A Macsyma[1] Implementation of Zeilberger's Fast Algorithm[2]

Fabrizio Caruso

August 20, 1999

[1]Comp. algebra system dev. by Macsyma Inc., ver. 419
[2]Supported by the SFB grant F1305 of the Austrian FWF

Abstract

We present the first implementation on the Macsyma computer algebra system of Zeilberger's fast algorithm for the definite summation problem for a very large class of sequences; i.e. given a hypergeometric sequence $F(n,k)$, we want to represent $f(n) = \sum_{k=0}^{n} F(n,k)$ in a "simpler" form. We do this by finding a linear recurrence for the summand $F(n,k)$, from which, by some conditions on the support of $F(n,k)$, we can obtain a homogeneous $k-$free recurrence for $f(n)$. The solution of this recurrence is left as a post-processing, and it will give the "simpler" form we were looking for.

Zeilberger's fast algorithm exploits a specialized version of Gosper's Algorithm for the indefinite summation problem; i.e. given a hypergeometric sequence $t(k)$, the problem of finding another sequence $T(k)$ such that $t(k) = \Delta_k T(k) = T(k+1) - T(k)$. The implementation of this algorithm has also been carried out in the Macsyma computer algebra system, and its details are also treated in this paper.

Contents

Chapter 1

Introduction

We present the first implementation within the Macsyma computer algebra system of Zeilberger's Fast Algorithm for the definite summation problem for the large class of proper hypergeometric sequences, i.e. given a 2-variable sequence $F(n,k)$, we want to rewrite the definite sum $f(n) = \sum_{k=0}^{n} F(n,k)$ in a form free of $\sum$. We do this by finding a special $k-$free linear recurrence with polynomial coefficients for the summand $F(n,k)$, which, under the condition of natural boundary for $F(n,k)$, can be extended into a $k-$free homogeneous linear recurrence with polynomial coefficients. The desired linear recurrence for the summand has the form: $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$. The search for this recurrence is done by guessing the order m of the left hand side of this recurrence and then trying to find the corresponding $G(n,k)$ of the right hand side by means of a specialized Gosper algorithm (We note that a priori upper bounds for m can be given; however they turn out to be too large and therefore not useful from practical use). A new version of Gosper's algorithm for this purpose has also been implemented and included in this package.

Gosper's algorithm for the indefinite summation problem solves the hypergeometric telescoping problem, i.e. given a hypergeometric sequence $t(k)$, it decides if there is a hypergeometric sequence $T(k)$ such that $t(k) = \Delta_k T(k) = T(k+1) - T(k)$, and if it exists, it finds it.

The standard Gosper's algorithm cannot be used in Zeilberger's algorithm because it does not take into account that there are some polynomial parameters. Therefore a specialized parametrized version of this algorithm, in which the polynomial unknown parameters a_i's are taken into account, has been implemented and it is used by our implementation of Zeilberger's algorithm to compute the right hand side of the desired recurrence.

These implementations do not consider the more general $q-$case, for which a Mathematica version has already been developed at RISC (see [7]). Moreover we remark that at RISC a Mathematica version of Zeilberger's fast algorithm for the ordinary case $q = 1$, considered in this report, has already been implemented by Paule and Schorn [6]. Both packages can be downloaded from the home page http://www.risc.uni-linz.ac.at/research/combinat/risc/.

Chapter 2

Some Theory

We now introduce the basic definitions of the theory of definite and indefinite hypergeometric summation and some fundamental results of the theory which are necessary for the correctness of Zeilberger's fast algorithm and Gosper's algorithm.

For a formal description of Gosper's algorithm and Zeilberger's fast algorithm and of the proofs of their correctness we refer respectively to the original paper [3] for Gosper's algorithm and to the papers [9], [10]. For a tutorial description of these algorithms we refer to [4] and to [8].

2.1 Some Fundamental Definitions

Definition Given a hypergeometric sequence $t(n)$ over a field K of characteristic 0, we call $t(n+1)/t(n)$ *shift quotient* of $t(n)$.

Definition A sequence $t(n)$ over a field K of characteristic 0 is said to be *hypergeometric* if and only if the shift quotient $t(n+1)/t(n)$ is a rational function in n with coefficients in K.

We can extend this definition to multivariate sequences in the natural way:

Definition A sequence $t(x_1, \ldots x_m)$ over a field K of characteristic 0 is said to be *hypergeometric* with respect to x_s, where $1 \leq s \leq m$, if and only if the quotient $t(x_1, \ldots, x_s + 1, \ldots x_m)/t(x_1, \ldots, x_m)$ is a rational function in n with coefficients in K.

