Interactive Computer Manipulation of Formal Sums

(Thesis format: Monograph)

by

Nivedita Patil

Graduate Program
in
Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
THE SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Supervisor:	Examination committee:
Dr. Stephen Watt	Dr. Marc Moreno Maza
	Dr. Olga Veksler
	Dr. Greg Reid

The thesis by

Nivedita Patil

entitled:

Interactive Computer Manipulation of Formal Sums

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date _____ Chair of the Thesis Examination Board

Abstract

The goal of this thesis is to explore how computer algebra systems can be augmented to allow user-guided transformation and simplification of expressions involving symbolic summation. Mathematical expressions represented as trees are one of the data objects of computer algebra systems. By accessing and manipulating these data objects we can simplify and transform expressions involving symbolic summation. To choose what transformations to be performed is under the discretion of the user. We present a conceptual framework to perform transformations on expressions involving symbolic summation. This is done by creating a set of library functions for interactive manipulation of formal sums. We base our design on the properties of summation. This idea is also extended to other associative operators such as product and definite integral.

Keywords. Formal Sums, Computer Algebra System, Associative Operators.

Acknowledgments

I would like to thank my thesis supervisor Dr. Stephen Watt from the Department of Computer Science at University of Western Ontario. He helped me understand the concepts and directed me in the right direction. I am grateful to him for his excellent support in all arenas.

I would also extend my thanks to the members of ORCCA (Ontario Research Centre for Computer Algebra) lab and the faculty and fellow students in Computer Science Department for their invaluable guidance and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Symbolic computation refers to automatic manipulation of mathematical expressions in symbolic form as opposed to finding numerical values for these expressions. A computer algebra system is a computer program that performs symbolic computation. Powerful interactive systems for doing symbolic computation have been designed and built. We still face challenging problems, however, while manipulating and simplifying expressions involving symbolic summation. Existing systems do not provide an adequate set of tools for transforming symbolic sums in the ways required for interactive simplification. In a computer algebra system performing direct transformations on symbolic summations involves a series of operations and can be very hard and tedious. Such manipulations and transformations are very much needed in many problems involving formal sums. Consider the following transformations,

$$\sum_{i=1}^{2n} x_i \implies \sum_{i=1}^{n} x_{2i} + \sum_{i=1}^{n} x_{2i-1}$$

$$\sum_{k=1}^{100} G(k) \implies \sum_{k=1}^{10} G(k) + \sum_{k=11}^{100} G(k)$$

$$\sum_{j=a}^{b} [P(j) + Q(j)] \implies \sum_{j=a}^{b} P(j) + \sum_{j=a}^{b} Q(j)$$

In the first transformation, we split the summation into even and odd terms. In the second we split the summation at a term and in the last transformation we split the summation based on the sum in the inner expression. To perform these transformations in current computer algebra systems we need to access the parameters of symbolic summation and then set the values of the resultant summation. This can be very laborious and often cannot be performed in a single step. Such transformations are naturally done by hand. There is a huge gap between solving symbolic summation problems by hand and performing the same operations by a computer algebra system. In our thesis we try to bridge this gap for formal sums.

The goal of this thesis is to explore how computer algebra systems can be augmented to allow user-guided transformation and simplification of expressions involving symbolic summation.

Symbolic summation manipulations arise often in practice. Computer algebra systems should be able to accommodate manipulations and computations of this type. Mathematical expressions represented as trees are one of the most important data objects of computer algebra systems. By accessing and manipulating these data objects we can simplify and transform expressions involving symbolic summation. The choice of transformations is given by the user. In our thesis we propose a technique to solve these problems by accessing the parameters of the formal sum. Then we exploit the axioms of summation to perform the required manipulations. Currently no symbolic computation system implements these axioms to solve such problems. We give a conceptual framework and provide symbolic tools to create a problem solving environment. This is done by creating a set of library functions for interactive manipulation of formal sums. We base our design on the properties of summation. This idea is also extended to other associative operators such as products and definite integrals. The idea is to computationally support the user in solving problems involving formal sums.

This thesis is organized as follows: In Chapter 2, through examples of symbolic summation, we show the necessity for user defined transformations and manipulation

of formal sums. Here we also discuss previously developed methods for expression manipulation and symbolic summation manipulation techniques.

The transformations and manipulations of expressions are governed by a set of rules or axioms. Properties of summation form the basis of our design. These details are covered in Chapter 3.

Chapter 4 gives details of Maple software package containing the library functions for manipulation of formal sums.

Chapter 5 describes how our algorithms have been extended to other associative operators such as products and definite integrals.

In Chapter 6 we show how our library functions provide the necessary tools to manipulate summations, such as the examples of Section 2.1.

Chapter 7 provides our final overview of the project, and gives conclusions and extensions for future work.

Chapter 2

Preliminaries

2.1 Motivation

The objective of this section is to present some examples of symbolic summation problems and discuss the difficulties of applying computer algebra to such problems. The leading computer algebra systems are powerful tools but they have many drawbacks. The motivation for our study came when we were trying to derive some identities involving formal sums on a computer algebra system. We saw that to perform transformations and manipulations on these sums is not so straightforward. The following examples illustrate typical problems which occur while obtaining these identities in a computer algebra system.

2.1.1 Example: The Binomial Theorem

The binomial theorem describes the algebraic expansion of powers of a binomial. According to this theorem, it is possible to expand any power of $x + y$ into a sum of the form as in (2.1). This example has summation involving binomial coefficients. There are other ways to do the proof, but this is just being used as an example of manipulating summations, not as the best way to prove the result. We prove this

theorem by induction. Note that we are not interested in proving the theorem given by (2.1), but rather in showing what sorts of transformations are useful in manipulating formal sums.

To be proved: For any $n \epsilon \mathbf{N}$ and any $x, y \epsilon \mathbf{R}$,

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} \tag{2.1}$$

Proof: We prove it by induction. For $n = 0$ we have

$$(x+y)^0 = \binom{0}{0} x^0 y^0 = 1$$

Assume the formula holds for some fixed arbitrary $n \epsilon \mathbf{N}$. Then,

$$\begin{aligned}
(x+y)^{n+1} &= (x+y)(x+y)^n \\
&= (x+y)\sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} && (2.2)\\
&= \sum_{k=0}^{n} \binom{n}{k} (x+y) x^k y^{n-k} && (2.3)\\
&= \underbrace{\sum_{k=0}^{n} \binom{n}{k} x^{k+1} y^{n-k}}_{\text{set } l = k+1} + \underbrace{\sum_{k=0}^{n} \binom{n}{k} x^k y^{n+1-k}}_{\text{set } l = k} && (2.4)\\
&= \sum_{l=1}^{n+1} \binom{n}{l-1} x^l y^{n+1-l} + \sum_{l=0}^{n} \binom{n}{l} x^l y^{n+1-l} && (2.5)\\
&= \sum_{l=1}^{n} \binom{n}{l-1} x^l y^{n+1-l} + x^{n+1} + y^{n+1} + \sum_{l=1}^{n} \binom{n}{l} x^l y^{n+1-l} && (2.6)\\
&= x^{n+1} + \sum_{l=1}^{n} \left[\binom{n}{l-1} + \binom{n}{l} \right] x^l y^{n+1-l} + y^{n+1} && (2.7)
\end{aligned}$$

$$= x^{n+1} + \sum_{l=1}^{n} \binom{n+1}{l} x^l y^{n+1-l} + y^{n+1} \tag{2.8}$$

$$= \sum_{l=0}^{n+1} \binom{n+1}{l} x^l y^{n+1-l} \tag{2.9}$$

which completes the proof by induction.

Proving this theorem by hand is much simpler than doing it on a computer algebra system. The transformation of multiplying the inner expression of the summation in (2.2) with $(x + y)$ and transforming it to (2.3) is extremely awkward in current computer algebra systems, such as Maple. Splitting the summation based on the sum $(x + y)$ as in (2.3) and transforming to (2.4) is also tedious. Then rearranging or shifting the first summation of (2.4) by 1 and transforming it to (2.5) involves a number of operations. Later combining the two summations which have the same range $l = 1 \,..\, n$ in (2.6) and forming (2.7) is also laborious. We apply the Pascal's identity,

$$\binom{n+1}{r} = \binom{n}{r-1} + \binom{n}{r}$$

to the binomial coefficients of (2.7) and then deduce to (2.8). Finally in (2.9) we prove the theorem for power $n + 1$. The intention is not to develop a theorem prover but rather to develop techniques to perform transformations and manipulations of summations on a computer algebra system.

2.1.2 Example: Summation involving Stirling Numbers

Stirling numbers often occur in combinatorics. There are two different kinds: the Stirling numbers of the first kind and the Stirling numbers of the second kind. In this example we make use of the second kind. The Stirling numbers of the second kind count the number of ways of partitioning n distinct objects into k non-empty sets. They are denoted by,

$$\left\{ \begin{array}{c} n \\ k \end{array} \right\} \qquad \text{or} \qquad S(n,k)$$

Falling power: The expression x to falling power m is denoted by $x^{\underline{m}}$ whose value is given by,

$$x^{\underline{m}} = x(x-1)(x-2)...(x-(m-1)) \tag{2.10}$$

According to Theorem [12,4.1] we can convert between powers and falling powers using the following formula,

$$x^n = \sum_{k=0}^{n} \left\{ \begin{array}{c} n \\ k \end{array} \right\} x^{\underline{k}} \quad \text{or} \quad x^n = \sum_{k=0}^{n} S(n,k) x^{\underline{k}}$$

Proof: We will prove this by induction on n.
The base case $n = 0$ is trivial.

$$x^0 = S(0,0)x^{\underline{0}} = 1 \tag{2.11}$$

Our induction hypothesis is that $x^n = \sum_{k=0}^{n} S(n,k)x^{\underline{k}}$ holds for all $n \leq r$

$$\begin{aligned}
x^{r+1} &= xx^r \\
&= x\sum_{k=0}^{r} S(r,k)x^{\underline{k}} && (2.12) \\
&= \sum_{k=0}^{r} xS(r,k)x^{\underline{k}} && (2.13) \\
&= \sum_{k=0}^{r} S(r,k)\left[x^{\underline{k+1}} + kx^{\underline{k}}\right] && (2.14) \\
&= \sum_{k=0}^{r} S(r,k)x^{\underline{k+1}} + \sum_{k=0}^{r} S(r,k)kx^{\underline{k}} && (2.15) \\
&= \sum_{k=1}^{r+1} S(r,k-1)x^{\underline{k}} + \sum_{k=0}^{r} S(r,k)kx^{\underline{k}} && (2.16)
\end{aligned}$$

$$= \sum_{k=1}^{r} S(r,k-1)x^{\underline{k}} + S(r,r)x^{\underline{r+1}} + S(r,0)x^{\underline{0}} \cdot 0 + \sum_{k=1}^{r} S(r,k)kx^{\underline{k}} \quad (2.17)$$

$$= S(r,r)x^{\underline{r+1}} + \sum_{k=1}^{r} \left[S(r,k-1)x^{\underline{k}} + S(r,k)kx^{\underline{k}} \right] \quad (2.18)$$

$$= x^{\underline{r+1}} + \sum_{k=0}^{r} S(r+1,k)x^{\underline{k}} \quad (2.19)$$

$$= \sum_{k=0}^{r+1} S(r+1,k)x^{\underline{k}} \quad (2.20)$$

In the first step, we applied our induction hypothesis, then in (2.12) we moved x inside the sum. The following equation from Lemma [12,4.2] for falling powers is used to transform (2.13) to (2.14).

$$xx^{\underline{k}} = x^{\underline{k+1}} + kx^{\underline{k}} \quad (2.21)$$

We then split the summation into one summation based on $x^{\underline{k+1}}$ and the other based on $x^{\underline{k}}$ and later shift the first summation by 1. Again we split the summations at a term to get them in the same range $k = 1 \,..\, r$. Now we can pull out the terms corresponding to $k = r+1$ and $k = 0$ and set $S(r,r) = 1$ and $S(r,0) = 0$. Next we combine the summations with the same range and transform from (2.18) to (2.19) using the following equation from Theorem [12,4.2]. Lastly in (2.20) we prove the theorem for $r+1$.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k) \quad (2.22)$$

The transformation of multiplying the inner expression of summation in (2.12) with x and transforming it to (2.13) involves a series of operations. Splitting the summation based on the sum $x^{\underline{k+1}} + kx^{\underline{k}}$ as in (2.14) and transforming to (2.15) is a bit laborious. Rearranging or shifting the first summation of (2.15) by 1 and transforming it to (2.16) involves a series of operations. Later combining the two summations of (2.17) as they

have the same range $k = 1 \,..\, r$ and forming (2.18) is very time consuming in the current computer algebra system.

