Further developments of FORM

Takahiro Ueda

Seikei University, Tokyo

with Toshiaki Kaneko, KEK, Tsukuba
Ben Ruijl, ETH Zürich
Jos Vermaseren, Nikhef, Amsterdam

14 March 2019
Saas Fee, Switzerland
ACAT 2019

Contents

Introduction

Form is

FORM is

a toolkit for formula manipulation

https://github.com/vermaseren/form
Vermaseren *et al.*

Form is

a toolkit for formula manipulation

https://github.com/vermaseren/form
Vermaseren *et al.*

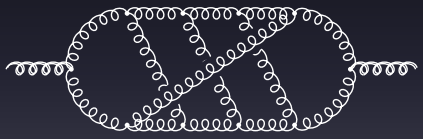
Efficient, especially for very big expressions

FORM is

a toolkit for formula manipulation

https://github.com/vermaseren/form
Vermaseren *et al.*

Efficient, especially for very big expressions

e.g., 

- TU's talk in ACAT 2016
- Ruijl's talk in ACAT 2017

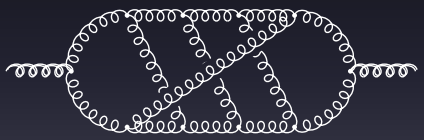
Parallelisation available with `Pthreads` or `MPI`

FORM is

a toolkit for formula manipulation

https://github.com/vermaseren/form
Vermaseren *et al.*

Efficient, especially for very big expressions

e.g., 

- TU's talk in ACAT 2016
- Ruijl's talk in ACAT 2017

Parallelisation available with `Pthreads` or `MPI`

Term rewriting with imperative programming:
Define mathematical expressions you want to manipulate
and specify how you want to manipulate

Example

fibonacci.frm

```
1  CFunction fib;
2  Symbol n;
3  Local F = fib(10);  * Find the 10th Fibonacci number.
4
5  repeat id fib(n?{>=3}) = fib(n - 1) + fib(n - 2);
6  id fib(2) = 1;
7  id fib(1) = 1;
8
9  Print;
10 .end
```

Run FORM as `form fibonacci.frm`, then...

Example

```
FORM 4.2.0 (Jul  6 2017, v4.2.0) 64-bits         Run: Sat Mar  9 12:39:06 2019
    CFunction fib;
    Symbol n;
    Local F = fib(10);  * Find the 10th Fibonacci number.

    repeat id fib(n?{>=3}) = fib(n - 1) + fib(n - 2);
    id fib(2) = 1;
    id fib(1) = 1;

    Print;
    .end

Time =       0.01 sec    Generated terms =         55
               F         Terms in output =          1
                         Bytes used      =         20

   F =
      55;

  0.01 sec out of 0.00 sec
```

Example

```
FORM 4.2.0 (Jul  6 2017, v4.2.0) 64-bits         Run: Sat Mar  9 12:39:06 2019
    CFunction fib;
    Symbol n;
    Local F = fib(10);  * Find the 10th Fibonacci number.

    repeat id fib(n?{>=3}) = fib(n - 1) + fib(n - 2);
    id fib(2) = 1;
    id fib(1) = 1;

    Print;
    .end

Time =       0.01 sec    Generated terms =         55
               F         Terms in output =          1
                         Bytes used      =         20

   F =
      55;

  0.01 sec out of 0.00 sec
```

← stupidly inefficient code but let's go further…

Example

```
3 Local F = fib(30);  * Find the 30th Fibonacci number.
```

Example

```
3 Local F = fib(30);  * Find the 30th Fibonacci number.
```

```
Time =       0.21 sec    Generated terms =     100000
               F       1 Terms left      =          1
                         Bytes used      =         20

Time =       0.43 sec    Generated terms =     200000
               F       1 Terms left      =          2
                         Bytes used      =         40

Time =       0.65 sec    Generated terms =     300000
               F       1 Terms left      =          3
                         Bytes used      =         60

Time =       0.87 sec    Generated terms =     400000
               F       1 Terms left      =          4
                         Bytes used      =         80
```

