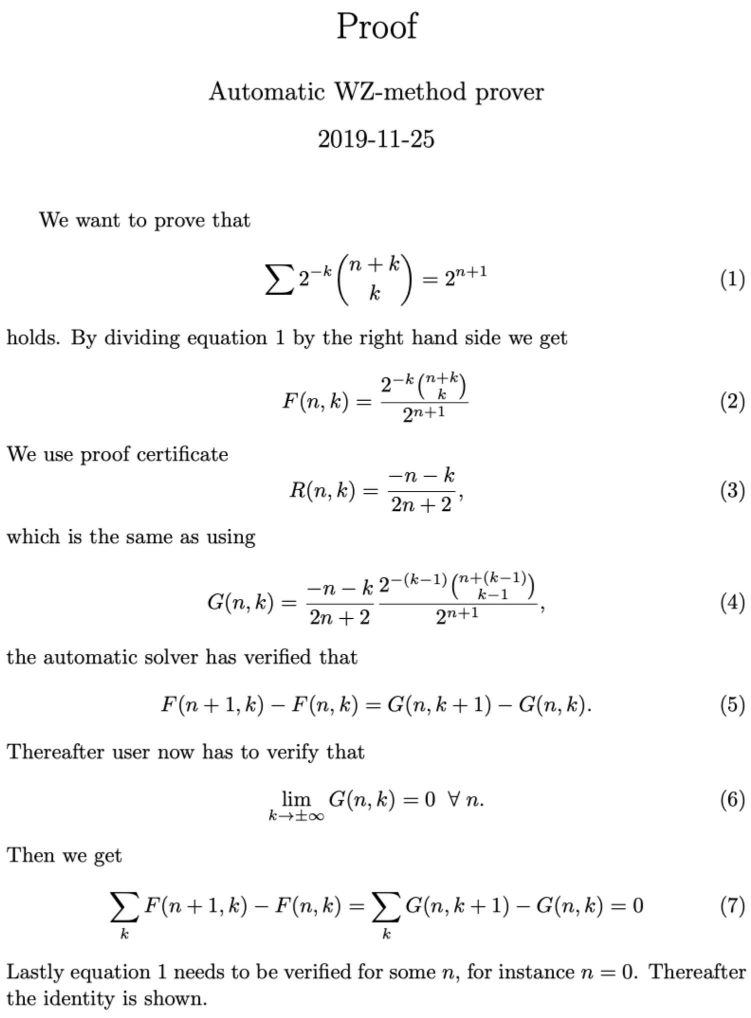
$$\lim_{k \to \pm\infty} G(n,k) = 0 \;\; \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.14 Validation of program. Proof of identity 14.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum 2^{-k}\binom{n+k}{k} = 2^{n+1} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{2^{-k}\binom{n+k}{k}}{2^{n+1}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-n-k}{2n+2}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-n-k}{2n+2}\frac{2^{-(k-1)}\binom{n+(k-1)}{k-1}}{2^{n+1}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

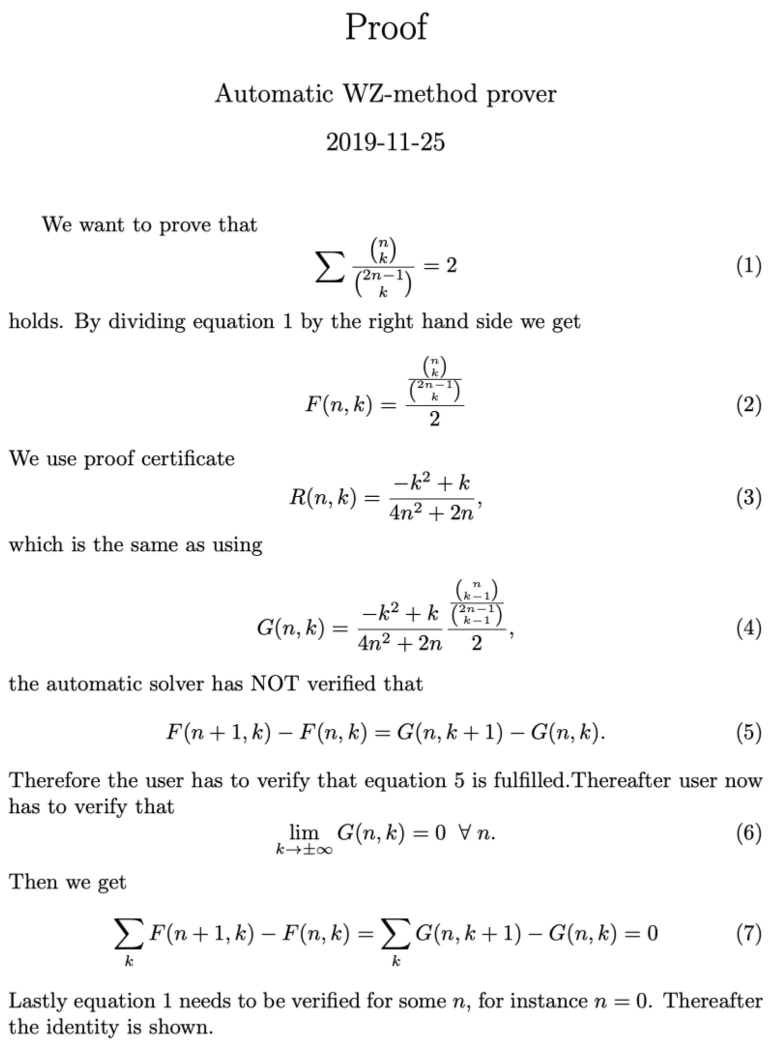
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.15 Validation of program. Proof of identity 15.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \frac{\binom{n}{k}}{\binom{2n-1}{k}} = 2 \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\frac{\binom{n}{k}}{\binom{2n-1}{k}}}{2} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-k^2+k}{4n^2+2n}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-k^2+k}{4n^2+2n}\frac{\frac{\binom{n}{k-1}}{\binom{2n-1}{k-1}}}{2}, \tag{4}$$