2.1.3 Example: Summation involving functions

In this example, we have an input expression involving summations as in (2.23). We need to manipulate this expression and transform it to the expression (2.24).

$$\text{Input: } \sum_{i=0}^{n-1} \big[f(i) + f(i+1)\big] + \sum_{i=1}^{n} \big[g(i) + g(i+1)\big] \tag{2.23}$$

$$\text{Output: } f(0) + 2f(1) + g(1) + 2\sum_{i=2}^{n-1} \big[f(i) + g(i)\big] + f(n) + 2g(n) + g(n+1) \tag{2.24}$$

The following steps represent the way we would derive (2.24) by hand.

$$\sum_{i=0}^{n-1} \big[f(i) + f(i+1)\big] + \sum_{i=1}^{n} \big[g(i) + g(i+1)\big]$$

$$\begin{aligned}
&= \sum_{i=0}^{0} \big[f(i) + f(i+1)\big] + \sum_{i=1}^{n-1} \big[f(i) + f(i+1)\big] + \sum_{i=1}^{n-1} \big[g(i) + g(i+1)\big] \\
&\quad + \sum_{i=n}^{n} \big[g(i) + g(i+1)\big] && (2.25) \\
&= f(0) + f(1) + \sum_{i=1}^{n-1} \big[f(i) + f(i+1)\big] + \sum_{i=1}^{n-1} \big[g(i) + g(i+1)\big] + g(n) \\
&\quad + g(n+1) && (2.26) \\
&= f(0) + f(1) + \sum_{i=1}^{n-1} f(i) + \sum_{i=1}^{n-1} f(i+1) + \sum_{i=1}^{n-1} g(i) + \sum_{i=1}^{n-1} g(i+1) + g(n) \\
&\quad + g(n+1) && (2.27) \\
&= f(0) + f(1) + \sum_{i=1}^{n-1} f(i) + \sum_{i=2}^{n} f(i) + \sum_{i=1}^{n-1} g(i) + \sum_{i=2}^{n} g(i) + g(n) \\
&\quad + g(n+1) && (2.28)
\end{aligned}$$

$$= f(0) + f(1) + \sum_{i=1}^{1} f(i) + \sum_{i=2}^{n-1} f(i) + \sum_{i=2}^{n-1} f(i) + \sum_{i=n}^{n} f(i) + \sum_{i=1}^{1} g(i)$$
$$+ \sum_{i=2}^{n-1} g(i) + \sum_{i=2}^{n-1} g(i) + \sum_{i=n}^{n} g(i) + g(n) + g(n+1) \tag{2.29}$$

$$= f(0) + f(1) + f(1) + \sum_{i=2}^{n-1} f(i) + \sum_{i=2}^{n-1} f(i) + f(n) + g(1) + \sum_{i=2}^{n-1} g(i)$$
$$+ \sum_{i=2}^{n-1} g(i) + g(n) + g(n) + g(n+1) \tag{2.30}$$

$$= f(0) + 2f(1) + g(1) + \sum_{i=2}^{n-1} \big[2f(i) + 2g(i)\big] + f(n) + 2g(n) + g(n+1) \tag{2.31}$$

$$= f(0) + 2f(1) + g(1) + 2\sum_{i=2}^{n-1} \big[f(i) + g(i)\big] + f(n) + 2g(n) + g(n+1) \tag{2.32}$$

On observing (2.23), we see that we can combine the two summations if we get them in the same range. To get the summations in the same range i.e. $i = 1 \,..\, n-1$, we perform a split in the first summation at $i = 0$ and the second summation at $i = n - 1$ and form (2.25). The summations with range $i = 0 \,..\, 0$ and $i = n \,..\, n$ are nothing but terms hence we get their values in (2.26). We split the summations based on the inner expression and form (2.27). The output does not contain any $i+1$ terms hence to get this we shift the summations with $i+1$ terms by 1 and form (2.28). We need to get the summations in range $i = 2 \,..\, n-1$, hence we split the summations at corresponding intervals and form (2.29). Summations with intervals $i = 1 \,..\, 1$ and $i = n \,..\, n$ are nothing but values hence we get these values and form (2.30). Then we combine the summations with range $i = 2 \,..\, n-1$ and form (2.31). Finally in (2.32) we factor out 2 from the summation to get the expected output.

As seen in the above three examples it is challenging to solve these symbolic summation problems in current algebraic systems. To perform such transformations in a computer algebra system such as Maple, we need to get each operand of summation and manipulate them and then set the values for the resultant summation. Hence to

do this we need to understand how expressions are edited, represented and manipulated in the current computer algebra systems, which is described in detail in the following section.

2.2 Expression Editors

An expression editor provides an interface to create and edit mathematical expressions in a computer. Expression editors are mathematical manipulation and typesetting (presentation of textual material in graphic form) programs which help to create and edit expressions. The main purpose of expression editors are to display and re-arrange expressions in mathematically meaningful ways and perform transformations to alter the data and evaluate expressions. User Interface plays an important role in such an application. Its intuitive GUI offers visual navigation of sub-expressions and an expression template palette for expression input. The main design criteria for these editors should be their math capabilities. Several mathematical editors have been developed in the past ten years. MathEdit, Matlab, Microsoft Equation Editor(for Microsoft Windows only), Mathcad, IBM MathML expression editor [2] developed by IBM Research and being acquired by Integre Research Group are a few examples which are used in computing science and education.

2.2.1 Expression Trees

An expression in an editor is usually structured internally as a tree. The structure of an expression involves the relationship between the operands and operators. Since expressions are one of the data objects of computer algebra systems, an understanding of this structure is essential.

Figure 2.1 shows the structure of an expression tree for $a + bx^2$. The operator at the root is called the main operator, a designation that emphasizes $a + b * x \wedge 2$ is viewed as a sum with two operands a and $b*x\wedge 2$. The structure of the tree makes the

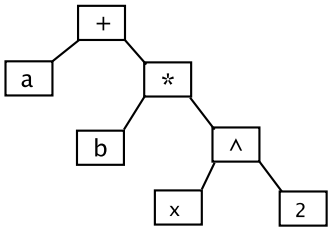

Figure 2.1: Expression tree for $a + bx^2$

order of operations for an expression implicit. It can be thought of as removing the precedence rules and just having parentheses. The plus sign has the least precedence and so is on the top. Similarly, x binds tighter to 2 through the power operation than to b through the times operation. A simple set of rules based on the precedence of each operator and whether it is left or right associative determines where parentheses are needed. To simplify this expression we need to perform some evaluations. Evaluation can be defined as mapping from an object which is the input to another simpler or more specific object i.e. output.

2.2.2 Serialization Format

Serialization is a process of converting an object or data structure to an ordered series of bytes for permanent storage or for transmission across the network. It needs to store enough information of the original object for recreation including other objects to which it refers. The format used to serialize the expressions should be simple to parse into an internal representation, easily understood by human to be applied for various tasks and compatible for wide range of applications. They produce a semantic representation of the expression. During serialization of the tree structure, the precedence of the operators is examined to determine where parentheses are needed to faithfully reconstruct the tree. Expression (2.34) illustrates the serialized version

of (2.33).

$$\frac{\sqrt{\cos\left(5x+2\right)}}{4+\sin x} \tag{2.33}$$

$$\text{sqrt(cos(5}*\text{x+2))/(4+sin(x))} \tag{2.34}$$

2.2.3 Presentation versus Content

There are two kinds of expression editors, presentation-based and content-oriented. Presentation-based editors produce representation of formulae and store the typesetting primitives rather than the meaning. They do not store a representation of the formula that can be manipulated and computed. For automatic computer processing the resulting expression in these editors is ambiguous or there is no interpretation to them as they are based on typesetting language. Content oriented editors are difficult to find. The few are at a developmental stage: JOME [7] and MathML Expression Editor [2]. There are some editors which are integrated to a broad spectrum computer algebra system to perform transformation and manipulations of expressions with a rich set of mathematical libraries. The math software provides an extensive function library supporting general math, trigonometric, chemistry, geometric, statistical, and numerous random number generators following a variety of statistical distribution and many other function categories.

2.2.4 Web Support

There are few standalone editors, but now most editors provide browser support either natively or through a plug-in. It must be flexible and customizable to support different category of users. For Web publications expressions are created and then converted to images. HTML has no real support for mathematics, often one must resort to gif images. Using images results in poor quality printing and loss of all semantic information. Editors should provide a translation system to mathematics

entered in the editing window to virtually any text based language. There has historically been no means to 'cut and paste' mathematical expressions from a Web page into (say) a computer algebra package. This is now possible with two new technologies OpenMath and MathML. These are two different encoding of mathematics on the Web. Editors can serialize the content tree to these encodings. MathML is an xml application to describe mathematical notation and capturing both its structure and content. It is a standard for representing mathematics on the Web. There are two kinds of MathML: Presentation and Content. Presentation focuses on how the expression looks such as color, size and position where as Content as the name implies focuses on expressing operations such as addition, integration, matrices, etc. OpenMath is another emerging standard for representing mathematical objects with their semantics for purposes such as exchanging between computer programs, to be published on the Web or to be stored in a database. MathML and OpenMath are highly complementary. Though MathML does have some limited facilities for dealing with content, it also allows semantic information encoded in OpenMath to be embedded inside a MathML structure.

Though expression editors provide a way to create formal sums it is not possible to manipulate and perform transformations on such sums. These editors support a set of library functions which can be extended. This thesis is an effort to provide this functionality by designing and implementing a set of functional commands for such transformations. Maple is a symbolic and numeric computation system with support for importing and exporting MathML 2.0, including both presentation and content forms of MathML.

2.3 Previous Work

It is quite important to understand the methods for expression manipulation and draw ideas for symbolic summation manipulation. In this section we explore various ways of expression manipulation and illustrate with examples the train of thought that was

followed while designing the manipulation of expressions. In Section 2.3.1 we describe the concept of direct manipulation and discuss some editors which have implemented this concept. Analytica is a theorem prover which uses summation properties to prove theorems involving symbolic summation and is discussed in Section 2.3.2. In Section 2.3.3 we discuss the Theorema system which defines rules for various operators and performs algebraic manipulations based on these rules. Finally in Section 2.3.4 we show how manipulation of expressions are performed using term rewriting techniques.

2.3.1 Direct Manipulation

Direct manipulation is a very interesting area of expression manipulation. The main goal of this concept is to combine the process of editing and simplification. In a direct-manipulation editor for mathematical content we can edit an expression "in place" as the expression is formatted and displayed on the screen in a traditional notation. The process of editing and displaying occur simultaneously and the expression is reformatted at every modification. Amaya, FrameMaker, MathType, LyX and Mathematica provide direct-manipulation editors. These work on a structured representation of the document. Mathematical structures contain empty regions for sub-expressions usually denoted by $\square$ for entering sub-expressions. $\sqrt[\square]{\square}$ and $\frac{\square}{\square}$ represent the structures for roots and fractions respectively. The users can click on any of these boxes to set the focus and enter the data. These editors have a lot of usability issues such as, performing another intermediate step of mentally parsing the expression to decide the sequence of structures to be chosen, coordinating between the presentation and structural representation of expressions and extensibility to new operators.

There are few editors such as MathEx which allow customization of structural palettes, keyboard editing and notational presentation. Another example of direct manipulation would be, for instance moving a symbol in an expression implicitly means solve the expression with respect to this variable. A few editors also provide the facility for the user to move sub-expressions around while the system maintains

the meaning of the surrounding expression or makes substitutions by dragging and dropping formulas. In most direct-manipulation editors content and presentation MathML, OpenMath semantic markup are used for internal representation of structural and presentational information of expressions. Let us see the concept of direct manipulation implemented in Amaya and JOME.

Amaya

Amaya [21] is a Web editor where in we can create and update documents directly on the Web. Amaya proposes a WYSIWYG interface where MathML mathematical expressions are handled as structured components. The input structure for expression $\frac{x^4}{\sqrt{y-z}}$ looks as shown in Figure 2.2. In editors such as Amaya the cursor navigation defines the visual presentation and the content representation is also indicated explicitly. In Figure 2.2 the user sets the position of the cursor to change the value of 4, automatically the pointer in the MathML representation goes to the corresponding data as shown in Figure 2.3 and updates the value simultaneously.

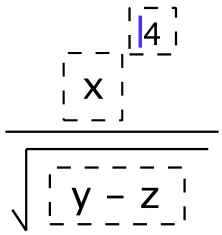

Figure 2.2: An input structure for expression $\frac{x^4}{\sqrt{y-z}}$ in Amaya showing the cursor positioned at 4 for updating