Example

```
Time =       1.50 sec    Generated terms =     700000
               F       1 Terms left      =          7
                         Bytes used      =        140

Time =       1.71 sec    Generated terms =     800000
               F       1 Terms left      =          8
                         Bytes used      =        160

Time =       1.78 sec    Generated terms =     832040
               F       1 Terms left      =          9
                         Bytes used      =        180

Time =       1.78 sec    Generated terms =     832040
               F         Terms in output =          1
                         Bytes used      =         20

   F =
      832040;

  1.78 sec out of 1.82 sec
```

Example

```
Time =       1.50 sec    Generated terms =     700000
               F       1 Terms left      =          7
                         Bytes used      =        140

Time =       1.71 sec    Generated terms =     800000
               F       1 Terms left      =          8
                         Bytes used      =        160

Time =       1.78 sec    Generated terms =     832040
               F       1 Terms left      =          9
                         Bytes used      =        180

Time =       1.78 sec    Generated terms =     832040
               F         Terms in output =          1
                         Bytes used      =         20

   F =
      832040;

  1.78 sec out of 1.82 sec
```

For big expressions FORM sorts terms in a hierarchical way (merge sort), which works well even on disk storage

This is why FORM is good for extremely big expressions

Preprocessor and $-variables

FORM has a powerful ‘preprocessor’ in compile-time

preprocessor instructions starting with ‘#’
preprocessor variables, conditional branching, loop constructs,
procedures (subroutines), . . . , metaprogramming

Preprocessor and $-variables

FORM has a powerful ‘preprocessor’ in compile-time

> preprocessor instructions starting with ‘#’
> preprocessor variables, conditional branching, loop constructs, procedures (subroutines), . . . , metaprogramming

$-variable: a variable storing a small expression, which can be accessed both in compile-time (i.e., by the preprocessor) and run-time

Preprocessor and $-variables

```
CFunction fib; Symbol n;

* Build a table with precomputed values.
#define N "1000"
CTable sparse, check, fibtab(1);
Fill fibtab(1) = 1;
Fill fibtab(2) = 1;
#do i=3,`N'
  #$value = fibtab(`i' - 1) + fibtab(`i' - 2);
  Fill fibtab(`i') = `$value';
#enddo

Local F = fib(1000);  * Find the 1000th Fibonacci number.
id fib(n?) = fibtab(n);
Print;
.end
```

Preprocessor and $-variables

```
Time =       0.01 sec    Generated terms =          1
               F         Terms in output =          1
                         Bytes used      =        188

   F =
      434665576869374564356885276750406258025646605173717804024817290895365554\
      17949051890403879840079255169295922593080322634775209689623239873322471 1\
      61642996440906533187938298969649928516003704476137795166849228875;

  0.01 sec out of 0.02 sec
```

Now fast enough ☺

Preprocessor and $-variables

What if the maximum argument of `fibtab` is not known?

```
Element in table is undefined
      fibtab(1001)
Program terminating at fibtab2.frm Line 15 -->
```

Power of metaprogramming

```
CFunction fib;
Symbol n;

* User input: suppose we don't
* know the maximum value.
Local F = fib(1001);

* Find the maximum argument.
#$nmax = 0;
if (match(fib(n?$n)));
  $nmax = max_($nmax,$n);
endif;
ModuleOption local, $n;
ModuleOption maximum, $nmax;
.sort
```

```
#define N "`$nmax'"
#if `N' > 0
* Build a table.
  CTable sparse, check, fibtab(1);
  Fill fibtab(1) = 1;
  Fill fibtab(2) = 1;
  #do i=3,`N'
    #$value = fibtab(`i'-1)
            + fibtab(`i'-2);
    Fill fibtab(`i') = `$value';
  #enddo
* And use the table.
  id fib(n?) = fibtab(n);
#endif
Print;
.end
```

