FROM POLYLOGS TO FORM DEVELOPER

Coenraad Marinissen
c.marinissen@nikhef.nl

Nikhef

OUTLINE

- What are the polylogs?
  - Classical
  - Harmonic
  - Multiple
- Properties of the polylogs
  - Product algebra
  - Singular behaviour
- Numerical evaluation
- Implementation
  - Interface to GiNaC
- Conclusion and outlook

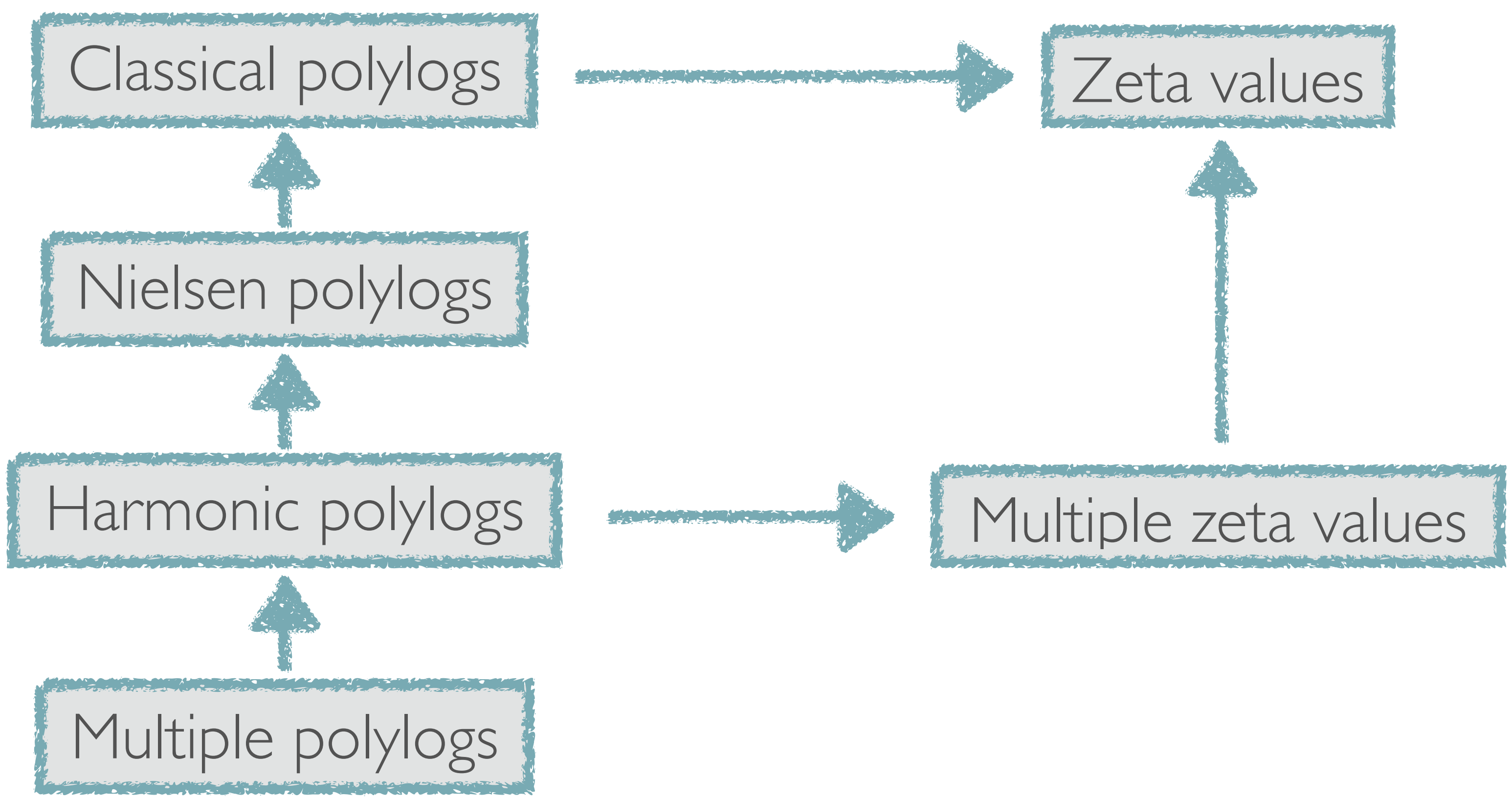
POLYLOGS, ZETA VALUES AND ALL THAT

- Polylogs come in many different varieties:

  → Classical
  Harmonic
  Multiple

- Closely related to the (multiple) zeta values

  → Multiple zeta value datamine and FORM
  → Polylogs have a natural place in FORM



CLASSICAL POLYLOGS

- Recursive integral representation:

$$Li_s(x) = \int_0^x dt \frac{Li_{s-1}(t)}{t} \qquad \text{with} \qquad Li_0(x) = \frac{x}{1-x}$$

- Series expansion:

$$Li_s(x) = \sum_{n=1}^{\infty} \frac{x^n}{n^s}$$

- Implemented in most commonly used software/packages/libraries:

Mathematica, Maple, mpfr, etc.

HARMONIC POLYLOGS

- Recursive integral representation:

$a_i \in \{0,1,-1\}$

$$H(a, a_1, \ldots, a_s; x) = \int_0^x dt\ f_a(t)\ H(a_1, \ldots, a_s; t)$$

- Where we defined

$$f_0(x) = \frac{1}{x} \qquad f_1(x) = \frac{1}{1-x} \qquad f_{-1}(x) = \frac{1}{1+x}$$

HARMONIC POLYLOGS

- Recursive integral representation:

$a_i \in \{0,1,-1\}$

$$H(a,a_1,\ldots,a_s;x) = \int_0^x dt\ f_a(t)\ H(a_1,\ldots,a_s;t)$$

- Where we defined

$$f_1(x) = \frac{1}{x} \qquad f_1(x) = \frac{1}{1-x} \qquad f_1(x) = \frac{1}{1+x}$$

Example:

$$H(1;x) = -\log(1-x) \qquad H(0;x) = \log(x) \qquad H(-1;x) = \log(1+x)$$

And with more indices:

$$H(0,1;x) = Li_2(x) \qquad H(0,-1;x) = -Li_2(-x) \qquad H(1,0;x) = -\log x \log(1-x) + Li_2(x)$$

HARMONIC POLYLOGS

- Recursive integral representation:

$a_i \in \{0,1,-1\}$

$$H(a, a_1, \ldots, a_s; x) = \int_0^x dt\, f_a(t)\, H(a_1, \ldots, a_s; t)$$

- Where we defined

$$f_0(x) = \frac{1}{x} \qquad f_1(x) = \frac{1}{1-x} \qquad f_{-1}(x) = \frac{1}{1+x}$$

- Series expansion (without trailing zeroes)

$$H_{s_1,\ldots,s_p}(x) = \sum_{n_1>n_2>\ldots>n_p>0} \frac{x^{n_1}}{n_1^{s_1}} \frac{1}{n_2^{s_2}} \cdots \frac{1}{n_p^{s_p}}$$

Compact notation

$$H(0,0,1,0,-1;x) = H_{3,-2}(x)$$

MULTIPLE POLYLOGS

- Recursive integral representation:

$a_i \in \mathbb{C}$

$$G(a_1, a_2, \ldots, a_s; x) = \int_0^x dt \; \frac{G(a_2, \ldots, a_s; t)}{t - a_1}$$

