Introduction to FORM

March 25, 2025

Bakar Chargeishvili

Computer Algebra and Particle Physics - CAPP 2025,
March 24 – 28, 2025

Karlsruhe Institute of Technology

Plan for today

- Preprocessor commands
- Dollar variables
- Alternatives to FORM's default preprocessor
- Parallelization
- Color algebra
- Diagram generation
- Floating point engine

FORM Preprocessor: The Engine Behind Symbolic Algebra

- Preprocessor = powerful text manipulation engine that runs **before** actual FORM code
- Handles code generation, repetitive tasks, and conditional compilation

Key Commands:

- `#define` - Create macros and constants
- `#if/#else/#endif` - Conditional code blocks
- `#do/#enddo` - Loops for generating repetitive expressions
- `#include` - Import external files/libraries
- `...`-operator is also a preprocessor command

The preprocessor transforms your code before FORM even sees it - essential for managing complex calculations

Preprocessor (dollar) variables

- Define new preprocessor variables using `#$a = 0` notation and use it without `#`-sign.
- Dollar variables are shared between preprocessor and algebraic compiler
  - Not all preprocessor variables are shared

Let's analyze this program:

```
1   #-
2   #$a = 25;
3   *Try commenting the next line
4   L test = $a;
5
6   $a = 30;
7   #do i=1,5
8       #message Iteration number `i'
9   #enddo
10
11  #message Can we print $a this way?: `$a'
12
13  .sort
14  L test = $a;
15  Print;
16  .end
```

Store captured wildcards

One can use dollar varibales to store captured wildcards for later use:

```
1   #-
2   CF func;
3   Auto S i;
4
5   L test1 = func(1,2,3,4);
6
7   id func(i1?$a,i2?$b,?k)=func(i1,i2,?k);
8
9   #message we got `$a' and `$b'
10
11  .sort
12  #message we got `$a' and `$b'
13  L test2 = func($a,$b);
14
15  Print;
16  .end
```

To .sort or not to .sort - preprocessor edition

Plug in the expansion of $\ln(1+x)$ in expansion of $e^x - 1$ around $x = 0$:

$$\ln(1+x) = \sum_{i=1}^{N} (-1)^{i+1} x^i / i + \ldots ,$$

$$e^x - 1 = \sum_{i=1}^{N} \frac{x^i}{i!} + \ldots = x\left(1 + \frac{x}{2}\left(1 + \frac{x}{3}\left(1 + \frac{x}{4}(\ldots)\right)\right)\right) .$$

```
1   #-
2   #define N "60"
3   On Statistics;
4   Symbol i, x(:`N'), y(:`N');
5   *define ln(1+x)
6   Local X =  - sum_(i, 1, `N', sign_(i)/i*x^i);
7   *tag x by y
8   id  x = x*y;
9   *Telescope formula
10  #do i=2,`N'+1
11      id  y = 1 + x*y/`i';
12  *     .sort
13  #enddo
14  Print;
15  .end
```

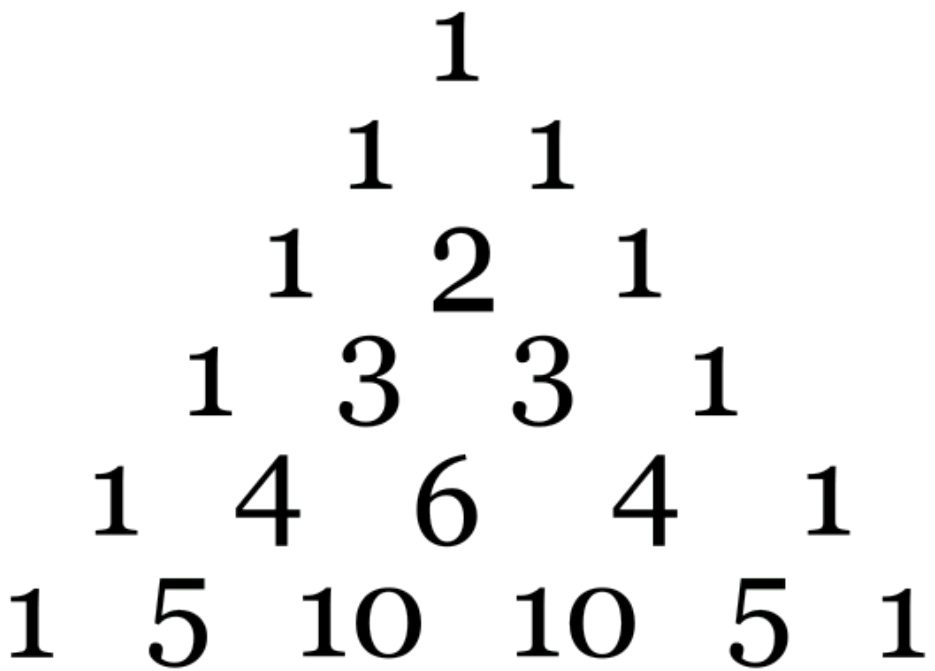
Preprocessing - Poweruser's Edition

- We can use other programming languages as a preprocessor:
- Write FORM code using other scripting languages and postprocess the result as desired.

