quarks Q2C cosmos APRIL MEETING 2017 January 28-31, 2017 Washington, DC

INTEGRAND REDUCTION RELOADED ALGEBRAIC GEOMETRY AND FINITE FIELDS

Ray D. Sameshima
New York City College of Technology
CUNY Graduate Center

Ph. D. Advisors: Andrea Ferroglia, Giovanni Ossola

Work done in collaboration with:
Pierpaolo Mastrolia, Amedeo Primo, William Javier Torres Bobadilla(University of Padova), Tiziano Peraro (University of Edinburgh)

Supported in part by NSF Grant PHY-1417354

AN OUTLINE

- Scattering amplitudes, integral and integrand decompositions
- Integrand level decomposition with algebraic geometry
- Analytic expression from numerical evaluation by means of finite fields
- First steps towards implementation

WHAT IS A SCATTERING AMPLITUDE?

- A connection between collider experiments and theoretical models.
- A scattering amplitude is a probability amplitude, and the cross section is derived from it.

In fact, it (the cross section) is the effective area of a chunk taken out of one beam, … (Peskin and Schroeder)

LEADING ORDER, NEXT LEADING ORDER, AND HIGHER

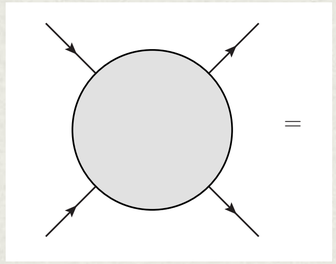
A scattering amplitude is computed using a perturbation expasion:

To understand interactions more deeply and to compare with experiments more accurately, it is required to reduce the theoretical uncertainty.

This will be achieved by taking higher loop amplitudes.

LEADING ORDER, NEXT LEADING ORDER, AND HIGHER

A scattering amplitude is computed using a perturbation expasion:

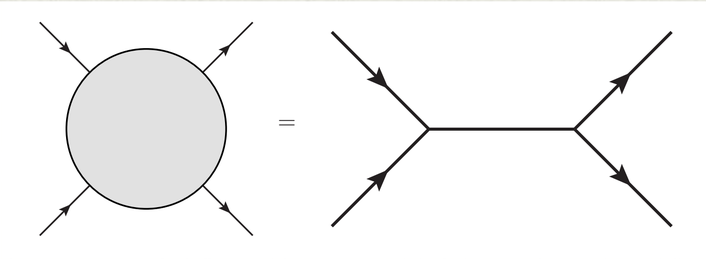


To understand interactions more deeply and to compare with experiments more accurately, it is required to reduce the theoretical uncertainty.

This will be achieved by taking higher loop amplitudes.

LEADING ORDER, NEXT LEADING ORDER, AND HIGHER

A scattering amplitude is computed using a perturbation expasion:

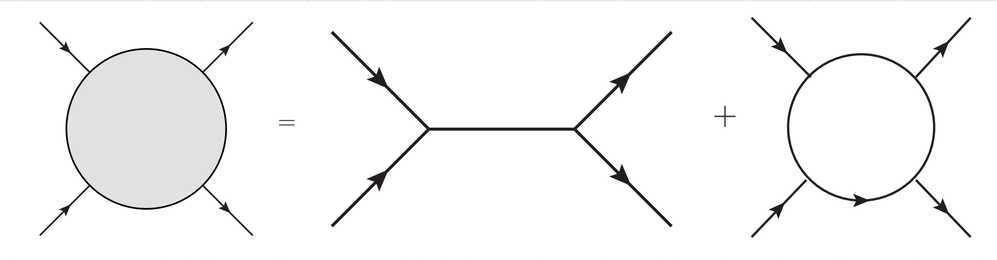


To understand interactions more deeply and to compare with experiments more accurately, it is required to reduce the theoretical uncertainty.

This will be achieved by taking higher loop amplitudes.

LEADING ORDER, NEXT LEADING ORDER, AND HIGHER

A scattering amplitude is computed using a perturbation expasion:

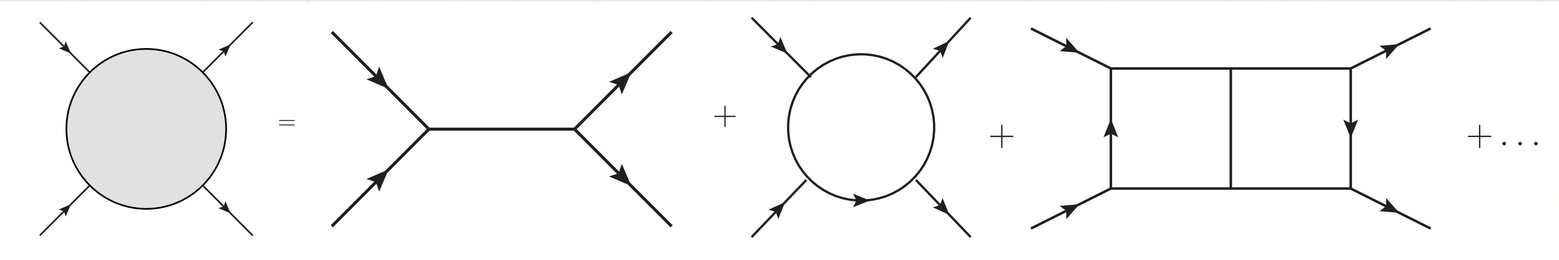


To understand interactions more deeply and to compare with experiments more accurately, it is required to reduce the theoretical uncertainty.