- with boundary conditions

$$G(;x) = 1 \qquad \text{and} \qquad G(a_1, \ldots, a_s; 0) = 0.$$

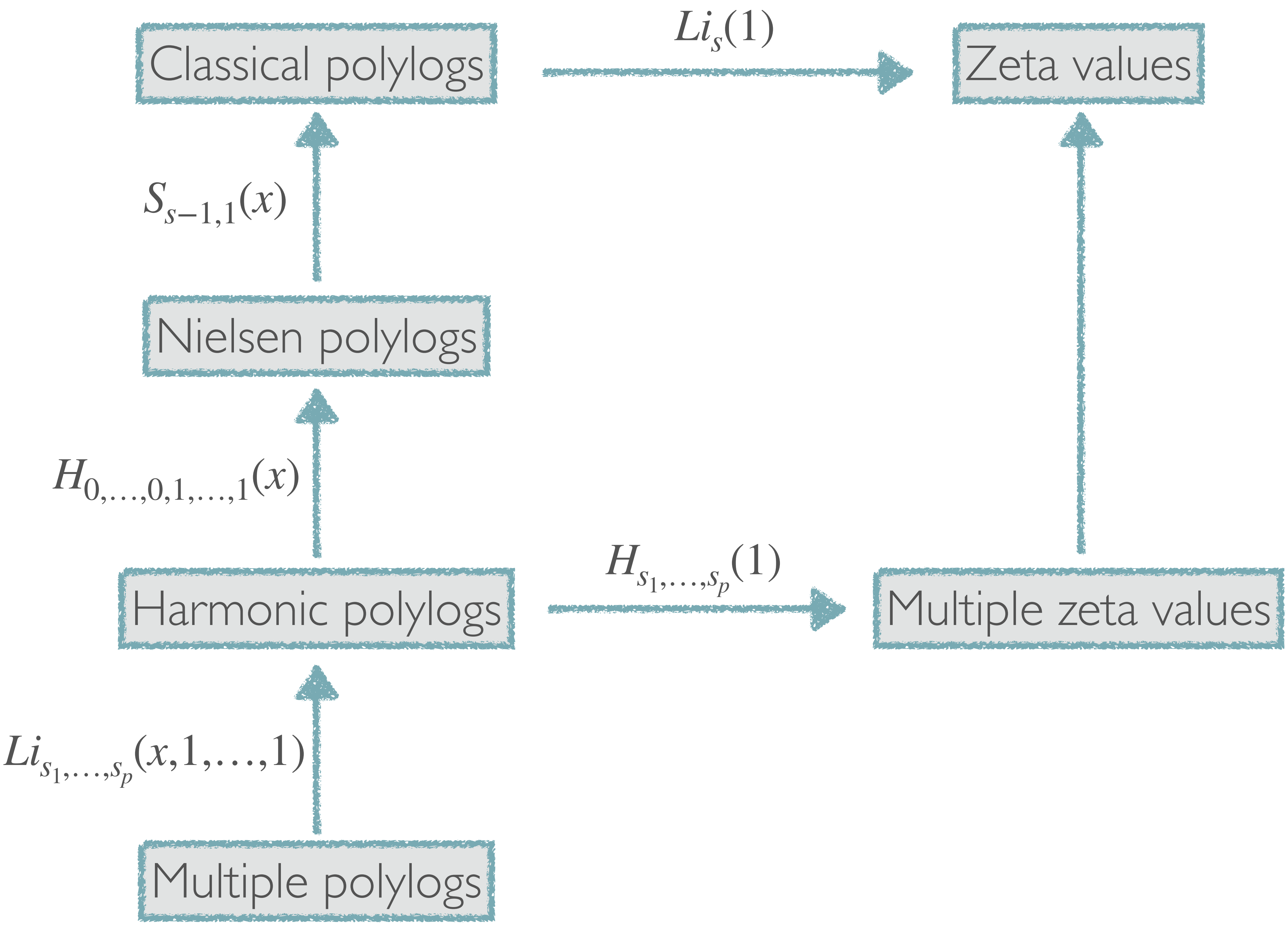
- Sum representation

$$Li_{s_1,\ldots,s_p}(x_1, \ldots, x_p) = \sum_{n_1 > n_2 > \ldots > n_p > 0} \frac{x_1^{n_1}}{n_1^{s_1}} \frac{x_2^{n_2}}{n_2^{s_2}} \cdots \frac{x_p^{n_p}}{n_p^{s_p}}$$