the automatic solver has NOT verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Therefore the user has to verify that equation 5 is fulfilled.Thereafter user now has to verify that

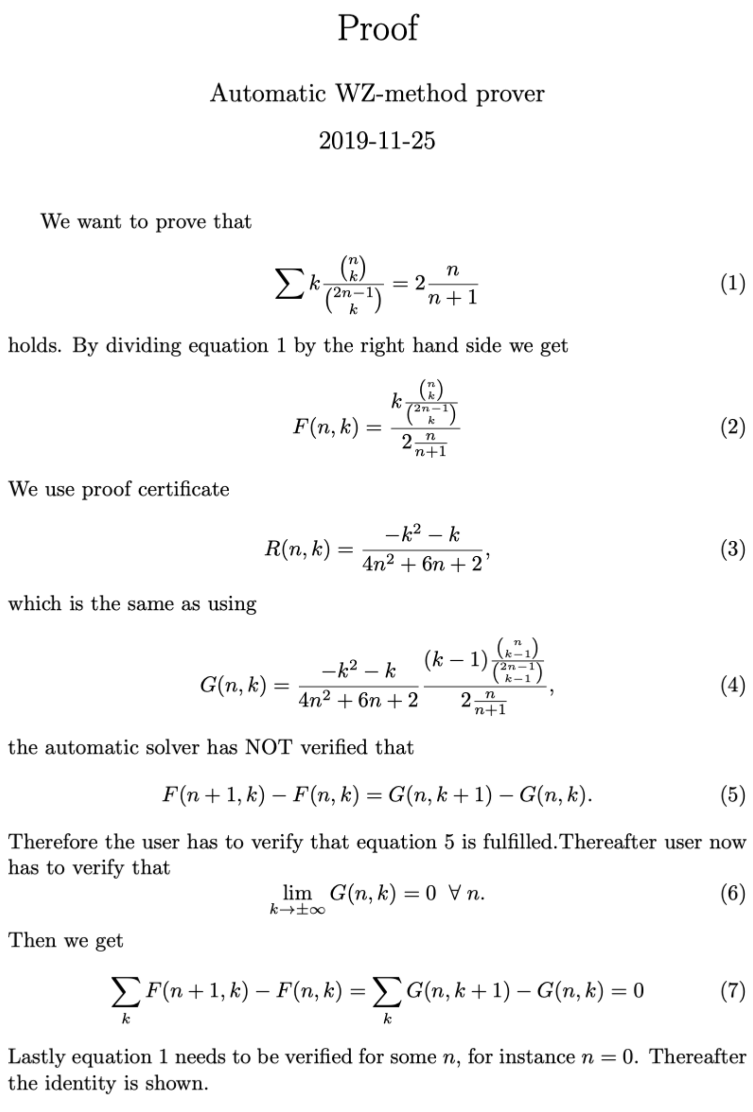
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.16 Validation of program. Proof of identity 16.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum k \frac{\binom{n}{k}}{\binom{2n-1}{k}} = 2\frac{n}{n+1} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{k\frac{\binom{n}{k}}{\binom{2n-1}{k}}}{2\frac{n}{n+1}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-k^2-k}{4n^2+6n+2}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-k^2-k}{4n^2+6n+2}\frac{(k-1)\frac{\binom{n}{k-1}}{\binom{2n-1}{k-1}}}{2\frac{n}{n+1}}, \tag{4}$$

the automatic solver has NOT verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Therefore the user has to verify that equation 5 is fulfilled.Thereafter user now has to verify that

