Polynomial Algebra in Form 4

Jan Kuipers

Nikhef theory group
Amsterdam, The Netherlands

ACAT — 6$^{\text{th}}$ September 2011

Introducing Form 4

- Form is a symbolic algebra package
- It is developed at Nikhef by Jos Vermaseren et al.
- Recently, version 4.0β has been released
- It includes many new features
- This talk is about one of them: polynomial algebra

Overview of polynomial algebra

- Polynomial algebra in Form basically consists of:
  - Greatest common divisor
  - Factorization
  - PolyRatFuns
- All three are discussed with examples
- Next, an analytic version of “Mincer” and future applications are discussed

Polynomial manipulation

- Distributed variable sparse and degree dense representation of polynomials is used:

$$p = \sum_{\text{terms}} a x^i y^j z^k$$

- Needs conversions from Form expressions, but is faster
- Fast algorithms for multiplying and division using heaps are implemented

Greatest common divisor

- The function gcd_ returns the greatest common divisor of its arguments
- Example code:

```
Symbols x,y;
Local E = gcd_(x^2+x*y, y^2+x*y);
Print;
.end

E =
   y + x;
```

- Note: Form 3 would return `E = 1`

Greatest common divisor algorithms

- For small polynomials, a heuristic that substitutes integers and performs integer gcd calculations is used
- For large polynomials, Zippel's modular algorithm is used
- Efficient for both sparse and dense polynomials
- Speed is comparable to Mathematica

Factorization of function arguments

- The statement `FactArg` factorizes the argument of a function
- Example code:

```
Symbols k,i,f,e,h;
CFunction N;
Local E = N(k*f+k*e+k*h+i*f+i*e+i*h);
FactArg N;
Print;
.end

E =
   N(i + k,h + e + f);
```

- Note: in Form 3 this function argument does not factorize
- For backward compatibility: `On OldFactArg;`

Factorization of dollar variables

- The preprocessor statement `#Factorize` factorizes a dollar variable
- Example code:

```
Symbols x,y;
#$a = x^2-y^2;
#Factorize $a;

#do i=1,`$a[0]'
  #write "%$", $a[`i'];
#enddo
```

```
-y+x
y+x
```

- Analogous `Factorize` for run time factorization
- Syntax for factorizing expressions is coming soon

Factorization algorithms

- For univariate polynomials Berlekamp's algorithm is used
- Multivariate polynomials are reduced to univariate and afterwards Hensel lifting is used to reconstruct multivariate factors
- To factorize this polynomial

```
-6272714818668017*a^35*b^22*c^20*d^9*e^21 -
 6867348605700329*a^34*b^33*c^19*d^11*e^36 +
 323798222821062*a^34*b^20*c^29*d^8*e^18 +
         (... 10 more terms ...)        +
 2081169781417560*a^28*b^10*c^13*d^27*e^12 -
 285878431480222*a^28*b^4*c^25*d^13*e^13 -
 520827763173144*a^27*b^4*c^19*d^24*e^11
```

Form takes 9 sec and Mathematica takes 900 sec

PolyRatFuns

- Analogous to Form's `PolyFun`, rational coefficients can be used with `PolyRatFun`
- The first argument of the function serves as numerator and the second as denominator
- Example code:

```
Symbols x,y,z;
CFunction f;
PolyRatFun f;

Local E = x * f(y,z) + x * f(y,1-z)
        + x^2 * f(y^2-1,y-1);
Print;
.end
```

```
E =
   x*f( - y,z^2 - z) + x^2*f(y + 1,1);
```

Application: MincerExact

- Mincer (program for 3-loop massless propagator diagrams) works in expansions in $\epsilon = (4 - D)/2$, typically up to 6th power
- Big tables with expansions of Pochhammer symbols and alike are needed
- Using PolyRatFuns MincerExact needs no expansions at all
- Code is much cleaner/shorter and only slightly slower
- Results for Mellin moments look like:

```
VALUE=GschemeConstants(0,0)^2*GschemeConstants(2,0)*
cf^2*rat(-192*ep^7+944*ep^6-1824*ep^5+1680*ep^4-640*
ep^3-48*ep^2+96*ep-16,12*ep^3+36*ep^2+33*ep+9)
```

Application: MincerExact

- Now have a closer look at an answer of MincerExact:

```
Symbol ep,a,b,c,d;
CFunction rat,num,den;
Local E = rat(-192*ep^7+944*ep^6-1824*ep^5+1680*ep^4
  -640*ep^3-48*ep^2+96*ep-16,12*ep^3+36*ep^2+33*ep+9);
id rat(a?,b?) = num(a)*den(b);
FactArg den;
ChainOut den;
id den(a?number_) = 1/a;
Print +f +s;
.sort

   E =+1/3*num(-16+96*ep-48*ep^2-640*ep^3+1680*ep^4
      -1824*ep^5+944*ep^6-192*ep^7)*den(1+ep)*
      den(1+2*ep)*den(3+2*ep);
```

Application: MincerExact

- Make a partial fraction expansion:

```
SplitArg den;
FactArg den;
id den(a?,ep,b?) = 1/b*den(a/b,ep);
Repeat id den(a?,ep)*den(b?,ep) =
          (den(a,ep)-den(b,ep)) / (b-a);
Print +f +s;
Bracket num;
.sort
```

```
   E=+num(-16+96*ep-48*ep^2-640*ep^3+1680*ep^4
          -1824*ep^5+944*ep^6-192*ep^7)*(
         +1/6*den(1/2,ep)
         +1/6*den(3/2,ep)
         -1/3*den(1,ep)
         );
```

- Rewrite it once more:

```
id num(a?) = a;
Repeat id ep*den(a?,ep) = 1 - a*den(a,ep);
Print +f +s;
.sort

   E= -7828/3
      +3803/3*ep
      -488*ep^2
      +380/3*ep^3
      -16*ep^4
      +243/8*den(1/2,ep)
      +153125/24*den(3/2,ep)
      -5120/3*den(1,ep);
```

Application: MincerExact

- Finally, expand around $\epsilon = 0$ up to 6th order:

```
Symbol ep(:6);
Repeat id den(a?,ep) = 1/a - ep/a * den(a,ep);
Print +f +s;
.end
```

```
   E= -16/9
      +464/27*ep
      -4960/81*ep^2
      +21152/243*ep^3
      +65264/729*ep^4
      -1744048/2187*ep^5
      +16761728/6561*ep^6;
```

Future work

- Code simplification
- Implement Buchberger's algorithm for finding Gröbner basis
- Implement LaPorta's algorithm for reducing Feynman integrals
- Determine all Mellin moments N for D.I.S.

The end

Questions?