The ColourMaths` package

Introduction

The Colour Maths package was developed to solve a **recurring** problem among *Mathematica* users – how to specify arbitrary algebraic transformations as elegantly and reliably as possible. Using this package a beginner user of *Mathematica* can specify complicated algebraic transformations with little or no grasp of functions such as Collect, Together, Part, etc. Furthermore, all the details of algebraic manipulations stay in the notebook for future reference and there is a much reduced chance of making incorrect algebraic operations. Version 2.3 contains mechanisms to enable a user to extend the scope of the package for his/her domain, and it contains a powerful mechanism to refer back to particular formulae (e.g. theorems) earlier in the notebook.

■ Getting started

ColourMaths is supplied as a ZIP file containing all the files required in their appropriate directories .. It is vital that this directory structure is preserved. Copy the file either to the directory specified by $BaseDirectory, or $UserBaseDirectory (as a general rule, use the former if you are the sole user of the computer, use the latter if several people log on to your machine). Unzip the file ColourMaths.ZIP using a tool that preserves the directory structure and handles long names correctly, e.g. PKZIP (R) Version 2.50, or WinZip (R).

As of version 2.20 of ColourMaths, the documentation is fully accessible via the Documentation Center.

Once the package is in place it can be read In the usual way:

```
<< ColourMaths`
```

ColourMaths will create a palette immediately it is read in. Using this palette you can specify algebraic manipulations using colour. Simply select a whole subexpression and press one of the three coloured square buttons (red, green, blue) at the top of the palette. If you selected an invalid subexpression you will receive a beep, otherwise you will produce a coloured expression. It is possible to colour any number of subexpressions with the same or different colours. To remove a colour simply make a selection which includes the colour you wish to remove and hit the black button (other methods will be explained later).

Hint: To ensure that you select a whole subexpression click repeatedly on the expression you wish to select. *Mathematica* will select successively larger subexpressions until you reach the one you require.

The colour acts as a tag to enable arbitrary algebraic transformations to be specified. Using ColourMaths you should be able to avoid re-typing expressions in almost all situations - avoiding the mistakes that this can produce. Your notebooks will also contain a full record of your work.

The palette also contains some of the colour manipulation functions described below. These paste at the end of the cell, since that is usually whee they are required. To avoid creating a gigantic palette, not all functions or options can be accessed in this way. For example, the RR function pastes on the assumption that you will apply it to a rule rather than a function (although you can, of course, edit the pasted code to change it). In general there is no substitute for reading the remaining sections of this manual!

■ What is a coloured expression?

Although it is possible to colour text in a notebook using the Format menu, this colour is not transmitted to the Kernel and is purely decorative. When you select a subexpression and press one of the colour buttons something quite different

occurs. The subexpression is surrounded by a function 'Red⌣Colour', 'Green⌣Colour', or 'Blue⌣Colour' as appropriate. You can see this function by examining the FullForm:

```
a + b + c // FullForm
```

```
Plus[a, c, Red\[Breve]Colour[b]]
```

The colour package programs the FrontEnd so that it normally displays these colour functions by colouring their arguments. Once you understand the structure of coloured expressions you can even manipulate them directly:

```
a + b + c /. Red⌣Colour[x_] -> Green⌣Colour[2 x]
```

```
a + c + 2 b
```

Because *Mathematica* does not 'know' anything about these functions except how to display them, they can be useful for preventing unwanted evaluation. For example, consider the following expression:

```
π / (E^(I π x) - E^(-I π x)) // ExpToTrig
```

$$-\frac{1}{2}\,i\,\pi\,\text{Csc}[\pi\,x]$$

Perhaps you feel more comfortable with the sine function rather than the cosecant function (reciprocal sine). By using colour we can prevent *Mathematica* performing this transformation:

```
π / (e^(i π x) - e^(-i π x)) // ExpToTrig
```

$$\frac{\pi}{-e^{-i\,\pi\,x}+e^{i\,\pi\,x}}$$

In larger algebraic calculations in which a long sequence of colour operations are applied this concept of preventing evaluations can become vital.

■ The colour manipulation functions

Once portions of an expression have been tagged they are usually manipulated using the colour manipulation functions provided by ColourMaths. Because these functions are intended to be used frequently, and in combination, they are given a short name as well as a more traditional *Mathematica* name. Those functions which remove the colour after doing their work have a full name which ends in 'AndClear', and a short form which ends in the digit '1'. These functions are known as 'non-colour preserving variants'.