The two definitions are related:

$$Li_{s_1,\ldots,s_p}(x_1, \ldots, x_p) = (-1)^p G\left(\underbrace{0,\ldots,0}_{s_p - 1}, \frac{1}{x_p}, \ldots, \underbrace{0,\ldots,0}_{s_1 - 1}, \frac{1}{x_1 \ldots x_p}\right)$$

SPECIAL VALUES



PRODUCT ALGEBRA

- Express product of any two HPLs/MPLs as a linear combination of single HPLs or MPLs:

$$H(a;x)H(b,c;x) = H(a,b,c;x) + H(b,a,c;x) + H(b,c,a;x)$$

- This is called a *shuffle algebra*.

PRODUCT ALGEBRA

- Express product of any two HPLs/MPLs as a linear combination of single HPLs or MPLs:

$$H(a;x)H(b,c;x) = H(a,b,c;x) + H(b,a,c;x) + H(b,c,a;x)$$

- This is called a *shuffle algebra*.
- Minimal set of polylogs/basis

Minimal: All other HPLs can be constructed from this set using the product algebra.

Irreducible: HPLs without divergences

Weight	Full basis	Irreducible set	Minimal set
1	3	3	3
2	9	4	3
3	27	12	8
4	81	36	18
5	243	108	48
6	729	324	116
7	2187	972	312
8	6561	2916	810

SINGULAR BEHAVIOUR

- The HPL's can have divergences in $x = 0$ and $x = 1$.
- The divergent part can be extracted with the help of the above product rules:

Trailing zero

➡ $x = 0$:

$$H(s_1, \ldots, s_p, 0; x) = \log(x)\, H(s_1, \ldots, s_p; x) - H(0, s_1, \ldots, s_p; x)$$
$$- H(s_1, 0, \ldots, s_p; x) - \ldots - H(s_1, \ldots, 0, s_p; x)$$

➡ $x = 1$:

$$H(1, s_1, \ldots, s_p; x) = -\log(1-x)\, H(s_1, \ldots, s_p; x) - H(s_1, 1, \ldots, s_p; x)$$
$$- H(s_1, s_2, 1 \ldots, s_p; x) - \ldots - H(s_1, \ldots, s_p, 1; x)$$

Leading one

- One can similarly remove trailing zeroes for the multiple polylogs

NUMERICAL EVALUATION

- Use series expansion:

$$\left\{ \begin{array}{ll} Li_s(x) = \sum_{n=1}^{\infty} \frac{x^n}{n^s} & \rightarrow |x| \leq 1 \text{ and } (s,x) \neq (1,1) \\ H_{s_1,\ldots,s_p}(x) = \sum_{n_1>n_2>\ldots>n_p>0} \frac{x^{n_1}}{n_1^{s_1}} \frac{1}{n_2^{s_2}} \cdots \frac{1}{n_p^{s_p}} & \rightarrow |x| \leq 1 \text{ and } (s_1,x) \neq (1,1) \\ Li_{s_1,\ldots,s_p}(x_1,\ldots,x_p) = \sum_{n_1>n_2>\ldots>n_p>0} \frac{x_1^{n_1}}{n_1^{s_1}} \frac{x_2^{n_2}}{n_2^{s_2}} \cdots \frac{x_p^{n_p}}{n_p^{s_p}} & \rightarrow |x_1 \ldots x_p| \leq 1 \text{ and } (s_1,x_1) \neq (1,1) \end{array} \right.$$

- Outside range of convergence: transformation of the argument.

$$Li_2(x) = -Li_2\left(\frac{1}{x}\right) - \zeta_2 - \frac{1}{2}[\log(-x)]^2$$

- Close to 1: have to improve rate of convergence

$$Li_2(x) = -Li_2(1-x) + \zeta_2 - \log(x)\log(1-x)$$

IMPROVE RATE OF CONVERGENCE

- Bernoulli substitution:

Bernoulli numbers

$$Li_2(x) = \sum_{n=1}^{\infty} \frac{x^n}{n^2} = \sum_{n=0}^{\infty} \frac{B_n}{(n+1)!} \left[-\log(1-x)\right]^{n+1}$$

Slow convergence in the range $\frac{1}{2} \leq |x| \leq 1$

➡ Can do a similar transformation for the other classical polylogs and HPLs

- Holder convolution:

$$G(z_1, \ldots, z_w) = \sum_{j=0}^{w} G\left(1 - z_j, 1 - z_{j-1}; 1 - \frac{1}{p}\right) G\left(z_{j+1}, \ldots, z_w; \frac{1}{p}\right)$$

➡ Already used to improve the rate of convergence for the MZV's in FORM.