This will be achieved by taking higher loop amplitudes.

LEADING ORDER, NEXT LEADING ORDER, AND HIGHER

A scattering amplitude is computed using a perturbation expasion:

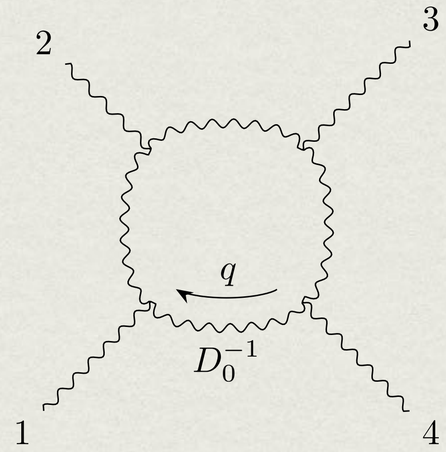


To understand interactions more deeply and to compare with experiments more accurately, it is required to reduce the theoretical uncertainty.

This will be achieved by taking higher loop amplitudes.

AN EXAMPLE

A one-loop $2 \to 2$ scattering diagram is depicted as

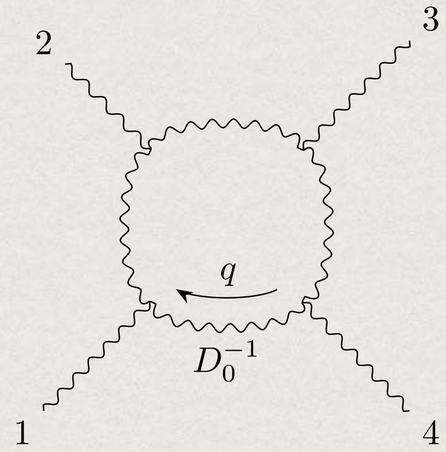


We translate it into an integral form using Feynman rules given by the theory we consider:

$$\int dq \mathcal{A}(q) = \int dq \frac{\mathcal{N}(q)}{D_0 D_1 D_2 D_3}$$

AN EXAMPLE

A one-loop $2 \to 2$ scattering diagram is depicted as



We translate it into an integral form using Feynman rules given by the theory we consider:

$$\int dq \mathcal{A}(q) = \int dq \frac{\mathcal{N}(q)}{D_0 D_1 D_2 D_3}$$

Numerator and denominators are all polynomials of integration variable, i.e., loop momenta.

INTEGRAL-LEVEL REDUCTION

Consider 1-loop Feynman integral with n denominators

$$\int dq \frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}}$$

The integral-level reduction leads to an expression in terms of Master Integrals which are easier to evaluate than the original integral above:

$$\int dq \frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}} = \sum_i c_i \mathcal{I}_i$$

If Master Integrals are known, all we need is to extract the coefficients (c's) that are given as rational functions of kinematics.

INTEGRAND-LEVEL REDUCTION

Alternatively, we can reduce the integral kernel before integration.

$$\frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}}$$

Integrand reduction allows us to fit the numerators as the following polynomial form

$$\mathcal{N}(q) = \sum_{i_1<<i_s} \Delta_{i_1 \cdots i_s}(q) \prod_{h \neq i_1, \cdots, i_s} D_h + \cdots + \sum_{i_1<<i_2} \Delta_{i_1 i_2}(q) \prod_{h \neq i_1, i_2} D_h$$

This expression allows us to compute all coefficients in a straightforward manner.

G. Ossola, C. G. Papadopoulos and R. Pittau, Nucl. Phys B 763, 147 (2007).
P. Mastrolia and G. Ossola, JHEP 1111, 014 (2011).

INTEGRAND-LEVEL REDUCTION

Alternatively, we can reduce the integral kernel before integration.

$$\frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}}$$

Integrand reduction allows us to fit the numerators as the following polynomial form

$$\mathcal{N}(q) = \sum_{i_1<<i_s} \Delta_{i_1 \cdots i_s}(q) \prod_{h \neq i_1,\cdots,i_s} D_h + \cdots + \sum_{i_1<<i_2} \Delta_{i_1 i_2}(q) \prod_{h \neq i_1,i_2} D_h$$

This expression allows us to compute all coefficients in a straightforward manner.

G. Ossola, C. G. Papadopoulos and R. Pittau, Nucl. Phys B 763, 147 (2007).
P. Mastrolia and G. Ossola, JHEP 1111, 014 (2011).

INTEGRAND-LEVEL REDUCTION

Alternatively, we can reduce the integral kernel before integration.

$$\frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}}$$

Integrand reduction allows us to fit the numerators as the following polynomial form

$$\mathcal{N}(q) = \sum_{i_1<<i_s} \Delta_{i_1 \cdots i_s}(q) \prod_{h \neq i_1, \cdots, i_s} D_h + \cdots + \sum_{i_1<<i_2} \Delta_{i_1 i_2}(q) \prod_{h \neq i_1, i_2} D_h$$

This expression allows us to compute all coefficients in a straightforward manner.