Function name	*Abbreviation*	*Action*
RedOperation	RR	Restricts functions/rules to red expressions
RedOperationAndClear	RR1	Restricts functions/rules to red expressions, and removes colour
GreenOperation	GG	Restricts functions/rules to green expressions
GreenOperationAndClear	GG1	Restricts functions/rules to green expressions, and removes colour
BlueOperation	BB	Restricts functions/rules to blue expressions
BlueOperationAndClear	BB1	Restricts functions/rules to blue expressions, and removes colour

RedGrow	Rgrow	Expands the red colouration to cover the containing expression
GreenGrow	Ggrow	Expands the green colouration to cover the containing expression
BlueGrow	Bgrow	Expands the blue colouration to cover the containing expression
CoalesceRed	RC	Coalesces red parts of an expression
CoalesceRedAndClear	RC1	Coalesces red parts of an expression, and removes colour
CoalesceGreen	GC	Coalesces green parts of an expression
CoalesceGreenAndClear	GC1	Coalesces green parts of an expression, and removes colour
CoalesceBlue	BC	Coalesces blue parts of an expression
CoalesceBlueAndClear	BC1	Coalesces blue parts of an expression, and removes colour
MoveRedOut	Rout	Moves red expression out one layer
MoveRedOutAndClear	Rout1	Moves red expression out one layer, and removes colour
MoveGreenOut	Gout	Moves green expression out one layer
MoveGreenOutAndClear	Gout1	Moves green expression out one layer, and removes colour
MoveBlueOut	Bout	Moves blue expression out one layer
MoveBlueOutAndClear	Bout1	Moves blue expression out one layer, and removes colour
MoveRedOver	Rover	Moves red expression over == and inequalities
MoveRedOverAndClear	Rover1	Moves red expression over == and inequalities, and removes colour
MoveGreenOver	Gover	Moves green expression over == and inequalities
MoveGreenOverAndClear	Gover1	Moves green expression over == and inequalities, and removes colour
MoveBlueOver	Bover	Moves blue expression over == and inequalities
MoveBlueOverAndClear	Bover1	Moves blue expression over == and inequalities, and removes colour
MoveRedIn	Rin	Moves red expression inside another
MoveRedInAndClear	Rin1	Moves red expression inside another and removes the colour
MoveGreenIn	Gin	Moves green expression inside another
MoveGreenInAndClear	Gin1	Moves green expression inside another and removes the colour
MoveBlueIn	Bin	Moves blue expression inside another
MoveBlueInAndClear	Bin1	Moves blue expression inside another and removes the colour
MoveRedInsideRed	RRin	Moves red expression inside red one

MoveRedInsideRedAndClear	RRin1	Moves red expression inside red one and remove the colour
MoveRedInsideGreen	RGin	Moves red expression inside green one
MoveRedInsideGreenAndClear	RGin1	Moves red expression inside green one and remove the colour
MoveRedInsideBlue	RBin	Moves red expression inside blue one
MoveRedInsideBlueAndClear	RBin1	Moves red expression inside blue one and remove the colour
MoveGreenInsideRed	GRin	Moves green expression inside red one
MoveGreenInsideRedAndClear	GRin1	Moves green expression inside red one and remove the colour
MoveGreenInsideGreen	GGin	Moves green expression inside green one
MoveGreenInsideGreenAndClear	GGin1	Moves green expression inside green one and remove the colour
MoveGreenInsideBlue	GBin	Moves green expression inside blue one and remove the colour
MoveGreenInsideBlueAndClear	GBin1	Moves green expression inside blue one
MoveBlueInsideRed	BRin	Moves blue expression inside red one
MoveBlueInsideRedAndClear	BRin1	Moves blue expression inside red one and remove the colour
MoveBlueInsideGreen	BGin	Moves blue expression inside green one
MoveBlueInsideGreenAndClear	BGin1	Moves blue expression inside green one and remove the colour
MoveBlueInsideBlue	BBin	Moves blue expression inside blue one
MoveBlueInsideBlueAndClear	BBin1	Moves blue expression inside blue one and remove the colour
RedAdd	Radd	Adds expr – Red⌣Colour[expr] to the red expression
GreenAdd	Gadd	Adds expr – Green⌣Colour[expr] to the green expression
BlueAdd	Badd	Adds expr – Blue⌣Colour[expr] to the blue expression
RedMultiply	Rmpy	Multiplies the red expression by expr Red⌣Colour$\left[\text{expr}^{-1}\right]$
GreenMultiply	Gmpy	Multiplies the green expression by expr Gree$\left[\text{⌣Colour}\left[\text{expr}^{-1}\right]\right.$
BlueMultiply	Bmpy	Multiplies the blue expression by expr Blue⌣Colour$\left[\text{expr}^{-1}\right]$
ColourSort	CS	CS[f] returns *a* function that sorts coloured parameters of f. CS[] returns *a* function that colour sorts lists.
ColourSortAndClear	CS1	CS1 [f] and CS1[] operate as CS[f] and CS[] except that all colour is removed after the operation completes.

The following functions are used to manipulate inert sums, integrals, products, etc. and to define and use assertions. Many of them are frequently used inside colour functions such as RR1 to restrict their action.

