FORM version 5 update

Takahiro Ueda

Juntendo U.

22 June 2026

FORM Developer’s Workshop 2026

FORM 5.0

Released	27 January 2026
Preprint	“FORM Version 5.0” Davies, Kaneko, Marinissen, TU, Vermaseren arXiv:2601.19982
Repository	github.com/form-dev/form

This talk FORM 5.0 recap for this workshop

FORM 5.0

Released	27 January 2026
Preprint	“FORM Version 5.0” Davies, Kaneko, Marinissen, TU, Vermaseren arXiv:2601.19982
Repository	github.com/form-dev/form

This talk FORM 5.0 recap

for this workshop

Outline

Key new features

- Diagram generation
- Floating-point arithmetic
- FLINT backend for polynomials

Other changes

Summary

Diagram generation

Generate and manipulate diagrams in one script

Built-in diagram generator

- FORM 5 can generate Feynman diagrams directly from a FORM script
  Based on the graph generator used in the GRACE system; Kaneko '95
- `Model` ... `EndModel`: define the fields and allowed vertices
- `diagrams_(...)`: specify the process, momenta, the loop order, and optional filters
- The result is a FORM expression, so diagrams can be selected, grouped, and further manipulated using standard pattern matching
  See also Marco Klann's talk

Built-in diagram generator

- FORM 5 can generate Feynman diagrams directly from a FORM script
  Based on the graph generator used in the GRACE system; Kaneko '95
- `Model` ... `EndModel`: define the fields and allowed vertices
- `diagrams_(...)`: specify the process, momenta, the loop order, and optional filters
- The result is a FORM expression, so diagrams can be selected, grouped, and further manipulated using standard pattern matching
  See also Marco Klann's talk

Built-in diagram generator

- FORM 5 can generate Feynman diagrams directly from a FORM script
  Based on the graph generator used in the GRACE system; Kaneko '95
- `Model` ... `EndModel`: define the fields and allowed vertices
- `diagrams_(...)`: specify the process, momenta, the loop order, and optional filters
- The result is a FORM expression, so diagrams can be selected, grouped, and further manipulated using standard pattern matching
  See also Marco Klann's talk

Built-in diagram generator

- FORM 5 can generate Feynman diagrams directly from a FORM script
  Based on the graph generator used in the GRACE system; Kaneko '95
- `Model` ... `EndModel`: define the fields and allowed vertices
- `diagrams_(...)`: specify the process, momenta, the loop order, and optional filters
- The result is a FORM expression, so diagrams can be selected, grouped, and further manipulated using standard pattern matching
  See also Marco Klann's talk

Models

```
Model PHI3;
  Particle phi,+1;
  Vertex phi,phi,phi:g;
EndModel;
```

$$\mathscr{L}_{\text{int}} = \frac{g}{3!}\phi^3$$

A model defines the fields and interactions for the generator

- `Particle phi,+1` declares the scalar field ϕ
- `Vertex phi,phi,phi:g` declares the cubic interaction with coupling g

Models

```
Model PHI3;
  Particle phi,+1;
  Vertex phi,phi,phi:g;
EndModel;
```

```
Model PHI4;
  Particle phi,+1;
  Vertex phi,phi,phi,phi:g^2;
EndModel;
```

Models

```
Model PHI3;
  Particle phi,+1;
  Vertex phi,phi,phi:g;
EndModel;
```

```
Model PHI4;
  Particle phi,+1;
  Vertex phi,phi,phi,phi:g^2;
EndModel;
```

```
Model QCD;
  Particle qua,QUA,-2;
  Particle gho,GHO,-1;
  Particle glu,+3;
  Vertex qua,QUA,glu:g;
  Vertex gho,GHO,glu:g;
  Vertex glu,glu,glu:g;
  Vertex glu,glu,glu,glu:g^2;
EndModel;
```

Generating diagrams

```
Local F = diagrams_(PHI3,{phi},{phi},{p1,p2},{k1,...,k5},2,
                    `NoTadpole_'+`OnePI_');
```

- `PHI3`: model name
- `{phi}` and `{phi}`: incoming and outgoing particles
- `{p1,p2}` and `{k1,...,k5}`: external and internal momenta
- `2`: two-loop order
- Options `NoTadpole_` and `OnePI_`: `QGRAF`-compatible graph filters that exclude tadpole graphs and keep only 1PI graphs

Generating diagrams

```
1 Local F = diagrams_(PHI3,{phi},{phi},{p1,p2},{k1,...,k5},2,
2                     `NoTadpole_'+`OnePI_');
```

