FORM course: lecture 1

Ben Ruijl

ETH Zurich

Jul 27, 2018

FORM

- FORM is a symbolic manipulation toolkit
- It supports basic algebraic operations, and can do them very fast
- Memory is no limitation, only disk space (no swapping)
- Terms are processed one-by-one
- Useful for HEP problems: often we have terabytes of terms!

Obtaining FORM

- Download FORM 4.2 from: `https://github.com/vermaseren/form/releases`
- Tutorial: `https://www.nikhef.nl/~form/maindir/documentation/tutorial/online/online.html`
- Reference manual: `https://www.nikhef.nl/~form/maindir/documentation/reference/online/online.html`
- Compile:

```
./configure
make -j4
make install
```

Example program

Save the following as `prog1.frm`:

```
Symbols a,b;
Local F = (a+b)^2;
Print;
.end
```

and run

```
form prog1
```

Example program

```
Symbols a,b;  * define symbols
Local F = (a+b)^2;  * define expression
Print;  * print the expression
.end;  * end the program (and sort)
```

```
Time =        0.00 sec    Generated terms =          3
F                         Terms in output =          3
                          Bytes used      =        108

F =
  b^2 + 2*a*b + a^2;
```

Example program

```
Symbols a,b;  * define symbols
Local F = (a+b)^2;  * define expression
Print;  * print the expression
.end;  * end the program (and sort)
```

```
Time =         0.00 sec    Generated terms =          3
F                          Terms in output =          3
                           Bytes used      =        108

F =
  b^2 + 2*a*b + a^2;
```

Operations on terms

Form can mutate terms with id:

```
Symbols a,b,c;
Local F = (a+b)^6;
id a^2*b = c;   * replacement
Print;
.end
```

```
F =
  15*c^2 + 15*b^3*c + b^6 + 20*a*b^2*c
  + 6*a*b^5 + 6*a^3*c + a^6;
```

Operations on terms

Form can mutate terms with `id`:

```
Symbols a,b,c;
Local F = (a+b)^6;
id a^2*b = c;   * replacement
Print;
.end
```

```
F =
   15*c^2 + 15*b^3*c + b^6 + 20*a*b^2*c
   + 6*a*b^5 + 6*a^3*c + a^6;
```

FORM behaviour

- Form always expands terms
- Form processes expressions term by term
- This means that *only 1 term* should fit in memory, the rest can be read/written to disk
- Pro: memory is no limitation
- Con: operations on expressions are more difficult

When confused why certain operations don't exist, imagine that every expression is too big to fit in memory

The following FORM program will run (try it):

```
Auto Symbols x;  * all starting with x is a symbol
Local F = (x1+x2+x3+x4+x5+x6)^100;
.end
```

```
Time =       0.45 sec    Generated terms =     100000
              F        1 Terms left      =     100000
                         Bytes used      =    6106364

Time =       1.00 sec    Generated terms =     200000
              F        1 Terms left      =     200000
                         Bytes used      =   12519468
....
```

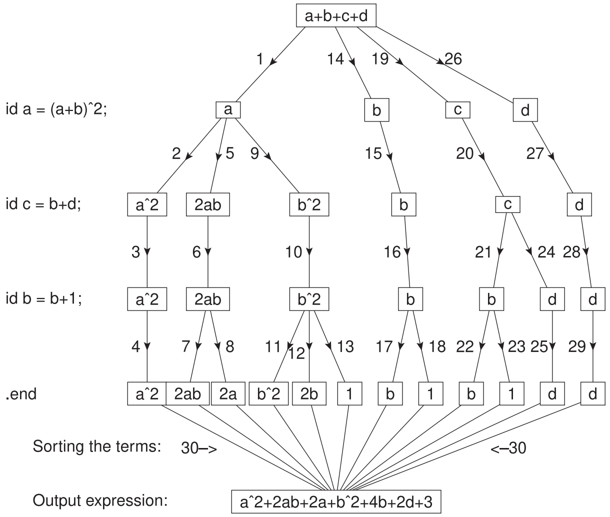
The following FORM program will run (try it):

```
Auto Symbols x;  * all starting with x is a symbol
Local F = (x1+x2+x3+x4+x5+x6)^100;
.end
```

```
Time =          0.45 sec    Generated terms =     100000
                  F       1 Terms left      =     100000
                            Bytes used      =    6106364

Time =          1.00 sec    Generated terms =     200000
                  F       1 Terms left      =     200000
                            Bytes used      =   12519468
....
```

Term flow


a+b+c+d
id a = (a+b)ˆ2;
id c = b+d;
id b = b+1;
.end
Sorting the terms:
30–>
<–30
Output expression:
aˆ2+2ab+2a+bˆ2+4b+2d+3
a
b
c
d
aˆ2
2ab
bˆ2
2a
2b
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Sorting

- At the end of a module, terms should be sorted to see if terms will merge
- A module is ended with `.sort` (or `.end`)
- Where to place the sort is up to the user

Sorting: good vs bad

```
Symbols a,b,c,d;
Local F = (a+b+c+1)^6;
id a = -c+d+1;
id b = -d+1;
Print;
.end
```

Generates 924 terms and has 1 in the output...