Function name	*No of arguments*	*Action*
DoIntegrals	**0**	***A* function that converts inactive integrals into the form recognised by Mathematica.**
DoSummations	**0**	***A* function that converts inactive summations into the form recognised by Mathematica.**
SwapOperations	**0**	***A* function that swaps nested inert summations / integrals.**
ByParts	**0 or 1**	**xxxx**
ChangeVariables	**2**	**Returns *a* function that performs *a* variable substitution on an inert integral.**
DoSpecialFunctions	**0**	**Returns *a* function that causes the inert forms of special functions to be replaced by the Mathematica equivalent.**

■ Selective application of rules and functions

Using colour it is possible to restrict the application of a rule, rule set, or function to just the coloured parts of an expression. This is done using the functions RR, GG, BB or their non colour preserving variants RR1, GG1, BB1. Here is an example in which the built-in function TrigToExp is applied selectively to the red subexpression:

```
Cos[a x] + Cos[b x] + Cos[c x] // RR[TrigToExp]
```

$$\text{Cos}[a\,x] + \text{Cos}[c\,x] + \left(\frac{1}{2}\,e^{-i\,b\,x} + \frac{1}{2}\,e^{i\,b\,x}\right)$$

Here s an example using a rule. Note carefully that RR always returns a pure function whatever its argument(s), and so should be preceded by the '//' operator:

```
Cos[a x] + Cos[b x] + Cos[c x] // RR[x -> y]
```

```
Cos[a x] + Cos[c x] + Cos[b y]
```

These functions can take multiple arguments, which are used in turn, so the previous two transformations could be combined thus:

```
Cos[a x] + Cos[b x] + Cos[c x] // RR1[TrigToExp, x -> y]
```

$$\frac{1}{2}\,e^{-i\,b\,y} + \frac{1}{2}\,e^{i\,b\,y} + \text{Cos}[a\,x] + \text{Cos}[c\,x]$$

Since we have presumably finished with the red colour at this point, we have used the non colour preserving variant RR1.

Notice that there is no reason why the function or rule need represent a mathematical identity, so these functions can be used to perform many kinds of operations. As just one example, suppose we are manipulating special functions and decide to use a private notation except when *Mathematica* evaluation is required. We might define a rule set such as this:

```
myrule := {Γ[x_] -> Gamma[x], ζ[z_] -> Zeta[z]}
```

Using colour we could apply this rule set exactly where required:

```
Γ[a] Γ[4] // RR1[myrule]
---------
   Γ[c]
```

```
6 Γ[a]
------
 Γ[c]
```

■ Collecting coloured expressions

Consider the following expression:

```
a^2 + b^2 - c^2 - d^2
```

Clearly, from a mathematical point of view $a^2 - d^2$ is a subexpression. However, because *Mathematica* has not printed these parts of the expression together, there is no way to select, and hence colour this subexpression. The solution is to use the RC (RedCoalesce), GC, and BC functions. These functions will force terms together thus:

```
a^2 + b^2 - c^2 - d^2 // RC
```

```
b^2 - c^2 + (a^2 - d^2)
```

Typically these functions are combined with one of the other colour manipulation functions, for example:

```
a^2 + b^2 - c^2 - d^2 // RC // RR1[Factor]
```

```
b^2 - c^2 + (a - d) (a + d)
```

Without coalescing the two red expressions Factor gets applied to them separately and achieves nothing.

```
a^2 + b^2 - c^2 - d^2 // RR1[Factor]
```

```
a^2 + b^2 - c^2 - d^2
```

Although non colour preserving versions of these functions exist, they are not often used because in many instances (such as the above) without the colour the expression collapses to its original form.

Rather than merge two coloured expressions, it is sometimes useful to expand a coloured region to cover the immediate containing expression. This can be done with the functions Rgrow, Ggrow, and Bgrow. Possibly the most application of these functions is to colour negative components of additions. For example:

```
a - b // FullForm
```

```
Plus[a, Times[-1, Red\[Breve]Colour[b]]]
```

```
a - b // Rgrow // FullForm
```

```
Plus[a, Red\[Breve]Colour[Times[-1, b]]]
```

Without this device there would be no way to pick out the subexpression (-b) as opposed to the subexpression b.

■ Moving expressions 'out'

■ Moving expressions 'in'

Although it is common to talk about moving expressions inside others, the concept can be rather more ambiguous than the reverse operation. Colour Math supports several such operations to deal with the different cases. You can always use

the most general form, which will work in all cases, if desired.

The simplest case is where the 'move in' operation is totally unambiguous, for example:

```
√a √b // Rin
```

$$\sqrt{a\ b}$$

```
(a + b) c d // Rin1
```

```
(a c + b c) d
```

An intermediate case exists in which it suffices to mark two subexpressions with the same colour to specify a 'move in' operation. For example:

```
Log[a] + Log[b] + Log[c] + Log[d] // RRin1
```

```
Log[a] + Log[c] + Log[b d]
```

In the most general case it is necessary to use two distinct colours:

```
a + b (c + d) e + f // RBin1
```

```
(c + d) ((a + b) e + (a + b) f)
```

Here are a few more examples:

```
s ∑_{i=0}^{∞} f[i] // Rin
```

$$\sum_{i=0}^{\infty} f[i]\ s$$

```
a b (c^2 + d^2) // Rin1
```

```
a (b c^2 + b d^2)
```

■ Moving expressions 'over'

The functions Rover, Gover, and Bover move subexpressions to the other side of equalities and inequalities. The precise meaning depends on the context. Note that it is easy to do invalid things with inequalities - such as dividing both sides by something negative - so it is vital not to use these functions blindly. The manipulation of equalities is much safer. Here are some examples:

```
x^2 + y^2 - a == 0 // Bover
```

```
x^2 + y^2 == a
```

$$\frac{(x + y)}{x - y} == p \text{ // Rover // Rin1}$$

```
x + y == p (x - y)
```

```
a b + b c d == e // Gover1
```

$$a + c\,d == \frac{e}{b}$$

■ Multiply top and bottom by ...

