Document Version 1.2 – FORMS.M Version 4.0

Simple Forms Calculator

William L. Burke

These notes describe a calculator implemented in Mathematica to simplify expressions involving differential forms. As the name implies, the intent was to make a minimal implementation, with little attention paid to efficient input or pretty output. It will get the signs right for you, and the simplicity of the implementation will demonstrate the efficiency of a rule-based system for this kind of programming. Setting up such a system is a good way to debug your understanding of forms.

The key idea behind all of this is that Mathematica manipulates expressions as trees. It looks for patterns, and when it finds a match, substitutes something else for the pattern. When it finds no further matches, or none at all, it returns the expression. I will use the "fallen tree" representation for drawing trees in ascii text streams. This is the style used in the trn newsreader. The trees are written on their sides, hence my name for it. The vector expression

$$\frac{\partial}{\partial x} + 2\frac{\partial}{\partial y} + \frac{\partial}{\partial z}$$

would be written as an expression tree in Mathematica which I will write in this notation

```
Plus--Vector--x
    |-Times--2
    |      \-Vector--y
    \-Vector--z
```

Coordinates

You might want to start working with forms in three dimensional Euclidean space. There are special definitions to simplify the input and output for this case in the file `euclid3.m`The coordinates there are x, y, and z. It is surprising how many of the rules for forms do not depend on any knowledge of the full list of coordinates. I didn't realize this until I developed this system.

The shorthand names for the basis forms defined in `euclid3.m` are `dx`, `dxdy`, `dxdydz` and so on. No spaces here, Mathematica treats these as a single symbol. The basis vectors are given by `px`, and so on. The "absent" symbol $\hat{dx}$ is denoted by `tx`.

There is also an output function `Efilter` that you can use to beautify the output, setting

```
In[]:=   $Post = Efilter
```

Vectors

A vector will be a linear combination of monomial terms. Each of these monomials will be a tree with one branch. The head of the tree is a node labeled "Vector". The symbol on the branch indicates the direction of the vector relative to the coordinate system. No rules are needed for these monomials to form a vector space.

Forms

Differential forms will be linear combinations of monomial terms. This time there will be a number of branches; an r-form will have r branches below a node labeled "Form". The 2-form $dx\,dy$ would be represented by the tree

```
         Form--x
             \-y
```

and written `Form[x,y]`. The shorthand in `euclid3.m` would let you write this `dxdy`, no space.

We need to explicitly accomodate the fact that a 0-form is a number with a rule

```
         Form[] := 1
```

The antisymmetry allows us to simplify expressions involving forms. This is not done continually, in the interests of efficiency, but the following rule is invoked as needed:

```
         FSimp := Form[ a__] :>
                    Signature[ Form[a]]  Sort[Form[a]]
```

Duality

The operation which contracts a vector with a form is denoted by the operator "angle". The expression

$$a\frac{\partial}{\partial x} \rfloor b\,dx\,dy\,dz$$

is represented by

```
angle--Times--a
     |      \-Vector--x
     \-Times--b
            \-Form--x
                  |-y
                  \-z
```

There are rules needed to enforce the bilinearity of this operation, such as

```
angle[ a_, b_ + c_] := angle[a,b] + angle[a,c]
```

which is the replacement

```
angle--a                  Plus--angle--a
     \-Plus--b       ->        |      \-b
           \-c                 \-angle--a
                                      \-c
```

The essential rule replaces

```
angle--Vector--u
     \-Form--u       ->          \-Form--a
           |-a                          \-b
           \-b
```

with due attention to the signs.

There are also rules needed to take care of the fact that we represent the zero vector and the zero forms by the number zero.

Wedge Product

The wedge produce is denoted by the tree

```
wedge--Form
     |-Form
     ...
     \-Form
```

I have put in some rules that let you denote the wedge product with `NonCommutativeMultiply`, `**`, but these are not fool proof at present. Intuitively you expect `**` to bind more weakly than `*`, but that can't be done in Mma as I understand it. So it gets confused on

```
dx ** 3 dy
```

Exterior Derivative

The exterior derivative operator is denoted `d[.]`. Functions need to have their arguments explicitly indicated, since the differentiations are done by the Mathematica operator `Dt`. Thus

```
In := d[ f[x,y] ]

        (1,0)                    (0,1)
Out = f      [x,y] Form[x] + f      [x,y] Form[y]
```

Lie Bracket

The Lie bracket (Jacobi bracket) is represented by

```
Lie--Vector
   \-Vector
```

and linear combinations.

Lie Derivative of Forms

This is represented by

```
Lie--Vector
   \-Form
```

and linear combinations.

Twisted Forms

These are put in using the "absent symbol" notation. The twisted form

$$\hat{d}x\hat{d}y$$

will be represented by the expression tree

```
TForm--x
     \-y
```

and written `TForm[x,y]`. The shorthand in `euclid3.m` would allow you to write this `txty`. The unit pseudoscalar is called `tmax`, and in Euclidean 3-space we have

```
tmax = TForm[x,y,z]
```

The `Length` of the variable `tmax` is used whenever a result depends upon the dimension of the space.

The wedge of an ordinary form with a twisted form in that order does not depend upon the dimension of the space, but the other two products do. Thus the exterior derivative of a twisted form can be computed without knowing the dimension of the space or the names of any coordinates beyond those given.

Pullback

Defined so far only for ordinary forms:

```
Pullback--Form
        \-Rule
```

where the second argument is the set of rules that describe the map. For the x-axis as a parametrized curve, the rules would be

```
pc = {x->s, y->0, z->0}
```

The points on the curve are found from the replacement

```
{x,y,z} /.  pc
```

and the pullback of a form alpha onto the x-axis would be found from

```
Pullback[pc, alpha]
```

Hodge Star

The present implementation is rather simple minded. It assumes that the coordinates are orthonormal, and that the coordinate t, if present, is timelike. Also both star and it's inverse are denoted by star:

```
star--Form          and          star--TForm
```

Testing

There is a file of test expressions that has been used to validate the system, and that should be used if you modify the rules to ensure that you have not broken anything. The file is called `trip.m`, and will run when you load it after `forms.m`.