Expression simplification and polynomial algebra in FORM

Ben Ruijl

Apr 12, 2023

Ruijl Research

Introduction

- This talk considers the C++ routines in FORM
- `optimize.cc`: expression simplification for efficient numerical evaluation of expressions
- `polygcd.cc`: polynomial GCD computation
- `polyfact.cc`: polynomial factorization

Expression simplification

$$+32o^3n^2m + 32o^3n^2l - 48o^3n^2km - 48o^3n^2kl + 32o^4j^2m + 32o^4j^2l + 64o^4ijm - 128o^4ijh + 64o^4ijl + 64o^4i^2m$$
$$-128o^4i^2h + 64o^4i^2l - 128o^4gjh + 64o^4gim - 128o^4gih + 64o^4gil + 32o^4g^2m + 32o^4g^2l - 64o^4km - 64o^4kl$$
$$+32o^4kjm + 32o^4kjl - 32o^4kj^2m - 32o^4kj^2l + 64o^4kim - 192o^4kih + 64o^4kil - 64o^4kijm + 128o^4kijh - 64o^4kijl$$
$$-64o^4ki^2m + 128o^4ki^2h - 64o^4ki^2l + 96o^4kgm - 192o^4kgh + 96o^4kgl + 128o^4kgjh - 64o^4kgim + 128o^4kgih$$
$$-64o^4kgil - 32o^4kg^2m - 32o^4kg^2l + 64o^4k^2m + 64o^4k^2l - 64o^4k^2jm - 64o^4k^2jl - 64o^4k^2im - 64o^4k^2il$$
$$-64o^4k^2gm - 64o^4k^2gl - 32o^4k^3m - 32o^4k^3l + 48fo^2n^2m + 32fo^2n^2h + 48fo^2n^2l - 48fo^2n^2jm + 64fo^2n^2jh$$
$$-48n^3h^2 - 48fo^2n^2jl - 96fo^2n^2im - 96fo^2n^2il - 64fo^2n^2gh - 48fo^2n^2km - 48fo^2n^2kl + 256fo^3jh + 32fo^3j^2m$$
$$-128fo^3j^2h + 32fo^3j^2l - 32fo^3j^3m - 32fo^3j^3l - 64fo^3im + 256fo^3ih - 64fo^3il + 128fo^3ijm - 448fo^3ijh$$
$$+128fo^3ijl - 128fo^3ij^2m + 64fo^3ij^2h - 128fo^3ij^2l + 192fo^3i^2m - 384fo^3i^2h + 192fo^3i^2l - 192fo^3i^2jm$$
$$+256fo^3i^2jh - 192fo^3i^2jl - 128fo^3i^3m + 128fo^3i^3h - 128fo^3i^3l + 64fo^3gm + 64fo^3gl - 448fo^3gjh$$
$$+64fo^3gj^2h + 192fo^3gim - 576fo^3gih + 192fo^3gil - 64fo^3gijm + 384fo^3gijh - 64fo^3gijl - 128fo^3gi^2m$$
$$+128fo^3gi^2h - 128fo^3gi^2l + 32fo^3g^2m - 64fo^3g^2h + 32fo^3g^2l - 32fo^3g^2jm + 128fo^3g^2jh - 32fo^3g^2jl$$
$$-64fo^3g^2im - 64fo^3g^2ih - 64fo^3g^2il - 64fo^3g^3h - 64fo^3km + 128fo^3kh - 64fo^3kl + 32fo^3kjm - 448fo^3kjh$$
$$+32fo^3kjl - 96fo^3kj^2m + 256fo^3kj^2h - 96fo^3kj^2l + 128f^2o^2i^2m - 384f^2o^2i^2h + 128f^2o^2i^2l - 384f^2o^2i^2jm$$

Expression optimisation

```
S x,y,z;
L F = (x*y+6*x+z^2)
 *(x^2+y^2+z^2+1);

Format O2;
.sort
#Optimize F
#write "%O";
Print F;
.end
```

```
Z1_=y + 6;
Z2_=z^2;
Z3_=Z1_*Z2_;
Z4_=x*Z1_;
Z4_=Z2_ + Z4_;
Z4_=x*Z4_;
Z1_=y*Z1_;
Z1_=1 + Z1_;
Z1_=y*Z1_;
Z1_=Z4_ + Z3_ + 6 + Z1_;
Z1_=x*Z1_;
Z3_=y^2;
Z3_=Z2_ + 1 + Z3_;
Z2_=Z3_*Z2_;
F=Z1_ + Z2_;
```