Definition Given a 2-variable sequence $f(k,n)$ over a field K of characteristic 0 is said *proper hypergeometric* if and only if it is in the following form:

$$f(k,n) = p(k,n)\frac{\prod_{i=0}^{I}(a_i k + b_i n + c)!}{\prod_{j=0}^{J}(d_j k + e_j n + f)!}x^k$$

where $p(k,n)$ is a polynomial in k and n, and c, f, x are complex numbers, I, J and the a_i's, b_i's, d_j's, e_j's are fixed integers,

This definition extends in the natural way to $m-$variable case, where $m \geq 2$.

2.2 Some Fundamental Facts

Proposition

Given a hypergeometric sequence $t(n)$ over a field K. If there is a hypergeometric sequence $T(n)$ such that $t(n) = T(n+1) - T(n)$, then we have $t(n+1)/t(n) = r(n)T(n)$ where $r(n)$ is a rational function.

Proposition

Any proper hypergeometric sequence $F(n,k)$ is hypergeometric w.r.t. n and k.

Theorem *(Existence of a k-free recurrence)*

Let $F(n,k)$ be a proper hypergeometric sequence, then F satisfies a nontrivial recurrence of the form:

$$\sum_{i=0}^{m} a_i(n)F(n,k) = G(n,k+1) - G(n,k),$$

where $G(n,k)/F(n,k)$ is a rational function in n and k.

Chapter 3

The Algorithms

3.1 Gosper's Algorithm

Gosper's algorithm solves the hypergeometric indefinite summation problem: given a hypergeometric sequence $t(n)$, it decides whether there exists another hypergeometric sequence $T(n)$ such that $t(n) = T(n+1) - T(n)$.

INPUT : A hypergeometric sequence $t(n)$ over a field K of characteristic 0.

OUTPUT : If there is a hypergeometric sequence $T(n)$ s.t. $t(n) = T(n+1) - T(n)$, then return $T(n)$, else return "NO HYPERGEOMETRIC SOLUTION"

Pseudocode

- **STEP 0** $rat_t(n) := t(n+1)/t(n)$

- **STEP 1** Write $rat_t(n)$ in the following form:

  $$rat_t(n) = \frac{p(n+1)}{p(n)} \frac{q(n)}{r(n+1)}$$

  where $gcd(q(n), r(n+j)) = 1$ for all positive integers j's, and p, q, and r are polynomials.

- **STEP 2** Solve the following linear polynomial recurrence equation:

  $$p(n) = q(n)s(n+1) - r(n)s(n)$$

  where the polynomial $s(n)$ is the unknown.

  If no such polynomial exists,
  then

RETURN "NO HYPERGEOMETRIC SOLUTION",

else
RETURN

$$T(n) = \frac{1}{(\frac{s(n+1)}{s(n)}\frac{q(n)}{r(n)} - 1)} \cdot t(n).$$

Remark *(Step 1)*
The special form for the rational function rat_t described in step 1 can be computed by forcing the gcd condition by resultant computation and simple rewriting. In the proper hypergeometric case, this step can also be done by simple inspection.

Remark *(Step 2)*
The linear recurrence equation of step2 can be computed by plugging into it a polynomial with unknown coefficients, whose degree can be properly bounded from the degrees of the polynomials p, $q^r_+ := q-r$, $q^r_- := q+r$ and by the leading coefficients of the q^r_+, q^r_-.

3.1.1 Why it works

For a formal proof of the correctness of this algorithm we refer to [4], [8], [3].
The idea of this algorithm is that we can rewrite an indefinite hypergeometric summation problem into a linear recurrence. In fact, from the propositions in section 2.2, we have that the solution $T(n)$ and the input must differ by a factor given by rational function $r(t)$:

$$T(n) = r(n)t(n).$$

By plugging this into the desired property: $t(n) = T(n+1) - T(n)$ and by dividing both sides by $t(n)$, we obtain the following recurrence equation over rational functions:

$$1 = r(n+1)rat_t(n) - r(n)$$

where $rat_t(n) := t(n+1)/t(n)$. Gosper could not solve this kind of recurrence equations, which today can be solved by Abramov's method [1]. For an alternative approach and a detailed explanation of the connection between these strategies see, for instance, [5]. Gosper had to find a way to get round this problem: he rewrites rat_t in a special form that leads to simpler linear recurrence equation in which the unknown is a polynomial, and that can be solved by simple linear algebra.

3.2 Zeilberger's Algorithm

Zeilberger's algorithm solves the proper hypergeometric problem: given a proper hypergeometric sequence $F(n,k)$ with natural boundary, (i.e. for any fixed n, $F(n,k)$ has finite support with respect to k), it finds a $k-free$ recurrence for $F(n,k)$ of the form: $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1)-G(n,k)$, which under the condition of natural boundary for $F(n,k)$ yields a homogeneous linear recurrence for the sum $f(n) = \sum_{k=0}^{n} F(n,k)$.

