A Fast Algorithm for Computing the Truncated Resultant

G. Moroz and É. Schost

Inria Nancy - Grand Est, France
Waterloo University, Canada

SpecFun - December 12, 2016

Resultant definition

Given two polynomials in $\mathbb{C}[y]$:

- $P = p_0 y^d + \cdots + p_d$ with roots $\sigma_1, \ldots, \sigma_d$
- $Q = q_0 y^d + \cdots + q_d$ with roots $\tau_1, \ldots, \tau_d$

Definition: Resultant

$$
\begin{aligned}
Res(P, Q) &= p_0^d q_0^d \prod_{i,j}(\sigma_i - \tau_j) \\
&= p_0^d\, Q(\sigma_1) \cdots Q(\sigma_d) \\
&= (-1)^{d^2} q_0^d\, P(\tau_1) \cdots P(\tau_d)
\end{aligned}
$$

Resultant definition

Coefficients in any ring R:

- $\mathbb{C}$
- $\mathbb{K}$
- $\mathbb{K}[x]$
- $\mathbb{K}[x]/\left\langle x^k \right\rangle$

The **Resultant** is the determinant of the Sylvester Matrix

Definition: Sylvester matrix

$$
\begin{array}{ccccccccccc}
1 & 2 & \cdots & & & d+1 & d+2 & d+3 & \cdots & \\
\end{array}
$$

$$
\begin{pmatrix}
p_0 & & & & & q_0 & & & & \\
p_1 & p_0 & & & & q_1 & q_0 & & & \\
p_2 & p_1 & p_0 & & & q_2 & q_1 & q_0 & & \\
p_3 & p_2 & p_1 & p_0 & & q_3 & q_2 & q_1 & q_0 & \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}
$$

Problem and motivation

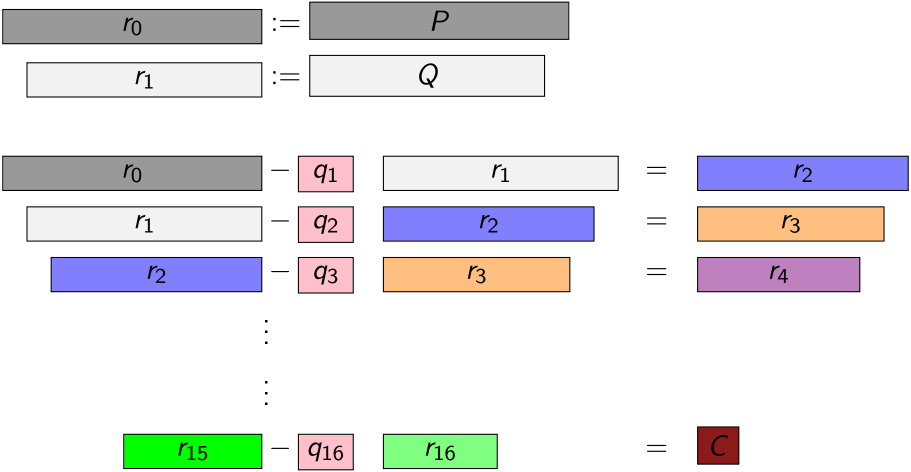
Why compute the first terms of the resultant?

- **Numeric-symbolic algorithms**: first 2 terms for Newton operator
- **Analytic functions**: often handled modulo x^k

Fast algorithm to compute the resultant

- Coefficients in a **field**
  - Half-gcd algorithm: $\tilde{O}(d)$ [Knuth, Schönhage 1970]
- Coefficients in the **ring** $R =$ p-**adic numbers**
  - Euclidean algorithm: precision k in $\tilde{O}(d^2 k)$ average time [Caruso 2015]
- Coefficients in the **ring** $R = \mathbb{K}[x]/\left\langle x^k \right\rangle$
  - R **not integral**
  - $R[y]$ **not factorial**
  - Regular case: $\tilde{O}(dk)$
  - Division-free determinant algorithm: $\tilde{O}(d^{2.698} k)$ [Kaltofen, Villard 2004]

Coefficients in a field: Euclidean algorithm



Coefficients in a field: Euclidean algorithm

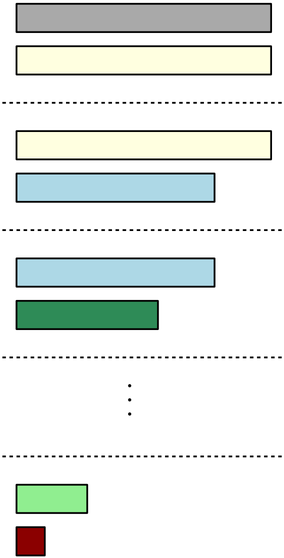
$$\begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$$