Frequently it is useful to manipulate an expression by adding and subtracting the same term, or by multiplying and dividing the same term. The result is unchanged of course, but the aim is to combine this maneuver with other operations to achieve something useful. You will encounter two problems if you try to perform such operations with 'raw' *Mathematica*:

• You may want to perform the manipulation deep inside an expression, not on the top level.

• Until you have performed some further operations *Mathematica* is quite likely to 'simplify' your expression back to its original form.

To solve these problems a set of six operations are supplied which operate as in the following example:

```
f[a, b] // Rmpy[2] // Gadd[n]
```

$$f\left[-n + (a + n),\ \frac{2\,b}{2}\right]$$

The desired operation is performed inside the coloured expression, and the inverse operation is applied outside the colour (so that the value of the expression, disregarding colour, remains the same). Since the result would at once collapse if the colour were removed before other transformations were applied, no non colour preserving version of these functions is supplied.

Here is a more interesting example:

```
(1 + q + q^2 + q^3) (1 + q + q^2) (1 + q) // Rmpy[1 - q] // Gmpy[1 - q] // Bmpy[1 - q] //
   RR1[Expand] // GG1[Expand] // BB1[Expand]
```

Notice how the coloured product or sum is ready for another colour manipulation operation, in this case Expand.

■ Extracting expressions to another notebook

The philosophy of using coloured expressions is perform as much as possible without re-writing or pasting operations, which do not leave any record in the notebook. However, sometimes it is convenient to extract part of an expression and manipulate it on its own in a separate notebook, pasting the result back in at the end of the process. If you do this you must accept that your notebook will not contain a full record of your manipulations.

To extract a subexpression first colour it in the usual way and then press the large 'X' on the palette of the appropriate colour. Note that there are no functions corresponding to these palette items. A new notebook will be created with the subexpression pasted into it. This notebook will also contain a button marked 'Return' which you should press to return the last result that you produce. Do not press this button until you have evaluated an expression of some sort.

When you press the 'Return' button the temporary notebook will close and the result will be pasted back into your full expression.

■ Functions with many parameters

Certain mathematical functions, such as the (generalised) hypergeometric functions and the MeijerG function depend on many parameters, which are arranged in lists. Consider for example the function HypergeometricPFQ[{3/2,2},{1,2},z]. Here we have two lists of parameters, and it is a basic fact – obvious from the definition of these functions – that the order of parameters within each list is irrelevant. Suppose now that you wish to apply a transformation rule to a hypergeo-

metric function, but the order of the parameters in the lists does not match those in the rule. Because *Mathematica* applies transformation rules in an essentially blind way it is necessary to sort the order of parameters before applying such a rule (Alternatively you could define many rules to cover each parameter ordering, but this quickly becomes unmanageable).

To handle such problems using colour, a set of functions are supplied to sort the order of parameters in lists by colour. Lists are sorted into the order red, then green, then blue, then uncoloured expressions. For example:

```
F[{a, b, 1, 2}, {d, e, f}, z] // CS[]
```

```
F[{2, b, a, 1}, {d, e, f}, z]
```

As usual, the sorting operation can be followed by removing the colour. In this case all three colours are removed:

```
F[{a, b, 1, 2}, {d, e, f}, z] // CS1[]
```

```
F[{2, b, a, 1}, {d, e, f}, z]
```

Hint: If you wish to manipulate special functions such as hypergeometric functions you may find it useful to use a notation, such as that above, which is not recognised by *Mathematica*. This prevents the system applying spurious 'simplifications' such as converting hypergeometric functions to Bessel functions. Later, when required, the rule F->HypergeometricPFQ will switch to standard *Mathematica* notation if desired.

By default these functions sort lists only. However, sometimes it is more useful to apply these functions in a more general context by specifying the head of the term to be sorted. For example:

```
g[a, b, c] // CS1[g]
```

```
g[b, a, c]
```

■ A word of caution

As you probably realise, *Mathematica* does not automatically expand terms such as $\sqrt{x^2}$. The reason for this is that if x is negative or complex this is not valid. Many mathematical operations are only valid under certain conditions. Since colour manipulations are designed for manual use, the system 'assumes' that you know what you are doing. Of course, many of the traditional ways of manipulating *Mathematica* expressions using Take, Part, etc. do not make these checks either. Here are some of the issues to watch for when using colour transformations:

• Operations involving exponents are performed under the same assumptions as are used by PowerExpand.

• Likewise, operations involving logarithms uses the transformation Log[a *b*]⟸Log[a] Log[b]. This can be invalid if complex numbers are involved.

