Simplification of Algebraic Expressions

Contributors: Michael Monagan, Petr Lisonek, Hans Bauck

This work was supported by the MITACS NCE and NSERC of Canada.

Simplification of expressions is a difficult but important task performed by every computer algebra system. In a very real sense it is perhaps the single most important task because many problems may be posed as simplification problems. In this MITACS project we are developing simplification algorithms that, for a given expression *e*, build a basis *B* such that elements of *B* are algebraically independent and *e* is a rational function over *B*. Some of the code developed in this project has been installed in Maple 6 and some is being installed in the current research version of Maple. This worksheet highlights some of the work.

1. Simplification of Elementary Functions

One of our focus points is the simplification of elementary functions, i.e. functions that can be obtained from the field of rational functions in one or more variables over a constant field by adding exponentials and logarithms. The elementary functions are important since they include, among other things, all trigonometric and hyperbolic functions as well as their inverses.
We are using the *Risch Structure Theorem* for elementary functions in its multivariate version, first proven in the paper by Epstein & Caviness: ''A structure theorem for the elementary functions and its application to the identity problem.'' (Internat. J. Comput. Inform. Sci. 1979). Here is a simple example:

```
> e:= 2*sin((x+y)/2)*cos((x+y)/2)- sin(x+y);
  simplify(e);
```

$$e := 2\sin\left(\frac{1}{2}x+\frac{1}{2}y\right)\cos\left(\frac{1}{2}x+\frac{1}{2}y\right)-\sin(x+y)$$

$$2\sin\left(\frac{1}{2}x+\frac{1}{2}y\right)\cos\left(\frac{1}{2}x+\frac{1}{2}y\right)-\sin(x+y)$$

The reason for the failure of the **simplify** command in Maple is that the dependency between trigonometric functions is not recognized. The Risch Structure Theorem is a rigorous tool for recognizing such dependencies, once the function is represented using exponentials and logarithms. Our implementation handles this automatically but here are the details carried out manually for the sake of illustration. First we convert to complex exponentials

```
> f := convert(e,exp);
```

$$f := -I\left(\mathbf{e}^{(I(1/2\,x+1/2\,y))}-\frac{1}{\mathbf{e}^{(I(1/2\,x+1/2\,y))}}\right)\left(\frac{1}{2}\mathbf{e}^{(I(1/2\,x+1/2\,y))}+\frac{\frac{1}{2}}{\mathbf{e}^{(I(1/2\,x+1/2\,y))}}\right)+\frac{1}{2}I\left(\mathbf{e}^{(I(x+y))}-\frac{1}{\mathbf{e}^{(I(x+y))}}\right)$$

The Risch Structure Theorem identifies the algebraic relationship between the two exponential terms, namely, that $\mathbf{e}^{(I(x+y))}=\left(\mathbf{e}^{\left(I\left(\frac{x}{2}+\frac{y}{2}\right)\right)}\right)^2$. Letting $\Theta_1=\mathbf{e}^{\left(I\left(\frac{x}{2}+\frac{y}{2}\right)\right)}$ and $\Theta_2=\mathbf{e}^{(I(x+y))}$ our input expression is

```
> f := subs(exp(I*(x+y))=Theta[2], exp(I*(x/2+y/2))=Theta[1] , f);
```

$$f := -I\left(\Theta_1-\frac{1}{\Theta_1}\right)\left(\frac{1}{2}\Theta_1+\frac{\frac{1}{2}}{\Theta_1}\right)+\frac{1}{2}I\left(\Theta_2-\frac{1}{\Theta_2}\right)$$

where the relationship is $\Theta_1^{\ 2}=\Theta_2$. We now have a choice, we may replace Θ_1 by $\sqrt{\Theta_2}$ or we may replace Θ_2 by $\Theta_1^{\ 2}$. Either approach will work but the latter is preferable. We have

```
> f := subs(Theta[2]=Theta[1]^2,f);
```

$$f := -I\left(\Theta_1 - \frac{1}{\Theta_1}\right)\left(\frac{1}{2}\Theta_1 + \frac{\frac{1}{2}}{\Theta_1}\right) + \frac{1}{2} I \left(\Theta_1^{\,2} - \frac{1}{\Theta_1^{\,2}}\right)$$

