Complex variables

Initialization

```
ClearAll["Global`*"];
Off[General::spell, General::spell1];
```

Introduction

Mathematica does not automatically assume that variables are real or that real variables are positive; its symbolic manipulations are intended to be as general as possible unless given specific information about the properties of particular variables. If you are certain that all variables in an expression are real and positive, PowerExpand will perform the desired simplification.

```
expr = Sqrt[a^2 b^2 c^2];
expr // Simplify
expr // PowerExpand
```

$$\sqrt{a^2\, b^2\, c^2}$$

$$a\, b\, c$$

Many other functions do not respond to **Simplify** without making assumptions about their arguments which determine the appropriate branch for functions of complex variables.

```
Log[expr] // Simplify
Log[expr] // PowerExpand
Log[expr] // PowerExpand // Simplify
```

$$\frac{1}{2}\,\text{Log}\left[a^2\,b^2\,c^2\right]$$

$$\frac{1}{2}\,\left(2\,\text{Log}[a]+2\,\text{Log}[b]+2\,\text{Log}[c]\right)$$

$$\text{Log}[a]+\text{Log}[b]+\text{Log}[c]$$

ComplexExpand separates an expression into its real and imaginary parts assuming that all variables within that expression are real.

```
z = x + I y;
{Cos[z], Cosh[z^2], Log[z]} // ComplexExpand
```

$$\left\{\text{Cos}[x]\,\text{Cosh}[y]-i\,\text{Sin}[x]\,\text{Sinh}[y],\right.$$
$$\text{Cos}[2\,x\,y]\,\text{Cosh}\left[x^2-y^2\right]+i\,\text{Sin}[2\,x\,y]\,\text{Sinh}\left[x^2-y^2\right],$$
$$\left.i\,\text{Arg}[x+i\,y]+\frac{1}{2}\,\text{Log}\left[x^2+y^2\right]\right\}$$

We can use **ComplexExpand** to simplify expressions involving complex variables by supplying a list of complex arguments as an optional argument, such that **ComplexExpand[expr,{z1,z2...}]** expands *expr* assuming that variables matching any of the x_i are complex.

```
Clear[k, z];
ComplexExpand[ Exp[I k z] + Exp[-I k z], k]
```

$$e^{-z\,\text{Im}[k]}\,\text{Cos}[z\,\text{Re}[k]]+e^{z\,\text{Im}[k]}\,\text{Cos}[z\,\text{Re}[k]]+$$
$$i\,\left(e^{-z\,\text{Im}[k]}\,\text{Sin}[z\,\text{Re}[k]]-e^{z\,\text{Im}[k]}\,\text{Sin}[z\,\text{Re}[k]]\right)$$

Another useful option for functions such as ComplexExpand that specifies what functions to attempt to generate in the output is TargetFunctions→ {Abs, Arg}

```
Clear[k, z];
ComplexExpand[Exp[I k z] + Exp[-I k z], k,
 TargetFunctions → {Abs, Arg}]
```

$$e^{-z\,\text{Abs}[k]\,\text{Sin}[\text{Arg}[k]]}\,\text{Cos}[z\,\text{Abs}[k]\,\text{Cos}[\text{Arg}[k]]] + e^{z\,\text{Abs}[k]\,\text{Sin}[\text{Arg}[k]]}\,\text{Cos}[z\,\text{Abs}[k]\,\text{Cos}[\text{Arg}[k]]] + i\left(e^{-z\,\text{Abs}[k]\,\text{Sin}[\text{Arg}[k]]}\,\text{Sin}[z\,\text{Abs}[k]\,\text{Cos}[\text{Arg}[k]]] - e^{z\,\text{Abs}[k]\,\text{Sin}[\text{Arg}[k]]}\,\text{Sin}[z\,\text{Abs}[k]\,\text{Cos}[\text{Arg}[k]]]\right)$$

```
Clear[k, z];
ComplexExpand[Exp[I k z] + Exp[-I k z], k,
  TargetFunctions → {Abs, Arg}] /.
 {Abs[k] → κ, Arg[k] → φ}
```