• If you select part of an expression by dragging the mouse (as opposed to repeated clicking) it is possible to select parts of an expression which appear to be legal sub-expressions, but which are not. Here is a simple example:

```
a + b - c + d // FullForm
```

```
Plus[a, d, Times[b, Red\[Breve]Colour[Times[-1, c]]]]
```

Here the legal expression, -c, was extracted from the outer expression in an illegal way - essentially because drag selection can easily be used to select invalid sub-expressions. The best way to avoid this problem is to select using repeated clicks (at least where confusion might exist) and use Rgrow etc. to pickup a minus sign:

```
a + b - c + d // Rgrow // FullForm
```

```
Plus[a, b, d, Red\[Breve]Colour[Times[-1, c]]]
```

• Care should be taken when moving a multiplicative factor to the other side of an inequality. Negative multipliers can give incorrect results.

■ Adding your own extensions

Because the colour functions are represented in the Kernel as functions with no definition, it is easy to create rules which use colour to perform an operation of your own. For example, suppose you needed to perform 2-term series expansions on subexpressions regularly. You could try something like:

```
f[(1 + x)^3, Exp[x]] // RR1[u_ :> Normal[Series[u, {x, 0, 2}]]]
```

$$f\left[(1+x)^3,\ 1+x+\frac{x^2}{2}\right]$$

However, this is very clumsy to use on a regular basis, so an alternative is to define a function to operate on the colour directly:

```
RedExpand[u_] := u /. Red⌣Colour[s_] :> Normal[Series[s, {x, 0, 2}]]
```

Now you can use this function directly whenever it is needed simply by colouring one or more subexpressions red:

```
f[(1 + x)^3, Exp[x]] // RedExpand
```

$$f\left[(1+x)^3,\ 1+x+\frac{x^2}{2}\right]$$

■ Defining inert mathematical operations

Consider the following summation:

$$\sum_{n=1}^{1000} a_n$$

If you give such a sum to *Mathematica*, it will 'evaluate' it - producing an expression with 1000 terms, which is almost certainly not what is required! Summations with infinite or variable limits will either evaluate or remain unevaluated, however there is no built-in way to represent a sum without letting the system take a shot at evaluating it. Clearly, there are many cases, such as when one is trying to prove results about a sum, where one does not want this evaluation to even be attempted. The same is true for definite integrals, products, and limits. To facilitate the manipulation of these objects, the ColourMaths package contains inert variants of these constructs. These can be created by selecting the expression in question and pressing the "Inert" button on the palette.

Active head	Inert head
Integrate	Integrate⌣Inert
Sum	Sum⌣Inert
Product	Product⌣Inert
Limit	Limit⌣Inert

All passive integrals and sums are assumed to be definite, indefinite integrals are converted to definite intert integrals by the "Inert" button. An indefinite operation is specified by using a variable as the upper limit.

Inert constructs have a shaded background to distinguish them from normal operations. Where inert operations are nested inside other inert operations, this shading is made deeper.

Clearly it would be possible to add further inert operations to this list, such as differentiation. As we will see in the next section, these inert operations allow us to represent theorems involving sums, integrals etc. (such as the binomial theorem) whereas if the sum in the binomial theorem were represented using Sum, *Mathematica* would simply apply the binomial theorem and evaluate the sum!

Although it would be possible to represent these objects using HoldForm (the invisible form of Hold), in practice this is not very convenient as it suppresses other evaluations which may well be required. For example:

```
HoldForm[Sum[a[n, 8], {n, 0, 1000}]] /. a[p_, q_] :> 2^q b[p] /; EvenQ[q]
```

$$\sum_{n=0}^{1000} 2^8\, b[n]$$

Compare this with the same operations performed with an inert sum and without HoldForm:

```
∑_{n=0}^{1000} a[n, 8] /. a[p_, q_] :> 2^q b[p] /; EvenQ[q]
```

$$\sum_{n=0}^{1000} 256\, b[n]$$

Somewhat curiously, Sum and Product have attribute HoldAll, whereas Integrate and Limit do not. None of the passive forms have been defined to hold their arguments - it is assumed that their dummy arguments do not have definitions outside of the construct. Since we often want to perform extensive manipulations of summations it is convenient to allow evaluation to take place.

In general, nested sums, and integrals can have the order of operations reversed subject to certain conditions regarding uniform convergence. A function SwapOperations is supplied to achieve this on the inert forms:

```
∑_{n=0}^{1} ∫_0^1 f_n[x] dx // SwapOperations
```

$$\int_0^1 \left(\sum_{n=0}^{1} f_n[x]\right) dx$$

```
∫_0^1 (∑_{n=0}^{1} f_n[x]) dx // SwapOperations
```

$$\sum_{n=0}^{1} \int_0^1 f_n[x]\, dx$$

Obviously this is a purely mechanical process, it could not even begin to check for uniform convergence, since in general it would not 'know' the meaning of many of the functions in use. The user must add this to his derivation if required.