* Compilation/running for each `.sort`/`.end`

Power of metaprogramming

```
F =
   703303677114228158218352548771835497701812698363587327426049050871545371\
   181969335797422494945626117334877504492417659910881863632654502236471060\
   120533741212738673391111981393731255987676900919022452453234035 01;

0.03 sec out of 0.04 sec
```

Result of a part of program can change program flow in another part of program

Such optimizations make a difference for millions of terms, $\mathcal{O}$(1TB) expressions

“There’s more than one way to do it”

With zero-dimensional sparse tables (v4.2.0)

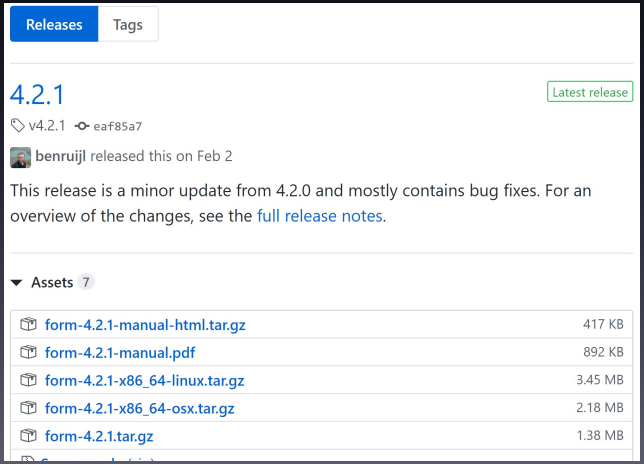
```
1  Symbol n, n1, n2;
2  CTable fib(n?int_);
3  CTable fibimpl(n?int_, n1?, n2?);
4  Fill fib = theta_(- 1 - n) * sign_(n + 1) * fib(- n)
5           + theta_(n - 1) * fibimpl(n-2, 1, 1);
6  Fill fibimpl = theta_(- n) * n2
7               + thetap_(n) * fibimpl(n-1, n2, n1+n2);
8
9  L F = fib(1001);
10 Print;
11 .end
```

See also FORM version 4.2 release notes, Ruijl, TU, Vermaseren, arXiv:1707.06453

Recent developments

FORM version 4.2.1

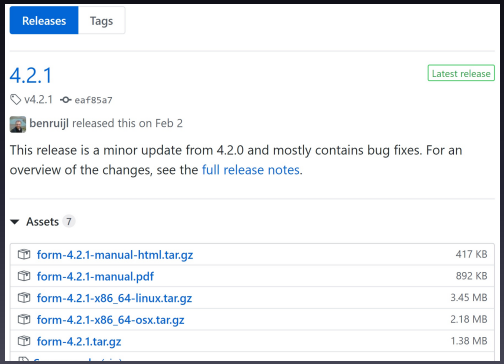
https://github.com/vermaseren/form/releases



FORM version 4.2.1

Over 80 commits since 4.2.0
(July 2017)

https://github.com/vermaseren/form/releases

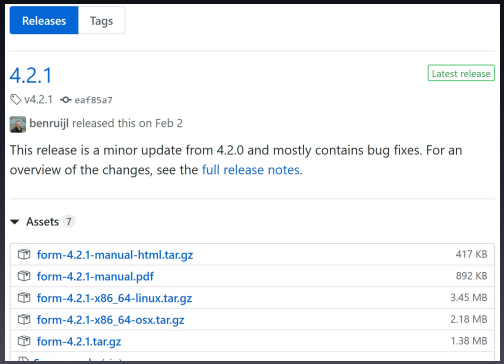


FORM version 4.2.1

Over 80 commits since 4.2.0
(July 2017)

Mainly bug fixes, but also contains new/experimental features

https://github.com/vermaseren/form/releases



FORM version 4.2.1

Over 80 commits since 4.2.0
(July 2017)