AVAILABLE TOOLS

Tool/Library	Language	Key Features
GiNaC	`C++`	Robust
Polylogtools	`Mathematica`	Robust
HPL	`Mathematica`	Direct Mathematica code
handyG	`Fortran 90`	Fast
FastGPL	`C++`	Fast

IMPLEMENTATION

- Coding from scratch is a lot of work and potentially error prone
  - ➡ Interface with GiNaC (`C++`)
- Build option: `./configure --with-ginac`

  Off by default?
- Checks if the GiNaC library is already available on the system

IMPLEMENTATION

- Coding from scratch is a lot of work and potentially error prone

  ➡ Interface with GiNaC (**`C++`**)

- Build option: `./configure --with-ginac`

  Off by default?

- Checks if the GiNaC library is already available on the system

	Classical	Harmonic	Multiple
Mathematical notation	$Li_s(x)$	$H_{s_1,\ldots,s_p}(x)$	$Li_{s_1,\ldots,s_p}(x_1,\ldots,x_p)$
Ginac	`Li(s,x)`	`H({s1,…,sp},x)`	`Li({s1,…,sp},{x1,…,xp})`
FORM	`lin_(s,x)`	`hpl_(s1,…,sp,x)`	`mpl_(lst_(s1,…,sp),lst_(x1,…,xp))`

IMPLEMENTATION

- Coding from scratch is a lot of work and potentially error prone
  - ➡ Interface with GiNaC (`C++`)
- Build option: `./configure --with-ginac`

Off by default?

- Checks if the GiNaC library is already available on the system
- Already reserved names:
  - ➡ `li2_` and `lin_`
- New reserved names
  - ➡ `hpl_` and `mpl_` and `lst_`

`ftypes.h`

```
#define LI2FUNCTION 89
#define LINFUNCTION 90
```

```
#ifdef WITHGINAC
#define HPLFUNCTION 124
#define MPLFUNCTION 125
#define LSTFUNCTION 126
#define MAXBUILTINFUNCTION 126
#else
#define MAXBUILTINFUNCTION 123
```

EVALUATE

- Evaluate all polylogs:

  ➡ `Evaluate;`

- Or evaluate one of the functions

  ➡ `Evaluate hpl_;`

Example:

```
  #StartFloat 64
  L F = lin_(2,1)*hpl_(2,-1,0.5);
  Evaluate;
  Print;
  .end

 F =
    1.21627718215376320202e-01;
```

```
  #StartFloat 64
  L F = lin_(2,1)*hpl_(2,-1,0.5);
  Evaluate hpl_;
  Print;
  .end

 F =
    7.39407862397919301042e-02*lin_(2,1);
```

EVALUATE

- Evaluate all polylogs:

  ➡ `Evaluate;`

- Or evaluate one of the functions

  ➡ `Evaluate hpl_;`

- In the compiler buffer we find:

```
Left Hand Sides:
  87  3  4294967295
  87  3  4294967294
  87  3  4294967293
```

ftypes.h

```
#define ALLFUNCTIONS -1
#define ALLMZVFUNCTIONS -2
#define ALLPOLYLOGFUNCTIONS -3
```

```
#define TYPEEVALUATE 87
```

EVALUATE

- Evaluate all polylogs:

  ➡ `Evaluate;`

- Or evaluate one of the functions

  ➡ `Evaluate hpl_;`

- In the compiler buffer we find:

```
Left Hand Sides:
  87  3  124
```

ftypes.h

```
#define HPLFUNCTION 124
#define MPLFUNCTION 125
#define LSTFUNCTION 126
```

```
#define TYPEEVALUATE 87
```