Two common hand manipulations of integrals is to change the variable of integration or to perform an integration by parts. Here we see a variable substitution with a subsequent simplification of the integrand:

```
∫_0^1 (1 - t)^(-1+q) t^(-1+p) dt // Iv[s -> t / (1 - t), t]
```

$$\int_0^\infty \frac{\left(\frac{s}{1+s}\right)^{-1+p} \left(1-\frac{s}{1+s}\right)^{-1+q}}{\frac{s}{(1+s)\left(1-\frac{s}{1+s}\right)^2} + \frac{1}{1-\frac{s}{1+s}}}\, ds$$

$$\int_0^\infty \frac{\left(\frac{s}{1+s}\right)^{-1+p}\left(1-\frac{s}{1+s}\right)^{-1+q}}{\frac{s}{(1+s)\left(1-\frac{s}{1+s}\right)^2}+\frac{1}{1-\frac{s}{1+s}}}\,ds \quad \text{// Simplify}$$

$$\int_0^\infty \frac{\left(\frac{1}{1+s}\right)^q\left(\frac{s}{1+s}\right)^p}{s}\,ds$$

The integration by parts operation assumes that the part to be integrated is picked out in red:

$$\int_a^b x^n \,\text{Sin}[x]\,dx \quad \text{// ByParts}$$

$$a^n \,\text{Cos}[a] - b^n \,\text{Cos}[b] + \int_a^b n\, x^{-1+n}\,\text{Cos}[x]\,dx$$

One problem with inert mathematical operations such as Integral is that it is impossible to define differentiation with respect to a parameter in a consistent way. Here is a simple definition assuming the limits are constants:

```
Integrate‿Inert /: D[Integrate‿Inert[F_, {x_, a_, b_}], k_] :=
 Integrate‿Inert[D[F, k], {x, a, b}]
```

Using this definition, we seem to be able to differentiate inert integrals:

$$D\left[\int_0^{\frac{\pi}{2}} \sqrt{1-k^2\,\text{Sin}[x]^2}\,dx,\, k\right]$$

$$-\int_0^{\frac{\pi}{2}} \frac{k\,\text{Sin}[x]^2}{\sqrt{1-k^2\,\text{Sin}[x]^2}}\,dx$$

However, if the inert integral is embedded inside a larger expression, things go wrong. It would seem that D uses the chain rule a little too eagerly.

$$D\left[k\int_0^{\frac{\pi}{2}} \sqrt{1-k^2\,\text{Sin}[x]^2}\,dx,\, k\right]$$

$$\int_0^{\frac{\pi}{2}} \sqrt{1-k^2\,\text{Sin}[x]^2}\,dx - \frac{k^2\,\pi\,\text{Sin}[x]^2}{2\sqrt{1-k^2\,\text{Sin}[x]^2}}$$

For this reason, the ColourMaths package supplies the Diff operation which fixes this problem.

$$\text{Diff}\left[k\int_0^{\frac{\pi}{2}} \sqrt{1-k^2\,\text{Sin}[x]^2}\,dx,\, k\right]$$

$$-k\int_0^{\frac{\pi}{2}} \frac{k\,\text{Sin}[x]^2}{\sqrt{1-k^2\,\text{Sin}[x]^2}}\,dx + \int_0^{\frac{\pi}{2}} \sqrt{1-k^2\,\text{Sin}[x]^2}\,dx$$

The ColourMaths package also contains inert forms for several common Special functions and related mathematical objects. In most cases these also have rules to display them in a traditional form from StandardForm. These functions do not display well in TraditionalForm, but by switching them to the active *Mathematica* equivalent it is possible to obtain an expression suitable for display in TraditionalForm.

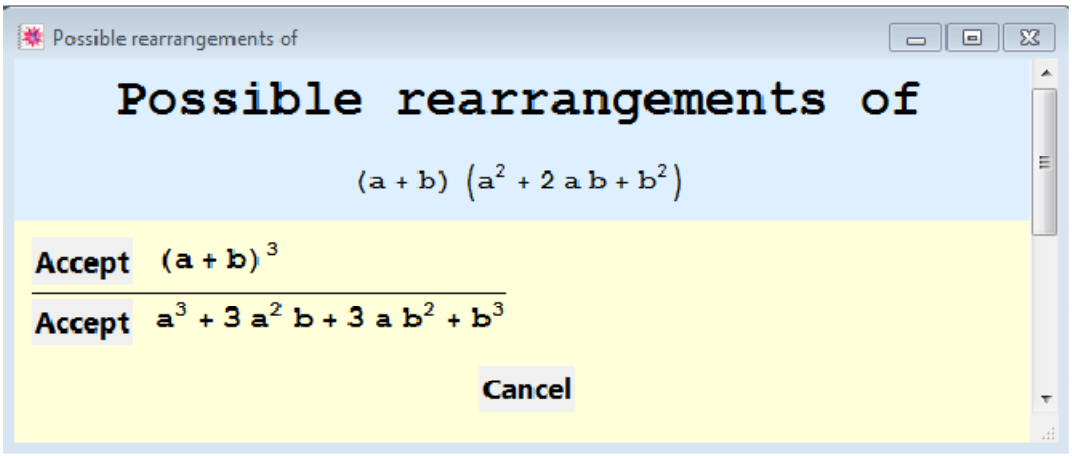
■ The Adjust button – Visual algebra

Most of us are familiar with doing algebra in a linear fashion. The 'point and click' culture has yet to make much impact

on this process. However, The Adjust button offers a step in that direction! Simply select the part of an expression to be manipulated, and press the Adjust button to see a menu of possible manipulations:

```
f[(a + b) (a^2 + 2 a b + b^2)]
```