Q_1 $\quad \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} \quad \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

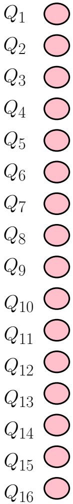
Q_2 $\quad \begin{pmatrix} 0 & 1 \\ 1 & -q_2 \end{pmatrix} \quad \begin{pmatrix} r_2 \\ r_3 \end{pmatrix}$

$\vdots$

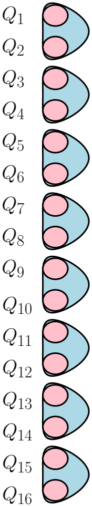
Q_{16} $\quad \begin{pmatrix} 0 & 1 \\ 1 & -q_{16} \end{pmatrix} \quad \begin{pmatrix} r_{16} \\ C \end{pmatrix}$



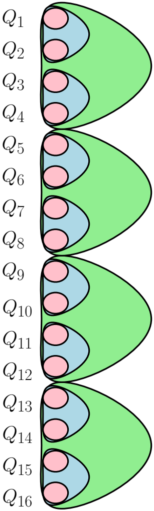
Divide and conquer



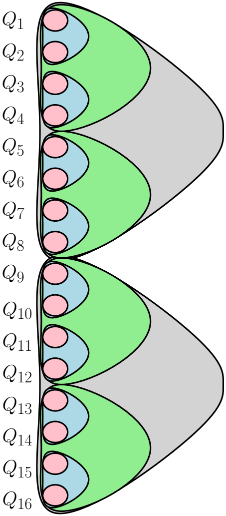
Divide and conquer



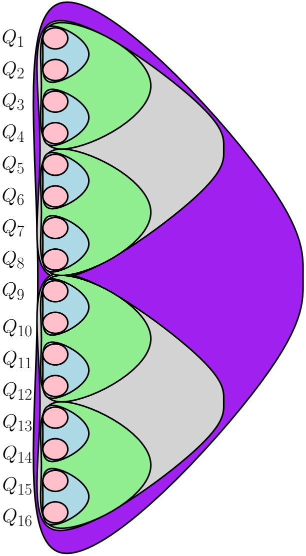
Divide and conquer



Divide and conquer

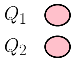


Divide and conquer

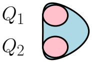


Half-gcd algorithm

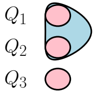
Half-gcd algorithm



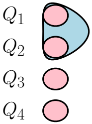
Half-gcd algorithm



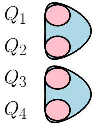
Half-gcd algorithm



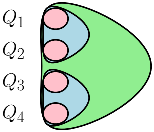
Half-gcd algorithm



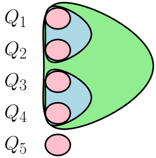
Half-gcd algorithm



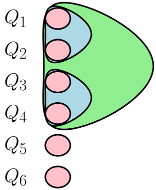
Half-gcd algorithm