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
        "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<!-- Created by amaya 11.3.1, see http://www.w3.org/Amaya/ -->
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mfrac>
    <msup>
      <mi>x</mi>
      <mn>4</mn>
    </msup>
    <msqrt>
      <mi>y</mi>
      <mo>&minus;</mo>
      <mi>z</mi>
    </msqrt>
  </mfrac>
</math>
```

Figure 2.3: Presentation MathML for expression $\frac{x^4}{\sqrt{y-z}}$ in Amaya showing the data being updated simultaneously with the input

JOME - Java OpenMath Editor

JOME [7] is a self-contained software component for visualization and manipulation of mathematical formulae. It uses OpenMath for the representation of semantically rich mathematical objects. It is written in Java and is conceptually based on Model-View-Controller(MVC) design pattern. It proposes selection/deselection mechanism for interacting with displayed formula. Selection determines the part of formula to which the desired manipulation will apply and is the first brick of functionality for interactivity. According to the element pointed to by the mouse, JOME automatically fits the selected sub-expression to be the smallest syntactically correct expression.

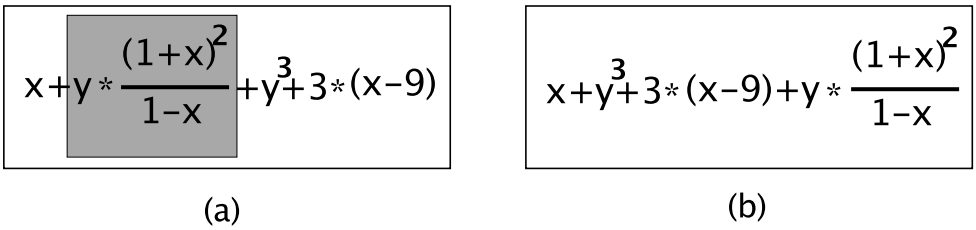

Figure 2.4: A sub-expression manipulated according to the commutative property of addition

This guarantees the coherence of subsequent operations on the expression. It is possible to select the whole formula or just parts of it. Once selected, depending on the properties of the operator for which the selected sub-expression is the operand, the sub-expression can be manipulated. JOME has some support for manipulating formulas with semantic drag and drop (the selection can be moved from one side of an operator to the other with a relevant mathematical transformation performed). For example consider Figure 2.4, it is possible to "move" the selected operands based on the commutative property of addition to the extreme right resulting in an expression as shown in Figure 2.4 (b).

2.3.2 Analytica

A technique similar to our approach exists in Analytica [5], an experimental automatic theorem prover written on top of Mathematica computer algebra system. The main problem to prove theorems is the large amount of domain knowledge needed for even the simplest of proofs. Hence Analytica combines theorem proving with the mathematical knowledge that is built in the symbolic computation so that each simplification step can be rigorously justified. Mathematica is based on term-rewriting and also provides a powerful rule-based programming language. It also deals with expressions involving symbolic summation and product operators. In addition to the simplification rules that are provided by the symbolic computation system, it makes use of summation properties to prove theorems. The rules in Analytica for summation are divided into two sets, **SumSimplifyRules** and **SumRewriteRules**. The

former is used to rewrite summation to simpler forms and the latter is used to rewrite to equivalent form but not necessarily simpler. The following comes from a lemma used in the proof of existence of a continuous nowhere differentiable function given by Weierstrass is an example [5] which demonstrates the way Analytica solves the summation problem.

Summation example in Analytica:

$$\sum_{n=0}^{\infty} b^n \cos(\pi a^n x) - (-1)^{\alpha}\Big(\sum_{n=m}^{\infty} b^n(1+\cos(\pi a^{-m+n}\xi(m)))\Big) - \sum_{n=0}^{\infty} b^n \cos(\pi a^{-m+n}(1+\alpha))$$
$$+ \sum_{n=0}^{m-1} b^n(-\cos(\pi a^n x) + \cos(\pi a^{-m+n}(1+\alpha))) = 0$$

simplify summations

$$-\Big((-1)^{\alpha}\Big(\sum_{n=m}^{\infty} b^n(1+\cos(\pi a^{-m+n}\xi(m)))\Big)\Big) + \sum_{n=0}^{\infty} b^n(\cos(\pi a^n x) - \cos(\pi a^{-m+n}(1+\alpha)))$$
$$+ \sum_{n=0}^{m-1} b^n(-\cos(\pi a^n x) + \cos(\pi a^{-m+n}(1+\alpha))) = 0$$

simplify summations

$$-\Big((-1)^{\alpha}\Big(\sum_{n=m}^{\infty} b^n(1+\cos(\pi a^{-m+n}\xi(m)))\Big)\Big) + \sum_{n=m}^{\infty} b^n(\cos(\pi a^n x) - \cos(\pi a^{-m+n}(1+\alpha)))$$
$$= 0$$

simplify summations

$$\sum_{n=m}^{\infty} (-(-1)^{\alpha} b^n(1+\cos(\pi a^{-m+n}\xi(m)))) + b^n(\cos(\pi a^n x) - \cos(\pi a^{-m+n}(1+\alpha))) = 0$$

reduces to

$$\sum_{n=m}^{\infty} (b^n(-\cos(\pi a^n x) + (-1)^{\alpha}(1+\cos(\pi a^{-m+n}\xi(m))) + \cos(\pi a^{-m+n}(1+\alpha)))) = 0$$

This can be simplified to **True** by trigonometric rules.

2.3.3 The Theorema System

A system for computer supported mathematical theorem proving and theory exploration. It is implemented in Mathematica system. In a non-interactive mode, Theorema solves the given reasoning problems automatically i.e. the inference rules are applied automatically and the user sees only the output. However in an interactive mode [18] the system is compelled to stop after every step of applying the inference rule to wait for a decision from the user which is quite similar to our user directed manipulation. In Theorema, a user interacts with three blocks of system components: reasoners, organization tools and libraries of mathematical knowledge. Basic building blocks of the system's reasoners are inference rules which are implemented as Mathematica functions that can be grouped into modules and then combined with reasoners. In a Theorema session reasoners which are basically proofs are accessed by the following call,

$$Reason[entity, using \rightarrow knowledge - base, by \rightarrow reasoner, options]$$

where Reason is one of Prove, Compute or Solve; entity is a mathematical entity to which Reason needs to be applied; knowledge-base is knowledge which needs to referred for reasoning; options are specific information which influence the behavior of reasoners. After this call is sent for evaluation to the Mathematica kernel, the system displays the set of inference rules to be applied to the next step and waits for the decision from the user to choose the rules listed or proceed with a different set of rules. Finally the output is in the form of a pretty-printed, textbook-style syntax.

2.3.4 Term Rewriting

A reasoning technique for manipulating large expressions by replacing sub-terms of the expressions with equivalent expressions giving the justification for simplification. This technique has been extensively used in computerized proof systems and computer algebra systems. In paper [6] they propose an interactive tool that makes it possible to

perform frequent rewrite operations using drag-and-drop paradigm. This tool applies only to certain class of rewrites where the operation can be intuitively interpreted as a movement of data. Consider the example where in we need to simplify the simple expression $(x+0)+(x+0)$ by rewriting. In the design of the editor proposed in [6], the user can select the second (rightmost) 0 and drag the mouse to the closest + sign (the third one) then the proof system generates and executes a command that makes a new goal appear, where the previous expression is replaced with $x+0+x$.

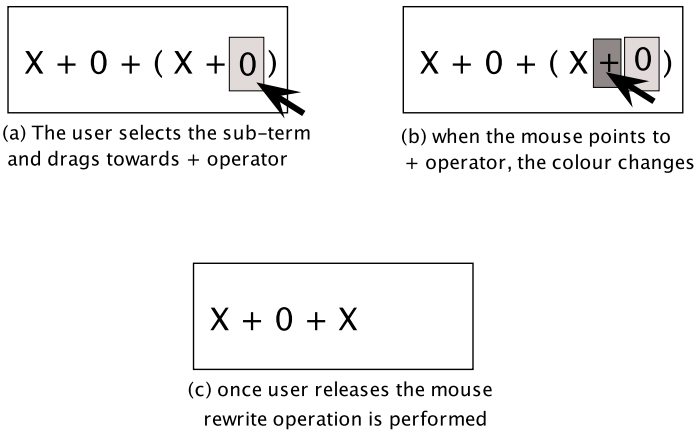

Figure 2.5: Algebraic manipulation in Term Rewriting

2.3.5 Summation in Finite Terms

This thesis studies the transformation and manipulation of formal sums. A related problem, that we do not study here, is finding closed-form expressions for summations. That is, given a formal sum, to find an equivalent expression that does not involve any summation operators. Computer algebra systems are generally quite good at this. In (2.35) we give an example of closed form of a summation.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{2.35}$$

There exist many algorithms which cover various classes of summands for computing closed forms of indefinite and definite sums. For example, Gosper's algorithm finds

closed form of hypergeometric identities. Michael Karr, in his paper[15] finds formulas for finite sums. The approach taken by him is to pose the problem in algebraic terms and then to derive constructive terms for summability. The Karr's summation algorithm has been implemented in Mathematica. Almost all computer algebra systems such as Maple, Mathematica, Macsyma can actually compute closed form of a sum. The SumTools package in Maple finds closed form of indefinite and definite sums. In our thesis, we are not computing closed form of sums, but we are manipulating summations.

Chapter 3

Summation

Summations play an important role in symbolic computation. Summation is an addition of a set of terms. The "terms" to be summed may be natural numbers, complex numbers, matrices, or still more complicated objects. In mathematics we have a special representation for addition of many similar terms. Greek letter $\sum$ (sigma) is used as a notation for such a summation. If $f(i)$ represents some function involving i, then this symbolic expression

$$\sum_{i=a}^{b} f(i) \tag{3.1}$$

is equivalent to the sum,

$$f(a) + f(a+1) + f(a+2) + ... + f(b-1) + f(b) \quad \text{when } a \leq b$$
$$-f(a+1) - f(a+2) - .... - f(b-1) \quad \text{when } a > b$$

$i \rightarrow$ index of summation which is always an integer.

$a \rightarrow$ lower limit of summation.

$b \rightarrow$ upper limit of summation.

This notation means evaluate the expression $f(i)$ of the summation for every value of i from the lower limit to the upper limit (inclusively) and add the results

together when the lower limit is less than or equal to the upper limit. We can explicitly expand this to a sum of $b - a + 1$ terms. The generalization to the case when the lower limit is greater than the upper limit is used by many authors, e.g.[15], and allows more natural identities. It means to find difference of all terms between lower and upper limit. When $a > b$ many authors take the sum to be zero. This, however, complicates many simple identities. Instead, we adopt the convention that the summands are negated, which avoids exceptions, as detailed below. It is hard in symbolic computation to define what simplest form of an expression means. In most of the computer algebra systems, the expressions are automatically simplified by applying algebraic transformations defined by axioms or rule groups. In our thesis we design and implement the manipulation of symbolic summation of the form as in (3.1). Automatic simplification is defined as a collection of algebraic transformations that is applied to an expression as a part of evaluation process. This is governed by rules (or rule groups). We use the same concept to transform and manipulate symbolic summation by axioms of summation. We will see these axioms in the following section.

3.1 Properties of Summation

In this section we introduce a number of rules to operate on summations. All these axioms are simple identities of summation.

I. If C is a constant in the summation and does not depend on k i.e. the iterating variable, we can factor out a constant i.e. C from the summation.

$$\sum_{k=s}^{n} Cf(k) = C \sum_{k=s}^{n} f(k)$$

II. If the inner expression of a summation has a sum (+) or difference (-) as the highest level operator, then the summation is equal to sum or difference of

individual summations of those inner expressions.

$$\sum_{k=s}^{n}\Big[f(k)\pm g(k)\Big]=\sum_{k=s}^{n}f(k)\pm\sum_{n=s}^{n}g(k)$$

III. This could be another condition of property (I) where the constant being multiplied is -1.

$$\sum_{n=s}^{t}-f(n)=-\sum_{n=s}^{t}f(n)$$

IV. We can split up a summation into several parts based on their limits as far as the value of indices are maintained. In (3.2) we split the summation with a range from s to n at jth term.

$$\sum_{k=s}^{n}f(k)=\sum_{k=s}^{j}f(k)+\sum_{k=j+1}^{n}f(k) \tag{3.2}$$

This is known as the "Chasles relation" in the French literature. This can have two conditions when $j\leq n$ and $j>n$. It is quite straightforward when $j\leq n$. But for $j>n$ consider this example,

$$s=1 \quad n=4 \quad j=6$$

$$\sum_{k=1}^{4}f(k) = \sum_{k=1}^{6}f(k)+\sum_{k=7}^{4}f(k) \tag{3.3}$$

$$\sum_{k=1}^{4}f(k) = f(1)+f(2)+f(3)+f(4) \tag{3.4}$$

$$\sum_{k=1}^{6}f(k) = f(1)+f(2)+f(3)+f(4)+f(5)+f(6) \tag{3.5}$$

$$\sum_{k=7}^{4}f(k) = -f(5)-f(6) \tag{3.6}$$

In (3.3), we split the summation at a value greater than the upper limit (i.e. $6>4$). To check whether the split is correct we get the actual terms in (3.4),

(3.5) and (3.6) and we can see that the split holds good because (3.4) is same as the sum of (3.5) and (3.6).

We can also split the summation based on even and odd terms. To find a formula for such a split is not straightforward especially when the lower limit is odd and upper limit is even and vice versa. Hence we devised the following formula which works for all conditions,

$$\sum_{k=a}^{b} A(k) \;=\; \sum_{k=\lceil \frac{a}{2} \rceil}^{\lfloor \frac{b}{2} \rfloor} A(2\,k) + \sum_{k=StartSplit}^{\lceil \frac{b}{2} \rceil} A\,(2\,k-1) \tag{3.7}$$

$$\text{where } StartSplit = \begin{cases} \lceil \frac{a}{2} \rceil & \text{if } a \text{ is odd} \\ \lceil \frac{a}{2} \rceil + 1 & \text{if } a \text{ is even} \end{cases}$$

Let us take an example and see how we can split the summation into even and odd terms,

$$a = 7 \quad b = 1$$

$$\sum_{k=7}^{1} A(k) = \sum_{n=4}^{0} A(2k) + \sum_{n=4}^{1} A(2k-1) \tag{3.8}$$

$$\sum_{k=4}^{0} A(2k) = -A(2) - A(4) - A(6) \tag{3.9}$$

$$\sum_{k=4}^{1} A(2k-1) = -A(3) - A(5) \tag{3.10}$$

$$\sum_{k=7}^{1} A(k) = -A(2) - A(3) - A(4) - A(5) - A(6) \tag{3.11}$$

Solving (3.7) for the values $a = 7$ and $b = 1$ we get (3.8). To check whether the split is correct we get the actual terms in (3.9), (3.10) and (3.11) and we can see that (3.11) is same as the sum of (3.9) and (3.10). We could also split a sum based on the congruence class of the index modulo any number, but the even and odd case is most common.

V. We can shift or rearrange the summation by any value p (as in our example). The summation remains unaltered as it is the same sequence of terms added because though we add p to the limits it is also being subtracted from the index variable.

$$\sum_{k=s}^{n} f(k) = \sum_{k=s+p}^{n+p} f(k-p)$$

VI. We can reverse the order of terms in a summation as shown below. This will rearrange the terms from upper limit to lower limit without altering the value.