INPUT : A proper hypergeometric sequence $F(n,k)$ w.r.t k over a field K and an a priori bound on the order m of the recurrence that will be searched.
OUTPUT : A $k-$free linear recurrence with polynomial coefficients for $F(n,k)$ of the form:

$$\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$$

Pseudocode

- **STEP 0** Try $m = 0$, i.e. Gosper's algorithm, on $F(n,k)$;
  if it finds a solution,
  then
  RETURN this solution, else continue.

- **STEP 1** Try a parametrized version of Gosper's algorithm on the sequence

$$\sum_{i=0}^{m} a_i(n)F(n,k)$$

  in which the a_i's are polynomial unknowns.

- **STEP 2** $m := 1$

- **STEP 3** If some polynomials a_i's and a hypergeometric sequence $G(n,k)$ can be found for which the above-mentioned special recurrence is satisfied,
  then
  RETURN $[G, [a_0, \dots, a_m]]$,
  else
  increase m and **GO TO STEP 1**.

Remark *(Step 1)*
The parametrized version of Gosper's algorithm is formally the same as the standard one; the only difference is that in this version the a_i's appear

as polynomial parameters and the solving of the recurrence equation $p(n) = q(n)s(n+1) - r(n)s(n)$ is now done by plugging in unknown polynomials whose coefficients are no more in the ground field K, but in the ring of rational functions with coefficients in K.
Therefore the problem of definite summation can be rewritten as the problem of solving systems of linear equations with polynomial coefficients.

Remark *(Step 3)*
Under the hypothesis of natural boundary (finite support) of $F(n,k)$, for instance let us assume that the support is strictly contained in the interval $[0,p]$, with $p > 0$, the special recurrence for the summand $F(n,k)$, namely $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$ leads to a (homogeneous) recurrence for the sum $f(n)$, which is what we are ultimately looking for.
In fact, summing the left hand side and right hand side of the special recurrence over a boundary larger than the support of $F(n,k)$ we get:

$$\sum_{k=0}^{p}\sum_{i=0}^{m} a_i(n)F(n+i,k) = \sum_{k=0}^{p}(G(n,k+1) - G(n,k)) = G(n,p+1) - G(n,0) = 0$$

Hence:

$$\sum_{i=0}^{n} f(n+i) = G(n,p+1) - G(n,0) = 0$$

When the summand does not have finite support the above equation will be non-homogeneous.

3.2.1 Why it works

For a proof of the correctness we refer to the literature on this topic ([8], [9], [10]).
Here we present the basic idea behind it:
From the theorem that states the existence of a $k-$free recurrence (see previous chapter) we know that for any proper hypergeometric sequence $F(n,k)$, there exists a recurrence of the form: $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$.
Zeilberger's algorithm tries to find this recurrence by iteratively increasing the order of the recurrence and by using a parametric version of Gosper's algorithm to find its right hand side; this procedure stops because of the termination of Gosper's algorithm and because of the theorem mentioned above.

Chapter 4

The Implementation

In this chapter we report on some of the details of our implementation of "Zeilberger's fast algorithm" and of "Gosper's algorithm".

The implementation of these two algorithms has been done in the internal LISP-like language of the Macsyma computer algebra system.

I implemented both algorithms in Macsyma in a straightforward way and no significant changes have been made in the original algorithms.

4.1 Gosper's Algorithm

- This version of Gosper'a algorithm works in the special case of proper hypergeometric sequences, but it can be easily extended to the general hypergeometric case by substituting the Gosper-form related routine with a more general routine in which the Gosper-form is achieved by a resultant computation. (See [4] for more details).

- As a consequence of this implementation choice the desired special form at step 1 for the rational function $rat_t(n)$ is constructed by initializing $p(n) := 1$, $q(n) := numerator(rat_t)$, $r(n+1) := denominator(rat_t)$ and by simple inspection and rewriting of the factors of the polynomials q and r.

4.2 Zeilberger's Fast Algorithm

- This version of Zeilberger's algorithm works in the proper hypergeometric case. We remark that there are implementations that could work at least in principle in a slightly larger class of sequences (holonomic hypergeometric sequences, see [2]). Our choice makes the implementation simpler and allows us to use our implementation of Gosper's algorithm for the proper hypergeometric case.

- The computation of the recurrence for the sum and its solution is left to the user as a post-processing. Moreover the existence of a solution expressible in elementary terms is not always guaranteed, therefore leaving the solution as a recurrence is a reasonable choice.

Chapter 5

Performance

First of all we must point out that this implementation is only the first attempt to incorporate Zeilberger's fast algorithm into the Macsyma computer algebra system and therefore it is very far from being competitive with the best existing optimized implementations (for example see [6]).