Note that I have used the shading to represent the selection – it isn't part of the expression. Here is what the system offers at this point:



Accepting one or other of the two choices will not simply alter the expression (which would leave no indication of how the transformation was produced). Instead, it annotates the expression, so that the desired result is obtained by evaluation:

```
f[(a + b) (a^2 + 2 a b + b^2)] // Red⌣ColourAdjust[Apart⌣Adjust]
```

$$f\left[a^3 + 3\,a^2\,b + 3\,a\,b^2 + b^3\right]$$

This illustrates the concept, but to obtain really interesting results, we need to supply some transformations that are specific to our area of interest. For example:

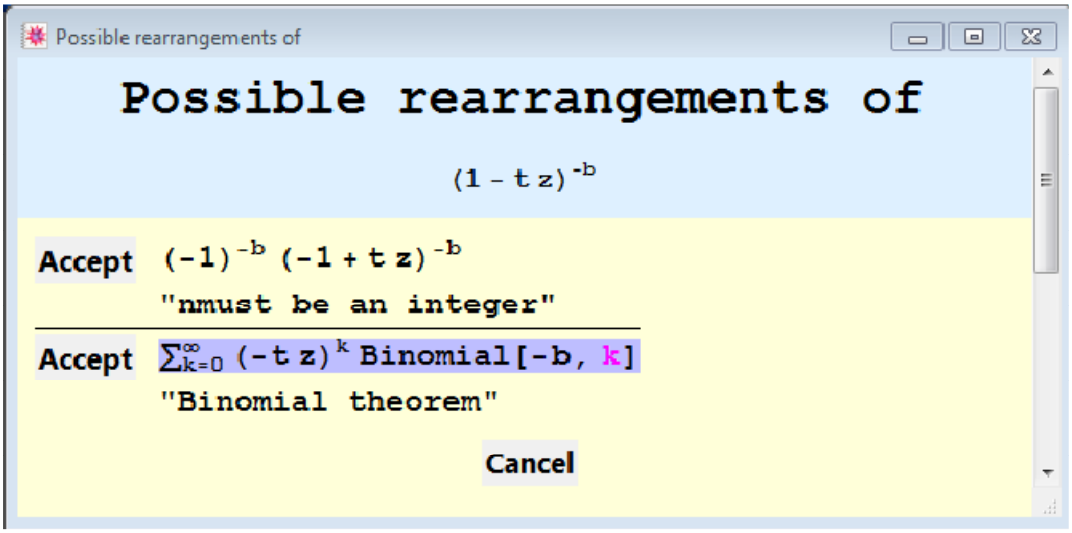
```
Define⌣Adjustment[Binomial⌣Theorem,
  (1 + x_)^p_ :> Sum[x^k Binomial[p, k], {k, 0, Infinity}], "Binomial theorem"];

Define⌣Adjustment[Beta⌣Function⌣Definition,
  Integrate⌣Inert[t_^(p_ - 1) * (1 - t_)^(q_ - 1), {t_, 0, 1}] :> B[p, q],
  "Beta function definition"];
```

The Define⌣Adjustment function takes an arbitrary transformation name (analogous to Apart⌣Adjust above), which must be unique, a transformation rule, and any additional comments. The last argument can be omitted.

Here is a typical use of these definitions. Select the shaded part of the expression, and press the Adjust button. You should see the following::

$$\int_0^1 (1 - t)^{-1-a+c}\, t^{-1+a}\, (1 - t\,z)^{-b}\, dt$$



This produces the following. As you can see, it is quite clear what step was chosen, and it could be re-executed at a later date provided the above call to Define␣Adjustment had been entered.

However, there is a complication here. The summation in the binomial expansion requires a dummy index – a problem which is very common in really useful transformations. It is for this reason that the dummy index, k, was coloured with the 'flag colour' when it was entered. The idea is that you fill in a suitable index to use thus:

$$\int_0^1 (1-t)^{-1-a+c}\, t^{-1+a}\, (1-t\,z)^{-b}\, dt \;//\; \text{Red␣ColourAdjust[Binomial␣Theorem, k} \to \blacksquare\text{]}$$

$$\int_0^1 (1-t)^{-1-a+c}\, t^{-1+a}\, (1-t\,z)^{-b}\, dt \;//\; \text{Red␣ColourAdjust[Binomial␣Theorem, k} \to \text{m]}$$

$$\int_0^1 (1-t)^{-1-a+c}\, t^{-1+a} \sum_{m=0}^{\infty} (-t\,z)^m\, \text{Binomial}[-b, m]\, dt$$

