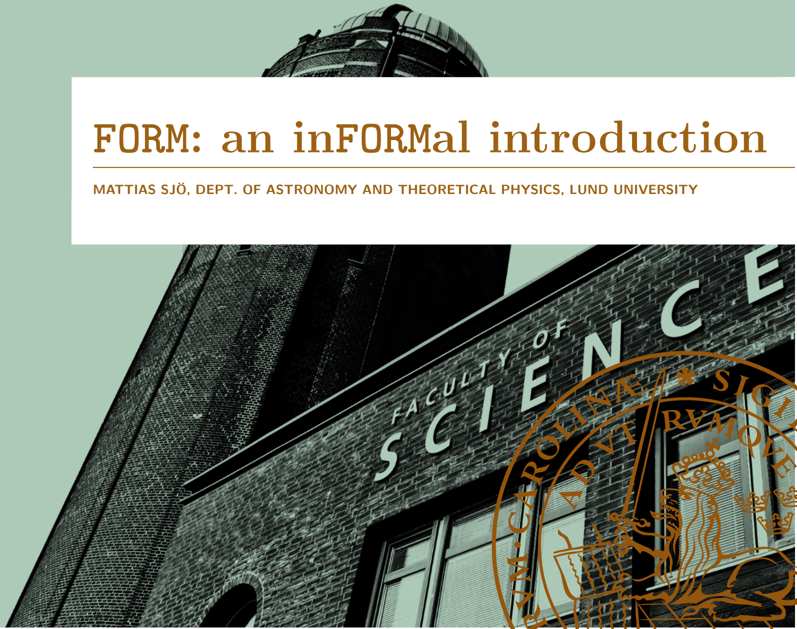
FORM: an inFORMal introduction

MATTIAS SJÖ, DEPT. OF ASTRONOMY AND THEORETICAL PHYSICS, LUND UNIVERSITY



What is FORM?

- Semi-specialised computer algebra system
- Strong ties to (and originally developed for) particle physics

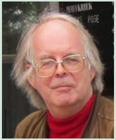


Image: chessprogramming.org

- Created by Dutch physicist Jos Vermaseren (also creator of Axodraw) in 1989
- Contributions from many others (it's open source!)
- Latest version (2018) is 4.2

Image: Accademia Degli Archi

- Spiritual successor to 1960's SCHOONSCHIP by Martinus Veltman
- Used renormalisation of Yang-Mills theory $\rightarrow$ Nobel Prize 1999
- Similarly, FORM is used for all kinds of research

FORM versus Mathematica (and Maple, etc.)

Mathematica	FORM
	
Lots of features	Limited features
Big and slow	Small and fast
Proprietary and expensive	Free and open source
Beautiful graphics and interface	The same rugged charm as FORTRAN
Lots of documentation and support	Decent documentation, mostly un-goolgleable

What can FORM do?

- Arbitrary-precision complex rational arithmetic and tensor algebra
- Special particle physics facilities (Dirac matrices, diagram generation,...)
- Complete procedural/declarative programming language (basically anything can be emulated)
- Simplify and modify **HUMONGOUS** expressions
- Use hard drive very efficiently — not limited by RAM
- Parallel and cluster computing (TFORM and ParFORM)

But...

- No numerics, no plotting, no rich text
- No "intelligent" algebra, no built-in calculus

The key point

FORM excels when things are **straightforward**,
but **big** enough to be **immensely difficult**!

Cool things done in FORM

> "We're typically one loop order higher
> than we'd be without FORM"
>
> — Common knowledge, quoted through Hans Bijnens

- Much of Hans Bijnens' research
- My master's project (incl. 12-point NLSM scattering) and subsequent work
- The high-order QCD β function
- A multiple zeta value data mine
- and many more...

A simple FORM program

LUND UNIVERSITY

A simple FORM program

Let's evaluate an integral:

$$\int_0^1 (6x + 8x^2 - 9x^3)^4 (1 - x)\,\mathrm{d}x$$

```
symbols x, n;

local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
id x^n? = x^(n+1) / (n+1);

* Insert limits
multiply replace_(x, 1) - replace_(x, 0);

print;
.end
```

Declaring variables


```
symbols x, n;

* local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
* id x^n? = x^(n+1) / (n+1);

* Insert limits
* multiply replace_(x, 1) - replace_(x, 0);

* print;
* .end
```

- All variables must be declared
- `symbol`: general-purpose commuting object

Declaring expressions


```
* symbols x, n;

local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
* id x^n? = x^(n+1) / (n+1);

* Insert limits
* multiply replace_(x, 1) - replace_(x, 0);

* print;
* .end
```

- Expressions are collections of terms
- Terms are the target of all FORM operations
- `local`: different scopes are possible in large programs

Identification statements

```
* symbols x, n;

* local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
id x^n? = x^(n+1) / (n+1);

* Insert limits
* multiply replace_(x, 1) - replace_(x, 0);

* print;
* .end
```