Sorting: good vs bad

```
Symbols a,b,c,d;
Local F = (a+b+c+1)^6;
id a = -c+d+1;
.sort
id b = -d+1;
.end
```

First sort:

```
Generated terms = 462
Terms in output = 28
```

Second sort:

```
Generated terms = 84
Terms in output = 1
```

Identity statements I

- Patterns for id statements can be any terms
- Use wildcards `var?` to match any object of the same type

```
S x,y;   * short for Symbol
L F = x^2 + y;
id x? = 5;   * match any symbol with x?
Print;
.end
```

yields

```
F = 30
```

Identity statements II

Patterns can be more complicated:

```
S x,y,n;
L F = x^2 + y;
id x?^n? = x^(n + 1);
Print;
.end
```

yields

```
F = y^2 + x^3
```

Identity statements III

- Restrictions can be placed by a set `{1,..}` or a number range `{>5}`
- A statement can be repeated with `repeat`

```
S x,n;
L F = x^10;
repeat id x^n?{>1} = x^(n-1) + x^(n-2);
Print;
.end
```

yields

```
F = 34 + 55*x;
```

Functions

- Functions for non-commutative functions
- CFunctions for commutative functions

```
S a,b,c;
CF f;  * short for CFunctions
Local F = f(1,2,c);
id f(1,2,b?) = f(1,2,b?+1);
Print;
.end
```

yields:

```
   F = f(1,2,c+1);
```

Ranged wildcards

A wildcard starting with a ? indicates a range:

```
S x;
L F = f(1,2,x,3,4);
id f(?a,x,?b) = f(?b,?a);
Print;
.end
```

yields

```
F = f(3,4,1,2);
```

Applying statements to arguments

id-statements are only applied at ground-level:

```
S x,y;
L F = f(x*y);
id x = 5;   * does not match
argument f;
  id x = 6;
endargument;
Print;
.end
```

yields

```
F = f(6*y)
```

If statements

```
S x,y;
L F = f(2) + f(5);
if (match(f(x?{>4})));
  id f(x?) = f(x + 1);
else;
  id f(x?) = f(x - 1);
endif;
Print;
.end
```

yields

```
F = f(1) + f(6)
```

Bracketing I

- Powers of variables can be extracted
- The terms are not nested for real, but information about brackets can be used in the next module

```
S x,y,z;
L F = x*y + x^2*y + x^2*z + 2;
Bracket x;   * extract powers of x
Print;
.end
```

```
F = + x * ( y )
    + x^2 * ( z + y )
    + 2;
```

Bracketing II

- Brackets can be indexed in the next module

```
S x,y,z;
L F = x*y + x^2*y + x^2*z + 2;
Bracket x;
.sort
L G = F[x^2];
Print G;   * only print G
.end
```

```
G = z + y
```

Bracketing III

- Bracketed content can be collected in a function if it fits in memory

```
S x,y,z;
L F = x*y + x^2*y + x^2*z + 2;
Bracket x;
.sort
CF f;
Collect f;
Print;
.end
```

```
F = f(z + y)*x^2 + f(y)*x + f(2)
```

Vectors and indices

Contraction and Einstein summation:

```
Index i1,i2,i3;
Vector p1,p2,p3;
Local F = p1(i1)*(p2(i1)+p3(i3))*(p1(i2)+p2(i3));
Print;
.end
```

yields:

```
F = p1(i1)*p1(i2)*p3(i3) + p1(i1)*p2.p3
    + p1(i2)*p1.p2 + p2(i3)*p1.p2;
```

Make sure an index does not appear more than twice in a term!

Traces and gamma matrices

```
S D;
Index i1=D,i2=D;  * D-dimensional indices
Vector p1,p2;
Local F1 = g_(1, i1, i1);  * gamma matrices
Local F2 = g_(1, p1, i2);
Local F3 = g_(1, p1, p2);
tracen 1;  * n-dimensional trace of spin line 1
Print;
.end
```

```
F1 = 4*D;
F2 = 4*p1(i2);
F3 = 4*p1.p2;
```

Feynman rule application

```
S vhhg, gh, gl;  * ghost-gluon vertex, ghost, gluon
I i1,i2;
V Q,p1,p2,p3,p4;
CF vx,prop;
L F = vx(Q,p1,p2,i1,vhhg)
      *vx(-p1,-Q,-p2,i2,vhhg)
      *prop(p1,i1,i2,gl)*prop(p2,gh);

id prop(p1?,i1?,i2?,gl) = d_(i1,i2)/p1.p1;
id prop(p1?,gh) = 1/p1.p1;
id vx(p1?,p2?,p3?,i1?,vhhg) = -i_*vx(p1,p2,p3)*p1(i1);
```

```
F = vx(-p1,-Q,-p2)*vx(Q,p1,p2)*Q.p1*p1.p1^-1*p2.p2^-1;
```

Exercises

- Write a program that can differentiate polynomials
- Write a program that expands $\ln(1-x)$ up to a certain power of x (see `sum_` in manual)
- Substitute the expanded form of $x = 1 - e^y$ into the result
- Try to make the program faster for higher powers of x