Now we swap the integral and sum (deferring any convergence checks until later), and move things about.

$$\int_0^1 (1-t)^{-1-a+c}\, t^{-1+a} \sum_{m=0}^{\infty} (-t\,z)^m\, \text{Binomial}[-b, m]\, dt \;//\; \text{SwapOperations} \;//\; \text{Rout1}$$

$$\sum_{m=0}^{\infty} \text{Binomial}[-b, m] \int_0^1 (1-t)^{-1-a+c}\, t^{-1+a}\, (-t\,z)^m\, dt$$

Now a few simple rearrangements leave the expression with a recognisable beta function integral

$$\sum_{m=0}^{\infty} \text{Binomial}[-b, m] \int_0^1 (1-t)^{-1-a+c}\, t^{-1+a}\, (-t\,z)^m\, dt \;//\; \text{Gmpy}[(-z)^{-m}]$$

$$\sum_{m=0}^{\infty} \text{Binomial}[-b, m] \int_0^1 (1-t)^{-1-a+c}\, t^{-1+a}\, (-z)^m\, \left((-z)^{-m}\, (-t\,z)^m\right) dt$$

$$\sum_{m=0}^{\infty} \text{Binomial}[-b, m] \int_0^1 (1-t)^{-1-a+c}\, t^{-1+a}\, (-z)^m\, \left((-z)^{-m}\, (-t\,z)^m\right) dt \;//\; \text{Rout1} \;//$$

$$\text{GG1[PowerExpand]}$$

$$\sum_{m=0}^{\infty} \text{Binomial}[-b, m] \int_0^1 (1-t)^{-1-a+c}\, t^{-1+a+m}\, (-z)^m\, dt \;//\; \text{Rout1}$$

$$\sum_{m=0}^{\infty} (-z)^m \text{Binomial}[-b, m] \int_0^1 (1-t)^{-1-a+c}\, t^{-1+a+m}\, dt$$ //

```
Red⌣ColourAdjust[Beta⌣Function⌣Definition]
```

$$\sum_{m=0}^{\infty} (-z)^m \text{Binomial}[-b, m]\, B[a+m, -a+c]$$

■ Highlight Formulae

Most serious mathematical derivations number certain formulae so that these can be referred to at subsequent points in the text. While *Mathematica* does permit equation numbering, it supplies no way to actually access an equation by number. Although it would probably be possible to create such a mechanism, another problem associated with relying on equation numbering, is that inserting an extra numbered equation into a notebook will cause the other equation numbers to change.

The ColourMaths package lets you highlight any cell and give it a name to identify it further down the notebook. This is done by clicking anywhere in the cell, and pressing the "Highlight" button on the palette. For example:

"Atomic_hydrogen_ground_state"

$$\text{E⌣Hydrogen} == \frac{e^4\, m_e\, m_p}{32\, \pi^2\, \hbar^2\, (m_e + m_p)\, \epsilon_0^2}$$

Once this highlight formula has been named in this way, it can be refereed to thus:

```
Get⌣Highlight⌣Formula["Atomic_hydrogen_ground_state"]
```

$$\text{E⌣Hydrogen} == \frac{e^4\, m_e\, m_p}{32\, \pi^2\, \hbar^2\, (m_e + m_p)\, \epsilon_0^2}$$

Using the following extra definitions (which might typically be included in the initialisation section of a notebook), we can access the formula in a more interesting way:

```
physicalConstants = {
     me -> 9.10938215`*^-31,
    mp -> 1.672621637`*^-27,
     ϵ0 -> 8.85419*^-12,
     e -> 1.6021766*^-19,
     ħ -> 1.0545717*^-34};

Get⌣Highlight⌣Formula["Atomic_hydrogen_ground_state"] /. physicalConstants
```

$$\text{E⌣Hydrogen} == 2.17868 \times 10^{-18}$$

If you prefer to number your equations, you can always give them a numerical name, e.g. "23" or "(23)" or "2.1.7" .

Highlight formulae can also be accessed from a menu using Pick⌣Highlight⌣Formula:

```
Pick⌣Highlight⌣Formula[]
```

$$\text{E⌣Hydrogen} == \frac{e^4\, m_e\, m_p}{32\, \pi^2\, \hbar^2\, \left(m_e + m_p\right)\, \epsilon_0^2}$$

Certain other functions, such as Left⌣Side and Right⌣Side can also take the string name of a suitable formula. For example:

```
Right⌣Side["Atomic_hydrogen_ground_state"]
```

$$\frac{e^4\, m_e\, m_p}{32\, \pi^2\, \hbar^2\, \left(m_e + m_p\right)\, \epsilon_0^2}$$

Note that these functions access the highlight formulae from the notebook,not from the kernel. This means that provided the ColourMaths package is loaded, it is not necessary to re-execute a notebook to pick up these formulae.

■ Future enhancements

Clearly the concept of using *Mathematica* in this way could be expanded in many directions, with specialised sets of transformation rules, etc being added as required.