Expression optimisation

```
S x,y,z;
L F = (x*y+6*x+z^2)
 *(x^2+y^2+z^2+1);

Format O2;
.sort
#Optimize F
#write "%O";
Print F;
.end
```

```
Z1_=y + 6;
Z2_=z^2;
Z3_=Z1_*Z2_;
Z4_=x*Z1_;
Z4_=Z2_ + Z4_;
Z4_=x*Z4_;
Z1_=y*Z1_;
Z1_=1 + Z1_;
Z1_=y*Z1_;
Z1_=Z4_ + Z3_ + 6 + Z1_;
Z1_=x*Z1_;
Z3_=y^2;
Z3_=Z2_ + 1 + Z3_;
Z2_=Z3_*Z2_;
F=Z1_ + Z2_;
```

Horner schemes

- Say we have $x^3y^2 + x^2y + x^3z$
- $x \cdot x \cdot x \cdot y \cdot y + x \cdot x \cdot y + x \cdot x \cdot x \cdot z$
- $9 \times \cdot$
- Horner scheme: $x^2(y + x(y^2 + z))$
- $x \cdot x \cdot (y + x \cdot (y \cdot y + z))$
- $4 \times \cdot$
- Other possibility: $x^3z + y(x^2(1 + xy))$
- $x \cdot x \cdot x \cdot z + y \cdot (x \cdot x \cdot (1 + x \cdot y))$
- $7 \times \cdot$
- Optimal order problem is NP-hard

- Say we have $x^3y^2 + x^2y + x^3z$
- $x \cdot x \cdot x \cdot y \cdot y + x \cdot x \cdot y + x \cdot x \cdot x \cdot z$
- $9 \times \cdot$
- Horner scheme: $x^2(y + x(y^2 + z))$
- $x \cdot x \cdot (y + x \cdot (y \cdot y + z))$
- $4 \times \cdot$
- Other possibility: $x^3z + y(x^2(1 + xy))$
- $x \cdot x \cdot x \cdot z + y \cdot (x \cdot x \cdot (1 + x \cdot y))$
- $7 \times \cdot$
- Optimal order problem is NP-hard

Horner schemes

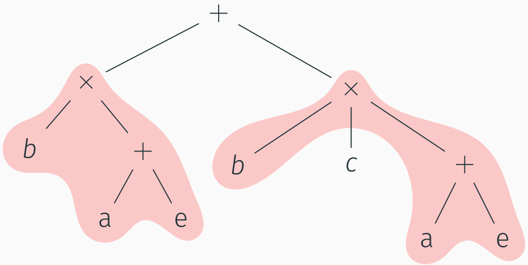
- Say we have $x^3y^2 + x^2y + x^3z$
- $x \cdot x \cdot x \cdot y \cdot y + x \cdot x \cdot y + x \cdot x \cdot x \cdot z$
- $9 \times \cdot$
- Horner scheme: $x^2(y + x(y^2 + z))$
- $x \cdot x \cdot (y + x \cdot (y \cdot y + z))$
- $4 \times \cdot$
- Other possibility: $x^3z + y(x^2(1 + xy))$
- $x \cdot x \cdot x \cdot z + y \cdot (x \cdot x \cdot (1 + x \cdot y))$
- $7 \times \cdot$
- Optimal order problem is NP-hard

Horner schemes

- Say we have $x^3y^2 + x^2y + x^3z$
- $x \cdot x \cdot x \cdot y \cdot y + x \cdot x \cdot y + x \cdot x \cdot x \cdot z$
- $9 \times \cdot$
- Horner scheme: $x^2(y + x(y^2 + z))$
- $x \cdot x \cdot (y + x \cdot (y \cdot y + z))$
- $4 \times \cdot$
- Other possibility: $x^3z + y(x^2(1 + xy))$
- $x \cdot x \cdot x \cdot z + y \cdot (x \cdot x \cdot (1 + x \cdot y))$
- $7 \times \cdot$
- Optimal order problem is NP-hard

Common Subexpression Elimination

- $b \times (a + e)$ is a common subexpression
- CSEE reduces both $\times$ and $+$