- `id[entify]`: apply substitution to all terms
- Wildcards: `n?` matches any symbol (including numbers)
- Matched value is substituted for `n` in RHS

Built-in functions


```
* symbols x, n;

* local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
* id x^n? = x^(n+1) / (n+1);

* Insert limits
multiply replace_(x, 1) - replace_(x, 0);

* print;
* .end
```

- multiply: multiply argument with all terms
- Built-in functions end with an underscore
- replace_: like id, do substitution in terms containing it — compare to $\int dx\,[\delta(x-1) - \delta(x-0)]Q$

Output control


```
* symbols x, n;

* local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
* id x^n? = x^(n+1) / (n+1);

* Insert limits
* multiply replace_(x, 1) - replace_(x, 0);

print;
* .end
```

- print: Causes expressions to be printed after execution
- Many options are available (formatting, selecting expressions,. . .)

Ending a module


```
* symbols x, n;

* local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
* id x^n? = x^(n+1) / (n+1);

* Insert limits
* multiply replace_(x, 1) - replace_(x, 0);

* print;
.end
```

- FORM programs are compiled and executed as modules
- Statements beginning with a period end modules
- `.end`: end module and terminate program

To recap...

LUND UNIVERSITY

Let's evaluate an integral:

$$\int_0^1 (6x + 8x^2 - 9x^3)^4 (1 - x)\,\mathrm{d}x$$

```
symbols x, n;

local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

* Integrate
id x^n? = x^(n+1) / (n+1);

* Insert limits
multiply replace_(x, 1) - replace_(x, 0);

print;
.end
```

The output

```
mssjo@mssjo-thinkpad:~/Documents/talks/form> form example.frm
FORM 4.2.1 (Feb  6 2019, v4.2.1-3-g558b01f) 64-bits  Run: Wed Jan 27 18:12:02 2021
    symbols x, n;

    local Q = (6*x + 8*x^2 - 9*x^3)^4 * (1-x);

    * Integrate
    id x^n? = x^(n+1) / (n+1);

    * Insert limits
    multiply replace_(x, 1) - replace_(x, 0);

    print;
    .end

Time =       0.00 sec    Generated terms =         30
               Q         Terms in output =          1
                         Bytes used      =         20

   Q =
      7183139/90090;

  0.00 sec out of 0.00 sec
```

- Run with $ `form <program>.frm`
- Echo input (can be turned off)
- Print statistics for module
- Print expressions as specified

The output. . . ×100!

```
mssjo@mssjo-thinkpad:~/Documents/talks/form> form example_var.frm
FORM 4.2.1 (Feb  6 2019, v4.2.1-3-g558b01f) 64-bits  Run: Wed Jan 27 18:07:58 2021
    symbols x, n;

    local Q = (6*x + 8*x^2 - 9*x^3)^400 * (1-x);

    * Integrate
    id x^n? = x^(n+1) / (n+1);

    * Insert limits
    multiply replace_(x, 1) - replace_(x, 0);

    print;
    .end

Time =      70.65 sec    Generated terms =     161202
               Q         Terms in output =          1
                         Bytes used      =        620

   Q =

      547435896601692498484126332755346156062798119729642480218695063033569824\
      451698074496925139709060453627961617491393452781283933794163422761087057\
      073954338185872992758769033411221453564519920458571994936968296092549413\
      940143110632653090849072436701904841850550136712737955688900209380364384\
      708411645533870056074258695133331866075716465683693384626491495900610528\
      359824317829440602828019657457547156457508158736082956436421338135200625\
      645761323882958755461074267146917971474576781122165914265047967201081431\
      502392383154321883303473708386577807549525128010918331241342850306495125\
      983778123775229496036675185893874317579990821115985354419192659143106403\
      049543046987261241143186522832084092167607588543336912758157191116877236\
      10298998997/
      255607776632285376121216484537866034629913342820002734644798140920975368\
      347071341729053444715371805094088075047827341991256556606187688001179715\
      054308092937273982685764817600837194808302551339656141645065419014770017\
```

The basics of FORM

LUND UNIVERSITY

Symbols

- Symbols are the most general objects:
  - Plain numbers `symbols pi, e, googol;`
  - Scalar constants/variables `symbols x, y, Zmass, Wmass;`
  - Tags to aid manipulation `symbols ONELOOP, TWOLOOP;`
  - etc...

- Symbols **do not have values**, everything is substitutions!

Wrong

```
symbol sqrt2 = 1.4142136;

* (Also, remember that
*  there are no floats,
*  only rationals!)
```

Right

```
symbol sqrt2;

... calculations ...

id sqrt2 ^ 2 = 2;
```

- There is a built-in symbol `i_`, the imaginary unit; FORM automatically substitutes `i_^2` → `-1`.

Functions

- Functions take arbitrarily many (or zero) arguments
- Symmetry information can be given
- Functions **are not evaluated**, everything is substitutions!