No large scale test has been carried out to assess the speed of this implementation. The only test that has been run on this implementation is the collection of sequences contained in the files `testGosper.macsyma` and `testZeilberger.macsyma`, among which there are some powers of the binomial coefficient. Increasing powers of the binomial coefficients have been used to test the stability of the system and to obtain a rough idea of the performance of the system and to compare it to some other implementations. I used this test suite because of its simplicity and because we know a priori the order of the recurrence for the summand ($\lceil \frac{exponent}{2} \rceil$).

5.1 Timings

We present some timings obtained by testing the function `parGosper` on increasing powers of the binomial coefficients in which no loop on the order of the recurrence is run (we know it in advance) and the environment variable *OUTPUT_FORMAT* is set to *NON_SIMPLIFIED*. In these tests the Paule-Schorn Mathematica implementation [6] and our Macsyma implementation are compared. These implementations have been run on an SGI Octane with 2 gigabytes of ram and two 250 Mhz RISC processors (although much less memory was necessary for running the program on these examples).

Results *Note:Timings are in seconds*

power	**order**	*Paule/Schorn Mathematica*	*Macsyma*	*difference ratio*
3	2	$0.36s$	$3.31s$	9.19
4	2	$0.92s$	$3.95s$	4.29
5	3	$4.47s$	$37.11s$	8.30
6	3	$16.98s$	$121.64s$	7.16
Average ratio :				7.23

5.2 Stability

Our Macsyma implementation tested on an SGI Octane equipped with 2 gigabytes of ram could no go beyond the sixth power of the binomial coefficient whereas Paule-Schorn Mathematica implementation was able to handle the eleventh power. The sixth power of the binomial coefficient required less than 50 megabytes of memory.

5.2.1 What can be done to improve this implementation

An analysis on the distribution of the computation time shows that in the "heavy cases" (binomial coefficient to the fifth and sixth powers) the bottle neck is the computation of the solution of the linear systems of equations that is required to solve the recurrence equation (*Gosper's equation*).

Therefore future optimized versions should use an ad hoc linear solver that could take advantage of the specific structure of the system.

Chapter 6

Manual

6.1 Loading the files

The entire package can be downloaded from the RISC Combinatorics home page at the following u.r.l. http://www.risc.uni-linz.ac.at/research/combinat/risc/
The user will find the following files:

1. algUtil.macsyma
2. shiftQuotient.macsyma
3. poly2quint.macsyma
4. makeGosperForm.macsyma
5. GosperEq.macsyma
6. Gosper.macsyma
7. Zeilberger.macsyma
8. LOADZeilberger.macsyma
9. testZeilberger.macsyma
10. testGosper.macsyma

The entire package can be loaded into memory by simply loading the file `LOADZeilberger.macsyma`; this file will take care of loading the other components except the files containing some examples on which the system has been tested, namely (`testZeilberger.macsyma`, `testGosper.macsyma`).

6.2 The Commands

Both Gosper and Zeilberger's algorithm have been implemented in two versions: a verbose version, which allows the user to choose different levels of verbosity, and a non-verbose version, which should provide a bit of better performance.

6.2.1 Verbosity

The levels of verbosity are selected by the user just adding a suffix to the command name (`Gosper`, `parGosper`, `Zeilberger`) or by passing a parameter in the generic verbose version of the algorithm. (No suffix is interpreted as non-verbose).

These are the levels of verbosity that have been implemented in both algorithms:

Level	**Suffix**	**Scope**
summary	`Summary`	*summary of the main computations*
normal	`Verbose`	*verbosity on the main procedure*
very	`VeryVerbose`	*verbosity on the subroutines*
debugging	`Debugging`	*deepest verbosity*
linsys	`LinSys`	*verbosity on the linear system*

Examples

`GosperVerbose(f,k)` invokes Gosper's algorithm in verbose mode

`ZeilbergerVeryVerbose(f,k,n)` invokes Zeilberger's algorithm in the very verbose mode

6.2.2 Gosper's Algorithm

- Gosper**(f, k)**
  It solves the indefinite summation problem of finding an hypergeometric sequence $g(k)$ such that $f(k) = \Delta_k g(k) = g(k+1) - g(k)$. If such a hypergeometric sequence exists it returns it as output, otherwise it will return **NO_HYP_SOL**.

- GosperVerboseOpt**(f, k, verbosity)**
  As `Gosper` but the level of verbosity is passed as a parameter.

- GosperSum**(f, k, a, b)**
  It computes $\sum_{k=a}^{b} f$ by using `Gosper` to solve the indefinite sum. (It only works if the indefinite sum is Gosper-summable).

- GosperSumVerboseOpt**(f, k, a, b, verbosity)**
  As `GosperSum` but the level of verbosity is passed as a parameter.