G. Ossola, C. G. Papadopoulos and R. Pittau, Nucl. Phys B 763, 147 (2007).
P. Mastrolia and G. Ossola, JHEP 1111, 014 (2011).

INTEGRAND-LEVEL REDUCTION

Alternatively, we can reduce the integral kernel before integration.

$$\frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}}$$

Integrand reduction allows us to fit the numerators as the following polynomial form

$$\mathcal{N}(q) = \sum_{i_1<<i_s} \Delta_{i_1 \cdots i_s}(q) \prod_{h \neq i_1, \cdots, i_s} D_h + \cdots + \sum_{i_1<<i_2} \Delta_{i_1 i_2}(q) \prod_{h \neq i_1, i_2} D_h$$

This expression allows us to compute all coefficients in a straightforward manner.

G. Ossola, C. G. Papadopoulos and R. Pittau, Nucl. Phys B 763, 147 (2007).
P. Mastrolia and G. Ossola, JHEP 1111, 014 (2011).

INTEGRAND-LEVEL REDUCTION

Alternatively, we can reduce the integral kernel before integration.

$$\frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}}$$

Integrand reduction allows us to fit the numerators as the following polynomial form

$$\mathcal{N}(q) = \sum_{i_1 << i_s} \Delta_{i_1 \cdots i_s}(q) \prod_{h \neq i_1, \cdots, i_s} D_h + \cdots + \sum_{i_1 << i_2} \Delta_{i_1 i_2}(q) \prod_{h \neq i_1, i_2} D_h$$

This expression allows us to compute all coefficients in a straightforward manner.

G. Ossola, C. G. Papadopoulos and R. Pittau, Nucl. Phys B 763, 147 (2007).
P. Mastrolia and G. Ossola, JHEP 1111, 014 (2011).

The multivariate polynomial division provides a systematic way to achieve the integrand reduction.

INTEGRAND-LEVEL REDUCTION

Alternatively, we can reduce the integral kernel before integration.

$$\frac{\mathcal{N}(q)}{D_0 \cdots D_{n-1}}$$

Integrand reduction allows us to fit the numerators as the following polynomial form

$$\mathcal{N}(q) = \sum_{i_1<<i_s} \Delta_{i_1 \cdots i_s}(q) \prod_{h \neq i_1, \cdots, i_s} D_h + \cdots + \sum_{i_1<<i_2} \Delta_{i_1 i_2}(q) \prod_{h \neq i_1, i_2} D_h$$

This expression allows us to compute all coefficients in a straightforward manner.

G. Ossola, C. G. Papadopoulos and R. Pittau, Nucl. Phys B 763, 147 (2007).
P. Mastrolia and G. Ossola, JHEP 1111, 014 (2011).

The multivariate polynomial division provides a systematic way to achieve the integrand reduction.

Y. Zhang, JHEP 1209, 042 (2012)
P. Mastrolia, E. Mirabella, G. Ossola and T. Peraro, Phys. Lett. B718 (2012)

An algorithmic recipe:

For a given Feynman integrand

$$\frac{N}{D_0 \cdots D_{n-1}}$$

take the ideal generated by the denominators

$$\langle D_0 \cdots D_{n-1} \rangle .$$

Construct a Gröbner basis of this ideal

$$g_1, \cdots, g_t \in \langle D_0 \cdots D_{n-1} \rangle .$$

Then the reminder Δ is unique, and the polynomial division becomes

$$N = \Gamma + \Delta, \Gamma = \sum_i Q_i g_i.$$

By construction, $\forall g_i \in \langle D_1 \cdots D_n \rangle$, we have

$$g_i = \sum_j R_{ij} D_j$$

and

$$N = \sum_j N_j D_j + \Delta, N_j = \sum_j Q_i R_{ij}.$$

Now we can recursively apply the above algorithm with N_j until we reach a fully reduced form:

$$\frac{N}{D_0 \cdots D_{n-1}} = \frac{\Delta}{D_0 \cdots D_{n-1}} + \sum_j \frac{N_j D_j}{D_0 \cdots D_{n-1}}$$

An algorithmic recipe:

For a given Feynman integrand

$$\frac{N}{D_0 \cdots D_{n-1}}$$

take the ideal generated by the denominators

$$\langle D_0 \cdots D_{n-1} \rangle .$$

Construct a Gröbner basis of this ideal

$$g_1, \cdots, g_t \in \langle D_0 \cdots D_{n-1} \rangle .$$

Then the reminder Δ is unique, and the polynomial division becomes

$$N = \Gamma + \Delta, \Gamma = \sum_i Q_i g_i.$$

By construction, $\forall g_i \in \langle D_1 \cdots D_n \rangle$, we have

$$g_i = \sum_j R_{ij} D_j$$

and

$$N = \sum_j N_j D_j + \Delta, N_j = \sum_j Q_i R_{ij}.$$

Now we can recursively apply the above algorithm with N_j until we reach a fully reduced form:

$$\frac{N}{D_0 \cdots D_{n-1}} = \frac{\Delta}{D_0 \cdots D_{n-1}} + \sum_j \frac{N_j D_j}{D_0 \cdots D_{n-1}}$$