$$\sum_{k=s}^{n} f(k) = \sum_{k=0}^{n-s} f(n-k)$$

Our simplification and transformation of formal sum is based on these identities but it is user directed simplification and not automatic.

Chapter 4

The Design Of A Software Package

There are numerous computer algebra systems, and these can be divided into general purpose and special purpose systems. Maple, Mathematica and Axiom are a few main general purpose software packages. CoCoA, SINGULAR and Macaulay are special purpose systems in the field of computational algebraic geometry. Both Maple and Mathematica provide a rich set of library functions to solve various kinds of problems in engineering applications and research.

While designing a computer algebra system, it is hard to build-in commands to anticipate each and every user requirement. Hence, except a few very specific application-oriented, computer algebra systems provide a programming language to write user specific commands. General purpose computer algebra systems provide a language to allow a user to deal with notation and semantics of programming as well as mathematics. We design and implement a software package in Maple 13.

4.1 Maple

Maple is one of the most powerful computer algebraic systems. It is user friendly, accurate and provides a high level programming language and provides two and three dimensional graphical output. It is a procedural programming language. It also has

a few functional programming constructs. Maple is dynamically typed. No typed declarations are required but types exist. Symbolic languages such as Maple can perform operations on unevaluated objects which is done by sequence of simplifications using built-in or user defined rules and assignments. The programmer has the control over the order and depth of evaluation of objects. Let us understand the internal organization of Maple.

4.1.1 Organization

Maple has three main components a kernel, a library and user interface. The kernel and library form the computational engine.

4.1.1.1 Kernel

The kernel written in C language is the core component of Maple. It performs fundamental or low level operations such as arbitrary precision arithmetic, execution of Maple language, file input and output etc. The primary responsibility is to interpret Maple language and represent data structures. It also saves the values of worksheet variables. While working on many worksheets in the same Maple session, it can be run in three different kernel modes shared, parallel and mixed modes. In shared mode, the variable assignments in one worksheet apply to all open worksheets. In parallel mode, variable assignment of one worksheet cannot be accessed in another. In mixed mode, the user is prompted to set the mode every time a new worksheet is open. Setting the mode can be done using the Options tab on the Tools menu. The compiled kernel of the system is about 100K bytes in size.

4.1.1.2 Library

The library contains most of the mathematical functionality which are written in Maple language. It consists of two parts main and packages. The main Maple library

contains the very frequently used commands and packages contain commands for very specific and related calculations. The library is stored in an archive and the kernel loads and interprets it on demand.

4.1.1.3 User Interface

Maple has two kinds of user interface: graphical and command-line. The former is for more interactive use while the latter could be used for batch processing and if we need to dedicate all the resources for computation.

4.1.2 Internal Functions

The evaluation of expressions are performed by internal functions in Maple. The internal functions are divided into five distinct groups which are as follows,

4.1.2.1 Evaluators

These are the main functions responsible for evaluation of expressions. There are six types of evaluations which are algebraic expression, boolean expression, name forming, statements, arbitrary precision and hardware floating point arithmetic. The statement evaluator is called after a user interface. After this a lot of interactions happen between the evaluators. Let us consider this example,

$$\text{if } c \geq 10 \text{ then } i := 10.2 * a \text{ end if;}$$

The statement evaluator first analyzes this statement and calls the boolean evaluator to resolve the if condition. If evaluation of the if statement is true then assignment of $i := 10.2 * a$ is performed by calling the statement evaluator again. This in turn calls the name forming evaluator to perform the left side of assignment and as the right side has a floating point number an arbitrary precision floating point evaluator is invoked.

4.1.2.2 Algebraic Functions

These are also called "basic" functions. These include finding the coefficients of polynomials (coeff), finding indeterminates (indets), mapping a function (map) etc.

4.1.2.3 Algebraic Service Functions

These functions serve the functions mentioned in the previous category which are also algebraic in nature. These cannot be directly called by the user. The examples include internal arithmetic packages, library functions retrieval etc.

4.1.2.4 Data Structure Manipulation Functions

These are algebraic functions which work on data structures instead of mathematical objects. The data structures include sums, products, expression sequences or lists. Examples include operand selection (op), operand substitution (subsop), searching (has) etc.

4.1.2.5 General Service Functions

These are general purpose functions and not very specific to numeric or symbolic computation. Examples could be storage allocation, table manipulation, garbage collection. These functions can be called by any other functions in the system as they are placed in the lowest hierarchical level.

4.1.3 Modules and Packages

The Maple system can be used interactively as an expression editor and computational procedures can be written using the high-level Maple programming language and libraries or packages can be added to the environment to extend capabilities. Procedures allow us to associate a sequence of commands with a single command.

Similarly modules associate related procedures and data.

Modules are a type of Maple expression which helps us to write generic algorithms, create packages or use Pascal-style records in programs. The use of modules satisfies the following important software engineering concepts,

- Encapsulation - It guarantees that an abstraction is used only according to its specified interface. It is possible to write significant software systems that are transportable and reusable. This makes code easier to understand and maintain important properties for large software systems.
- Packages - These gather a number of procedures that enable us to perform computations in some well-defined domain.
- Object Modeling - Objects are easily represented using modules. In object-oriented programming and software engineering an object is defined as something that has both state and behavior. Similarly with modules, we compute with objects by sending them messages to which they respond by performing services.
- Generic Programs - These accept objects that possess specific properties or behaviors. The underlying representation of the object is transparent to generic programs.

4.2 The Software Package: OpManipulate

Maple has a powerful ability to compute symbolic summations as closed form expressions, but its ability to do symbolic manipulations on the summation expressions themselves is limited. The 'Sum' command is a general purpose command to create summation. If the summation's value can be calculated then sum computes a closed form for the summation, else returns it unevaluated. A summation with symbolic

range will not be simplified. We restrict ourselves to finite sums. In this section we explain the need for designing these library functions but a detailed description for each of these library functions (calling sequence and examples) is given in Section 5.3.

There are numerous computations and mathematical problems involving summations. Here we take one example from [1] to demonstrate the need for designing and implementing a few library functions based on the summation properties. Here we intend to show the summation manipulations and please refer to [1] for mathematical aspects and terminologies. In statistics, if E[X] is the expected value of a random variable X, uniformly taking integer values 0, 1, ..., n, then E[X^2] may be calculated as follows:

$$\begin{aligned}
\mathrm{E}[X^2] &= \sum_{k=0}^{n} (k(k-1)+k)\binom{n}{k}p^k(1-p)^{n-k} && (4.1)\\
&= \sum_{k=0}^{n} k(k-1)\binom{n}{k}p^k(1-p)^{n-k} + \sum_{k=0}^{n} k\binom{n}{k}p^k(1-p)^{n-k} && (4.2)\\
&= \sum_{k=2}^{n} k(k-1)\binom{n}{k}p^k(1-p)^{n-k} + \sum_{k=1}^{n} k\binom{n}{k}p^k(1-p)^{n-k} && (4.3)\\
&= \sum_{k=2}^{n} n(n-1)\binom{n-2}{k-2}p^k(1-p)^{n-k} + \sum_{k=1}^{n} n\binom{n-1}{k-1}p^k(1-p)^{n-k} && (4.4)\\
&= \sum_{k=0}^{n-2} n(n-1)\binom{n-2}{k}p^{k+2}(1-p)^{n-2-k} + \sum_{k=0}^{n-1} n\binom{n-1}{k}p^{k+1}(1-p)^{n-1-k} && (4.5)\\
&= n(n-1)\sum_{k=0}^{n-2}\binom{n-2}{k}p^{k+2}(1-p)^{n-2-k} + n\sum_{k=0}^{n-1}\binom{n-1}{k}p^{k+1}(1-p)^{n-1-k} && (4.6)
\end{aligned}$$

The basic purpose for presenting this proof is to show the kinds of operations that our library functions must be able to perform. Such transformations are seen in most of the computations involving summations. Based on this example we list all the functions designed to implement the basic properties of summation which have

been explained in detail in Section 3.1.

SplitFunctions

To transform from (4.1) to (4.2) we split the summation based on the sum $(k(k-1)+k)$, which is the property II of summation. SplitFunctions function splits the summation based on the additive term in the inner expressions. This property is very common in most of the summation problems.

ShiftNTerms

While transforming from (4.4) to (4.5), we shift the first summation by 2 and second summation by 1 which is based on the property V of summation. This function shifts or rearranges the summation. Though index and the limit values change during this process, the value of the summation remains the same.

GetNonVariantsOutside

While transforming from (4.5) to (4.6), we get the expression $n(n-1)$ outside the first summation and n from the second summation as n is not dependent on the index variable which is the property I of summation. GetNonVariantsOutside function gets the variables and constants not dependent on index outside the summation.

SplitOp

In most of the examples involving summation we need to split the summation at a term that is property IV of summation. SplitOp function implements this property

as shown in the following example from (2.29).

$$\sum_{i=1}^{n-1} f(i) = \sum_{i=1}^{1} f(i) + \sum_{i=2}^{n-1} f(i)$$

ReverseOrder

This function implements property VI of summation.

In Table 4.1 we list all the functions designed and implemented based on summation properties. The function column shows the calling sequence of the functions and the second column lists the output of these functions for which the values of S, P and T are assigned.

Table 4.1: Library Functions - Implementing Properties

S=$\sum_{i=1}^{12} x^i f(i)$ P = $\sum_{i=a}^{b} (f(i) + g(i))x^i$ T=$\sum_{i=1}^{12} 45y^5 x^i f(i)$

Function	Output
SplitOp(S,2)	$\sum_{i=1}^{2} x^i f(i) + \sum_{i=3}^{12} x^i f(i)$
ShiftNTerms(S,5)	$\sum_{i=6}^{17} x^{i-5} f(i-5)$
ReverseOrder(S)	$\sum_{i=0}^{11} f(12-i)x^{12-i}$
SplitFunctions(P)	$\sum_{i=a}^{b} f(i)x^i + \sum_{i=a}^{b} g(i)x^i$
GetNonVariantsOutside(T)	$45y^5 \sum_{i=1}^{12} x^i f(i)$

We can represent a polynomial as a summation. The following is an univariate poly-

nomial in x,

$$\sum_{k=0}^{n} a_k x^k \tag{4.7}$$

GetCoefficient

As in (4.7) when the univariate polynomial is represented as a summation, then we can access the coefficient of any term of the polynomial using GetCoefficient function as an e.g. a_4 is the coefficient of x^4 term.

TakeNTermsHigh and TakeNTermsLow

There are a lot of manipulations of summations at the boundary values. In most of the polynomial manipulations we need the upper and lower terms hence these functions could be very handy to access the terms of summations towards the boundaries.

TakeTermsValue

Many a times while manipulating summation, we need to calculate the value of few terms or for all the terms as shown in the following example from (2.25).

$$\sum_{i=0}^{0} \big[f(i) + f(i+1)\big] = f(0) + f(1)$$

There is lot of need in various summation problems to calculate such values. TakeTermsValue calculates the value of the summation.

SplitTermsEvenOdd

This function segregates the summation into even and odd terms and implements (3.7).

MultiplyOp

This functions helps us to multiply the inner expression of the summation with a value and is very useful in few of the manipulations.

The above explained functions have been listed in Table 4.2 are those that cannot be related to any property of summation but we need them for various manipulations hence we call them helper functions. These are very handy while manipulating expressions and in performing computations.

Table 4.2: Library Functions - Helper Functions 1

S=$\sum_{i=1}^{12} x^i f(i)$ P = $\sum_{i=a}^{b} (f(i) + g(i))x^i$ T:=$45y^5 \sum_{i=1}^{12} x^i f(i)$

Function	Output
GetCoefficient(S,r,x)	$f(r)$
TakeTermsValue(S,2,3)	$x^2 f(2) + x^3 f(3)$
SplitTermsEvenOdd(S)	$\sum_{i=1}^{6} x^{2i} f(2i) + \sum_{i=2}^{6} x^{2i-1} f(2i-1)$
MultiplyOp(S, $45y^5$)	$\sum_{i=1}^{12} 45y^5 x^i f(i)$
TakeNTermsHigh(S,2)	$\sum_{i=11}^{12} x^i f(i)$
TakeNTermsLow(S,k)	$\sum_{i=1}^{k} f(i)x^i$
PutMultiplicandsInside(T)	$\sum_{i=1}^{12} 45y^5 x^i f(i)$

We now try to manipulate summations when they are operands of an expression. We pick this theorem from [9,3.8] to see the summation manipulations and please

refer to [9,3.8] for technical details.

$$\begin{aligned}
\frac{1}{\lambda_{k+1}^2} - \frac{1}{\lambda_k^2} &= 1 + \sum_{p=2}^{k} (-1)^p \binom{k}{p} (\alpha_0^2 ... \alpha_{p-2}^2) - \sum_{p=2}^{k} (-1)^p \binom{k+1}{p} (\alpha_0^2 ... \alpha_{p-2}^2) \\
&\quad -(-1)^{k+1} (\alpha_0^2 ... \alpha_{k-1}^2) \qquad (4.8) \\