6.2.3 Zeilberger's Fast Algorithm

- Zeilberger**(F, k, n)**
  Given a 2-variable proper hypergeometric sequence $F(n,k)$, it computes by Zeilberger's algorithm a recurrence equation for F of the form: $\sum_{i=0}^{m} a_i(n)F(n+i,k) = \delta_k(Cert(n,k)F(n,k))$, where the a_i's are polynomials free of k and $Cert$ ("rational certificate") is a rational function in n and k. The output will be a print-out of the recurrence and the explicit values of the polynomial parameters a_i and the "rational certificate" $Cert(n,k)$.

- ZeilbergerVerboseOpt**(F, k, n, verbosity)**
  As `Zeilberger` but the level of verbosity is passed as a parameter.

- parGosper**(F, k, n, ord)**
  Given a 2-variable proper hypergeometric sequence, it computes, when it exists, a recurrence equation of order ord for F of the form: $\sum_{i=0}^{ord} a_i(n)F(n+i,k) = \Delta_k(R(n,k)F(n,k))$, where a_i's are polynomials and R is a rational function (the *certificate*), and it yields a sequence $[R,[a_0,\ldots,a_{ord}]]$; if no such recurrence exists them it yields $[0,[\text{dummy value},\ldots,\text{dummy value}]]$.

- parGosperVerboseOpt**(F, k, n, ord, verbosity)**
  As `parGosper` but the level of verbosity is passed as a parameter.

6.2.4 Settings

No much settings and fine-tuning is necessary to use this package. The only settings are done through the environment variables `MAX_ORD` and the `OUTPUT_FORMAT`.

`MAX_ORD` sets an a priori bound on the order of the recurrence that Zeilberger's algorithm iterativelly tries to find by applying `parGosper` with increasing order. The default value of `MAX_ORD` is 3.

`OUTPUT_FORMAT` is a flag that controls whether the final output of the routines `parGosper`, `Gosper` and `Zeilberger` has to be simplified or not. `OUT_FORMAT` can have two values: 0 (or equivalently $NON\_SIMPLIFIED$) or 1 (or equivalently $SIMPLIFIED$). The default value of `OUTPUT_FORMAT` is $NON\_SIMPLIFIED$.

Chapter 7

Some Examples

Let us take a look at some examples, that can be found in the files `testGosper.macsyma` and `testZeilberger.macsyma`.

Example *(A simple Gosper-summable example)*
Let us try to telescope $\frac{1}{4k^2-1}$ by Gosper's algorithm:
(prompt) `Gosper(1/4*k^2-1,k);`

output: $-\frac{1}{2(2k-1)}$

Let us now try to sum it over the interval $[1, 4]$, i.e. evaluate $\sum_{k=1}^{4} \frac{1}{4k^2-1}$

(prompt) `GosperSum(1/4*k^2-1, k, 1, 4);`

output: $\frac{4}{9}$

Example *(A less simple Gosper-summable example)*
Let us try to telescope $\frac{(-1)^k k}{4k^2-1}$ by Gosper's algorithm:
(prompt) `Gosper(((-1)^k*k)/(4k^2-1),k);`

output: $-\frac{(-1)^k}{4(2k-1)}$

Example *(A more complicated example)*

(prompt) `Gosper((a!*(-1)^k)/((a-k)!*k!),k);`

output: $-\frac{a!\,k\,(-1)^k}{a\,(a-k)!\,k!}$

Example *(An impossible case: binomial coefficient)*

(prompt) `Gosper(binomial(n,k),k,n);`

output: NO_HYP_SOL

To handle this we must use `parGosper` or `Zeilberger` as shown in the following example.

Example *(Binomial coefficient)*

To evaluate $\sum_{i=0}^{m} \binom{n}{k}$ we can use `Zeilberger` or `parGosper` and use the fact the we know that we expect a first order recurrence:

(prompt) `Zeilberger(binomial(n,k),k,n);`

output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \binom{n}{k}$$

and

$$Cert(n,k) = -\frac{k}{n-k+1}$$

and

$$a[0](n) = -2$$

$$a[1](n) = 1$$

Summing both sides of the recurrence and computing the sum is left as simple post-processing.