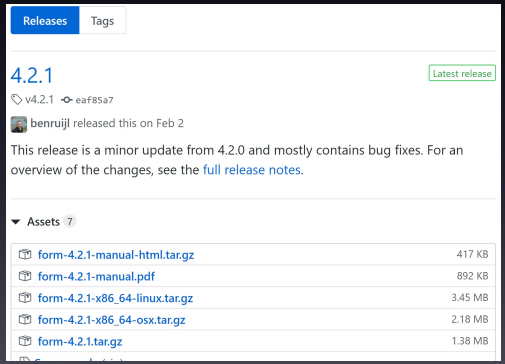
Mainly bug fixes, but also contains new/experimental features

Contributors include:
(in alphabetical order)

- Stephen Jones
- Alex Myczko
- Maximilian Reininghaus

and many bug reporters

https://github.com/vermaseren/form/releases



Thank you!

Installing FORM

```
Description-md5: 84468d585ba68000c5b3f146029e3dc6

# apt-get -y install form
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  form
0 upgraded, 1 newly installed, 0 to remove and 25 not upgraded.
Need to get 1371 kB of archives.
After this operation, 4929 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu disco/universe amd64 form amd64 4.2.1-1 [
1371 kB]
Fetched 1371 kB in 1s (2025 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package form.
(Reading database ... 13477 files and directories currently installed.)
Preparing to unpack .../form_4.2.1-1_amd64.deb ...
Unpacking form (4.2.1-1) ...
Setting up form (4.2.1-1) ...
# form -v
FORM 4.2.1 (Nov 21 2018, v4.2.1) 64-bits
  0.00 sec out of 0.00 sec
#
```

If you have latest OSes…

Canonical will ship Ubuntu 19.04 ‘Disco Dingo’ in April, which includes FORM 4.2.1

```
$ sudo apt-get update
$ sudo apt-get install form
```

Installing FORM

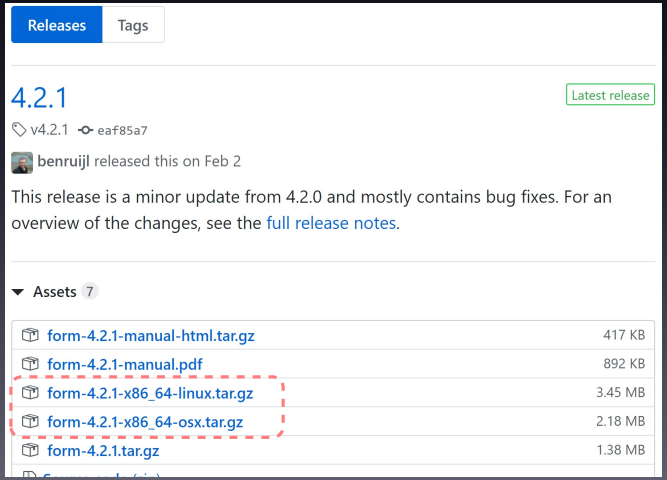
You can use package repositories of

- AUR (Arch Linux)
- Homebrew (macOS) / Linuxbrew (Linux/WSL)

or build FORM yourself. See instructions in
`https://github.com/vermaseren/form/wiki/Installation`

Installing FORM

There are also Linux and macOS binaries in the release page



Big bug fixes

When a big calculation is running, FORM uses gzip compression to store expressions on disk, but routines calling a library (`zlib`) was so buggy

- Randomly stopped with non-sense error messages
- Big memory leaks continuously increased memory usage

They were fixed (at least for many cases)

Some improvements

Just upgrading FORM may give some speed-up

More bugs?

The new release 4.2.1 contains improvements and bug fixes
But might have introduced other bugs

Please file bugs you found as well as questions/suggestions
`https://github.com/vermaseren/form/issues`

Further developments

Current projects

Namespace (Issue #236)

Graph manipulation

Everything is global

In FORM, (almost) everything is put in the ‘global namespace’

expressions, symbols, variables etc.

No local objects scoped in any parts of programs

Everything is global

In FORM, (almost) everything is put in the ‘global namespace’

expressions, symbols, variables etc.