The ideal generated by denominators is defined by

$$\langle D_1 \cdots D_n \rangle := \left\{ \sum_i f_i * D_i \middle| f_i \text{ is a polynomial} \right\}$$

An algorithmic recipe:

For a given Feynman integrand

$$\frac{N}{D_0 \cdots D_{n-1}}$$

take the ideal generated by the denominators

$$\langle D_0 \cdots D_{n-1} \rangle .$$

Construct a Gröbner basis of this ideal

$$g_1, \cdots, g_t \in \langle D_0 \cdots D_{n-1} \rangle .$$

Then the reminder Δ is unique, and the polynomial division becomes

$$N = \Gamma + \Delta, \Gamma = \sum_i Q_i g_i.$$

By construction, $\forall g_i \in \langle D_1 \cdots D_n \rangle$, we have

$$g_i = \sum_j R_{ij} D_j$$

and

$$N = \sum_j N_j D_j + \Delta, N_j = \sum_j Q_i R_{ij}.$$

Now we can recursively apply the above algorithm with N_j until we reach a fully reduced form:

$$\frac{N}{D_0 \cdots D_{n-1}} = \frac{\Delta}{D_0 \cdots D_{n-1}} + \sum_j \frac{N_j D_j}{D_0 \cdots D_{n-1}}$$

The ideal generated by denominators is defined by

$$\langle D_1 \cdots D_n \rangle := \left\{ \sum_i f_i * D_i \middle| f_i \text{ is a polynomial} \right\}$$

Since each element of Gröbner basis is also given by a finite sum of the denominators, and

$$\Gamma = \sum_i Q_i \sum_j R_{ij} D_j$$

An algorithmic recipe:

For a given Feynman integrand

$$\frac{N}{D_0 \cdots D_{n-1}}$$

take the ideal generated by the denominators

$$\langle D_0 \cdots D_{n-1} \rangle .$$

Construct a Gröbner basis of this ideal

$$g_1, \cdots, g_t \in \langle D_0 \cdots D_{n-1} \rangle .$$

Then the reminder Δ is unique, and the polynomial division becomes

$$N = \Gamma + \Delta, \Gamma = \sum_i Q_i g_i.$$

By construction, $\forall g_i \in \langle D_1 \cdots D_n \rangle$, we have

$$g_i = \sum_j R_{ij} D_j$$

and

$$N = \sum_j N_j D_j + \Delta, N_j = \sum_j Q_i R_{ij}.$$

Now we can recursively apply the above algorithm with N_j until we reach a fully reduced form:

$$\frac{N}{D_0 \cdots D_{n-1}} = \frac{\Delta}{D_0 \cdots D_{n-1}} + \sum_j \frac{N_j D_j}{D_0 \cdots D_{n-1}}$$

The ideal generated by denominators is defined by

$$\langle D_1 \cdots D_n \rangle := \left\{ \sum_i f_i * D_i \,\middle|\, f_i \text{ is a polynomial} \right\}$$

Since each element of Gröbner basis is also given by a finite sum of the denominators, and

$$\Gamma = \sum_i Q_i \sum_j R_{ij} D_j$$

Recursive algorithm

$$\frac{N_j D_j}{D_0 \cdots D_{n-1}} = \frac{N_j}{D_0 \cdots D_{j-1} D_{j+1} \cdots D_{n-1}}$$

TOWARD NEW EFFICIENT ALGORITHMS

- Integrand reduction method makes our problem and its solution clear, but it requires to generate a Gröbner basis for each diagram.

- A new algorithm (adaptive integrand decomposition) has been published recently, which enable to reduce the numerator using clever choice of momentum basis without generating Gröbner basis. This also can reduce the number of parameters for denominators.

$$q = q_{\parallel} + q_{\perp}$$

P. Mastrolia, T. Peraro and A. Primo, JHEP 1680, 164(2016)

FUNCTIONAL RECONSTRUCTION

There are a list of well-tested numerical implementations for the evaluation of one-loop Feynman Integrals:

```
Samurai, Ninja, Golem,…
```

FUNCTIONAL RECONSTRUCTION

There are a list of well-tested numerical implementations for the evaluation of one-loop Feynman Integrals:

```
Samurai, Ninja, Golem,…
```

Can we extract analytic expression from them?

FUNCTIONAL RECONSTRUCTION

There are a list of well-tested numerical implementations for the evaluation of one-loop Feynman Integrals:

```
Samurai, Ninja, Golem,…
```

Can we extract analytic expression from them?

Given sufficient input-output data, we can reconstruct the generic, analytic expressions for these integrands.

- Newton interpolation for polynomials
- Thiele interpolation for rational functions

FUNCTIONAL RECONSTRUCTION

There are a list of well-tested numerical implementations for the evaluation of one-loop Feynman Integrals:

```
Samurai, Ninja, Golem,…
```

Can we extract analytic expression from them?

Given sufficient input-output data, we can reconstruct the generic, analytic expressions for these integrands.