&= 1 - \sum_{p=2}^{k} (-1)^p \left[\binom{k+1}{p} - \binom{k}{p} \right] (\alpha_0^2 ... \alpha_{p-2}^2) - (-1)^{k+1} (\alpha_0^2 ... \alpha_{k-1}^2) \\
&\qquad (4.9) \\
&= 1 + \sum_{p=2}^{k} (-1)^{p+1} \binom{k}{p-1} (\alpha_0^2 ... \alpha_{p-2}^k) + (-1)^k (\alpha_0^2 ... \alpha_{k-1}^2) \qquad (4.10) \\
&= 1 + \sum_{p=1}^{k} (-1)^p \binom{k}{p} (\alpha_0^2 ... \alpha_{p-1}^2) \qquad (4.11)
\end{aligned}$$

CombineSplitFunctions

To transform from (4.8) to (4.9), we combine the summations as they have the same range $p = 2 \,..\, k$. This is the reverse of Splitfunctions function. We implement this property as CombineSplitFunctions function.

CombineSplitOp

During the manipulation from (4.10) to (4.11) there exist the following intermediate steps,

$$\begin{aligned}
&= 1 + \sum_{p=2}^{k} (-1)^{p+1} \binom{k}{p-1} (\alpha_0^2 ... \alpha_{p-2}^k) + (-1)^k (\alpha_0^2 ... \alpha_{k-1}^2) \qquad (4.12) \\
&= 1 + \sum_{p=1}^{k-1} (-1)^p \binom{k}{p} (\alpha_0^2 ... \alpha_{p-1}^k) + \sum_{p=k}^{k} (-1)^p \binom{k}{p} (\alpha_0^2 ... \alpha_{p-1}^2) \qquad (4.13) \\
&= 1 + \sum_{p=1}^{k} (-1)^p \binom{k}{p} (\alpha_0^2 ... \alpha_{p-1}^2) \qquad (4.14)
\end{aligned}$$

The summation in the (4.12) is shifted by -1 and transformed to (4.13). We rewrite the expression $(-1)^k(\alpha_0^2...\alpha_{k-1}^2)$ as a summation, then the summation in (4.13) has been split at the kth term. In (4.14) we combine the summations split at a term. CombineSplitOp performs such an operation.

Table 4.3: Library Functions - Helper Functions 2

L=$\sum_{i=a}^{b} f(i)+\sum_{i=a}^{b} g(i)$ and M=$\sum_{i=a}^{b} f(i)x^i+\sum_{i=b+1}^{c} f(i)x^i$

Function	Output
CombineSplitFunctions(L)	$\sum_{i=a}^{b} f(i)+g(i)$
CombineSplitOp(M)	$\sum_{i=a}^{c} f(i)x^i$

We group these functions as Helper Functions 2 and are listed in Table 4.3. We implement the procedures as shown in Tables 4.1, 4.2 and 4.3 and put them all together in a module called "OpManipulate". These functions make use of the following internal functions implemented to access the parameters of summation. The functions basically manipulate the summation by accessing the operands using the op command in Maple.

- GetRange - This function returns the range value of the summation.
  If S=$\sum_{i=a}^{b} G(i)x^i$, then GetRange(S) returns $i = a$ .. b.
- GetRangeInitialValue - This function returns the lower limit of the summation.
  If S=$\sum_{i=a}^{b} G(i)x^i$, then GetRangeInitialValue(S) returns a.
- GetRangeFinalValue - This function returns the upper limit of the summation.
  If S=$\sum_{i=a}^{b} G(i)x^i$, then GetRangeFinalValue(S) returns b.

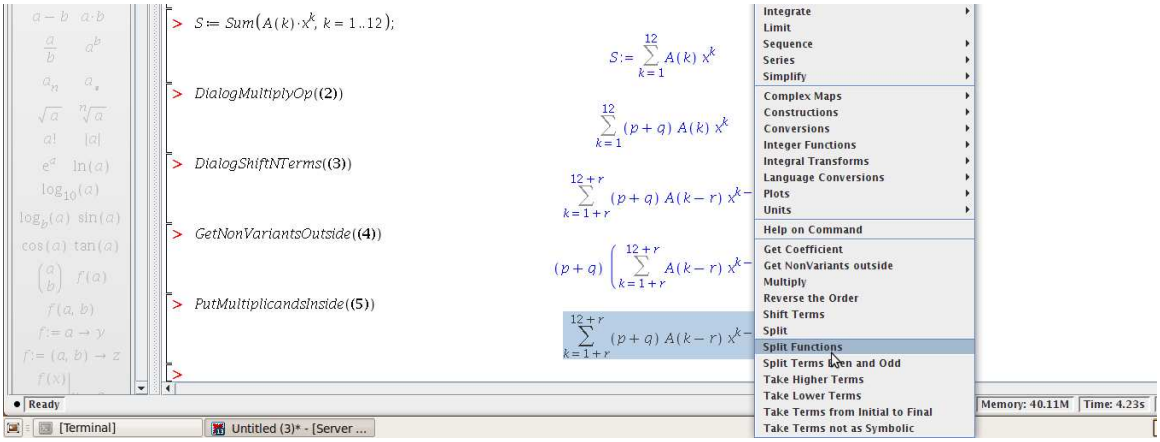

Figure 4.1: Library functions implemented as context-sensitive menus

- GetExpression - This function returns the inner expression or summand of the summation.

  If S=$\sum_{i=a}^{b} G(i)x^i$, then GetExpression(S) returns $G(i)x^i$.

- GetIteratingVariableName - This function returns the index of the summation.

  If S=$\sum_{i=a}^{b} G(i)x^i$, then GetIteratingVariableName(S) returns i.

- GetSymbolOperator - This function returns the operator whether sum, product or definite integral.

  If S=$\sum_{i=a}^{b} G(i)x^i$, then GetSymbolOperator(S) returns Sum.

Adding the package 'OpManipulate' to the installation

Depending on the installation, we may add our own modules to the standard library, or create new libraries in the directory specified by the command libname. Both of these are bad ideas. Adding a broken module to the standard library can make Maple non-functioning. We usually create a new library in the empty directory. These functions are also accessible as context-sensitive menus based on the operators being selected as shown in Figure 4.1.

Chapter 5

Generalization To Associative Operators

In this chapter we show how the design has been extended to other associative operators such as product and definite integral. To understand the design issues, we need to examine the properties of associative operators.

5.1 Associativity

In mathematics, associativity is the property that determines the way operators of the same precedence are grouped when there are no parenthesis.

Consider this expression $8+3+5 = 10$, it could be grouped to the left $(8+3)+5 = 16$ or right $8+(3+5) = 16$. Even though the parentheses were rearranged (the first expression requires adding 8 and 3 first, then adding 5 to the result, whereas the second expression requires adding 3 and 5 first, then 8), the value of the expression remains the same. This is true while adding any real numbers. We say operators with this property, for all arguments, are "associative".

More formally a binary operation $*$ on Set S is called associative if it satisfies the

associative law.

$$(a * b) * c = a * (b * c) \quad \text{for all} \quad a, b, c \, \epsilon \, S$$

If we apply this to multiplication the law holds

$$(ab)c = a(bc) \quad \text{for all} \quad a, b, c \, \epsilon \, S$$

The law can be generalized to when there are many $*$ operators. A binary operation $*$ on Set S is called non-associative if it does not satisfy the associative law.

$$(a * b) * c \neq a * (b * c) \quad \text{for some} \quad a, b, c \, \epsilon \, S$$

The order of evaluation does matter for these operators. Subtraction, division and exponentiation are non-associative which is shown in the following examples.

$$(8 \div 4) \div 2 \neq 8 \div (4 \div 2)$$
$$(9 - 2) - 3 \neq 9 - (2 - 3)$$

The associative law holds for summation, product and definite integral, hence they are all associative operators, when viewed in the right way. We also need to know the properties of the product and definite integral to design our library functions to be generic and work in these cases. We examine these properties in the following sections.

5.2 Mathematical Aspects

5.2.1 Definite Integral

Let f be a function which is continuous on a closed interval $[a, b]$. In mathematics, a continuous function is one in which a small change in the input results in small change in the output. The definite integral f from a to b is defined to be the limit,

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i) \triangle x$$

$$\text{where } \sum_{i=1}^{n} f(x_i) \triangle x \text{ is a Riemann Sum of } f \text{ on } [a, b].$$

A Riemann Sum is a method of approximating the total area underneath a curve on a graph otherwise known as an integral. The variable x which appears in the definition is dummy variable since the value of integral does not depend on x and it can be any other variable. There are two ways to think of a definite integral, one way would be to compute areas and volumes and the other way would be as a "sum".

5.2.1.1 Properties of Definite Integral

If $f(x)$ and $g(x)$ are defined and continuous on $[a, b]$, except maybe at a finite number of points, then we have the following linearity principle for the integral.

I. If C is a constant and does not depend on x, we can factor out a constant i.e. C from the definite integral. When f is a positive function, the height of Cf at a point x is C times the height of function f, hence the area under the curve $Cf(x)$ is C times the area under the curve of $f(x)$.

$$\int_a^b Cf(x)\,dx = C\int_a^b f(x)\,dx$$

II. If the inner expression of a definite integral has a sum or difference then the summation is equal to sum or difference of individual definite integrals of those inner expressions.

$$\int_a^b \big[f(x) \pm g(x)\big]\,dx = \int_a^b f(x)\,dx \pm \int_a^b g(x)\,dx$$

III. If the lower limit is equal to the upper limit the definite integral value is zero. As the base length of area bounded by $f(x)$ is zero, the area bounded by $f(x)$ is also zero.

$$\int_c^c f(x)dx = 0$$

IV. As definite integral of f on the interval $[a, b]$ is the area bounded by f and the x-axis between $x = a$ and $x = b$, then it is also true that this area is the sum of the area bounded between $x = a$ to $x = c$ and $x = c$ to $x = b$ where $x = c$ splits the region. This holds when the integral is defined as a connected region containing a, b and c.

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx$$

V. Reversing the limits will negate the value of the integral.

$$\int_a^b f(x)\,dx = -\int_b^a f(x)\,dx$$

VI. As long as the function and limits are the same the variable of integration that we use in the definite integral will not affect the answer. Hence the integral is independent of variable x or t.

$$\int_a^b f(x)\,dx = \int_a^b f(t)\,dt$$

VII. We can shift the integral by a value i.e. p. Doing so the value is unaltered as the integral is still $f(x)$ w.r.t x between the interval $[a, b]$.

$$\int_a^b f(x)\,dx = \int_{a+p}^{b+p} f(x-p)\,dx$$

5.2.2 Product

Product can be defined as multiplication of a sequence of similar terms. The "terms" to be multiplied may be natural numbers, complex numbers, matrices, or still more complicated objects. Greek symbol $\prod$ (Pi) notation is used to write complicated series of products in a compact way.

$$\prod_{i=a}^{b} f(i)$$

This symbolic expression is equivalent to the product,

$$f(a)f(a+1)f(a+2)....f(b) \quad \text{when } a \leq b$$

$$\frac{1}{f(a+1)f(a+2)...f(b-1)} \quad \text{when } a > b$$

$i \rightarrow$ index of product which is always an integer.

$a \rightarrow$ lower limit of product.

$b \rightarrow$ upper limit of product.

This notation means evaluate the expression $f(i)$ of the product for every value of i from the lower limit to the upper limit (inclusively) and multiply the results together when lower limit is less than or equal to upper limit. If lower limit is greater than the upper limit then it is the inverse of the product of $f(i)$ for every value of i between the limits. Most of the properties of summation hold true for product as well.

5.2.2.1 Properties of Product

I. We can shift the product by a value i.e. p. The product remains unaltered as it is the same sequence of terms multiplied because though we add p to the limits

it is also being subtracted from the index variable.

$$\prod_{k=s}^{n} f(k) = \prod_{k=s+p}^{n+p} f(k-p)$$

II. We can split a product into several parts as far as the value of indices are maintained. In (5.1) we split the product with range from $k = s \,..\, n$ at jth term.

$$\prod_{k=s}^{n} f(k) \;=\; \prod_{k=s}^{j} f(k) \prod_{k=j+1}^{t} f(k) \tag{5.1}$$

This can have two conditions when $j \leq n$ and $j > n$. It is quite straightforward when $j \leq n$, but for $j > n$ we generalize the product in the same way we did the summation, by considering it to be the inverse of the product on an inverted range. Consider this example,

$$s = 1 \quad n = 3 \quad j = 6$$

$$\prod_{k=1}^{3} G(k) \;=\; \prod_{k=1}^{6} G(k) \prod_{k=7}^{3} G(k) \tag{5.2}$$

$$\prod_{k=1}^{6} G(k) \;=\; G(1)G(2)G(3)G(4)G(5)G(6) \tag{5.3}$$

$$\prod_{k=7}^{3} G(k) \;=\; \frac{1}{G(4)G(5)G(6)} \tag{5.4}$$

$$\prod_{k=1}^{3} G(k) \;=\; G(1)G(2)G(3) \tag{5.5}$$

We split (5.2) at 6 which is for k greater than 4, the split holds good because (5.5) is same as the product of (5.3) and (5.4).

We can also split the terms of product on their even and odd indices. (5.6) has

been split for even and odd terms.

$$\prod_{k=a}^{b} A(k) \;=\; \prod_{k=\lceil \frac{a}{2} \rceil}^{\lfloor \frac{b}{2} \rfloor} A(2\,k) \prod_{k=StartSplit}^{\lceil \frac{b}{2} \rceil} A\,(2\,k-1) \tag{5.6}$$

$$\text{where } StartSplit = \begin{cases} \lceil \frac{a}{2} \rceil & \text{if } a \text{ is odd} \\ \lceil \frac{a}{2} \rceil + 1 & \text{if } a \text{ is even} \end{cases}$$

We can test (5.6) with the following values,

$$a = -11 \quad b = -8$$

$$\prod_{k=-11}^{-8} A(k) = \prod_{n=-5}^{-4} A(2k) \prod_{n=-5}^{-4} A(2k-1) \tag{5.7}$$

$$\prod_{k=-5}^{-4} A(2k) = A(-10)A(-8) \tag{5.8}$$

$$\prod_{k=-5}^{-4} A(2k-1) = A(-11)A(-9) \tag{5.9}$$

$$\prod_{k=-11}^{-8} A(k) = A(-11)A(-10)A(-9)A(-8) \tag{5.10}$$

For the given values of a and b, the split is as shown in (5.7), we can see that (5.10) is same as the product of (5.8) and (5.9). We had to take care of various conditions when the lower and upper limits are even and odd and vice versa, keeping this in mind (5.6) was devised.

III. If the inner expression of a product has a '$\times$' then the product is equal to multiplication of individual products of those inner expressions.

$$\prod_{k=s}^{n} \left[f(k)g(k) \right] = \prod_{k=s}^{n} f(k) \prod_{k=s}^{n} g(k)$$

IV. We can reverse the order of terms in a product as shown below. This will rearrange the terms from upper limit to lower limit without altering the value.

$$\prod_{k=s}^{n} f(k) = \prod_{k=0}^{n-s} f(n-k)$$

Stating the axioms of definite integral and product we can see that they are so much similar to formal sums. Hence we designed and implemented library functions to be more generic and also accommodated manipulations for product and definite integral other than summation. In the following section we see in detail the calling sequence and description for all the functions.

5.3 Software Aspects

In this section we explain each of the library function's calling sequence, parameters to be passed and few samples on a Maple session. The functions are written in a very generic way to accommodate similar operators. There are many computations involving definite integrals and product manipulations that can be solved with these functions. As these functions are accessible through context-sensitive menus, performing manipulations is much easier. All examples given in this paper are executed on Maple 13.

SplitOp

The **SplitOp** function splits the summation and product at a term and definite integral at an interval. This implements property IV of summation, property IV of definite integral and property II of product.

- Calling Sequence
  SplitOp(**expr**, **c**)
- Parameters
  expr - summation, product or definite integral
  c - value at which **expr** should be split
- Description

  - if **expr** is a summation then the split is at **c** th term and the second summation starts at **c**+1 th term.
  - if **expr** is a definite integral the split is at interval **c** and the second integral starts at interval **c**.
  - if **expr** is a product then the split is at **c** th term and the second product starts at **c**+1 th term.

- Maple Sample Output