Assuming that the recurrence has order 1, we could have used `parGosper`:

(prompt) `parGosper(binomial(n,k),k,n,1);`

output:

$$\left\{-\frac{k}{n-k+1}, \{-2,1\}\right\}$$

Example *(Squared binomial coefficient)*

We can try to sum the squared binomial coefficient with `parGosper`, but to do this we must guess the order of the recurrence:
(prompt) `parGosper(binomial(n,k)^2,k,n,1);`

output:

$$\left\{-\frac{k^2\ (3\,n-2\,k+3)}{(n-k+1)^2}, \{-2\ (2\,n+1)\ , n+1\}\right\}$$

Example *(Binomial Theorem)*

Let us try a similar example:
(prompt) `Zeilberger(binomial(n,k)*x^k,k,n);`

output:

$$a[0]f(n,k)+a[1]f(n+1,k)=\delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k)=\binom{n}{k}x^k$$

and

$$Cert(n,k)=-\frac{k}{n-k+1}$$

and

$$a[0](n)=-(x+1)$$
$$a[1](n)=1$$

Example *(Vandermonde Identity recurrence)*

(prompt) `Zeilberger(binomial(a,k)*binomial(b,n-k),k,n);`

output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \binom{A}{k}\binom{b}{n-k}$$

and

$$Cert(n,k) = \frac{k(-n+k+b)}{n-k+1}$$

and

$$a[0](n) = -(n-b-A)$$
$$a[1](n) = -(n+1)$$

Example *(First Karlsson-Gosper identity identity)*

(prompt) `Zeilberger(binomial(n,k)*(n-1/4)!/(n-k-1/4)!/(2*n+k + 1/4)!*`
`9^(-k),k,n);`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \frac{(n-\frac{1}{4})!\binom{n}{k}}{9^k(n-k-\frac{1}{4})!(2n+k+\frac{1}{4})!}$$

and

$$Cert(n,k) = \frac{144k(8n+4(k-1)+13)(52n^2+16kn+75n-32k^2+24k+26}{(n-k+1)(4n-4k+3)(8n+4k+5)(8n+4k+9)}$$

and

$$a[0](n) = 2^8$$
$$a[1](n) = -27(3n+2)(12n+13)$$

Example *(Second Karlsson-Gosper identity)*

(prompt) `Zeilberger(binomial(n,k)*(n-1/4)!/(n-k-1/4)!/(2*n+k+5/4)!*9^(-k),k,n);`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \frac{(n-\frac{1}{4})!\binom{n}{k}}{9^k(n-k-\frac{1}{4})!(2n+k+\frac{5}{4})!}$$

and

$$Cert(n,k) = \frac{144k(8n+4(k-1)+17)(52n^2+16kn+127n-32k^2-4k+72)}{(n-k-1)(4n-4k+3)(8n+4k+9)(8n+4k+13)}$$

and

$$a[0](n) = 2^8$$
$$a[1](n) = -27(3n+4)(12n+17)$$

Example *(Trinomial coefficients)*

(prompt) `{Zeilberger(n!/k!/(k+m)!/(-2*k-m+n)!,k,n);`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) + a[2]f(n+2,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \frac{n!}{k!(n+k)!(n-m-2k)!}$$

and

$$Cert(n,k) = \frac{4k(m+k)(n+1)(n+2)}{(n-m-2k+1)(n-m-2k+2)}$$

and

$$a[0](n) = 3(n+1)(n+2)$$
$$a[1](n) = (n+2)(2n+3)$$
$$a[2](n) = -(n-m+2)(n+m+2)$$

Example *(Special case of the Strehl identity)*
(prompt) `Zeilberger(binomial(2*k,k)*binomial(n,k)^2,k,n);`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) + a[2]f(n+2,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$Cert(n,k) = -\frac{k^3(n+1)^2(4n-3k+8)}{(n-k+1)^2(n-k+2)^2}$$

and

$$a[0](n) = 9(n+1)^2$$
$$a[1](n) = -(10n^2+30n+23)$$
$$a[2](n) = (n+2)^2$$

Example *(Fibonacci sequence-related recurrence)*
(prompt) `{Zeilberger((n!*(n+k)!)/(k!^3*(n-k)!^2),k,n));`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) + a[2]f(n+2,k) + a[3]f(n+3,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \frac{n!(n+k)!}{k!^3(n-k)!^2}$$

and

$$Cert(n,k) = -\frac{k^3(n+1)(11n^2-6kn+37n-k^2-7k+30)}{(n-k+1)^2(n-k+2)^2}$$

and

$$a[0](n) = -(n+1)^2$$
$$a[1](n) = -(11n^2+33n+25)$$
$$a[2](n) = (n+2)^2$$

Example *(The binomial coefficient to the third power)*

We can also use `parGosper` to find a recurrence on such sequence, but we must guess its order:

(prompt) `parGosper(binomial(n,k)^3,k,n,1);`
output: $\{0, \{0, 0\}\}$

This means that `parGosper` could not find a first order recurrence of the desired form but we don't give up. Let us look for a second order recurrence:

(prompt) `parGosper(binomial(n,k)^3,k,n,2);`
output:

$$\left\{\frac{k^3\ (n+1)^2\ \left(14\,n^3 - 27\,k\,n^2 + 74\,n^2 + 18\,k^2\,n - 93\,k\,n + 128\,n - 4\,k^3 + 30\,k^2 - 78\,k + 72\right)}{(n-k+1)^3\ (n-k+2)^3},\right.$$
$$\left.\left\{8\ (n+1)^2, 7\,n^2 + 21\,n + 16, -(n+2)^2\right\}\right\}$$