A. von Manteuff el and R. M. Schabinger, Phys. Lett. B 744, 101 (2015).
T. Peraro, JHEP 1612 (2016) 030

- Newton interpolation for polynomials
- Thiele interpolation for rational functions

FUNCTIONAL RECONSTRUCTION NEWTON'S ALGORITHM

Newton interpolation for integer sampling:

$$
\begin{aligned}
p(x) &= \sum_{i=0}^{R} \frac{\Delta^i(p)(0)}{i!} (x)_i \\
&\Delta^0(p) := p,\ \Delta(p)(n) = p(n+1) - p(n) \\
&(x)_0 := 1,\ (x)_n = x * (x-1) * \cdots * \{x - (n-1)\}
\end{aligned}
$$

FUNCTIONAL RECONSTRUCTION NEWTON'S ALGORITHM

The first differences are constructed from the functional values.

Newton interpolation for integer sampling:

$$p(x) = \sum_{i=0}^{R} \frac{\Delta^i(p)(0)}{i!} (x)_i$$

$$\Delta^0(p) := p, \ \Delta(p)(n) = p(n+1) - p(n)$$

$$(x)_0 := 1, \ (x)_n = x * (x-1) * \cdots * \{x - (n-1)\}$$

FUNCTIONAL RECONSTRUCTION NEWTON'S ALGORITHM

The first differences are constructed from the functional values.

Newton interpolation for integer sampling:

$$p(x) = \sum_{i=0}^{R} \frac{\Delta^i(p)(0)}{i!}(x)_i$$

Difference analysis

$$\Delta^0(p) := p, \; \Delta(p)(n) = p(n+1) - p(n)$$

$$(x)_0 := 1, \;\; (x)_n = x * (x-1) * \cdots * \{x - (n-1)\}$$

FUNCTIONAL RECONSTRUCTION NEWTON'S ALGORITHM

The first differences are constructed from the functional values.

Newton interpolation for integer sampling:

$$p(x) = \sum_{i=0}^{R} \frac{\Delta^i(p)(0)}{i!}(x)_i$$

Difference analysis

$$\Delta^0(p) := p, \; \Delta(p)(n) = p(n+1) - p(n)$$

$$(x)_0 := 1, \;\; (x)_n = x * (x-1) * \cdots * \{x - (n-1)\}$$

```
> firstDifs
>    :: (Eq a, Num a) =>
>       [a] -- map f [0..]
>    -> [a]
> firstDifs xs = reverse . map head . difLists $ [xs]
```

Haskell Road to Logic, Maths and Programming, Kees Doets and Jan van Eijck, 2004.

FUNCTIONAL RECONSTRUCTION THIELE'S ALGORITHM

Thiele interpolation for integer sampling:

$$r(x) = a_0 + \cfrac{x}{a_1 + \cfrac{x-1}{a_2 + \cfrac{x-2}{\ddots + \cfrac{\ddots}{a_{r-2} + \cfrac{\ddots}{a_{R-1} + \cfrac{x-(R-1)}{a_R}}}}}}$$

$$a_0 = f(0),\ a_n = \cfrac{1}{\cfrac{2}{\cfrac{3}{\ddots - a_{n-2}} - a_{n-1}} - a_2} - a_1$$

FUNCTIONAL RECONSTRUCTION THIELE'S ALGORITHM

Thiele interpolation for integer sampling:

Continuous fraction form

$$r(x) = a_0 + \cfrac{x}{a_1 + \cfrac{x-1}{a_2 + \cfrac{x-2}{\ddots + \cfrac{\ddots}{a_{r-2} + \cfrac{\ddots}{a_{R-1} + \cfrac{x-(R-1)}{a_R}}}}}}$$

$$a_0 = f(0),\ a_n = \cfrac{1}{\cfrac{2}{\cfrac{3}{\ddots - a_{n-2}} - a_{n-1}} \\ \ddots - a_2 \\ \cfrac{n}{f(n) - a_0} - a_1}$$

FUNCTIONAL RECONSTRUCTION THIELE'S ALGORITHM

Thiele interpolation for integer sampling:

Continuous fraction form

$$r(x) = a_0 + \cfrac{x}{a_1 + \cfrac{x-1}{a_2 + \cfrac{x-2}{\ddots + \cfrac{\ddots}{a_{r-2} + \cfrac{x-(R-1)}{a_{R-1} + \cfrac{}{a_R}}}}}}$$

$$a_0 = f(0),\ a_n = \cfrac{1}{\cfrac{2}{\cfrac{3}{\ddots - a_{n-2}} - a_{n-1}} - a_2} - a_1$$

Reciprocal differences are also given by the functional values.

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION

Some numerical algorithms can be solved efficiently over finite fields, i.e., prime fields.

We map our problems and solve on finite fields, then we remap them to rational field with high probability.