Now we do rational arithmetic on this expression and we find that it simplfies to zero.

```
> normal(f);
```

$$0$$

Apparently very little has been published regarding the implementation of the multivariate Risch Structure Theorem. The references usually only hint at the fact that a system of linear equations is set up for the unknown coefficients, see e.g., ''Algorithms for computer algebra'' by Geddes, Czapor and Labahn (which only deals with the univariate case). We have realized that the most obvious implementation of this idea, i.e. setting up the linear system by clearing denominators and equating coefficients at the monomials, would be computationally very costly and also it normally leads to a highly overdetermined linear system, hence a lot of unnecessary work is done. Therefore we use a *modular* approach. We set up a linear system by substituting random values for all of the variables and transcendental extensions in the Risch equation, then we solve the system modulo a single large prime p and then we obtain the unknown coefficients in the Risch equation by rational reconstruction (since they are known to be rationals by the statement of the theorem).

2. Simplification of Trigonometric Functions

A problem with this approach is that trigonometric and hyperbolic functions are converted to complex logarithms and exponentials. After doing the rational step it is not always possible to convert the complex logarithms and exponentials back to a real trigonometric form. Therefore, we use a modified Risch Structure Theorem that applies directly to trigonometric and hyperbolic functions. Consider the following simple example

```
> e:= 2*sin((x+y)/2)*cos((x+y)/2)- sin(x+y) + cos(x+y);
```

$$e := 2\sin\left(\frac{1}{2}x + \frac{1}{2}y\right)\cos\left(\frac{1}{2}x + \frac{1}{2}y\right) - \sin(x+y) + \cos(x+y)$$

Here we have four trigonometric functions $\Theta_1 = \sin(x+y)$, $\Theta_2 = \cos(x+y)$, $\Theta_3 = \sin\left(\frac{x}{2} + \frac{y}{2}\right)$ and $\Theta_4 = \cos\left(\frac{x}{2} + \frac{y}{2}\right)$.

```
> f := subs( sin(x+y)=Theta[1], cos(x+y)=Theta[2],
  sin(x/2+y/2)=Theta[3], cos(x/2+y/2)=Theta[4], e );
```

$$f := 2\,\Theta_3\,\Theta_4 - \Theta_1 + \Theta_2$$

There are different orderings in which the functions may be processed. In our code what happens is that Θ_1 and Θ_2 are processed first. At this point we have $B = [\Theta_1, \Theta_2, \Theta_1^{\,2} + \Theta_2^{\,2} = 1]$ where we keep the known algebraic relation $\Theta_1^{\,2} + \Theta_2^{\,2} = 1$ in the basis. When Θ_3 is processed, the modified Risch Structure Theorem tells us that $\Theta_1 = 2\,\Theta_3\,\Theta_4$. We have a choice here in that we could eliminate any of $\Theta_1, \Theta_3, \Theta_4$. Our code makes the first choice

```
> f := subs( Theta[1]=2*Theta[3]*Theta[4], f );
```

$$f := \Theta_2$$

At this point the algorithm restarts itself and we recognize that there is nothing to do because there is only one function in the expression. Hence the simplified result is $\Theta_2 = \cos(x+y)$.

3. Use of the Unwinding Number

In Maple all variables are implicitly assumed to be complex. As is well known from complex analysis,

$$\mathbf{e}^z = \mathbf{e}^{(z+2\,I\,k\,\pi)}$$

for any integer k. Thus the function $\mathbf{e}^z$ is not injective in the complex plane, and its inverse (i.e. $\log(u)$) is not defined uniquely (has infinitely many branches). But in scientific computing systems, it is impractical to work with such functions. Instead one works with the ***principal branch of the logarithm*** which is defined by $\Im(\log(z))\ \varepsilon\ (-\pi, \pi]$.