```
> S:=Sum(f(i)*x^i,i=1..10);
```

$$S := \sum_{i=1}^{10} f(i)\, x^i$$

```
> SplitOp(S,3);
```

$$\sum_{i=1}^{3} f(i)\, x^i + \sum_{i=4}^{10} f(i)\, x^i$$

```
> T:=Product(A(k),k=a..b);
```

$$T := \prod_{k=a}^{b} A(k)$$

```
> SplitOp(T,c);
```

$$\prod_{k=a}^{c} A(k) \prod_{k=c+1}^{b} A(k)$$

```
> Q:=Int(x^2,x=1..12);
```

$$Q := \int_1^{12} x^2 dx$$

```
> SplitOp(Q,6);
```

$$\int_1^6 x^2 dx + \int_6^{12} x^2 dx$$

MultiplyOp

The **MultiplyOp** function exponentiates the base operator, "+" to "⋆", "⋆" to "∧". The multiplicand can contain the iterating variable or index of summation, product or definite integral.

- Calling Sequence
  MultiplyOp(**expr**, **c**)
- Parameters
  expr - summation, product or definite integral
  c - value to be multiplied.
- Description
  - multiplies inner expression of **expr** with **c**.
- Maple Sample Output

```
> S:=Sum(G(k),k=1..4);
```

$$S := \sum_{k=1}^{4} G(k)$$

> MultiplyOp(S,x^k);

$$\sum_{k=1}^{4} x^k G(k)$$

> P := Product(x[k], k = 1 .. 14);

$$P := \prod_{k=1}^{14} x_k$$

> MultiplyOp(P,c);

$$\prod_{k=1}^{14} {x_k}^c$$

> T:=Int(e^x,x=1..100);

$$T := \int_1^{100} e^x dx$$

> MultiplyOp(T,(a+b));

$$\int_1^{100} (a+b)\, e^x dx$$

SplitFunctions

The **SplitFunctions** function splits summation and definite integral based on the additive term and product based on the multiplicative term in the inner expression. This implements property II of summation, property II of definite integral and property III of product.

- Calling Sequence
  SplitFunctions(**expr**)
- Parameters
  expr- summation, product or definite integral
- Description
  - if the **expr** is a summation and the inner expression has a sum, then it splits **expr** for each operand of the inner sum.
  - if the **expr** is a definite integral and the inner expression has a sum, then it splits **expr** for each operand of the inner sum.
  - if the **expr** is a product and the inner expression has multiplication, then it splits **expr** for each operand of the inner expression.
- Maple Sample Output

> S:=Sum(x*(g(k)+h(k)),k=1..4);

$$S := \sum_{k=1}^{4} x\,(g(k) + h(k))$$

> SplitFunctions(S);

$$\sum_{k=1}^{4} xg\left(k\right)+\sum_{k=1}^{4} xh\left(k\right)$$

> P:=Product(E*F,i=a..b);

$$P := \prod_{i=a}^{b} EF$$

> SplitFunctions(P);

$$\prod_{i=a}^{b} E \prod_{i=a}^{b} F$$

> T:=Int(e^x+f^x,x=1..100);

$$T := \int_{1}^{100} e^{x}+f^{x}dx$$

> SplitFunctions(T);

$$\int_{1}^{100} e^{x}dx+\int_{1}^{100} f^{x}dx$$

TakeNTermsHigh

The **TakeNTermsHigh** function gets the higher terms of summation, product and definite integral.

- Calling Sequence
  TakeNTermsHigh(**expr**, **n**)
- Parameters
  expr - summation, product or definite integral
  n - number of terms to be fetched if **expr** is summation or product else the limit if **expr** is a definite integral
- Description
  - if **expr** is a summation or product then gets the **n** number of higher terms of **expr**.
  - if **expr** is a definite integral then returns the integral with interval [upper limit-**n**+1,upper limit].
  - **n** should be a positive integer if not it throws an exception.
- Maple Sample Output

> S:=Sum(x^j,j=a..b);

$$S := \sum_{j=a}^{b} x^{j}$$

> TakeNTermsHigh(S,2);

$$\sum_{j=b-1}^{b} x^{j}$$

> P:=Product(f(i)*x^i,i=1..12);

$$P := \prod_{i=1}^{12} f\left(i\right) x^{i}$$

> TakeNTermsHigh(P,1);

$$\prod_{i=12}^{12} f(i)\, x^i$$

> Q:=Int(x,x=1..4);

$$Q := \int_1^4 x dx$$

> TakeNTermsHigh(Q,4);

$$\int_1^4 x dx$$

TakeNTermsLow

The **TakeNTermsLow** function gets the lower terms of summation, product and definite integral.

- Calling Sequence
  TakeNTermsLow(**expr**, **n**)
- Parameters
  expr - summation, product or definite integral
  n - number of terms to be fetched if **expr** is summation or product else the limit if **expr** is a definite integral
- Description
  - if **expr** is a summation or product then gets the **n** number of lower terms of **expr**.
  - if **expr** is a definite integral then returns the integral with interval [lower limit,lower limit+**n**].
  - **n** should be a positive integer if not it throws an exception.
- Maple Sample Output

> S:=Sum(xˆj,j=a..b);

$$S := \sum_{j=a}^{b} x^j$$

> TakeNTermsLow(S,2);

$$\sum_{j=a}^{a+1} x^j$$

> P:=Product(f(i)*xˆi,i=1..12);

$$P := \prod_{i=1}^{12} f(i)\, x^i$$

> TakeNTermsLow(P,1);

$$\prod_{i=1}^{1} f(i)\, x^i$$

> Q:=Int(x,x=1..4);

$$Q := \int_1^4 x dx$$

> TakeNTermsLow(Q,4);

$$\int_1^4 x dx$$

ShiftNTerms

The **ShiftNTerms** function shifts or rearranges summation, product and definite integral. This implements property V of summation, property VII of definite integral and property I of product.

- Calling Sequence
  ShiftNTerms(**expr**, **n**)
- Parameters
  expr - summation, product or definite integral
  n - value by which **expr** should be shifted
- Description
  - if **expr** is summation then it shifts or rearranges the terms of **expr** by **n**, this is done by adding **n** to the lower and upper limit and subtracting the index variable in the inner expression by **n**.
  - if **expr** is a definite integral then it shifts the interval of **expr** by **n**, this is done by adding **n** to the lower and upper limit and subtracting the index variable in the inner expression by **n**.
  - if **expr** is product then it shifts or rearranges the terms of **expr** by **n**, this is done by adding **n** to the lower and upper limit and subtracting the index variable in the inner expression by **n**.
- Maple Sample Output

> S:=Sum(x^j,j=a..b);

$$S := \sum_{j=a}^{b} x^j$$

> ShiftNTerms(S,2);

$$\sum_{j=a+2}^{b+2} x^{j-2}$$

> P:=Product(f(i)*x^i,i=1..12);

$$P := \prod_{i=1}^{12} f(i)\, x^i$$

> ShiftNTerms(P,c);

$$\prod_{i=1+c}^{12+c} f(i-c)\, x^{i-c}$$

> Q:=Int(x,x=1..4);

$$Q := \int_1^4 x dx$$

> ShiftNTerms(Q,4);

$$\int_5^8 x - 4dx$$

SplitTermsEvenOdd

The **SplitTermsEvenOdd** function splits summation and product into even and odd terms.

- Calling Sequence
  SplitTermsEvenOdd(**expr**)
- Parameters
  expr- summation or product
- Description
  - splits the **expr** into even and odd terms as in (3.7) for summation and (5.6) for product.
- Maple Sample Output

  > S:=Sum(x^j,j=2..11);

  $$S := \sum_{j=2}^{11} x^j$$

  > SplitTermsEvenOdd(S);

  $$\sum_{j=1}^{5} x^{2\,j} + \sum_{j=2}^{6} x^{2\,j-1}$$

  > P:=Product(f(i)*x^i,i=1..12);

  $$P := \prod_{i=1}^{12} f\left(i\right) x^i$$

  > SplitTermsEvenOdd(P);

  $$\prod_{i=1}^{6} f\left(2\,i\right) x^{2\,i} \prod_{i=1}^{6} f\left(2\,i-1\right) x^{2\,i-1}$$

ReverseOrder

The **ReverseOrder** function reverses the order of terms for summation and product and intervals for definite integral. This implements property VI of summation, property V of definite integral and property IV of product.

- Calling Sequence
  ReverseOrder(**expr**)
- Parameters
  expr- summation, product or definite integral
- Description

  - if **expr** is summation then it reverses the order of terms.
  - if **expr** is a definite integral then it reverses the interval.
  - if **expr** is product then it reverses the order of terms.

- Maple Sample Output

```
> S:=Sum(x^j,j=2..11);
```

$$S := \sum_{j=2}^{11} x^j$$

```
> ReverseOrder(S);
```

$$\sum_{j=0}^{9} x^{11-j}$$

```
> P:=Product(f(i)*x^i,i=1..12);
```

$$P := \prod_{i=1}^{12} f(i)\, x^i$$

```
> ReverseOrder(P);
```

$$\prod_{i=0}^{11} f(12-i)\, x^{12-i}$$

```
> Q:=Int(x,x=1..3);
```

$$Q := \int_1^3 x dx$$

```
> ReverseOrder(Q)
```

$$-\int_3^1 x dx$$

GetCoefficient

The **GetCoefficient** function gets the coefficient of a term in a univariate polynomial which is represented as a summation.

- Calling Sequence
  GetCoefficient(**expr**,**k**,**invar**)
  GetCoefficient(**expr**,**invar**∧**k**)
- Parameters
  expr - summation
  k - coefficient for **k**th term
  invar - variable for which coefficient needs to be fetched
- Description
  - gets the coefficient of **invar**∧**k** when the summation represents a polynomial.
- Maple Sample Output