EVALUATE

- Evaluate all polylogs:

  ➡ `Evaluate;`

- Or evaluate one of the functions

  ➡ `Evaluate hpl_;`

- Would it be nice to also have `Evaluate sin_;`?

```
    #StartFloat 64
    L F = sin_(pi_)*cos_(pi_/2);
    Evaluate sin_;
test_lin.frm Line 4 --> cos_(pi_/2) should be a built in function that can be e
valuated numerically.
    Print;
    .end
Program terminating at test_lin.frm Line 5 -->
```

EVALUATE

- Evaluate all polylogs:

  ➡ `Evaluate;`

- Or evaluate one of the functions

  ➡ `Evaluate hpl_;`

- Would it be nice to also have `Evaluate sin_;`?
- Or evaluate everything, except for one function not?

  ➡
```
Evaluate;
NEvaluate mzv_;
```

CHANGES TO THE SOURCE CODE

- Interface/wrapper to connect to the `c++` code of GiNaC:

  ➡ `ginacwrapp.cc`

- Connecting point between FORM and and the GiNaC interface:

  ➡ `EvaluatePolylog();`

  Called from `Generator();`

- Structure of `EvaluatePolylog()`:

1. Locate a `lin_`, `hpl_` or `mpl_` function
2. Get argument and check for correctness
3. Evaluate

   ➡ `CalculateLin(), CalculateHpl(), or CalculateMpl()`

   Actual translation to c++/GiNaC is done here
4. Put the new term together

   ➡ Both *real* and *imaginary* part

ANALYTIC CONTINUATION

- Use FORM's build in symbol `i_` which already uses `i_^2 = -1.`

```
#StartFloat 64
L F = hpl_(2,1,0,-0.5);
Evaluate;
Print;
.end

F =
   1.474552234138308216 51e-01*i_ - 1.330715221990351833 62e-01;
```

- Using integral representation, we can also go outside the range of where the series representation converges.
- Dangerous business, i.e. pick a branch

  ➡ For now, copy the choice used for GiNaC.

- Not implemented for the other mathematical functions which use the `mpfr`–library

  ➡ What about the dilog `li2_`?

THINGS TO DECIDE

For the polylogs:

- The mzv's need MaxWeight. `#StartFloat Precision, MaxWeight`

➡ Also for the polylogs?

- Allow both `hpl_(s1,…,sp,x)` as `hpl_(lst_(s1,…,sp),x)?`
- Wat to do with the dilog? Now implemented using `mpfr`.
- Also implement the multiple polylogs of Goncharov or the Nielsen polylogs?

➡ `gpl_(lst_(a1,…,as),x)` and `npl_(n,p,x)`

THINGS TO DECIDE

For the polylogs:

- The mzv's need MaxWeight. `#StartFloat Precision, MaxWeight`

  ➡ Also for the polylogs?

- Allow both `hpl_(s1,…,sp,x)` as `hpl_(lst_(s1,…,sp),x)?`
- Wat to do with the dilog? Now implemented using `mpfr`.
- Also implement the multiple polylogs of Goncharov or the Nielsen polylogs?

  ➡ `gpl_(lst_(a1,…,as),x)` and `npl_(n,p,x)`

More general:

- Precision in bits or digits?
- What to do with infinity?

  ➡ `tan_(pi_/2)` is unaltered, `mzv_(1,2)` terminates the program.

- Merge some of the code with evaluate.h?

CONCLUSION AND OUTLOOK

- Finish the GiNaC interface for the polylogs
  - ➡ Also take imaginary arguments
  - ➡ Get ready for version 5.0.0
- Implement finite field methods in FORM:
  - ➡ Native implementation: a lot of work
  - ➡ Interface to FiniteFlow (**c++**)
- Test suite
- Outreach:
  - ➡ User side
  - ➡ Developers side

QUESTIONS?