- `PHI3`: model name
- `{phi}` and `{phi}`: incoming and outgoing particles
- `{p1,p2}` and `{k1,...,k5}`: external and internal momenta
- `2`: two-loop order
- Options `NoTadpole_` and `OnePI_`: `QGRAF`-compatible graph filters that exclude tadpole graphs and keep only 1PI graphs

Generating diagrams

```
1 Local F = diagrams_(PHI3,{phi},{phi},{p1,p2},{k1,...,k5},2,
2                     `NoTadpole_'+`OnePI_');
```

- `PHI3`: model name
- `{phi}` and `{phi}`: incoming and outgoing particles
- `{p1,p2}` and `{k1,...,k5}`: external and internal momenta
- `2`: two-loop order
- Options `NoTadpole_` and `OnePI_`: `QGRAF`-compatible graph filters that exclude tadpole graphs and keep only 1PI graphs

Generating diagrams

```
1 Local F = diagrams_(PHI3,{phi},{phi},{p1,p2},{k1,...,k5},2,
2                     `NoTadpole_'+`OnePI_');
```

- `PHI3`: model name
- `{phi}` and `{phi}`: incoming and outgoing particles
- `{p1,p2}` and `{k1,...,k5}`: external and internal momenta
- `2`: two-loop order
- Options `NoTadpole_` and `OnePI_`: `QGRAF`-compatible graph filters that exclude tadpole graphs and keep only 1PI graphs

Generating diagrams

```
1 Local F = diagrams_(PHI3,{phi},{phi},{p1,p2},{k1,...,k5},2,
2                     `NoTadpole_'+`OnePI_');
```

- `PHI3`: model name
- `{phi}` and `{phi}`: incoming and outgoing particles
- `{p1,p2}` and `{k1,...,k5}`: external and internal momenta
- `2`: two-loop order
- Options `NoTadpole_` and `OnePI_`: `QGRAF`-compatible graph filters that exclude tadpole graphs and keep only 1PI graphs

Generating diagrams

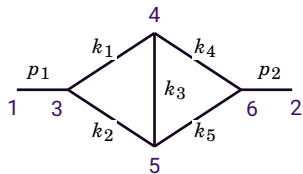
```
1 Local F = diagrams_(PHI3,{phi},{phi},{p1,p2},{k1,...,k5},2,
2                     `NoTadpole_'+`OnePI_');
```

- `PHI3`: model name
- `{phi}` and `{phi}`: incoming and outgoing particles
- `{p1,p2}` and `{k1,...,k5}`: external and internal momenta
- `2`: two-loop order
- Options `NoTadpole_` and `OnePI_`: `QGRAF`-compatible graph filters that exclude tadpole graphs and keep only 1PI graphs

Diagrams as graph data

Each generated diagram is represented as a single term
Graph data is encoded by `topo_` and `node_` factors

```
1/2
*topo_(1)
*node_(1,1,phi(-p1))
*node_(2,1,phi(-p2))
*node_(3,g,phi(p1),phi(-k1),phi(-k2))
*node_(4,g,phi(k1),phi(-k3),phi(-k4))
*node_(5,g,phi(k2),phi(k3),phi(-k5))
*node_(6,g,phi(p2),phi(k4),phi(k5))
```



- `topo_(1)`: topology label
- `node_(4,...)`: vertex with fields and momenta

Floating-point arithmetic

Finally, floats in FORM

Floating-point coefficients in action

Evaluate numerical constants inside a symbolic expression

https://gmplib.org/ https://www.mpfr.org/

```
#StartFloat 40d

Symbols r;

Local F = 4*pi_*r^3/3;

Evaluate pi_;
Print +s;
```

```
   F =
       + 4.188790204786390984616857844372670512263e+00*r^3
       ;
```

- `#StartFloat 40d` sets the precision to 40 decimal digits
- `Evaluate pi_` evaluates π in the coefficient $4\pi/3$
- The rest of the expression remains symbolic

MZVs and Euler sums

Evaluate supported special sums numerically

```
#StartFloat 10d, MZV=3

Symbols a,b;

Local F = a*mzv_(2,1)+b*euler_(-1,2);

Evaluate mzv_, euler_;
Print +s;
```

```
   F =
        + 2.695764795e-01*b
        + 1.202056903e+00*a
       ;
```