```
> S:=Sum(A(j)*x^j,j=2..11);
```

$$S := \sum_{j=2}^{11} A(j)\, x^j$$

> GetCoefficient(S,2,x);

$$A(2)$$

> GetCoefficient(S,x^2);

$$A(2)$$

TakeTermsValue

The **TakeTermsValue** function gets the terms of summation and product and value of the definite integral. If the input is a summation then it returns the closed form of summation if it can be evaluated else it returns the summation unevaluated.

- Calling Sequence
  TakeTermsValue(**expr**,**start**,**end**)
  TakeTermsValue(**expr**)
- Parameters
  expr - summation, product or definite integral
  start - lower limit of **expr** starting from which the terms or value need to be fetched
  end - upper limit of **expr** until which the terms or value need to be fetched
- Description
  - if **expr** is summation or product then gets the actual terms of **expr** starting from **start** to **end** (if **start** and **end** parameters are passed) else returns the terms with the lower and upper limits of **expr** (actual terms as an expression and not symbolic).
  - if **expr** is a definite integral then gets the value of **expr** starting from **start** to **end** (if **start** and **end** parameters are passed) else returns the value with the lower and upper limits of **expr** (actual value as an expression and not symbolic).
- Maple Sample Output

> S:=Sum(x^j,j=2..4);

$$S := \sum_{j=2}^{4} x^j$$

> TakeTermsValue(S);

$$x^2\left(1 + x + x^2\right)$$

> P:=Product(f(i)*x^i,i=a..b);

$$P := \prod_{i=a}^{b} f(i)\, x^i$$

> TakeTermsValue(P,1,1);

$$f(1)\,x$$

> Q:=Int(x,x=1..3);

$$Q := \int_1^3 x dx$$

> TakeTermsValue(Q);

$$4$$

CombineSplitFunctions

The **CombineSplitFunctions** function combines the summations and definite integrals split on additive terms and products split on multiplicative terms. This performs reverse operation as in case of **SplitFunctions** function.

- Calling Sequence
  CombineSplitFunctions(**expr**)
- Parameters
  expr - expression
- Description
  - if the highest level operator is '+', it scans through **expr** and combines all the summations which have the same range value.
  - if the highest level operator is '*', it scans through **expr** and combines all the products which have the same range value.
  - if the highest level operator is '+', it scans through **expr** and combines all the definite integrals which have the same interval.
- Maple Sample Output

> S:=Sum(x^j,j=2..4)+Sum(y^j,j=2..4)+Sum(z^j,j=2..4);

$$S := \sum_{j=2}^{4} x^j + \sum_{j=2}^{4} y^j + \sum_{j=2}^{4} z^j$$

> CombineSplitFunctions(S);

$$\sum_{j=2}^{4} x^j + y^j + z^j$$

> P:=Product(f(i)*x^i,i=a..b)*Product(g(i)*x^i,i=a..b);

$$P = \prod_{i=a}^{b} f(i)\, x^i \prod_{i=a}^{b} g(i)\, x^i$$

> CombineSplitFunctions(P);

$$\prod_{i=a}^{b} f(i) \left(x^i\right)^2 g(i)$$

> Q:=Int(x,x=1..3)+Int(e^x,x=1..3);

$$Q := \int_1^3 x dx + \int_1^3 e^x dx$$

> CombineSplitFunctions(Q);

$$\int_1^3 x + e^x dx$$

CombineSplitOp

The **CombineSplitOp** function combines summation and product split at a term and definite integral split at an interval. It performs reverse operation as in case of **SplitOp** function.

- Calling Sequence
  CombineSplitOp(**expr**)
- Parameters
  expr - expression
- Description
  - if the highest level operator is '+', it scans through **expr** and combines all the summations which are split at a term.
  - if the highest level operator is '*', it scans through **expr** and combines all the products which are split at a term.
  - if the highest level operator is '+', it scans through **expr** and combines all the definite integrals which are split at an interval.
- Maple Sample Output

> S:=Sum(xˆj,j=2..4)+Sum(xˆj,j=5..8);

$$S := \sum_{j=2}^{4} x^j + \sum_{j=5}^{8} x^j$$

> CombineSplitOp(S);

$$\sum_{j=2}^{8} x^j$$

> P:=Product(f(i)*xˆi,i=a..b)*Product(f(i)*xˆi,i=b+1..c);

$$P := \prod_{i=a}^{b} f(i)\, x^i \prod_{i=b+1}^{c} f(i)\, x^i$$

> CombineSplitOp(P);

$$\prod_{i=a}^{c} f(i)\, x^i$$

> Q:=Int(eˆx,x=1..3)+Int(eˆx,x=3..15);

$$Q := \int_1^3 e^x dx + \int_3^{15} e^x dx$$

> CombineSplitOp(Q);

$$\int_1^{15} e^x dx$$

GetNonVariantsOutside

The **GetNonVariantsOutside** function gets the constants and variables not dependent on iterating variable outside summation and definite integral.

- Calling Sequence
  GetNonVariantsOutside(**expr**)
- Parameters
  expr - summation or definite integral
- Description
  - gets the non variants (constants or variables not depending on the index variable) in the inner expression of **expr** outside.
- Maple Sample Output

  > S:=Sum(a^c*x^j,j=2..4);

  $$S := \sum_{j=2}^{4} a^c x^j$$

  > GetNonVariantsOutside(S);

  $$a^c \sum_{j=2}^{4} x^j$$

  > Q:=Int(45*y*x,x=1..3);

  $$Q := \int_1^3 45\, yx dx$$

  > GetNonVariantsOutside(Q);

  $$45y \int_1^3 x dx$$

PutMultiplicandsInside

The **PutMultiplicandsInside** function moves the multiplicands to the inner expression of summation and definite integral.

- Calling Sequence
  PutMultiplicandsInside(**expr**)
- Parameters
  expr - summation or definite integral
- Description
  - moves the outer variables and constants to the inner expression of **expr**.

- Maple Sample Output

  > S:=45*y*Sum(x^j,j=2..4);

  $$S := 45y \sum_{j=2}^{4} x^j$$

  > PutMultiplicandsInside(S);

  $$\sum_{j=2}^{4} 45\, yx^j$$

  > P:=e^x*Int(x,x=2..4);

  $$P := e^x \int_2^4 x dx$$

  > PutMultiplicandsInside(P);

  $$\int_2^4 e^x x dx$$

5.3.1 Testing Tool

An automated testing tool has been written in Maple to test these library functions. Basically a file has been created for each library function that contains the test cases. Each test case has the parameters to be passed for the library function and the expected output. The tool takes each file at a time and runs the corresponding library function with each test case and gets the actual output. If the actual output is the same as the expected output, then the output file is updated with 'OK' for the corresponding test case otherwise an 'Error'. This process is repeated for all the library functions. We now show an example of the input test case and the way the output file is updated,

An input test case for the CombineSplitFunctions function:

$$\underbrace{\text{Sum(A(i), i = 10.. - 10) + Sum(B(i), i = 10.. - 10)}}_{\text{Input Parameter}} \qquad \underbrace{\text{Sum(A(i) + B(i), i = 10.. - 10)}}_{\text{Expected Output}}$$

The output file is updated with the following row when the above test case is run:

$$\text{Sum(A(i), i = 10.. - 10) + Sum(B(i), i = 10.. - 10)} \quad \text{Sum(A(i) + B(i), i = 10.. - 10)} \quad \text{OK}$$

Chapter 6

Examples Using the New Package

6.1 Example: The Binomial Theorem

At the beginning of this thesis, in Example 2.1.1, we discussed how it was difficult to use a computer algebra system to transform expressions involving summations, illustrating with the manipulations for an explicit proof of the Binomial theorem. We designed and implemented the library functions for performing manipulations and transformations on expressions involving summations. Now let us use these functions to perform the manipulations. The functions here are called from the command line in Maple 13 worksheet which can also be called using context sensitive menus.

Solution as a Maple session:

First we assign the expression to be manipulated to XplusYraisetoN.

> XplusYraisetoN:=Sum(binomial(n, k)*xˆk*yˆ(n-k), k = 0 .. n);

$$XplusYraisetoN := \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$

We multiply XplusYraisetoN with (x+y) to prove the theorem for $n + 1$.

> S := (x+y)*XplusYraisetoN;

$$S := (x + y) \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$

We need to move $x + y$ to the summand of S. Now it is effortlessly done by calling PutMultiplicandsInside.

> S1 := PutMultiplicandsInside(S);

$$S1 := \sum_{k=0}^{n} (x + y) \binom{n}{k} x^k y^{n-k}$$

Then we split S1 based on the sum $(x + y)$ using SplitFunctions.

> SplitS1 := SplitFunctions(S1);

$$SplitS1 := \sum_{k=0}^{n} \binom{n}{k} x^{k+1} y^{n-k} + \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k+1}$$

To get the first operand of SplitS1 we make use of Maple's op.

> S2 := op(1, SplitS1);

$$S2 := \sum_{k=0}^{n} \binom{n}{k} x^{k+1} y^{n-k}$$

We need to shift S2 by 1, which is done by calling ShiftNTerms. While shifting S2 to ShiftS2 the value remains unchanged. This is because even if the lower bound k is increased by 1 the value of k in the inner expression is changed to $k - 1$. Hence S2 is same as ShiftS2.

> ShiftS2 := ShiftNTerms(S2, 1);

$$ShiftS2 := \sum_{k=1}^{n+1} \binom{n}{k-1} x^k y^{n-k+1}$$

We need to get the summation with range $k = 1 \,..\, n$. Hence ShiftS2 is split at nth term using SplitOp which also separates the $(n + 1)$ th term.

> SplitShiftS2 := SplitOp(ShiftS2, n);

$$SplitShiftS2 := \sum_{k=1}^{n} \binom{n}{k-1} x^k y^{n-k+1} + \sum_{k=n+1}^{n+1} \binom{n}{k-1} x^k y^{n-k+1}$$

> result1 := op(2, SplitShiftS2);

$$result1 := \sum_{k=n+1}^{n+1} \binom{n}{k-1} x^k y^{n-k+1}$$

Summation with range $k = n + 1 \,..\, n + 1$ is nothing but a term which has a value. To get this value we call TakeTermsValue and assign it to result1value.

> result1value := TakeTermsValue(result1);

$$result1value := x^{n+1}$$

The steps explained are also true when we transform S3 to result2value.

> S3 := op(2, SplitS1);

$$S3 := \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k+1}$$

> SplitS3 := SplitOp(S3, 0);

$$SplitS3 := \sum_{k=0}^{0} \binom{n}{k} x^k y^{n-k+1} + \sum_{k=1}^{n} \binom{n}{k} x^k y^{n-k+1}$$

> result2 := op(1, SplitS3);

$$result2 := \sum_{k=0}^{0} \binom{n}{k} x^k y^{n-k+1}$$

> result2value := TakeTermsValue(result2);

$$result2value := y^{n+1}$$

> result3 := op(1, SplitShiftS2)+op(2, SplitS3);

$$result3 := \sum_{k=1}^{n} \binom{n}{k-1} x^k y^{n-k+1} + \sum_{k=1}^{n} \binom{n}{k} x^k y^{n-k+1}$$

We combine the summations with the same range $k = 1 \,..\, n$ and form result3value using CombineSplitFunctions.

> result3value := CombineSplitFunctions(result3);

$$result3value := \sum_{k=1}^{n} x^k y^{n-k+1} \left(\binom{n}{k-1} + \binom{n}{k} \right)$$

The final output is the sum of result1value, result3value and result2value.

> result1value+result3value+result2value;

$$x^{n+1} + \sum_{k=1}^{n} x^k y^{n-k+1} \left(\binom{n}{k-1} + \binom{n}{k} \right) + y^{n+1}$$

We stop the transformations here and conclude that the manipulated expression is the same as the following expression.

$$x^{n+1} + \sum_{k=1}^{n} \binom{n+1}{k} x^k y^{n+1-k} + y^{n+1}$$

6.2 Example: Summation involving Stirling Numbers

We take Example 2.1.2 and perform symbolic summation manipulations using our library functions. We implement a function for falling powers ff(x,k) which performs the operation as shown below where x is any arbitrary expression and k is an integer.

$$ff(x,k) = x(x-1)(x-2)...(x-k+1)$$

Solution as a Maple session:

In Maple, Stirling numbers are implemented in package combinat.

```
> with(combinat);
```

[Chi, bell, binomial, cartprod, character, choose, composition, conjpart, decodepart, encodepart, eulerian1, eulerian2, fibonacci, firstpart, graycode, inttovec, lastpart, multinomial, nextpart, numbcomb, numbcomp, numbpart, numbperm, partition, permute, powerset, prevpart, randcomb, randpart, randperm, setpartition, stirling1, stirling2, subsets, vectoint]

We assign the expression to be transformed to S. We can observe that stirling2 gets automatically associated to combinat:-stirling2.