No local objects scoped in any parts of programs

When one uses a library made by another, the library user (and library creator) must be very careful not to break anything

Using libraries considered dangerous

Simple library with a procedure to compute derivatives of polynomials

deriv.h

```
1 * Find the derivative of a polynomial w.r.t. `x'.
2 Symbol n;
3 #procedure Derivative(x)
4   id `x'^n? = n * `x'^(n - 1);
5 #endprocedure
```

Using libraries considered dangerous

deriv.h

```
* Find the derivative of a polynomial w.r.t. `x'.
Symbol n;
#procedure Derivative(x)
  id `x'^n? = n * `x'^(n - 1);
#endprocedure
```

can be used as

```
#include deriv.h
Symbol x;
Local F = (1 + x)^2;
#call Derivative(x)
Print;
.end
```

Using libraries considered dangerous

deriv.h

```
* Find the derivative of a polynomial w.r.t. `x'.
Symbol n;
#procedure Derivative(x)
  id `x'^n? = n * `x'^(n - 1);
#endprocedure
```

can be used as

```
#include deriv.h
Symbol x;
Local F = (1 + x)^2;
#call Derivative(x)
Print;
.end
```

```
   F =
       2 + 2*x;
```

Using libraries considered dangerous

deriv.h

```
* Find the derivative of a polynomial w.r.t. `x'.
Symbol n;
#procedure Derivative(x)
  id `x'^n? = n * `x'^(n - 1);
#endprocedure
```

also works for multivariate polynomials

```
#include deriv.h
Symbol x, y;
Local F = (x + y)^2;
#call Derivative(y)
Print;
.end
```

Using libraries considered dangerous

deriv.h

```
* Find the derivative of a polynomial w.r.t. `x'.
Symbol n;
#procedure Derivative(x)
  id `x'^n? = n * `x'^(n - 1);
#endprocedure
```

also works for multivariate polynomials

```
#include deriv.h
Symbol x, y;
Local F = (x + y)^2;
#call Derivative(y)
Print;
.end
```

```
   F =
      2*y + 2*x;
```

Using libraries considered dangerous

deriv.h

```
* Find the derivative of a polynomial w.r.t. `x'.
Symbol n;
#procedure Derivative(x)
  id `x'^n? = n * `x'^(n - 1);
#endprocedure
```

but does not work for a corner case

```
#include deriv.h
Symbol n;
Local F = (1 + n)^2;
#call Derivative(n)
Print;
.end
```

Using libraries considered dangerous

deriv.h

```
* Find the derivative of a polynomial w.r.t. `x'.
Symbol n;
#procedure Derivative(x)
  id `x'^n? = n * `x'^(n - 1);
#endprocedure
```

but does not work for a corner case

```
#include deriv.h
Symbol n;
Local F = (1 + n)^2;
#call Derivative(n)
Print;
.end
```

```
F =
    6;
```

Conflict for the same symbol `n`

Classical solutions

① Put a prefix for private symbols

color.h

```
AutoDeclare Index cOli,cOlj,cOlk,cOln;
AutoDeclare Symbol cOlI;
AutoDeclare Vector cOlp,cOlq;
AutoDeclare Symbol cOlx,cOly,cOlc;
AutoDeclare Tensor cOld;
AutoDeclare Tensor cOldr(symmetric),cOlda(symmetric);
```

Developer-unfriendly, spoils readability

Classical solutions

② Put the responsibility on users

forcer.h

```
1 **
2 * The input can consist of the following symbols
3 * in a proper format.
4 * Any use of other symbols is at your own risk.
5 *
6 CF vx,ex;
7 V p1,...,p11,Q;
```

User-unfriendly, the user may need to know everything in the library (223049 lines for FORCER)

Namespace as the solution?