Detecting common subexpressions

```
typedef struct node {
  const WORD* data;
  struct node* l;
  struct node* r;
  UWORD hash;

  void calcHash() {
    if (data[0] == SYMBOL || data[0] == SNUMBER) {
      hash = hash_range(data, data[1] + mod);
    } else {
      if (l->hash == 0) l->calcHash();
      if (r->hash == 0) r->calcHash();
      size_t newr[] = {data[0], l->hash, r->hash};
      hash = hash_range(newr, 5);
    }
}
```

count_operators_cse_topdown

- Recursively walk through the tree and increment number of operations for every node
- Add each node to a hashset of visited nodes
- Skip nodes that are already found: common subexpressions are counted only once

Optimisation options

- O1: Horner order sorted by occurrence
- O2: Horner order sorted by occurrence + greedy optimisations
- O3: Monte Carlo Tree Search
- O4: Stochastic Local Search

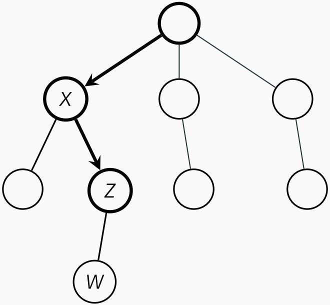
Monte Carlo Tree Search (MCTS)

- Successful for Go (AlphaGo, etc.)
- Build a state tree selectively
- Each node is a variable

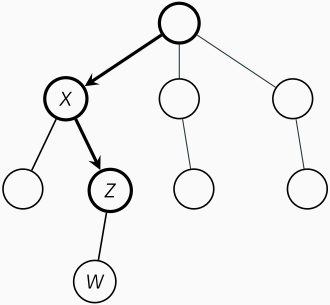
Idea

Use MCTS to find near optimal Horner scheme

Selection



Selection

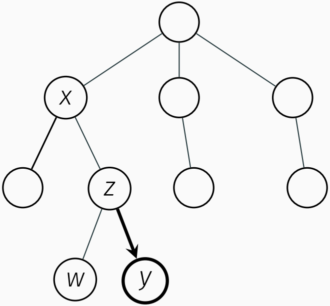


Criterion:

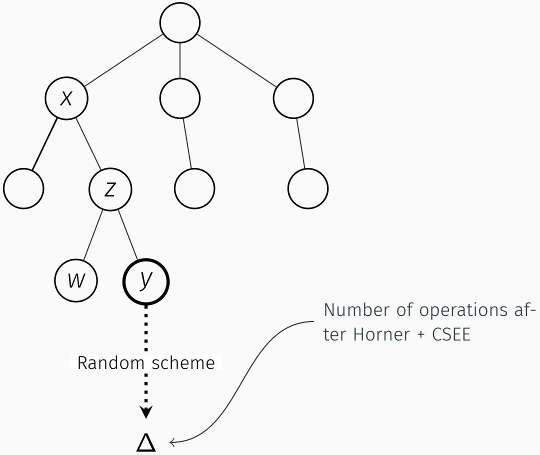
$$\underset{\text{children } c \text{ of } s}{\operatorname{argmax}} \frac{x(c)}{n(c)} + 2C_p\sqrt{\frac{2\ln n(s)}{n(c)}}$$

- $x(c)$ is score of node c
- $n(s)$ is visits at node s
- C_p is exploration-exploitation constant [expensive tuning]

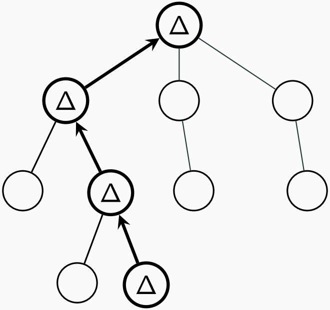
Expansion



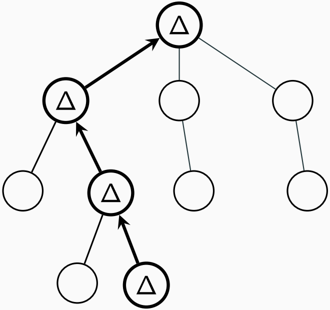
Simulation


x
z
w
y
Random scheme
Δ
Number of operations af-
ter Horner + CSEE

Backpropagation



Backpropagation



MCTS loop:

- Keep on sampling and updating the tree in a best-first way

Downsides:

- Evaluations take a long time for large expressions
- Tuning C_p is hard

Local Stochastic Search

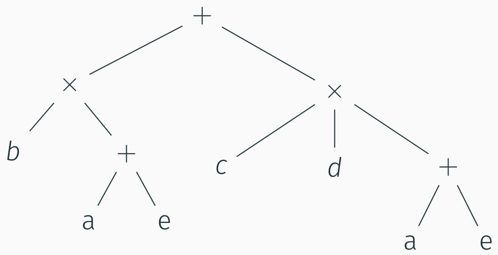
- State space is rather flat, so simpler search methods are sufficient
- Define a neighbour as a random swap in Horner scheme:

a b c d $\longrightarrow$ c b a d

- Move to a random neighbour if it has a better score

generate_instructions

Convert the expression into a sequence of instructions by reading it out depth-first and assign a number




```
Z0 = a + e;
Z1 = b * Z0;
Z2 = c * d * Z0;
F = Z1 + Z2;
```

Further manipulations

- Binary exponentiation, e.g. x^7:

```
Z1 = x * x;
Z2 = Z1 * Z1;
Z3 = Z2 * Z1;
F = Z3 * x;
```

- Pull out content in Horner scheme to increase odds of common subexpresions: $x + 2x^2 + 2x^3 \rightarrow x(1 + 2(x + x^2))$

partial_factorize

```
Z1 = x*a*b;
Z2 = x*c*d*e;
Z3 = 2*x + Z1 + Z2 + ...;
```

are replaced by

```
Z1 = a*b;
Z2 = c*d*e;
Zi = 2 + Z1 + Z2;
Zj = x*Zi;
Z3 = Zj + ...;
```

recycle_variables

- Linear scan register allocation
- On the linearized output, find lifetime of each variable
- `Z1` lives for 3 instructions and becomes available after `Z1+Z2`

```
Z1 = w^2
Z2 = y + z
Z3 = Z1 * Z2
Z4 = x + Z2
Z5 = w * Z4
F = Z3 + Z5
```

```
Z1 = w^2
Z2 = y + z
Z1 = Z1 * Z2
Z2 = x + Z2
Z2 = w * Z2
F = Z1 + Z2
```

optimize_greedy

- `find_optimizations`: find all occurrences of x^n, xy, cx, $x+c$, $x+y$ or $x-y$.
- Create new temporary variables or recycle existing temporary variables

```
Z1 = w^2;
Z2 = y + z;
Z3 = Z1 * Z2;
Z4 = x + y + z;
Z5 = w * Z4;
F = Z3 + Z5;
```

```
Z1 = w^2;
Z2 = y + z;
Z3 = Z1 * Z2;
Z4 = x + Z2;
Z5 = w * Z4;
F = Z3 + Z5;
```

Polynomial algebra

Zippel's algorithm

Compute $G = gcd(A, B)$ where $A, B \in \mathbb{Z}[x_1, \ldots, x_n]$

- Compute $G_i = gcd(A \mod p_i, B \mod p_i)$ for primes $p_1, p_2, \ldots$ and reconstruct G from these images by applying the Chinese Remainder Theorem (`gcd_modular`)
- Compute $G_{ij} = gcd(A(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i, B(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i)$ for randomly sampled α_is (`gcd_modular_dense_interpolation`)
- G_{i1} is univariate and is computed using the Euclidean algorithm
- G_1 is computed using Newton interpolation and will give the *shape* of the gcd
- G_i are computed using sparse interpolation by fitting the shape

Zippel's algorithm

Compute $G = gcd(A, B)$ where $A, B \in \mathbb{Z}[x_1, \ldots, x_n]$

- Compute $G_i = gcd(A \mod p_i, B \mod p_i)$ for primes $p_1, p_2, \ldots$ and reconstruct G from these images by applying the Chinese Remainder Theorem (`gcd_modular`)
- Compute $G_{ij} = gcd(A(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i, B(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i)$ for randomly sampled α_is (`gcd_modular_dense_interpolation`)
- G_{i1} is univariate and is computed using the Euclidean algorithm
- G_1 is computed using Newton interpolation and will give the *shape* of the gcd
- G_i are computed using sparse interpolation by fitting the shape