```
> S := Sum(stirling2(r, k)*ff(x,k), k = 0 .. r);
```

$$S := \sum_{k=0}^{r} \text{`combinat:-stirling2`}(r,k)\, ff\,(x,k)$$

Next we multiply S with x.

> S := x*S;

$$S := x \sum_{k=0}^{r} \text{`combinat:-stirling2`}(r,k)\, ff\,(x,k)$$

We move x to the inner expression of the S by calling PutMultiplicandsInside.

> S := PutMultiplicandsInside(S);

$$S := \sum_{k=0}^{r} x\,\text{`combinat:-stirling2`}(r,k)\, ff\,(x,k)$$

From (2.21), we know that xff(x,k)=ff(x,k+1)+xff(x,k). To substitute $xff\,(x,k)$, we call MultiplyOp passing the second parameter as (ff(x,k+1)+k*ff(x,k))/(x*ff(x,k)).

> S := MultiplyOp(S, (ff(x,k+1)+k*ff(x,k))/(x*ff(x,k)));

$$S := \sum_{k=0}^{r} (ff\,(x,k+1) + kff\,(x,k))\,\text{`combinat:-stirling2`}(r,k)$$

We perform a split based on the addition in the inner expression of the summation using SplitFunctions.

> S := SplitFunctions(S);

$$S := \sum_{k=0}^{r} \text{`combinat:-stirling2`}(r,k)\, ff\,(x,k+1) + \sum_{k=0}^{r} \text{`combinat:-stirling2`}(r,k)\, kff\,(x,k)$$

To get the first term of S we make use of Maple's op function and assign it to S1.

> S1 := op(1, S);

$$S1 := \sum_{k=0}^{r} \text{`combinat:-stirling2`}(r,k)\, ff\,(x,k+1)$$

We need to shift S1 by 1, which is done by calling ShiftNTerms function and assign it to ShiftS1. While shifting S1 to ShiftS1 the value remains unchanged. This is because even if the lower bound k is increased by 1 the value of $k+1$ in the inner expression is changed to k. Hence S1 is same as ShiftS1.

> ShiftS1 := ShiftNTerms(S1, 1);

$$ShiftS1 := \sum_{k=1}^{r+1} \text{‘combinat:-stirling2‘}\,(r, k-1)\,ff\,(x, k)$$

We need to get the summation with range $k = 1 \,..\, r$. Hence, ShiftS1 is split at rth term using SplitOp which also separates the $(r+1)$th term.

> SplitShiftS1 := SplitOp(ShiftS1, r);

$$SplitShiftS1 := \sum_{k=1}^{r} \text{‘combinat:-stirling2‘}\,(r, k-1)\,ff\,(x, k) + \sum_{k=r+1}^{r+1} \text{‘combinat:-stirling2‘}\,(r, k-1)\,ff\,(x, k)$$

Summation with range $k = r+1 \,..\, r+1$ is nothing but a term which has a value. To get this value we call TakeTermsValue and assign it to result1.

> result1 := TakeTermsValue(op(2, SplitShiftS1));

$$result1 := \text{‘combinat:-stirling2‘}\,(r, r)\,ff\,(x, r+1)$$

We substitute ‘combinat:-stirling2‘ (r, r) value as 1.

> result1:=ff(x,r+1);

$$result1 := ff\,(x, r+1)$$

The steps explained are similar when we transform S2 to result2.

> S2 := op(2, S);

$$S2 := \sum_{k=0}^{r} \text{‘combinat:-stirling2‘}\,(r, k)\,kff\,(x, k)$$

> SplitS2 := SplitOp(S2, 0);

$$SplitS2 := \sum_{k=0}^{0} \text{‘combinat:-stirling2‘}\,(r, k)\,kff\,(x, k) + \sum_{k=1}^{r} \text{‘combinat:-stirling2‘}\,(r, k)\,kff\,(x, k)$$

> result2 := TakeTermsValue(op(1, SplitS2));

$$result2 := 0$$

We combine the summations with the same range $k = 1 .. r$ and form result3 using CombineSplitFunctions.

> result3 := CombineSplitFunctions(op(2, SplitS2)+op(1, SplitShiftS1));

$$result3 :=$$
$$\sum_{k=1}^{r} ff\,(x,k)\,(\text{`combinat:-stirling2`}\,(r,k)\,k + \text{`combinat:-stirling2`}\,(r,k-1)\,)$$

The final output is the sum of result1, result2 and result3.

> result1+result2+result3;

$$ff\,(x,r+1) +$$
$$\sum_{k=1}^{r} ff\,(x,k)\,(\text{`combinat:-stirling2`}\,(r,k)\,k + \text{`combinat:-stirling2`}\,(r,k-1)\,)$$

Now we stop performing transformations using the library functions. We can further substitute the inner expression of summation and perform transformation as in (2.18) to (2.19). We can finally prove it for $r + 1$.

6.3 Example: Summation involving functions

In Example 2.1.3 we explained the difficulties of transforming (2.23) to (2.24) in a computer algebra system. Now we perform the transformations with the library functions on a Maple worksheet. There are different ways to transform the input expression to output, but we use the following way to demonstrate TakeNTermsHigh and TakeNTermsLow functions.

Input: $\displaystyle\sum_{i=0}^{n-1}[f(i)+f(i+1)] + \sum_{i=1}^{n}[g(i)+g(i+1)]$

Output: $\displaystyle f(0) + 2f(1) + g(1) + 2\sum_{i=2}^{n-1}\big[f(i)+g(i)\big] + f(n) + 2g(n) + g(n+1)$

Solution as a Maple session:

We first assign the expression to be transformed to Input.

> Input := Sum(f(i)+f(i+1), i = 0 .. n-1)+Sum(g(i)+g(i+1), i = 1 .. n);

$$Input := \sum_{i=0}^{n-1} f(i) + f(i+1) + \sum_{i=1}^{n} g(i) + g(i+1)$$

We can combine these summations if we get them in the same range i.e. $i = 1 \,..\, n-1$. To do this we get the first summation using Maple's op and assign it to Input_T1.

> Input_T1 := op(1, Input);

$$Input\_T1 := \sum_{i=0}^{n-1} f(i) + f(i+1)$$

Next we get n-1 number of higher terms using TakeNTermsHigh and assign to Input_H.

> Input_H := TakeNTermsHigh(Input_T1, n-1);

$$Input\_H := \sum_{i=1}^{n-1} f(i) + f(i+1)$$

We also assign the remaining terms of Input_T1 to Input_T1_L i.e. nothing but the zeroth term of Input_T1.

> Input_T1_L := TakeNTermsLow(Input_T1, 1);

$$Input\_T1\_L := \sum_{i=0}^{0} f(i) + f(i+1)$$

TakeTermsValue gets the value of Input_T1_L and sets it to Input_T1_L_val

> Input_T1_L_val:=TakeTermsValue(Input_T1_L);

$$Input\_T1\_L\_val := f(0) + f(1)$$

Next we perform similar transformations on second operand of Input until we get Input_T2_H_val.

> Input_T2 := op(2, Input);

$$Input\_T2 := \sum_{i=1}^{n} g(i) + g(i+1)$$

> Input_L := TakeNTermsLow(Input_T2, n-1);

$$Input\_L := \sum_{i=1}^{n-1} g(i) + g(i+1)$$

> Input_T2_H := TakeNTermsHigh(Input_T2, 1);

$$Input\_T2\_H := \sum_{i=n}^{n} g\left(i\right) + g\left(i+1\right)$$

> Input_T2_H_val:=TakeTermsValue(Input_T2_H);

$$Input\_T2\_H\_val := g\left(n\right) + g\left(1+n\right)$$

We split Input_H based on the additive term $f(i) + f(i+1)$ in the inner expression and assign it to Input_H_Split.

> Input_H_Split := SplitFunctions(Input_H);

$$Input\_H\_Split := \sum_{i=1}^{n-1} f\left(i\right) + \sum_{i=1}^{n-1} f\left(i+1\right)$$

Next we manipulate the first operand of Input_H_Split. We need to get the summations in the range $i = 2 \,..\, n-1$. To do this, we split the first operand of Input_H_Split at 1 and assign it to Input_H_SpSh1 by calling SplitOp.

> Input_H_SpSh1 := SplitOp(op(1, Input_H_Split), 1);

$$Input\_H\_SpSh1 := \sum_{i=1}^{1} f\left(i\right) + \sum_{i=2}^{n-1} f\left(i\right)$$

Then we assign Input_H_SpSh11Val with the value of first operand of Input_H_SpSh1 by calling TakeTermsValue.

> Input_H_SpSh11Val := TakeTermsValue(op(1, Input_H_SpSh1));

$$Input\_H\_SpSh11Val := f\left(1\right)$$

There is no summation in the output with summand containing $i+1$. Hence we need to remove these terms from the summation. We shift second operand of Input_H_Split by 1 using ShiftNTerms and assign it to Input_H_SpSh2.

> Input_H_SpSh2 := ShiftNTerms(op(2, Input_H_Split), 1);

$$Input\_H\_SpSh2 := \sum_{i=2}^{n} f\left(i\right)$$

We split Input_H_SpSh2 at n-1 and assign it to Input_H_SpSh2Split for the same reason i.e. to get them in range $i = 2 \,..\, n-1$.

> Input_H_SpSh2Split := SplitOp(Input_H_SpSh2, n-1);

$$Input\_H\_SpSh2Split := \sum_{i=2}^{n-1} f(i) + \sum_{i=n}^{n} f(i)$$

> Input_H_SpSh2Split2Val := TakeTermsValue(op(2, Input_H_SpSh2Split));

$$Input\_H\_SpSh2Split2Val := f(n)$$

Then similar steps are worked on Input_L until Input_L_SpSh2Split2Val.

> Input_L_Split := SplitFunctions(Input_L);

$$Input\_L\_Split := \sum_{i=1}^{n-1} g(i) + \sum_{i=1}^{n-1} g(i+1)$$

> Input_L_SpSh1 := SplitOp(op(1, Input_L_Split), 1);

$$Input\_L\_SpSh1 := \sum_{i=1}^{1} g(i) + \sum_{i=2}^{n-1} g(i)$$

> Input_L_SpSh11Val := TakeTermsValue(op(1, Input_L_SpSh1));

$$Input\_L\_SpSh11Val := g(1)$$

> Input_L_SpSh2 := ShiftNTerms(op(2, Input_L_Split), 1);

$$Input\_L\_SpSh2 := \sum_{i=2}^{n} g(i)$$

> Input_L_SpSh2Split := SplitOp(Input_L_SpSh2, n-1);

$$Input\_L\_SpSh2Split := \sum_{i=2}^{n-1} g(i) + \sum_{i=n}^{n} g(i)$$

> Input_L_SpSh2Split2Val := TakeTermsValue(op(2, Input_L_SpSh2Split));

$$Input\_L\_SpSh2Split2Val := g(n)$$

Input_Combine_H is obtained by adding the second operand of Input_H_SpSh1 and first operand of Input_H_SpSh2Split. The constant value 2 is moved inside the summation using PutMultiplicandsInside.

> Input_combine_H := PutMultiplicandsInside(op(2,Input_H_SpSh1)+op(1,Input_H_SpSh2Split));

$$Input\_combine\_H := \sum_{i=2}^{n-1} 2\, f\,(i)$$

Similarly, we get Input_Combine_L.

> Input_combine_L := PutMultiplicandsInside(op(2,Input_L_SpSh1)+op(1,Input_L_SpSh2Split));

$$Input\_combine\_L := \sum_{i=2}^{n-1} 2\, g\,(i)$$

Then we combine Input_Combine_H and Input_Combine_L using CombineSplitFunctions and assign it to combined_Input.

> combined_Input := CombineSplitFunctions(Input_combine_H+Input_combine_L);

$$combined\_Input := \sum_{i=2}^{n-1} 2\, f\,(i) + 2\, g\,(i)$$

Then we factor out 2 from the inner expression using GetNonVariantsOutside and assign it to Combined_Input_2.

> combined_Input_2 := GetNonVariantsOutside(combined_Input);

$$combined\_Input\_2 := 2 \sum_{i=2}^{n-1} f\,(i) + g\,(i)$$

The sum of Input_T1_L_val, Input_L_SpSh2Split2Val and Input_L_SpSh11Val is assigned to PartialOutput1.

> PartialOutput1 := Input_T1_L_val+Input_L_SpSh2Split2Val+Input_L_SpSh11Val

$$PartialOutput1 := f\,(0) + f\,(1) + g\,(n) + g\,(1)$$

Similarly, we assign the value for PartialOutput2.

> PartialOutput2 := Input_T2_H_val+Input_H_SpSh2Split2Val+Input_H_SpSh11Val;

$$PartialOutput2 := g\left(n\right) + g\left(1+n\right) + f\left(n\right) + f\left(1\right)$$

Then we add PartialOutput1, combined_Input_2 and PartialOutput2 to get the expected output.

> Output := PartialOutput1+combined_Input_2+PartialOutput2;

$$Output := f\left(0\right) + 2\,f\left(1\right) + g\left(n\right) + g\left(1+n\right) + 2\sum_{i=2}^{n-1} f\left(i\right) + g\left(i\right) + g\left(1\right) + f\left(n\right)$$

Chapter 7

Conclusion and Future Work

We have investigated and demonstrated how to perform transformations and manipulate symbolic summations. While doing so we have shed some light on current symbolic computation inabilities in computer algebra systems. We have shown how these ideas can be extended to develop a more general set of tools to manipulate a wider range of expressions.

We illustrated our work with some examples and solved them using our package. These transformations were based on a user's directions.

There are a number of interesting problems which would provide a path for future work. One of them would be to transform and manipulate a symbolic summation involving multinomial expansion in the inner expression.

Bibliography

[1] Binomial distribution. http://en.wikipedia.org/wiki/Binomial_distribution. (valid on June 16, 2010).

[2] Integre MathML Equation Editor. http://www.integretechpub.com/zed/. (valid on June 16, 2010).

[3] Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors. *Mathematical Knowledge Management, Third International Conference, MKM 2004, Bialowieza, Poland, September 19-21, 2004, Proceedings*, volume 3119 of *Lecture Notes in Computer Science*. Springer, 2004.

[4] R. Ausbrooks, S. Buswell, D. Carlisle, S. Dalmas, S. Devitt, A. Diaz, M.Froumentin, R. Hunter, P. Ion, M. Kohlhase, R. Miner, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, and S. Watt. Mathematical Markup Language (MathML) Version 2.0 (Second Edition). http://www.w3.org/TR/MathML2/. (valid on June 16, 2010).

[5] Andrej Bauer, Edmund M. Clarke, and Xudong Zhao. Analytica - an experiment in combining theorem proving and symbolic computation. *Journal of Automated Reasoning*, 21(3):295–325, 1998.

[6] Yves Bertot. Direct manipulation of algebraic formulae in interactive proof systems. In *In Electronic proceedings for the conference UITP'97, Sophia Antipolis*, pages 17–24.

[7] Laurent Dirat. JOME, a software component for interactive and distributed mathematics. *SIGSAM Bull.*, 34(2):38–42, 2000.

[8] Samuel S. Dooley. Editing mathematical content and presentation markup in interactive mathematical documents. In *ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 55–62, New York, NY, USA, 2002. ACM.

[9] George Exner, Il Bong Jung, and Chunji Li. On k-hyperexpansive operators. *Journal of Mathematical Analysis and Applications*, 323(1):569 – 582, 2006.

[10] Richard J. Fateman. Symbolic mathematics system evaluators. In *In Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, pages 86–94. ACM Press, 1996.

[11] Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[12] David Gleich. Finite calculus. www.stanford.edu/˜dgleich/publications/finite-calculus.pdf. (valid on June 16, 2010).

[13] Andre Heck. *Introduction to Maple*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[14] Norbert Kajler and Neil Soiffer. A survey of user interfaces for computer algebra systems. *Journal of Symbolic Computation*, 25(2):127–159, 1998.

[15] Michael Karr. Summation in finite terms. *J. ACM*, 28(2):305–350, 1981.

[16] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, J. McCarron, and P. DeMarco. *Maple 10 Advanced Programming Guide*. Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario, Canada, 2005. 442 pages.

[17] Luca Padovani and Riccardo Solmi. An investigation on the dynamics of direct-manipulation editors for mathematics. In Asperti et al. [3], pages 302–316.

[18] Florina Piroi and Temur Kutsia. The theorema environment for interactive proof development. In Sutcliffe and Voronkov [20], pages 261–275.

[19] Wei Su, Paul. Wang, Lian Li, Guanyu Li, and Yanjuan Zhao. A browser-based visual mathematics expression editor. Technical report, 2006.

[20] Geoff Sutcliffe and Andrei Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*. Springer, 2005.

[21] Irène Vatton, Laurent Carcone, and Vincent Quint. Amaya W3C'S Editor/Browser. http://www.w3.org/Amaya/. (valid on June 16, 2010).

Curriculum Vitae

Name:	**Nivedita Patil**
Post-Secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada M.Sc. Computer Science, December 2009 S.D.M College of Engineering & Technology (SDMCET) Dharwad, India B.E. in Computer Science and Engineering Sept. 1999
Work Experience:	Research Assistant, Teaching Assistant. University of western Ontario, London, Canada. Sept. 2008 - August 2009 Programmer Analyst eBRP Solutions, Toronto, Canada. October 2006 - August 2008 Senior Software Engineer Hewlett Packard., Bangalore , India. Oct 2000 - Dec 2005 Software Engineer Verifone Private Limited., Bangalore , India. Oct 1999 - Oct 2000