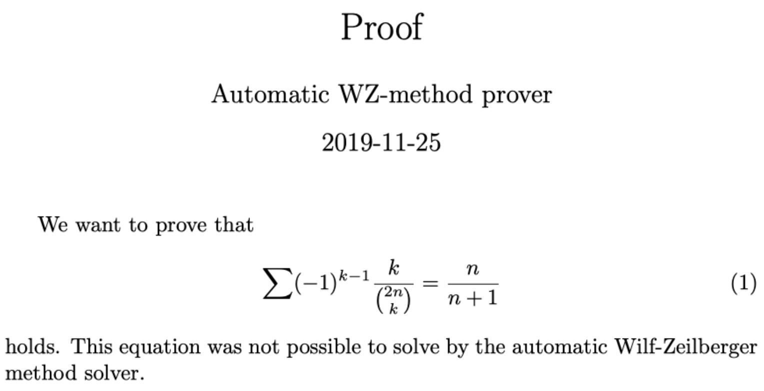
$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.17 Validation of program. Proof of identity 17.



Proof

Automatic WZ-method prover

2019-11-25

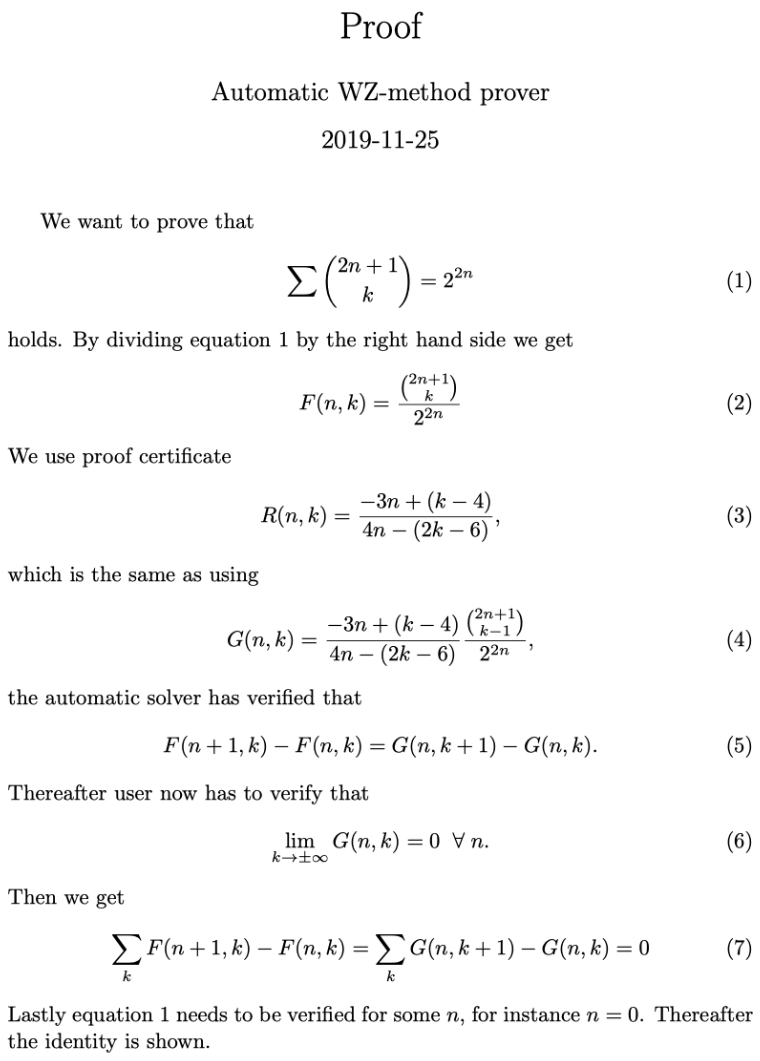
We want to prove that

$$\sum(-1)^{k-1}\frac{k}{\binom{2n}{k}}=\frac{n}{n+1} \tag{1}$$

holds. This equation was not possible to solve by the automatic Wilf-Zeilberger method solver.

Figure 7.18 Validation of program. Proof of identity 18.