Zippel's algorithm

Compute $G = gcd(A, B)$ where $A, B \in \mathbb{Z}[x_1, \ldots, x_n]$

- Compute $G_i = gcd(A \mod p_i, B \mod p_i)$ for primes $p_1, p_2, \ldots$ and reconstruct G from these images by applying the Chinese Remainder Theorem (`gcd_modular`)
- Compute $G_{ij} = gcd(A(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i, B(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i)$ for randomly sampled α_is (`gcd_modular_dense_interpolation`)
- G_{i1} is univariate and is computed using the Euclidean algorithm
- G_1 is computed using Newton interpolation and will give the *shape* of the gcd
- G_i are computed using sparse interpolation by fitting the shape

Zippel's algorithm

Compute $G = gcd(A, B)$ where $A, B \in \mathbb{Z}[x_1, \ldots, x_n]$

- Compute $G_i = gcd(A \mod p_i, B \mod p_i)$ for primes $p_1, p_2, \ldots$ and reconstruct G from these images by applying the Chinese Remainder Theorem (`gcd_modular`)
- Compute $G_{ij} = gcd(A(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i, B(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i)$ for randomly sampled α_is (`gcd_modular_dense_interpolation`)
- G_{i1} is univariate and is computed using the Euclidean algorithm
- G_1 is computed using Newton interpolation and will give the *shape* of the gcd
- G_i are computed using sparse interpolation by fitting the shape

Zippel's algorithm

Compute $G = gcd(A, B)$ where $A, B \in \mathbb{Z}[x_1, \ldots, x_n]$

- Compute $G_i = gcd(A \mod p_i, B \mod p_i)$ for primes $p_1, p_2, \ldots$ and reconstruct G from these images by applying the Chinese Remainder Theorem (`gcd_modular`)
- Compute $G_{ij} = gcd(A(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i, B(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \mod p_i)$ for randomly sampled α_is (`gcd_modular_dense_interpolation`)
- G_{i1} is univariate and is computed using the Euclidean algorithm
- G_1 is computed using Newton interpolation and will give the *shape* of the gcd
- G_i are computed using sparse interpolation by fitting the shape

Example

$$G = (y + 50)x^3 + 100y, \; A = (x - y + 1)G, \; B = (x + y + 1)G$$

- Select $p_1 = 13$:

$$A_1 = (x - y + 1)((y + 11)x^3 + 9y), \quad B_1 = (x + y + 1)((y + 11)x^3 + 9y)$$

- Choose $y = 1$:

$$A_{11} = x(12x^3 + 9), \; B_{11} = (x + 2)(12x^3 + 9), \; G_{11} = 12x^3 + 9$$

- Choose $y = 2$: $(y + 11)x^3 = 0$; BAD SAMPLE
- Choose $y = 3$:

$$A'_{11} = (x - 2)(x^3 + 1), \; B'_{11} = (x + 4)(x^3 + 1), \; G'_{11} = x^3 + 1$$

- Newton interpolation: $\{(1, 12x^3 + 9), (3, x^3 + 1)\} \to (y + 11)x^3 + 9y$

Example

$$G = (y+50)x^3 + 100y,\ A = (x-y+1)G,\ B = (x+y+1)G$$

- Select $p_1 = 13$:

$$A_1 = (x-y+1)((y+11)x^3+9y), \quad B_1 = (x+y+1)((y+11)x^3+9y)$$

- Choose $y = 1$:

$$A_{11} = x(12x^3+9),\ B_{11} = (x+2)(12x^3+9),\ G_{11} = 12x^3+9$$

- Choose $y = 2$: $(y+11)x^3 = 0$; BAD SAMPLE
- Choose $y = 3$:

$$A'_{11} = (x-2)(x^3+1),\ B'_{11} = (x+4)(x^3+1),\ G'_{11} = x^3+1$$

- Newton interpolation: $\{(1, 12x^3+9), (3, x^3+1)\} \to (y+11)x^3+9y$

Example

$$G = (y + 50)x^3 + 100y,\ A = (x - y + 1)G,\ B = (x + y + 1)G$$

- Select $p_1 = 13$:

$$A_1 = (x - y + 1)((y + 11)x^3 + 9y), \quad B_1 = (x + y + 1)((y + 11)x^3 + 9y)$$