Let p be an arbitrary prime, then

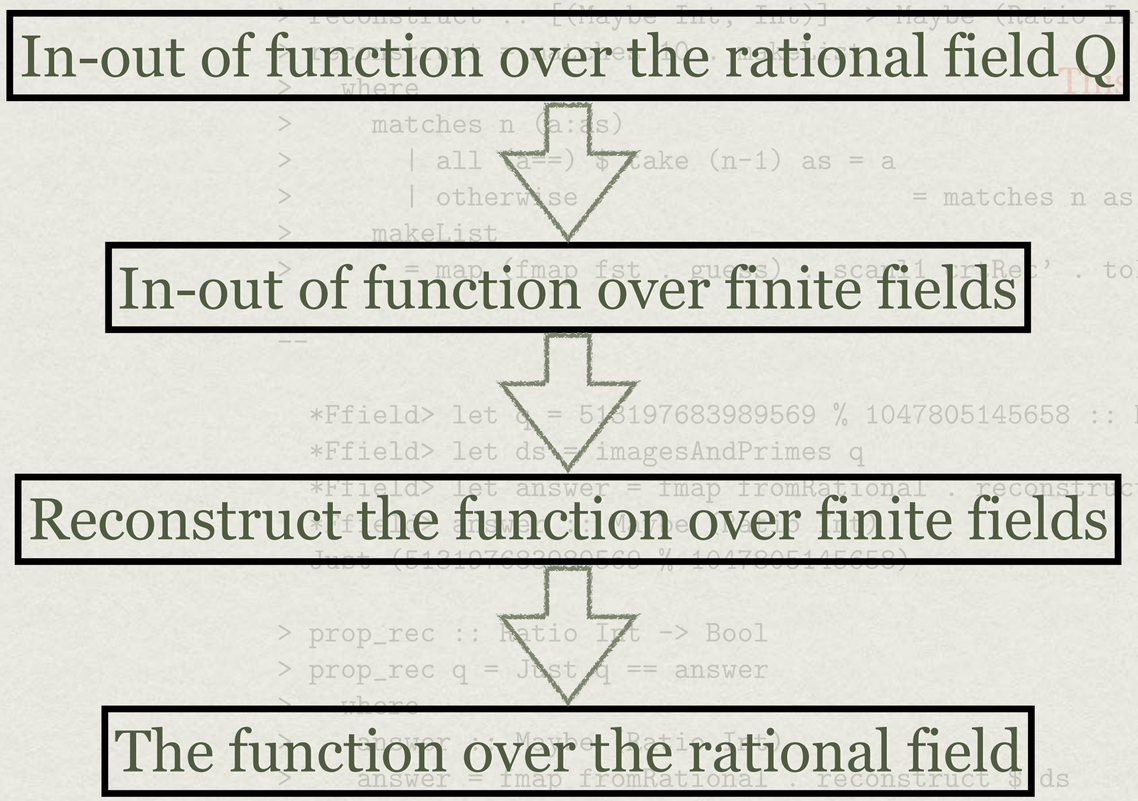
$$\mathbb{Z}_p := \mathbb{Z}/p\mathbb{Z} \cong \{0, 1, \cdots, (p-1)\}$$

becomes a field, where

$$\begin{aligned} + : \quad & (\mathbb{Z}_p, \mathbb{Z}_p) \to \mathbb{Z}_p; \quad (a, b) \mapsto a + b \mod p \\ * : \quad & (\mathbb{Z}_p, \mathbb{Z}_p) \to \mathbb{Z}_p; \quad (a, b) \mapsto a * b \mod p \end{aligned}$$

P. S . Wang, SYMSAC '81, ACM, 1981
T. Peraro, JHEP 1612 (2016) 030

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION



This function takes a list of data on several finite fields, and returns the original rational number.

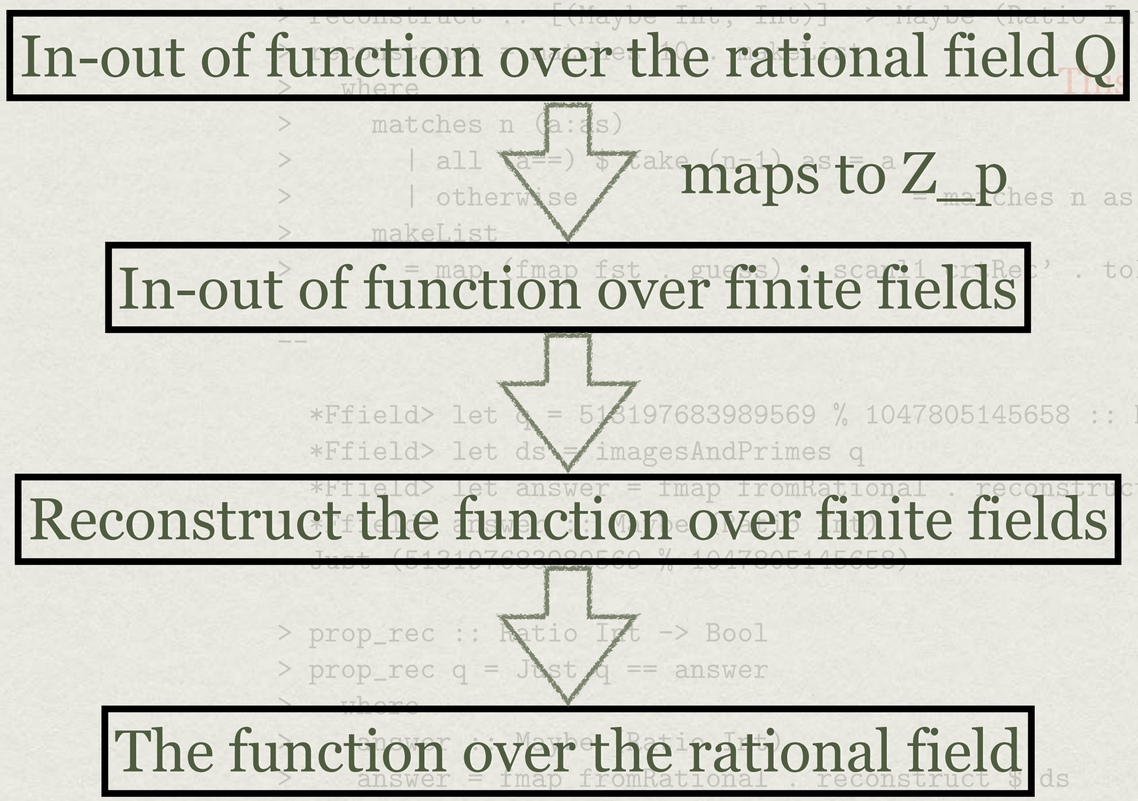
Here is a simple test.

```
*Ffield> let q = 513197683989569 % 1047805145658 :: Ratio Int
*Ffield> let ds = imagesAndPrimes q
*Ffield> let answer = fmap fromRational . reconstruct $ ds
*Ffield> answer
Just (513197683989569 % 1047805145658)
```