What we need is something like

```
1 * Find the derivative of a polynomial w.r.t. `x'.
2 #namespace deriv
3   Symbol n;
4   #procedure Derivative(x)
5     id `x'^n? = n * `x'^(n - 1);
6   #endprocedure
7 #endnamespace
```

such that the private symbol `n` is hidden from the outside
Tough to implement, still in a discussion stage

Graph manipulation

Manipulating graph structure is useful/mandatory for HEP computations

- Generating Feynman diagrams
- UV/IR subdivergences originated from subdiagrams

Idea: Incorporating the graph generator of GRACE Kaneko '95

Technical preview: topologies_ function

An experimental function `topologies_` is in v4.2.1

(will be deprecated)

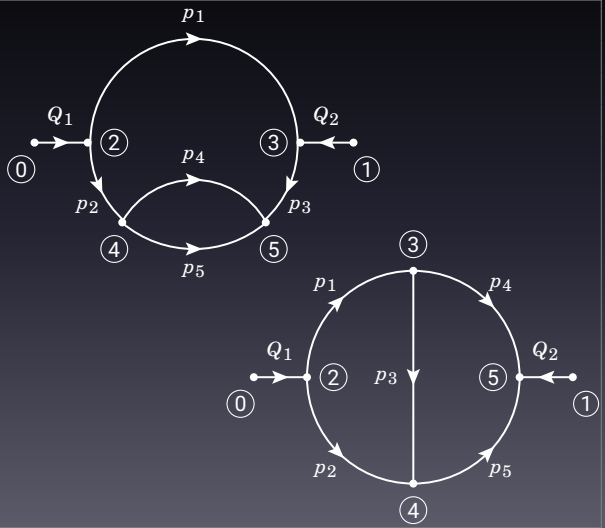
```
Vectors Q1,...,Q99;
Vectors p1,...,p99;
Set QQ: Q1,...,Q99;  * for external lines
Set pp: p1,...,p99;  * for internal lines
#define NLOOPS "2"
#define NLEGS  "2"
Local F = topologies_(`NLOOPS',`NLEGS',{3,},QQ,pp);
Print +sss;
.end
```

Technical preview: topologies_ function

```
F =
    +
       node_(0,-Q1)
      *node_(1,-Q2)
      *node_(2,Q1,-p1,-p2)
      *node_(3,Q2,p1,-p3)
      *node_(4,p2,-p4,-p5)
      *node_(5,p3,p4,p5)
    +
       node_(0,-Q1)
      *node_(1,-Q2)
      *node_(2,Q1,-p1,-p2)
      *node_(3,p1,-p3,-p4)
      *node_(4,p2,p3,-p5)
      *node_(5,Q2,p4,p5)
   ;
```

Technical preview: topologies_ function

```
F =
    +
      node_(0,-Q1)
     *node_(1,-Q2)
     *node_(2,Q1,-p1,-p2)
     *node_(3,Q2,p1,-p3)
     *node_(4,p2,-p4,-p5)
     *node_(5,p3,p4,p5)
    +
      node_(0,-Q1)
     *node_(1,-Q2)
     *node_(2,Q1,-p1,-p2)
     *node_(3,p1,-p3,-p4)
     *node_(4,p2,p3,-p5)
     *node_(5,Q2,p4,p5)
    ;
```



Technical preview: topologies_ function

NLOOPS	# of topologies	Time*
2	2	< 0.01s
3	10	< 0.01s
4	64	< 0.01s
5	519	0.05s
6	4999	0.75s
7	55758	10.12s

* On my Windows laptop (Surface Pro 4/i5-6300U/WSL)

Summary

As symbolic manipulation is important for HEP community and other fields, FORM evolves with new features as well as bug fixes and improvements

FORM 4.2.1 released:
`https://github.com/vermaseren/form/releases`

(Near) future developments: graph generations (work in progress) and namespaces (still discussion stage), hopefully in version 4.3?

Backup

Non-trivial conflict in preprocessor

Preprocessor variables have a ‘stack’, but still non-trivial conflict may occur

```
* Store a magic number into the given variable.
#procedure Get(x)
  #redefine `x' "123"
#endprocedure

#define a
#call Get(a)
#message a = `a'
* a = 123

#define x
#call Get(x)
#message x = `x'
* x = 1
.end
```