$$e^{-z\,\kappa\,\text{Sin}[\varphi]}\,\text{Cos}[z\,\kappa\,\text{Cos}[\varphi]] + e^{z\,\kappa\,\text{Sin}[\varphi]}\,\text{Cos}[z\,\kappa\,\text{Cos}[\varphi]] + i\left(e^{-z\,\kappa\,\text{Sin}[\varphi]}\,\text{Sin}[z\,\kappa\,\text{Cos}[\varphi]] - e^{z\,\kappa\,\text{Sin}[\varphi]}\,\text{Sin}[z\,\kappa\,\text{Cos}[\varphi]]\right)$$

Built-in Functions: Re, Im, Abs, Conjugate

The built-in functions Re, Im, Abs, and Conjugate return the real part, imaginary part, absolute value, and complex conjugate of complex numbers, but make no *a priori* assumptions about their arguments. Hence, the evaluation of many expressions appears incomplete.

```
Clear[a, b, z];
z = a + 2 I b;
{Re[z], Im[z], Abs[z], Conjugate[z]}
```

```
{-2 Im[b] + Re[a], Im[a] + 2 Re[b],
 Abs[a + 2 i b], Conjugate[a] - 2 i Conjugate[b]}
```

Now suppose that we declare *a* and *b* to be real by associating rules with those symbols that nullify their imaginary parts.For that reason we tray to use *f* **/:** *lhs* **=** *rhs* assigns *rhs* to be the value of *lhs*, and associates the assignment with the symbol *f*.

```
Clear[a, b, z];
a /: Im[a] = 0;
b /: Im[b] = 0;
z = a + 2 I b;
{Re[z], Im[z], Abs[z], Conjugate[z]}
```

```
{Re[a], 2 Re[b], Abs[a + 2 i b],
 Conjugate[a] - 2 i Conjugate[b]}
```

Better results are obtained, but notice that setting Im[*a*] → 0 does not automatically ensure that Re[*a*] → *a*.

```
Clear[a, b, z];
a /: Im[a] = 0;
a /: Re[a] = a;
b /: Im[b] = 0;
b /: Re[b] = b;
z = a + 2 I b;
{Re[z], Im[z], Abs[z], Conjugate[z]}
```

```
{a, 2 b, Abs[a + 2 i b],
 Conjugate[a] - 2 i Conjugate[b]}
```

Abs and **Conjugate** still do not behave as we would like. These deficiencies arise because this family of functions is defined for numerical arguments and do not have rules attached to guide the evaluation of expressions. Although we have supplemented these rules by attaching declarations to some of the symbols, it is a rather incomplete solution. For example, even if we declare *a* to be real, **Re** does not recognize that Cos[*a*] is then real and so we can use **ComplexExpand** to obtain the real or imaginary parts of functions.

```
Re[Cos[a]]
```

```
Re[Cos[a]]
```

```
Re[Cos[a]] // ComplexExpand
```

```
Cos[a]
```

Conjugate does not utilize associate or distributive properties.

```
Conjugate[z]
```

```
Conjugate[a] - 2 i Conjugate[b]
```

```
Conjugate[z] // ComplexExpand
```

```
a - 2 i b
```

The symbol **I** nor its internal representation **Complex[0,1]** is present in the expression for **z**. Thus, the replacement rule has no effect. Fortunately, this problem is easily circumvented by defining our own complex conjugation function as suggested by Zimmerman and Olness by introducing a rule that transforms lhs to rhs, evaluating rhs after the rule is used:

lhs :> rhs or lhs :-> rhs

```
conjugate::usage =
  "A simple method for computing the conjugate
    of an object which is explicitly complex.";

conjugateRule =
  Complex[re_, im_] :> Complex[re, -im];
conjugate[exp__] := exp /. conjugateRule;
```

```
conjugate[z]
```

```
a - 2 i b
```

```
conjugate[x - I y]
```

```
x + i y
```

However, this function will not work for arbitrary expressions because it assumes that all symbols are real and conjugates only the numerical coefficients expressed in terms of **Complex**.