- Choose $y = 1$:

$$A_{11} = x(12x^3 + 9),\ B_{11} = (x + 2)(12x^3 + 9),\ G_{11} = 12x^3 + 9$$

- Choose $y = 2$: $(y + 11)x^3 = 0$; BAD SAMPLE
- Choose $y = 3$:

$$A'_{11} = (x - 2)(x^3 + 1),\ B'_{11} = (x + 4)(x^3 + 1),\ G'_{11} = x^3 + 1$$

- Newton interpolation: $\{(1, 12x^3 + 9), (3, x^3 + 1)\} \to (y + 11)x^3 + 9y$

Example

$$G = (y+50)x^3 + 100y,\ A = (x-y+1)G,\ B = (x+y+1)G$$

- Select $p_1 = 13$:

$$A_1 = (x-y+1)((y+11)x^3+9y), \quad B_1 = (x+y+1)((y+11)x^3+9y)$$

- Choose $y = 1$:

$$A_{11} = x(12x^3+9),\ B_{11} = (x+2)(12x^3+9),\ G_{11} = 12x^3+9$$

- Choose $y = 2$: $(y+11)x^3 = 0$; BAD SAMPLE
- Choose $y = 3$:

$$A'_{11} = (x-2)(x^3+1),\ B'_{11} = (x+4)(x^3+1),\ G'_{11} = x^3+1$$

- Newton interpolation: $\{(1, 12x^3+9), (3, x^3+1)\} \to (y+11)x^3+9y$

Example

$$G = (y+50)x^3 + 100y,\ A = (x-y+1)G,\ B = (x+y+1)G$$

- Select $p_1 = 13$:

$$A_1 = (x-y+1)((y+11)x^3+9y), \quad B_1 = (x+y+1)((y+11)x^3+9y)$$

- Choose $y = 1$:

$$A_{11} = x(12x^3+9),\ B_{11} = (x+2)(12x^3+9),\ G_{11} = 12x^3+9$$

- Choose $y = 2$: $(y+11)x^3 = 0$; BAD SAMPLE
- Choose $y = 3$:

$$A'_{11} = (x-2)(x^3+1),\ B'_{11} = (x+4)(x^3+1),\ G'_{11} = x^3+1$$

- Newton interpolation: $\{(1, 12x^3+9), (3, x^3+1)\} \to (y+11)x^3+9y$

Example

$$G = (y + 50)x^3 + 100y,\ A = (x - y + 1)G,\ B = (x + y + 1)G$$

- Select $p_1 = 13$:

$$A_1 = (x - y + 1)((y + 11)x^3 + 9y), \quad B_1 = (x + y + 1)((y + 11)x^3 + 9y)$$

- Choose $y = 1$:

$$A_{11} = x(12x^3 + 9),\ B_{11} = (x + 2)(12x^3 + 9),\ G_{11} = 12x^3 + 9$$

- Choose $y = 2$: $(y + 11)x^3 = 0$; BAD SAMPLE
- Choose $y = 3$:

$$A'_{11} = (x - 2)(x^3 + 1),\ B'_{11} = (x + 4)(x^3 + 1),\ G'_{11} = x^3 + 1$$

- Newton interpolation: $\{(1, 12x^3 + 9), (3, x^3 + 1)\} \to (y + 11)x^3 + 9y$

gcd_modular_sparse_interpolation

- Guess GCD shape: $G = (\alpha_1 y + \alpha_2)x^3 + \beta_1 y$
- Try to fit $G = (\alpha_1 y + \alpha_2)x^3 + (1)y$
- Sample $p_2 = 17$ with $y = 5$ and $y = 6$
- We get $x^3 + 6$ and $x^3 + 9$, rescaled: $15x^3 + 5$ and $14x^3 + 7$
- Solve:

$$\begin{aligned}(\alpha_1 5 + \alpha_2)x^3 + 5 &= 15x^3 + 5 \mod 17\\ (\alpha_1 7 + \alpha_2)x^3 + 7 &= 14x^3 + 7 \mod 17\end{aligned}$$