- `Evaluate` replaces supported sums by floating-point values
- `MZV=3` sets the maximum weight for the MZV/Euler-sum tables
- Numerical evaluation is currently limited to real arguments

Rational and floating coefficients

Convert coefficients between exact and floating-point form

```
#StartFloat 11d

Local F = 103993/33102;

ToFloat;
Print "<float> %t";

ToRational;
Print "<rational> %t";
```

```
<float>   + 3.141592653e+00
<rational>   + 103993/33102
```

- `ToFloat` converts rational coefficients to floating-point coefficients
- `ToRational` reconstructs rationals using continued fractions

Practical details

Internal representation is protected; cleanup is explicit

```
float_(...)
#EndFloat
StrictRounding 20d;
Chop 1e-30;
```

- `float_` is protected from pattern matching and `Normalize`
- `#EndFloat` closes the floating-point system
- `StrictRounding` rounds floating-point numbers to a chosen precision
- `Chop` removes values below a given absolute threshold

FLINT backend for polynomials

FLINT speeds your code up

User-transparent FLINT backend

Same FORM script, faster polynomial arithmetic

```
On flint;   * by default
```

- When FORM is linked against FLINT, supported polynomial operations use it internally https://flintlib.org
- It is enabled by default, and normal scripts do not need to change
- Use `Off flint` when you need the old backend

What is accelerated?

FLINT is used for selected low-level polynomial operations

Uses FLINT internally

- `PolyRatFun`, `FactArg`, `FactDollar`
- `div_`, `rem_`, `mul_`
- `gcd_`, `inverse_`

Not currently routed through FLINT

- factorisation of full expressions
- `Modulus`

Caveat

The printed output may differ depending on whether FLINT is enabled, but the mathematical expression remains the same

Benchmark profile

The gain depends on the polynomial workload

Workload	`Off flint;`	`On flint;`	Speed-up
Univariate moments (minceex $N = 12$)	556 s	475 s	1.2x
Four-loop propagator IBP (forcer 17-prop)	641 s	308 s	2.1x
Multivariate box IBP (mbox1l (3,3,3,3))	588 s	11.2 s	53x

Wall time, 24 `TFORM` workers, FLINT 3.2.1

Largest gains appear when multivariate rational-polynomial arithmetic dominates

See also Joshua Davies's Friday talk

Other changes

Small changes that matter in practice

Free placement of dollar variable ModuleOptions

Dollar-variable ModuleOptions

```
#procedure SortLessProcedure
  $prcLocalDollar = count_(x,1);
  ModuleOption local $prcLocalDollar;
* more executable statements here
#endprocedure

#call SortLessProcedure
* more executable statements here
.sort
```

- No longer restricted to the end of the module
- Useful for procedures that do not perform their own `.sort`

Safer Mathematica output

Safer interchange output for Mathematica

FORM input

```
Vector p,q;
Symbol x;

Local F = (1 + x) * (1 + p.q)^2;

Format Mathematica;
Bracket x;
Print +s;
```

version 4.3.1

```
F =

    + x * (
        + 1
        + 2*p.q
        + p.q^2
        )

    + 1
        + 2*p.q
        + p.q^2
      ;
```

version 5.0.0

```
F = (

    + x * (
        + 1
        + 2*(p.q)
        + (p.q)^2
        )

    + 1
        + 2*(p.q)
        + (p.q)^2
      );
```

Dot products and whole expressions are now enclosed in parentheses, making multi-line output safer to import into Mathematica

Sorting large jobs: memory and sort files

Tuning memory use and sort-file I/O

Compress temporary sort files

```
On compress zstd;
```

Helpful when sort-file I/O dominates

https://facebook.github.io/zstd/

Release buffers after one large module

```
#SortReallocate
```

Useful after an unusually large module

Release buffers at module boundaries

```
On SortReallocate;
```

More automatic, but may add overhead

Diagnostics

Logs, warnings, and bug reports become easier to read

Readable logs

```
On HumanStats;
```

Make large counters readable

New warnings

```
missing $-variable ModuleOption
conflicting ModuleOptions
```

`TFORM` reports missing or conflicting dollar-variable `ModuleOptions`

For bug reports

```
form -vv
```

Print build features and linked library versions

Summary

FORM 5: more workflow inside FORM

Diagrams

Generate diagrams as FORM data

Floats

Evaluate numerical constants inside symbolic expressions

FLINT

Same scripts, faster polynomial arithmetic

Practical updates

Other changes include safer output, sorting, and diagnostics

Together, these are practical building blocks for FORM workflows