QuickCheck for random input; generate 100000 rational numbers.

```
*Ffield> quickCheckWith stdArgs { maxSuccess = 100000 } prop_rec
+++ OK, passed 100000 tests.
```

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION



This function takes a list of data on several finite fields, and returns the original rational number.

Here is a simple test.

QuickCheck for random input; generate 100000 rational numbers.

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION

In-out of function over the rational field Q

maps to Z_p

The input is a sub set of {0, 1, ... (p-1)}.

In-out of function over finite fields

Reconstruct the function over finite fields

The function over the rational field

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION

In-out of function over the rational field Q

maps to Z_p

The input is a sub set of {0, 1, ... (p-1)}.

In-out of function over finite fields

Newton and Thiele methods

Reconstruct the function over finite fields

The function over the rational field

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION

In-out of function over the rational field Q

maps to Z_p

The input is a sub set of {0, 1, ... (p-1)}.

In-out of function over finite fields

Newton and Thiele methods

Coefficients over finite fields

Reconstruct the function over finite fields

The function over the rational field

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION

In-out of function over the rational field Q

maps to Z_p

The input is a sub set of {0, 1, ... (p-1)}.

In-out of function over finite fields

Newton and Thiele methods

Coefficients over finite fields

Reconstruct the function over finite fields

Guess and comparison

The function over the rational field

FINITE FIELDS; AN EFFICIENT WAY OF RECONSTRUCTION

In-out of function over the rational field Q

maps to Z_p

The input is a sub set of {0, 1, ... (p-1)}.

In-out of function over finite fields

Newton and Thiele methods

Coefficients over finite fields

Reconstruct the function over finite fields

Guess and comparison

extended Euclidean algorithm
Chinese Reminder Theorem

The function over the rational field

TOWARD HIGHER LOOPS AND IMPLEMENTATIONS

- We aim at building a fully automated tool for the evaluation of Feynman integrals with 2 or more loops, with adaptive integrand decomposition algorithm, using combinations of numerical evaluations and reconstructions over finite fields.

- The reconstructed analytic expressions can be used to evaluate processes of interest for various phenomenological analyses.

SUMMARY

- Integral-level (integral) and integrand-level (polynomial)
- Algebraic geometry as tools, and a new algorithm for decomposition
- The usage of finite fields in reconstructions will provide huge improvement on computation time
- Toward implementations

SUMMARY

- Integral-level (integral) and integrand-level (polynomial)
- Algebraic geometry as tools, and a new algorithm for decomposition
- The usage of finite fields in reconstructions will provide huge improvement on computation time
- Toward implementations

Thank you very much for your attention.

ADAPTIVE INTEGRAND DECOMPOSITION

- Clever choice of parameter for momentum space reveals, so called, spurious terms which will vanish under integration.

- This choice makes the polynomial division problems into a set of liner equation.

$$q = q_{\parallel} + q_{\perp}$$

P. Mastrolia, T. Peraro and A. Primo, JHEP 1680, 164(2016)

RECONSTRUCTION

- Black-box problem, i.e., we can access only input-output data.
- From the table of in-out, i.e., a sub-graph of the function, we build the function itself.

For an arbitrary function $f : A \to B$,

$$G(X \subset A) := \{ (x, f(x)) | \, x \in X \} .$$

From this sub-graph, the reconstruction means to find

$$f' : X \to B \text{ s.t. } \; f'|_X = f|_X .$$

OVER FINITE FIELDS

- For efficiency, but it can also prevent from the (machine size) integer overflow.