Example, using python:

```
1  import subprocess, re
2  code = '''
3  #-
4  S a,b;
5
6  L test = (a+b)^30;
7  '''
8  form_file_name = '/tmp/FORM_Code.frm'
9  with open(form_file_name,'w') as f:
10     f.write(code)
11 result = subprocess.run(["form","-f",form_file_name],
12                         capture_output=True,text=True).stdout
13 result=result.replace('\n','')
14 result=result.replace(' ','')
15 result=result.replace(';','')
16 result=re.sub('^.*=','',result)
17 print(result)
```

Revisiting Pascal's triangle



- Try to calculate the sum of all elements at the depth i which are multiplets of 3.
  - Using only `FORM`.
  - Using `FORM` to construct the triangle, but another language as pre- (and post-)processor.

Setup parameters of FORM

- FORM uses setup parameters to set the size of various buffers needed during the calculation.
- The default parameters might not be suited for your problem.
- Example of modified values which works for moderate size problems (up to couple million terms depending on the size of a term):

```
#-
#:SmallSize 5000000
#:LargeSize 20000000
#:WorkSpace 50000000
#:MaxTermSize 300000
#:TermsInSmall 30000
```

- A script to determine the sane values for your specific machine: https://github.com/tueda/formset
  - Might not work on *BSD-based systems as intended.
- Check out Chapter 17 in the manual to understand how to choose the values.

Parallelization

- Parallelization in FORM is as easy as calling `tform` (installed by default) executable instead of `form`.

- For example

```
tform -w 64 YourFile.frm
```

  Parallelizes the calculation automatically on 64 cores.

Caveat

- Preprocessor calls and dollar variables might still force the linear execution. Use them economically.

```
S   x,a,b;
CF  f;
L   F = f(a+b) + f(a+2*b);
.sort
id  f(x?$x) = f(x);
#do i=1,1
    Multiply,$x;
#enddo
.end
```

Color algebra

- FORM can be expanded with the powerful addon for the color algebra
- Obtain the library and place it in your workdir:
  https://www.nikhef.nl/~form/maindir/packages/color/color.h

Let's calculate:

$$\mathrm{tr}\left(T^{\alpha}\,T^{\beta}\right)$$

```
1  #include- color.h
2  S Na, Nc;
3  Auto I i=N,j=Na,k=N;
4  G Q1 = T(i1,i2,j1)*T(i2,i1,j2);
5
6  .sort
7  #call docolor
8  Print;
9  .end
```

Diagram generation

- Since v5 FORM supports diagram generation

Example from ϕ^3 theory:

```
1   Model PHI3;
2       Particle phi,1;
3       Vertex phi,phi,phi:g;
4   EndModel;
5   Vector Q,Q1,...,Q7,p,p0,...,p21;
6   Indices j1,j2,i1,...,i21;
7   Set QQ:Q1,...,Q4;
8   Set pp:p1,...,p4;
9
10  L test = diagrams_(PHI3,{phi,phi},{phi},QQ,pp,0);
11  Print;
12  .end
```

Output:

```
test =
     topo_(1)*node_(1,1,E(-Q1))*node_(2,1,e(-Q2))*node_(3,1,mu(-Q3))*
     node_(4,1,MU(-Q4))*node_(5,Qe,e(Q1),E(Q2),photon(-p1))*
     node_(6,Qmu,mu(Q4),MU(Q3),photon(p1));
```

Floating point evaluations

- Starting with version 5.0 FORM is also equiped with arbitrary floating point engine:

Evaluate $\pi\zeta(5,3)$:

```
1   #-
2   #define Pi "22/7"
3   #startfloat 500,15
4
5   L test = `Pi'*mzv_(5,3);
6
7   Evaluate;
8
9   Print;
10  .end
```

Moving forward

- It's impossible to cover all the great features of FORM in details in 3 lecturs
- Read the manual: https://www.nikhef.nl/~form/maindir/documentation/reference/html/manual.html
- Beginner-friendly (somewhat outdated, but still very useful) manual by Andr´e Heck, *FORM for Pedestrians*: https://www.nikhef.nl/~form/maindir/documentation/tutorial/online/online.html
- When judging FORM against other computer algebra systems remember slide 6 from yesterday.

Thanks for your attention!

Karlsruhe Institute of Technology