Also in this example the automatic solver cannot prove the identity, and the problem lies in step 2 of Gosper's algorithm.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum \binom{2n+1}{k} = 2^{2n} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{\binom{2n+1}{k}}{2^{2n}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{-3n+(k-4)}{4n-(2k-6)}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{-3n+(k-4)}{4n-(2k-6)} \frac{\binom{2n+1}{k-1}}{2^{2n}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

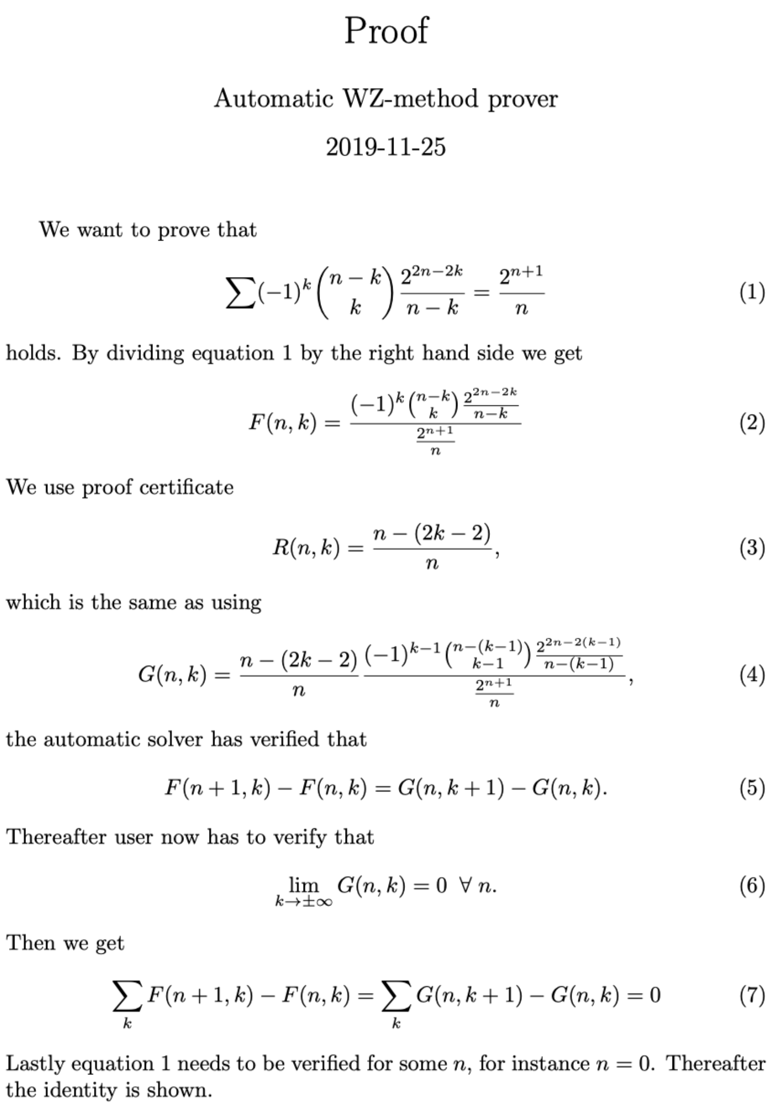
$$\lim_{k \to \pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.19 Validation of program. Proof of identity 19.



Proof

Automatic WZ-method prover

2019-11-25

We want to prove that

$$\sum(-1)^k \binom{n-k}{k} \frac{2^{2n-2k}}{n-k} = \frac{2^{n+1}}{n} \tag{1}$$

holds. By dividing equation 1 by the right hand side we get

$$F(n,k) = \frac{(-1)^k \binom{n-k}{k} \frac{2^{2n-2k}}{n-k}}{\frac{2^{n+1}}{n}} \tag{2}$$

We use proof certificate

$$R(n,k) = \frac{n-(2k-2)}{n}, \tag{3}$$

which is the same as using

$$G(n,k) = \frac{n-(2k-2)}{n} \frac{(-1)^{k-1} \binom{n-(k-1)}{k-1} \frac{2^{2n-2(k-1)}}{n-(k-1)}}{\frac{2^{n+1}}{n}}, \tag{4}$$

the automatic solver has verified that

$$F(n+1,k) - F(n,k) = G(n,k+1) - G(n,k). \tag{5}$$

Thereafter user now has to verify that

$$\lim_{k\to\pm\infty} G(n,k) = 0 \ \ \forall\, n. \tag{6}$$

Then we get

$$\sum_k F(n+1,k) - F(n,k) = \sum_k G(n,k+1) - G(n,k) = 0 \tag{7}$$

Lastly equation 1 needs to be verified for some n, for instance $n = 0$. Thereafter the identity is shown.