As a consequence of restricting the logarithm to its principal branch, we lose some algebraic identities which are crucial for simplification. In particular, $\log(x\,y) = \log(x) + \log(y)$ is no longer valid for all complex x, y.A trivial example follows:

```
> z1:=-3; z2:=-4;
```

$$z1 := -3$$

$$z2 := -4$$

```
> Im(log(z1)), Im(log(z1));
```

$$\pi, \pi$$

```
> simplify( log(z1*z2)-log(z1)-log(z2) );
```

$$-2\,I\,\pi$$

We can consider an integer valued function that counts how far away (how many branches) is z from the principal branch of the logarithm, i.e., from $\log(\mathbf{e}^z)$. Because the exponential function ''winds'' its argument around the origin of the complex plane, we think of the difference between z and $\log(\mathbf{e}^z)$ as ''unwinding'' of $\mathbf{e}^z$.

Here is the implicit form *definition of the Unwinding Number K(z):*

```
> log(exp(z)) = z + 2*Pi*I*K(z);
```

$$\ln(\mathbf{e}^z) = z + 2\,I\,\pi\,\mathrm{K}(z)$$

A recent reference for the Unwinding Number is the article ''Reasoning about the elementary functions of complex analysis'' by Corless, Davenport, Jeffrey, Litt and Watt (to appear in Springer Lecture Notes on Artificial Intelligence LNAI Vol. **1930**), in which the definition of *K(z)* differs from ours by a sign change. The unwinding number allows us to recover the ''lost'' identities for the log. Example:

$$\ln(x\,y) = \ln(x) + \ln(y) + 2\,I\,\pi\,\mathrm{K}(\ln(x) + \ln(y))$$

The following examples were obtained using our implementation of the function *K(z):*

```
> K(17);
```

$$0$$

```
> K(7-3*Pi*I);
```

$$2$$

```
> K( log(1-I+t));
```

$$0$$

```
> K(log(x)+log(y));
```

$$\mathrm{K}(\ln(x) + \ln(y))$$

```
> assume(y>0);
> K(log(x)+log(y));
```

$$0$$

The main identities for simplifying expressions with logarithms with $\mathrm{K}(z)$ are

$$\log(z^a) = a\,\log(z) + 2\,\pi\,I\,\mathrm{K}(a\,\log(z)) \text{ and}$$

$$\log\left(\prod_{k=1}^{n} z_k\right) = \left(\sum_{k=1}^{n} \log(z_k)\right) + 2\,\pi\,I\,\mathrm{K}\left(\sum_{k=1}^{n} \log(z_k)\right).$$

We did a prototype implementation of these identities inside the logarithm function. We call the new function Log to distinguish it from the standard library function ln. Here are some examples obtained using our implementation:

```
> Log(x*y);
```

$$\ln(x)+\ln(y)+2\,I\,\pi\,\mathrm{K}(\ln(x)+\ln(y))$$

```
> assume(x>0);
> Log(x*y);
```

$$\ln(x\sim)+\ln(y)$$

```
> assume(x<0);assume(y<0);
> Log(x*y*z)-Log(x*y);
```

$$\ln(z)$$

```
> x:='x': y:='y':
  Log(x^2*y);
```

$$2\ln(x)+2\,I\,\pi\,\mathrm{K}(2\ln(x))+\ln(y)+2\,I\,\pi\,\mathrm{K}(2\ln(x)+2\,I\,\pi\,\mathrm{K}(2\ln(x))+\ln(y))$$

```
> assume(x<0);
> Log(x^2*y);
```

$$2\ln(x\sim)-2\,I\,\pi+\ln(y)$$

Because of the general definition of the power function in the complex plane, i.e. $a^b=\mathbf{e}^{(b\ln(a))}$ we can use the Unwinding Number in simplifying expressions with roots.

A challenging task, which we are still working on, is a proper integration of the Unwinding Number into the Risch Structure Theorem, since the Risch theorem itself deals with algebraic objects (not with analytic objects), thus it does not deal with branches.