Extended Euclidean algorithm as a constructive proof for Bezöut's lemma

Chinese remainder theorem

HASKELL LANGUAGE

- A standardized purely functional lazy language for general purpose.

www.haskell.org

Haskell is a computer programming language.
In particular, it is a polymorphically statically typed, lazy, purely functional language, quite different from most other programming languages.
The language is named for Haskell Brooks Curry, whose work in mathematical logic serves as a foundation for functional languages.
Haskell is based on the lambda calculus, hence the lambda we use as a logo.

https://wiki.haskell.org/Introduction

EEA WITH HASKELL CODE

Extended Euclidean Algorithm

The inputs are two integers a, b and the out puts are $\gcd(a, b)$ and four lists $\{q_i\}_i, \{r_i\}_i, \{s_i\}_i, \{t_i\}_i$. The base cases are

$$\begin{aligned}(r_0, s_0, t_0) &:= (a, 1, 0)\\ (r_1, s_1, t_1) &:= (b, 0, 1)\end{aligned}$$

and inductively, for $i \geq 2$,

$$\begin{aligned}q_i &:= \mathrm{quot}(r_{i-2}, r_{i-1})\\ r_i &:= r_{i-2} - q_i * r_{i-1}\\ s_i &:= s_{i-2} - q_i * s_{i-1}\\ t_i &:= t_{i-2} - q_i * t_{i-1}.\end{aligned}$$

The termination condition is

$$r_k = 0$$

for some $k \in \mathbb{N}$ and

$$\begin{aligned}\gcd(a, b) &= r_{k-1}\\ x &= s_{k-1}\\ y &= t_{k-1}.\end{aligned}$$

```
> exGCD' :: (Integral n) => n -> n -> ([n], [n], [n], [n])
> exGCD' a b = (qs, rs, ss, ts)
>   where
>     qs = zipWith quot rs (tail rs)
>     rs = takeUntil (==0) r'
>     r' = steps a b
>     ss = steps 1 0
>     ts = steps 0 1
>     steps a b = rr
>       where
>         rr@(_:rs) = a:b: zipWith (-) rr (zipWith (*) qs rs)
>
> takeUntil :: (a -> Bool) -> [a] -> [a]
> takeUntil p = foldr func []
>   where
>     func x xs
>       | p x = []
>       | otherwise = x : xs
```

EEA WITH HASKELL CODE

Extended Euclidean Algorithm

The inputs are two integers a, b and the out puts are $\gcd(a, b)$ and four lists $\{q_i\}_i, \{r_i\}_i, \{s_i\}_i, \{t_i\}_i$. The base cases are

$$\begin{aligned}(r_0, s_0, t_0) &:= (a, 1, 0)\\ (r_1, s_1, t_1) &:= (b, 0, 1)\end{aligned}$$

and inductively, for $i \geq 2$,

$$\begin{aligned}q_i &:= \mathrm{quot}(r_{i-2}, r_{i-1})\\ r_i &:= r_{i-2} - q_i * r_{i-1}\\ s_i &:= s_{i-2} - q_i * s_{i-1}\\ t_i &:= t_{i-2} - q_i * t_{i-1}.\end{aligned}$$

The termination condition is

$$r_k = 0$$

for some $k \in \mathbb{N}$ and

$$\begin{aligned}\gcd(a, b) &= r_{k-1}\\ x &= s_{k-1}\\ y &= t_{k-1}.\end{aligned}$$

```
> exGCD' :: (Integral n) => n -> n -> ([n], [n], [n], [n])
> exGCD' a b = (qs, rs, ss, ts)
>   where
>     qs = zipWith quot rs (tail rs)
>     rs = takeUntil (==0) r'
>     r' = steps a b
>     ss = steps 1 0
>     ts = steps 0 1
>     steps a b = rr
>       where
>         rr@(_:rs) = a:b: zipWith (-) rr (zipWith (*) qs rs)
>
> takeUntil :: (a -> Bool) -> [a] -> [a]
> takeUntil p = foldr func []
>   where
>     func x xs
>       | p x = []
>       | otherwise = x : xs
```

Bezöut's identity:

$$\forall a, b \in \mathbb{N}, \exists x, y \in \mathbb{Z} \text{ s.t. } a * x + b * y = \gcd(a, b),$$

where gcd is the greatest common divisor.

CRT WITH HASKELL CODE

Chinese Remainder Theorem

Let $p_1, p_2 \in \mathbb{N}$ be coprime (not necessarily primes), then for arbitrary $a_1, a_2 \in \mathbb{N}$,

$$\begin{aligned} x &= a_1 \mod p_1 \\ x &= a_2 \mod p_2 \end{aligned}$$

have a unique solution modulo $p_1 * p_2$. Indeed, the solution is

$$a := a_1 * p_2 * m_1 + a_2 * p_1 * m_2,$$

where $m_1 := p_1^{-1} \mod p_2, m_2 := p_2^{-1} \mod p_1$

```
> crtRec'
>   :: Integral a =>
> .    (Maybe a, a) -> (Maybe a, a) -> (Maybe a, a)
> crtRec' (Nothing,p) (_,q)       = (Nothing, p*q)
> crtRec' (_,p)       (Nothing,q) = (Nothing, p*q)
> crtRec' (Just a1,p1) (Just a2,p2) = (Just a,p)
>   where
>     a = (a1*p2*m2 + a2*p1*m1) `mod` p
>     Just m1 = p1 `inversep` p2
>     Just m2 = p2 `inversep` p1
>     p = p1*p2
```