Figure 7.20 Validation of program. Proof of identity 20.

7.3 Popular Scientific Paper

Computer Algebra – use a computer to solve difficult maths problems!

Lars Åström[1],
Supervisor: Victor Ufnarovski[1]

[1]Department of Mathematics, Lund University

In mathematics it is not only important to solve a problem, but the proof of correctness is just as important. Proving equalities turns out to be hard, which is why an automatic prover is highly useful! In the thesis a program was developed that can prove certain types of combinatorial identities that involve summations.

Computer algebra is a field in mathematics where a computer is used to manipulate mathematical expressions, for instance add, multiply and divide polynomials. There are many existing softwares that do this, such as Maple and Mathematica, but even the simplest operations turn out to be quite tricky to implement.

The thesis produced a software library which can prove a specific type of equations. The software works by getting an equation as input and then produces a formal proof for that the equation actually holds. In the proof process an method called Wilf-Zeilberger's method (Wilf, 1994) is used to produce the intermediate steps.

Combinatorics is a part of mathematics concerned with counting, for instance "In how many ways can one pick a three cookies out of five different types of cookies?". The answer to this question is usually denoted by using a binomial coefficient – $\binom{5}{3}$. Here the bottom and top numbers are the number we choose and the number we choose from, respectively. In combinatorics one often encounters the problem of summation, and proving that two expressions are equal. This can be solving the question "In how many ways can one pick cookies (any number) out of five different types of cookies?". This can be done by adding the number of solutions if we pick exactly $0, 1, 2, 3, 4$ and 5, respectively. We can also solve the problem by noting that we have two choices for each cookie, to take it or not. Since we have five cookies the total number of choices becomes $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^5$. Therefore we should have that

$$\sum_{i=0}^{5} \binom{5}{i} = 2^5.$$

By computing the left and right side of the equation one can verify that equality occurs, but can this result be generalized? What if we have n cookies instead of 5?

The program produced in this thesis addresses this problem and can in an automatic way produce a proof of an equality. The program outputs a proof which the user can read and understand, where the user only need to verify a few statements – which usually is an easy task. This provides a great help for a user in the process of proving an equality.

Producing an automatic equation prover required a few steps – such as reading the equation that should be proved, running the method and printing the proof in the right format. The most important point I learned was the importance of continuous testing. Every single part of the program was tested both individually and as a part of the bigger picture in the program. This resulted in that as much as 30% of the almost 2300 lines of code that were written were purely for testing.

Producing mathematical proofs automatically is quite cool, and indeed the main contribution of the thesis. Still one major thing that came out of the thesis work was the understanding of Computer Algebra, especially how something can seem easy but is terribly hard to implement. The number of times I underestimated the time to implement something were truly uncountable.

References

1. Gosper, R.W. Jr. (1978). *Decision procedure for indefinite hypergeometric summation.* Proc. Natl. Acad. Sci. USA. Vol. 75, No. 1, pp. 40–42, January 1978. Palo Alto, California.
2. Gould, H.W. (1972). *Combinatorial Identities.* Morgamtown Printing and Binding Co. West Virginia.
3. Güyer, T. (2008). 'Computer Algebra Systems as the Mathematics Teaching Tool.' *World Applied Sciences Journal* 3 (1): 132-139, 2008.
4. Johnson, M. (2007). *Handout 08 Bottom-up Parsing.* Lecture notes Stanford University CS143. Delivered 2 July 2007.
5. Lopez, R. *Computer Algebra Systems.* Maplesoft, viewed 26 November 2019. `https://www.maplesoft.com/ns/maple/cas/computer-algebra-systems-math-education.aspx`.
6. Paule, P. and Schorn, M. (1994). 'A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities.' *J. Symbolic Computation* (1995) 20, pp. 673–698.
7. Tesler (2017). *Chapter 3.3, 4.1, 4.3. Binomial Coefficient Identities.* Lecture notes University of California San Diego Math 184A. Delivered winter 2019.
8. Veltman, M.J.G. and Williams, D.N. (1991). 'Schoonschip '91.'
9. Wilf, H.S. (1994). *Generating functionology.* Academic Press, Inc. Philadelphia, Pennsylvania.
10. Wilf, H.S., and Zeilberger, D. (1990). 'Rational functions certify combinatorial identities.' *J. Amer. Math. Soc. 3* (1990), pp. 147–158.