4. Simplification of Constants

Here our contribution so far has been mainly in a better treatment of combining several logarithmic constants together. Some examples follow. The code is integrated in Maple 6.

```
> Pi*log(2)  + 1/3*log(7) -2/5*log(11) - Pi/2*log(5);
```

$$\pi \ln(2) + \frac{1}{3}\ln(7) - \frac{2}{5}\ln(11) - \frac{1}{2}\pi\ln(5)$$

```
> combine(%);
```

$$\frac{1}{2}\pi\ln\left(\frac{4}{5}\right) + \frac{1}{15}\ln\left(\frac{16807}{1771561}\right)$$

```
> Machin:=4*arctan(1/5)-arctan(1/239)-Pi/4;
```

$$Machin := 4\arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right) - \frac{1}{4}\pi$$

```
> convert(Machin,ln);
```

$$2\,I\left(\ln\left(1-\frac{1}{5}I\right)-\ln\left(1+\frac{1}{5}I\right)\right)-\frac{1}{2}I\left(\ln\left(1-\frac{1}{239}I\right)-\ln\left(1+\frac{1}{239}I\right)\right)-\frac{1}{4}\pi$$

```
> combine(%);
```

$$0$$

Consider this integration example. Since the integral is real we expect a real output instead of the complex logarithms. The new code properly simplifies the output. When integrated properly to the integrator the nice real result will appear automatically.

```
> Int(x/tan(x),x=0..Pi/2) = int( x/tan(x), x=0..Pi/2 );
```

$$\int_0^{1/2\,\pi} \frac{x}{\tan(x)}\,dx = \frac{1}{2}\ln(1-I)\,\pi + \frac{1}{2}\ln(1+I)\,\pi$$

```
> combine(%,ln);
```

$$\int_0^{1/2\,\pi} \frac{x}{\tan(x)}\,dx = \frac{1}{2}\pi\ln(2)$$

Note, the complex logarithms come from the complex form for the antideritive, namely

```
> Int(x/tan(x),x) = int(x/tan(x),x);
```

$$\int \frac{x}{\tan(x)}\,dx = -\frac{1}{2}I\,x^2 + x\ln(1-\mathbf{e}^{(I\,x)}) - I\,\mathrm{polylog}(2,\mathbf{e}^{(I\,x)}) + x\ln(\mathbf{e}^{(I\,x)}+1) - I\,\mathrm{polylog}(2,-\mathbf{e}^{(I\,x)})$$

5. Complexity Measures

A ‘‘simple’’ (or simplified) expression may mean different things in different contexts, and sometimes the user is not satisfied with the output of the simplification because he/she was not able to specify the criteria for simplification. We propose to resolve this problem by introducing ‘‘complexity measures.’’ Here are two examples of useful complexity measures. The first is a syntactic measure which measures the printed size of the expression. The second measure counts the number of different functions appearing in the expression.

```
> synt := e -> length(e):
> funct := e -> nops(indets(e,function)):
```

In the following example f and g are two represenations of the same function:

```
> f := sin(x)/(cos(x)+2);
```

$$f := \frac{\sin(x)}{\cos(x)+2}$$

```
> g := normal( convert(f,tan) );
```

$$g := 2\,\frac{\tan\left(\frac{1}{2}\,x\right)}{3+\tan\left(\frac{1}{2}\,x\right)^2}$$

Here are the values of the syntactical and functional measures, respectively:

```
> synt(f), synt(g);
```

$$33, 57$$

```
> funct(f), funct(g);
```

$$2, 1$$

It often happens that several different results are generated (or can be generated) while simplifying the input expression; in that case the system should return the one that minimizes the complexity measure chosen by the user. Thus, if both f and g shown above are obtained during a simplification process, then f will be returned if the user wants short expressions, whereas g will be returned if the user wants an expression with a minimum number of function calls (other than the basic arithmetic operations). It is planned that this approach will be adopted in one of the next versions of Maple, probably by introducing an environment variable for the **simplify** command to be used as the complexity measure.