Wrong

```
symbols x, y, z;
function kallen(x,y,z)
    = (x^2 - y^2 - z^2)^2
    - 4 * y^2 * z^2;

* Note: only ASCII, so no
* 'Källén', unfortunately
```

Right

```
symbols x, y, z;
function kallen(symmetric);

... calculations ...

id kallen(x?, y?, z?)
    = (x^2 - y^2 - z^2)^2
    - 4 * y^2 * z^2;
```

- There are many useful built-in functions:
  `binom_`, `fac_`, `sum_`, `replace_`...
  Automatically substituted, just like `i_`.

Wildcards and identification

- Symbol wildcards match most things
- A symbol can be both "itself" and a wildcard simultaneously

```
symbols x, n;
local expr = x^5 - n*x^3;

id x^n? = n * x^(n-1);
id n = 11;
* Result: 5*x^4 - 33*x^2
```

- Function wildcards match functions
- Argument field wildcards like ?a match sequences of arguments

```
functions f, g;
symbols x, y, z;
local A = f(x,y,z);
local B = g(z,z,f(x,y));

id f?(x?, ?a) = f(?a, x);
* A: f(y,z,x)
* B: g(z,f(x,y),z)
```

Repetition

Let's compute Fibonacci numbers!

```
symbol n;
function F;
local fib36 = F(36);

* Repeats until no more changes are made
repeat;
    id F(0) = 0;
    id F(1) = 1;
    id F(n?) = F(n-1) + F(n-2);
endrepeat;

print;
.end
```

Returns 14 930 352 after summing 14 930 352 terms (about 30 seconds) — Perhaps not the smartest solution...

Repetition

Let's compute Fibonacci numbers without using an exponential-time algorithm!

```
symbol n, m, dummy;
function F;
local fib36 = F(1,0) * dummy^36;

* Inline version for single-statement repeats
repeat id F(m?,n?) * dummy = F(n, m+n);
id F(m?,n?) = n;

print;
.end
```

Happily computes Fibonacci numbers hundreds of digits long.

Important technique

"Dummy" tricks are key to "thinking in FORM"!

Vectors, Tensors and Indices

- FORM uses Einstein summation notation:

```
vectors u,v;
tensor T;
indices mu, nu;

local X = u(mu) * v(mu) + T(mu, nu) * u(mu) * v(nu);
```

  means $X = u_\mu v^\mu + T_{\mu\nu} u^\mu v^\nu$
- No distinction between upper and lower indices (but can be emulated)
- Alternatively, dot products can be used:

```
* Same as u(mu) * v(mu)
local Y = u.v;
```

- Or the so-called SCHOONSCHIP notation:

```
* Same as T(mu, nu) * u(mu) * v(nu)
local Z = T(u, v);
```

- There are built-in tensors `e_` (Levi-Civita ϵ, antisymmetric) and `d_` (Kronecker δ, symmetric).

Example: Gram determinant

Let's compute the Gram determinant

$$G(v_1, v_2, \ldots, v_n) = \begin{vmatrix} v_1 \cdot v_1 & v_1 \cdot v_2 & \ldots & v_1 \cdot v_n \\ v_2 \cdot v_1 & v_2 \cdot v_2 & \ldots & v_2 \cdot v_n \\ \vdots & \vdots & \ddots & \vdots \\ v_n \cdot v_1 & v_n \cdot v_2 & \ldots & v_n \cdot v_2 \end{vmatrix}$$

for $n = 12$, using $|A| = \epsilon_{i_1 i_2 \ldots} \epsilon_{j_1 j_2 \ldots} A_{i_1 j_1} A_{i_2 j_2} \ldots$

```
vectors v1,...,v10;
local G10 = e_(v1,...,v10)^2;
* Apply built-in tensor algebra
contract;
.end;
```

500 million terms (170 million in result) — 10 minutes of FORM!

The FORM preprocessor

- FORM has a C-style preprocessor
  — but better!
- A full procedural language with loops, recursion, etc.
- Can define #procedures for code reuse
- Also does a lot of I/O for combining FORM with external programs

Our Gram program, generalised

```
#define N "12"

vectors v1,...,v`N';
local G`N' = e_(v1,...,v`N')^2;
contract;

* Only print small determinants
#if(`N' < 7)
    print;
#endif
.end;
```

Some FORM resources

- Official FORM webpage — download, manuals, references, etc.
  `www.nikhef.nl/~form/`
- Editor support:
  - Vermaseren's own super-obscure editor (not recommended)
  - Vi/Vim has full support
  - Hans has a hacky Emacs mode
  - For Gnome editors: `www.github.com/vsht/form.lang`
  - For KDE editors: `www.github.com/mssjo/form-utils` (my own)
- My repository also has other homebrew FORM utilities.
- FormCalc: a Mathematica interface for loop calculations
  `www.feynarts.de/formcalc/`
- There will be a follow-up talk
  (based on your interest and your questions)

If you need any FORM assistance, just ask me!