  Thus we get $G_2' = (8y + 9)x^3 + y$
- $A_1/G_2' = 15(x - y + 1)$, $B_1/G_2' = 15(x + y + 1)$, so $G_2 = (y + 16)x^3 + 15y \mod 17$

gcd_modular_sparse_interpolation

- Guess GCD shape: $G = (\alpha_1 y + \alpha_2)x^3 + \beta_1 y$
- Try to fit $G = (\alpha_1 y + \alpha_2)x^3 + (1)y$
- Sample $p_2 = 17$ with $y = 5$ and $y = 6$
- We get $x^3 + 6$ and $x^3 + 9$, rescaled: $15x^3 + 5$ and $14x^3 + 7$
- Solve:

$$(\alpha_1 5 + \alpha_2)x^3 + 5 = 15x^3 + 5 \mod 17$$
$$(\alpha_1 7 + \alpha_2)x^3 + 7 = 14x^3 + 7 \mod 17$$

  Thus we get $G_2' = (8y + 9)x^3 + y$
- $A_1/G_2' = 15(x - y + 1), B_1/G_2' = 15(x + y + 1)$, so $G_2 = (y + 16)x^3 + 15y \mod 17$

- Guess GCD shape: $G = (\alpha_1 y + \alpha_2)x^3 + \beta_1 y$
- Try to fit $G = (\alpha_1 y + \alpha_2)x^3 + (1)y$
- Sample $p_2 = 17$ with $y = 5$ and $y = 6$
- We get $x^3 + 6$ and $x^3 + 9$, rescaled: $15x^3 + 5$ and $14x^3 + 7$
- Solve:

$$(\alpha_1 5 + \alpha_2)x^3 + 5 = 15x^3 + 5 \mod 17$$
$$(\alpha_1 7 + \alpha_2)x^3 + 7 = 14x^3 + 7 \mod 17$$

  Thus we get $G_2' = (8y + 9)x^3 + y$
- $A_1/G_2' = 15(x - y + 1)$, $B_1/G_2' = 15(x + y + 1)$, so $G_2 = (y + 16)x^3 + 15y \mod 17$

gcd_modular_sparse_interpolation

- Guess GCD shape: $G = (\alpha_1 y + \alpha_2)x^3 + \beta_1 y$
- Try to fit $G = (\alpha_1 y + \alpha_2)x^3 + (1)y$
- Sample $p_2 = 17$ with $y = 5$ and $y = 6$
- We get $x^3 + 6$ and $x^3 + 9$, rescaled: $15x^3 + 5$ and $14x^3 + 7$
- Solve:

$$(\alpha_1 5 + \alpha_2)x^3 + 5 = 15x^3 + 5 \mod 17$$
$$(\alpha_1 7 + \alpha_2)x^3 + 7 = 14x^3 + 7 \mod 17$$

  Thus we get $G_2' = (8y + 9)x^3 + y$
- $A_1/G_2' = 15(x - y + 1)$, $B_1/G_2' = 15(x + y + 1)$, so $G_2 = (y + 16)x^3 + 15y \mod 17$

gcd_modular_sparse_interpolation

- Guess GCD shape: $G = (\alpha_1 y + \alpha_2)x^3 + \beta_1 y$
- Try to fit $G = (\alpha_1 y + \alpha_2)x^3 + (1)y$
- Sample $p_2 = 17$ with $y = 5$ and $y = 6$
- We get $x^3 + 6$ and $x^3 + 9$, rescaled: $15x^3 + 5$ and $14x^3 + 7$
- Solve:

$$(\alpha_1 5 + \alpha_2)x^3 + 5 = 15x^3 + 5 \mod 17$$
$$(\alpha_1 7 + \alpha_2)x^3 + 7 = 14x^3 + 7 \mod 17$$

  Thus we get $G_2' = (8y + 9)x^3 + y$
- $A_1/G_2' = 15(x - y + 1)$, $B_1/G_2' = 15(x + y + 1)$, so $G_2 = (y + 16)x^3 + 15y \mod 17$

gcd_modular_sparse_interpolation

- Guess GCD shape: $G = (\alpha_1 y + \alpha_2)x^3 + \beta_1 y$
- Try to fit $G = (\alpha_1 y + \alpha_2)x^3 + (1)y$
- Sample $p_2 = 17$ with $y = 5$ and $y = 6$
- We get $x^3 + 6$ and $x^3 + 9$, rescaled: $15x^3 + 5$ and $14x^3 + 7$
- Solve:

$$(\alpha_1 5 + \alpha_2)x^3 + 5 = 15x^3 + 5 \mod 17$$
$$(\alpha_1 7 + \alpha_2)x^3 + 7 = 14x^3 + 7 \mod 17$$