Example *(Binomial coefficient to the fourth power)*

(prompt) `Zeilberger(binomial(n,k)^4,k,n);`

output:

$$a[0]f(n,k) + a[1]f(n+1,k) + a[2]f(n+2,k) = \delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \binom{n}{k}^4$$

and

$$Cert(n,k) = -\frac{\begin{array}{c} k^4(n+1)(74n^6 - 260kn^5 + 725n^5 + 374k^2n^4 - 2056kn^4 + 2885n^4 - \\ 276k^3n^3 - 6420kn^3 + 6045n^3 + 104k^4n^2 - 1244k^3n^2 + 5298k^2n^2 - \\ 9892kn^2 + 7030n^2 - 16k^5n + 298k^4n - 1884k^3 + 5322k^2n - 7520kn + \\ 4300n - 20k^5 + 210k^4 - 900k^3 + 1980k^2 - 2256k + 1080 \end{array}}{(n-k+1)^4(n-k+2)^4}$$

and

$$a[0](n) = -4(n+1)(4n+3)(4n+5)$$
$$a[1](n) = -2(2n+3)(3n^2+9n+7)$$
$$a[2](n) = (n+2)^3$$

Chapter 8

The Code

The implementations of both Gosper's algorithm and Zeilberger's fast algorithm have been entirely coded in the internal LISP-like language of the computer algebra system Macsyma vers. 419. The implementation exploits Macsyma computer algebra engine for factorizing, simplifying and normalizing polynomials and rational functions, and the modularity of the Macsyma language, but it does not use Macsyma hypergeometric tools like Gosper's implementation of his own algorithm because Zeilberger's algorithm requires a parametrized version of Gosper's algorithm.

For a print-out of the entire code we refer to the RISC Combinatorics home page at the U.R.L.: http://www.risc.uni-linz.ac.at/research/combinat/risc/.

The code is contained in the following files:

`algUtil.macsyma`	$algebraic\ utilities$
`shiftQuotient.macsyma`	$shiftQuotient\ computation$
`poly2quint.macsyma`	$internal\ data\ structures\ conversions$
`makeGosperForm.macsyma`	$Gosper's\ form\ related\ routines$
`GosperEq.macsyma`	$Gosper's\ equation\ related\ routines$
`Gosper.macsyma`	$Gosper's\ algorithm\ main\ routines$
`Zeilberger.macsyma`	$Zeilberger's\ algorithm\ main\ routines$
`LOADZeilberger.macsyma`	$Zeilberger's\ routines\ loader$
`testZeilberger.macsyma`	$Some\ examples$
`testGosper.macsyma`	$Some\ Gosper'salgorithm - related\ examples$

8.1 Low Level Routines

The low level routines are contained in the files `algUtil.macsyma`, `shiftQuotient.macsyma` and `poly2quint.macsyma`.

The first file contains the lowest level algebraic routines for handling polynomials (extracting components of polynomials, degree, etc).

The second file contains routines necessary for computing the shift-quotient of a hypergeometric sequence and for computing the result of the application of a linear recurrence operator to a hypergeometric sequence, which is computed by a Horner-like algorithm:

```
niceForm(hyp,var,parName,ord) :=
  block(
  [shQuo,num,den,res],
  res:parName[ord],
  shQuo :  shiftQuo(hyp,n),
  for i :  ord step -1 thru 1 do
   res :  xthru(parName[i-1] +
     shiftFactPoly(shQuo,n,i-1)*res),
  return(res)
  );
```

The third file contains the routines necessary for storing polynomials in special data structures; namely *quintuples* that are well-suited for building the desired *Gosper form* of the shift-quotient of a proper hypergeometric sequence. Such *quintuples* are arrays in which the following information of a polynomial is stored:

1. degree
2. leading coefficient
3. second leading coefficient
4. tail (polynomial without the first two monomials)
5. multiplicity (number of occurrences) of the polynomial in the shift quotient

Example

If the polynomial $(7x^3 - 5x^2 + 4x + 8)$ appears in the shift quotient with a multiplicity, say 5, it will be encoded in the following quintuple: $[3, 7, -5, x^2 + 4x, 5]$.