Exercises

Explain why `Re[ ComplexExpand[a + I b] ]` and `ComplexExpand[Re[a + I b]]` give different results. Which should be preferred?

Determine the real and imaginary parts of `Cosh[k z]` for real *z* and complex *k*.

Plot the real and imaginary parts of `BesselJ[3, z]` for complex *z*.

Write a function which properly simplifies `Abs[expr]` assuming that the variables in *expr* are real. Thus, your function should yield `abs[x + I y]→Sqrt[x²+y²]` and similar results for related expressions. Use your function to obtain a simple expression for `abs[ Cos[x+I y] ]` involving only real quantities.

Write a function which declares a variable to be real. Your

function should specify values for **Re**, **Im**, and **Conjugate**.

Using the package ReIm

The package is used to extend the functionality of the built-in functions that manipulate complex variables.

```
Needs["Algebra`ReIm`"];
```

If we declare certain variables to be real, expressions involving those variables can be simplified in the expected manner.

```
Clear[a, b, z];
a /: Im[a] = 0;
b /: Im[b] = 0;
z = a + I b;
```

```
{Re[z], Im[z], Abs[z], Conjugate[z],
  z Conjugate[z]} // Simplify
```

```
{Re[a], Re[b], Abs[a + i b],
 Conjugate[a] - i Conjugate[b],
 (a + i b) (Conjugate[a] - i Conjugate[b])}
```

Notice that now it is not necessary to give two rules to get a real number, simply nullifying the imaginary part is sufficient when **ReIm** is active. However, the absolute value function still does not operate as we would like, but we can work around that with a replacement rule. Furthermore, expressions involving variables of unspecified type are handled as if they are complex.

```
Conjugate[x - I y]
```

```
Conjugate[x] + i Conjugate[y]
```

```
Log[z] // ComplexExpand
```

$$i\,\text{Arg}[a + i\,b] + \frac{1}{2}\,\text{Log}\left[a^2 + b^2\right]$$

Efficiency issues for ReIm

Although **ReIm** provides a set of rules which performs many simplifications of expressions involving complex variables in a manner that is transparent to the user, this package is quite inefficient and can slow down calculations enormously. To illustrate this problem, we borrow the following example from Bahder. Before evaluating the following expression, quit the kernel in order to unload the package.

```
expr = Integrate[ (a x^2 + b x + c)/(c x + a), x]
```

$$\frac{\left(-a^2 + b\,c\right)\,x}{c^2} + \frac{a\,x^2}{2\,c} + \frac{\left(a^3 - a\,b\,c + c^3\right)\,\text{Log}[a + c\,x]}{c^3}$$

The time needed to evaluate the real part of this result, assuming that the coefficients *a*, *b*, *c* are complex while the integration variable *x* is real can now be obtained using **Timing**.

```
ComplexExpand[Re[expr], {a, b, c}]; // Timing
```

```
{0., Null}
```

Now we reload the package, declare *x* to be real, and evaluate the Re[expr] using the rules from the package. Note that **ComplexEx-**

pand is not needed when using the package.

```
Needs["Algebra`ReIm`"]
```

```
x /: Im[x] = 0;
Re[expr]; // Timing
```

```
{0., Null}
```

Thus, similar evaluations take much longer using **ReIm** compared with **ComplexExpand**. Bahder ascribes this inefficiency to the need to test all of the rules associated with **Re** that are contained in the package, whether or not they are actually relevant to the expression being evaluated. Therefore, for most purposes it is better to simplify expressions manually, with appropriate replacement rules, or to supply a set of user-defined rules pertinent to the expressions encountered in one's applications. Some guidance in constructing those rules can be obtained from the package itself because the package is a text file which can be edited by the user. Of course, the user is advised to work on a copy rather than the original!