  Thus we get $G_2' = (8y + 9)x^3 + y$
- $A_1/G_2' = 15(x - y + 1), B_1/G_2' = 15(x + y + 1)$, so $G_2 = (y + 16)x^3 + 15y \mod 17$

`chinese_remainder`

- $G \equiv (y + 11)x^3 + 9y \mod 13$
- $G \equiv (y + 16)x^3 + 15y \mod 17$
- Extended Euclidean algorithm: $m_1 p_1 + m_2 p_2 = 1$ gives $m_1 = 4$, $m_2 = -3$
- $G = ((y + 11)x^3 + 9y)(-3 \cdot 17) + ((y + 16)x^3 + 15y)(4 \cdot 13) = 100y + x^3(50 + y) \mod 221$

GCD of multiple polynomials

- How to do GCD of $gcd(F_1, F_2, F_3, ...)$ faster than $gcd(F_1, gcd(F_2, \ldots$?
- Solve simpler problem where p_i are distinct primes and F_s is the smallest:

$$gcd(F_s, p_1 F_1 + p_2 F_2 + \ldots) = A = gcd(F_1, F_2, F_3, ...)R$$

- In rare cases, $R \neq 1$. Then, for the set $G \subset F$ indivisible by A, do the same algorithm with $gcd(A, G)$

GCD of multiple polynomials

- How to do GCD of $gcd(F_1, F_2, F_3, ...)$ faster than $gcd(F_1, gcd(F_2, \ldots ?$
- Solve simpler problem where p_i are distinct primes and F_s is the smallest:

$$gcd(F_s, p_1 F_1 + p_2 F_2 + \ldots) = A = gcd(F_1, F_2, F_3, ...) R$$

- In rare cases, $R \neq 1$. Then, for the set $G \subset F$ indivisible by A, do the same algorithm with $gcd(A, G)$

Factorisation

- Square-free factorisation in `squarefree_factors`
- Obtain a form

$$a(\vec{x}) = \prod_{i=1}^{k} a_i(\vec{x})^k$$

  using repeated gcd computations of a and da/dx_1
- For each square-free factor: `factorize_squarefree`

factorize_squarefree

- Convert multivariate factoring problem to univariate problem over prime field
- Select prime and go to p-adic representation:

$$u(x) = u_0(x) + u_1(x)p + u_2(x)p^2 + \dots$$

- e.g. $u(x) = 14x^2 - 11x - 15$ for $p = 5$:

$$\begin{aligned} u_0(x) &= -x^2 - x \mod 5 \\ u_1(x) &= -2x^2 - 2x + 2 \mod 5 \\ u_2(x) &= x^2 - 1 \mod 5 \\ u(x) &= -x^2 - x + (-2x^2 - 2x + 2)5 + (x^2 - 1)5^2 \end{aligned}$$

- Same can be done with *ideals*$(I = \{x_2 - c_2, \dots x_n - c_n\})$ to go from $u(x_1, x_2, \dots)$ to $u(x_1)$

Berlekamp's algorithm

- Factorization of square-free univariate polynomial $a(x)$ with degree n in $\mathbb{Z}_p$.
- Factors are given by:

$$a(x) = \prod_{s \in \mathbb{Z}_p} gcd(v(x) - s, a(x))$$

where $v(x)$ from the set

$$W = \{v(x) \in Z_p[x] : v(x^p) - v(x) = 0 \mod a(x)\}$$

- `Berlekamp_Qmatrix`: construct $n \times n$ matrix and find basis vectors
- `Berlekamp_find_factors`: perform the gcd
- `combine_factors`: fixups when more factors are found by accident

lift_variables

Hensel 'lift' p-adic, I-adic representation to original representation:

1. Start with $a(x) \mod p = u_1(x)w_1(x) \mod p$
2. Compute error $e_i(x) = a(x) - u_i(x)w_i(x)$
3. Solve
$$s(x)u_i(x) + t(x)w_i(x) \equiv e_1(x)/p^i \mod p$$
4. Update $u_{i+1}(x) = u_i + t(x)p^i, w_{i+1}(x) = w_i + s(x)p^i$
5. Done when the error is 0, else go to step 2 with $i + 1$

Thank you for your attention.