8.2 Gosper-form Related Routines

All the Gosper-form related routines are contained in the file makeGosperForm.macsyma. The main routine builds the desired form by checking iteratively the gcd condition required by the Gosper-form; whenever the condition is violated the undesired factors will be moved from the polynomials q and r to the polynomial p. This procedure exploits the ad hoc data structure *quintuple*, which has been coded in poly2quint.macsyma:

```
...
for i:1 thru sizeListNum do
 (
 for j:1 thru sizeListDen do
  (
  if (quintDegree(listNum[i])=quintDegree(listDen[j])) and
   (quintLeadCoeff(listNum[i])=quintLeadCoeff(listDen[j])) then
   (
   /* Checking the preliminary cond on the 2nd highest coeffs */
   test:(quintLastButOne(listNum[i])-quintLastButOne(listDen[j]))/
    (quintDegree(listNum[i])*quintLeadCoeff(listNum[i])),
   if integerp(test) then
    if test>0 then
     (
     /* Checking the condition */
     if expand(sqfQuint2Poly(listNum[i],k)-
      subst(k+test,k,sqfQuint2Poly(listDen[j],k)))=0 then
       if test > maxTest then maxTest:test
     ) /* end if */
   ) /* end if */
  ) /* end for */
 ), /* end for */
...
```

8.3 Gosper's Equation Routines

The computation of the degree of the solution and the solution of the so-called "Gosper's equation" has been coded in the file `GosperEq.macsyma`. Solving is done by simple linear algebra on the unknown coefficients of the solution:

```
SolveZSys(Gpoly,unkPar1,numUnkPar1,unkPar2,numUnkPar2,indet) :=
  linsolve(poly2List(Gpoly,indet),
   union(
    makelist(unkPar1[i],i,0,numUnkPar1-1),
    makelist(unkPar2[i],i,0,numUnkPar2-1)
    )
  );
```

8.4 Gosper's Algorithm and Zeilberger's Fast Algorithm

Gosper's and Zeilberger's algorithm have been coded in two versions (verbose and non-verbose) in the file `Gosper.macsyma`. The decision to duplicate the code in the verbose and non-verbose code comes from the fact that we did not want to slow down the non-verbose version by inserting conditions in the code and that there is no preprocessor in the Macsyma language that could have automatically generated the two versions.

Zeilberger's fast algorithm is a just a parametrized version of Gosper's algorithm, in which the linear solving of the recurrence equation is done with polynomial parameters.

The coding follows in a straightforward way the standard Gosper algorithm described in the previous chapters:

```
GosperNonVerbose(f,k) :=
  block(
 ...
  fRat:  fRat(f,k), /* Computation of the shift quotient */
 ...
  /* Computation of the Gosper form */
  GosperForm:  makeGosperForm(fRat,k),
  p:  takeP(GosperForm),
  q:  takeQ(GosperForm)*signFlag, /* Adjusting the sign */
  r:  takeR(GosperForm),
  /* Solution of the Gosper equation */
  AnsatzDegree:  AnsatzDegree(p,q,r,k),
```

```
Gpoly: expand(GosperPoly(p,q,r,y,c,AnsatzDegree)),
sysSol: SolveSys(Gpoly,c,AnsatzDegree+1,y),
GosperSol: sol2Poly(first(sysSol),k),
/* Postprocessing: build the sol. to the telesc. prob. */
if GosperSol = 0 then
 tlscope: NO_HYP_SOL
else
 tlscope:telescope(f,GosperSol,q,r,k) /* OUTPUT */
);
```

Bibliography

[1] **Abramov, S.A.**

Rational solutions of linear differential and difference equations with polynomial coefficients,

Proc. ISSAC '95, ACM Press, 1995, 285-289.

[2] **Chyzak, F.**

Fonctions holonomes en calcul formel

Thèse universitaire no. TU 0531, INRIA. Defended on May 27, 1998. 227 pages.

[3] **Gosper, R.W.**

Decision procedure for indefinite hypergeometric summation

Proceedings of the National Academy of Sciences of USA 75 (1978), 40-42.

[4] **Graham, R., Knuth, D. E., Patashnik, O.**

Concrete Mathematics - A Foundation for Computer Science,

Addison Wesley, 1994.

[5] **Paule, P.**

Greatest factorial factorization and symbolic summation

J. Symbolic Computation, **20**, 1995, 235-268.

[6] **Paule, P., Schorn, M.**

A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities

J. Symbolic Computation, **20**, 1995, 673-698.

[7] **Paule, P., and Riese, A.**

A Mathematica q-Analogue of Zeilberger's Algorithm Based on an Algebraically Motivated Approach to q-Hypergeometric Telescoping

Special Functions, q-Series and Related Topics, Fields Institute Communications, Vol. 14, pp. 179-210, 1997.

[8] **Petkovšek, M., Wilf, H., Zeilberger, D.**
A=B,
A K Peters, MA (USA), 1996.

[9] **Zeilberger, Doron**
A fast algorithm for proving terminating hypergeometric identities
Discrete Mathematics, 80 (1990), 207-211.

[10] **Zeilberger, Doron**
The method of creative telescoping
Journal of Symbolic Computation 11 (1991), 195-204.