1.1 Introduction to package fsimp

The `fsimp` package provides a 'fullsimp' simplification function for expressions. This code was adapted from David Scherfgen's dssimp.mac code posted on the Maxima discussion list.

This code implements a simplification routine for Maxima CAS expressions using the core simplification routines for expressions invovling trigonometric, logarithmic, exponential, polynomial and other functions.

To use the package, place the files in your maxima search path and

load("fsimp.mac");

To simplify an expression `expr` call

`fullsimp(expr);`

The algorithm begins by searching through an expression for e.g. trig, log, exp, etc. and choosing a set of core simplification routines. Then it runs a poor-man's kind of simulated annealing routine. It's poor-man's because it's not really simulated annealing, but while it cycles through various core simplification routines (several times) it allows the intermediate expression to become slightly larger if it ultimately becomes smaller. The output is always the smallest size expression, where the size is computed by the lisp `consize` function plus the string length of the expression.

E. Majzoub.

1.2 Functions and Variables for fsimp

`fsdebug` [Variable]

Setting the variable `fsdebug` to 1 will allow the user to see the fsimp function operations, list selection of simplification routines, etc.

`fullsimp` (`expr`,[*simp1,simp2,...*]) [Function]

The `fullsimp` function takes an input expression `expr` and attempts to find a simpler form using the builtin simplification routines in core Maxima. These include: `[resimplify, expand, combine, radcan, ratsimp, rootscontract, xthru, multthru, factor, sqrtdenest, triglist,exptlist, loglist]`. Additional trigonometric simplifications are applied as well. The full list of simplifications is shown in the variable `simplist` which will be shown if the variable `fsdebug` is set to 1.

There are two ways to manipulate the simplifications in `simplist`. First, the option arguments to `fullsimp`, `simp1, simp2, etc.` are simplification routines the user wants *excluded* from `simplist`.

Second, the user may create a *list* called `fs_custom_simp`, and place any user defined simplification routines. For example the code

```
fs_custom_simp:[lambda([q], block([algebraic : true], ratsimp(q)))]
```

will create a simplification routine where the variable `algebraic` is set to `true`, while the `ratsimp` simplification is applied.

In general, `fullsimp` tries to find the smallest equivalent expression based on the Common Lisp `conssize` and string size of an expression.

The user may manipulate the order of simplifications through use of the exclusion option and the custom simplification option.

`fullsimp` is a simplifying function and can be added to the command line as shown in the last example below.

```
(%i1) load("fsimp.mac")$
(%i2) fullsimp((1+cos(t))/sin(t));
                                          t
(%o2)                                 cot(-)
                                          2
(%i3) fullsimp(cos(x)+%i*sin(x));
                                        %i x
(%o3)                                 %e
(%i4) (cos(t)+1)/((%i*sin(p)-cos(p))*sin(t)),fullsimp;
                                      %i p     t
(%o4)                              - %e     cot(-)
                                               2
```

If you wish to exclude particular simplification routines, add them to the arguments of fullsimp as shown below.

```
(%i1) load("fsimp.mac")$
(%i2) fullsimp((1+cos(t))/sin(t),fstrigsimp,trigsimp,trigreduce);
                                    cos(t) + 1
(%o2)                               ----------
                                      sin(t)
```

Appendix A Function and Variable index