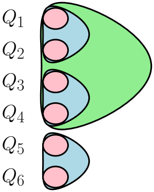
Half-gcd algorithm



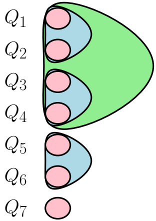
Half-gcd algorithm



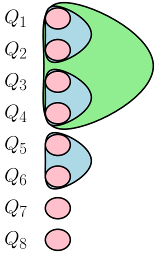
Half-gcd algorithm



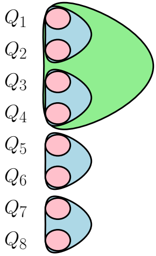
Half-gcd algorithm



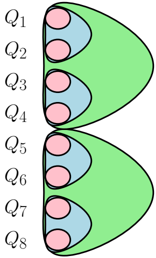
Half-gcd algorithm



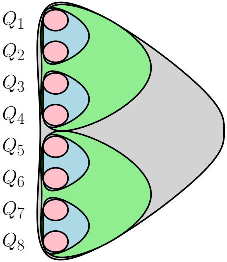
Half-gcd algorithm



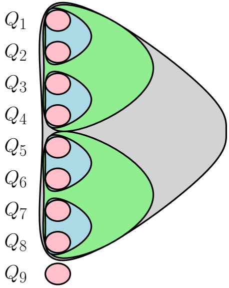
Half-gcd algorithm



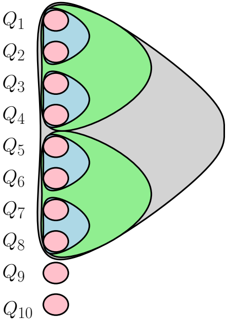
Half-gcd algorithm



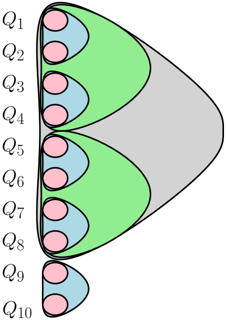
Half-gcd algorithm



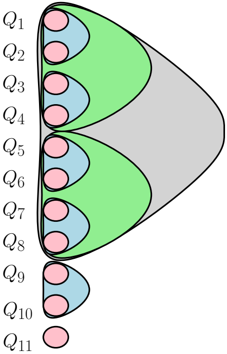
Half-gcd algorithm



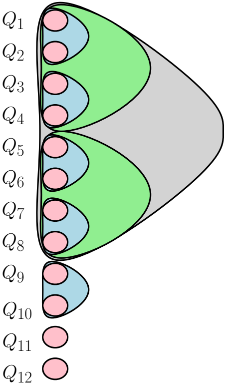
Half-gcd algorithm



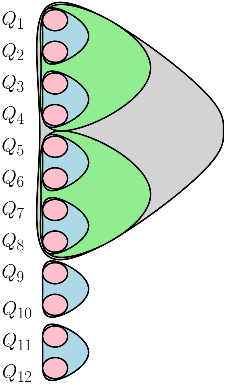
Half-gcd algorithm



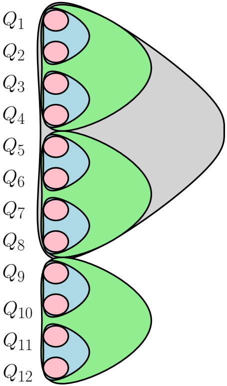
Half-gcd algorithm



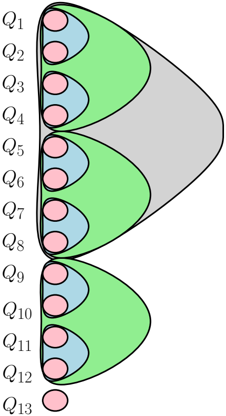
Half-gcd algorithm



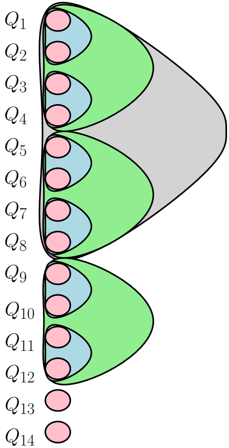
Half-gcd algorithm



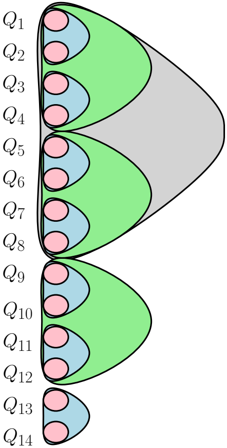
Half-gcd algorithm



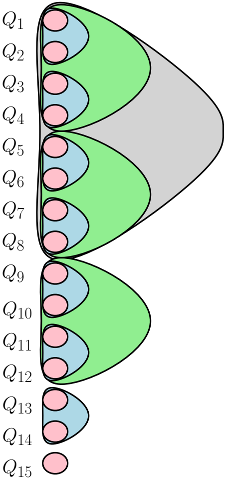
Half-gcd algorithm



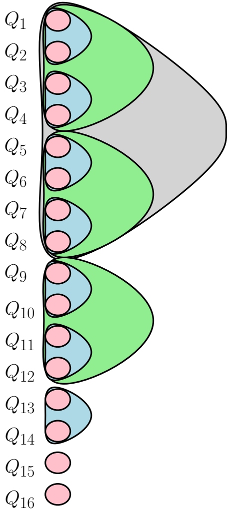
Half-gcd algorithm



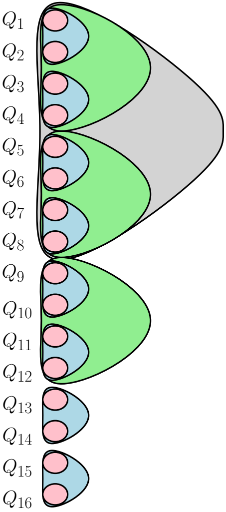
Half-gcd algorithm



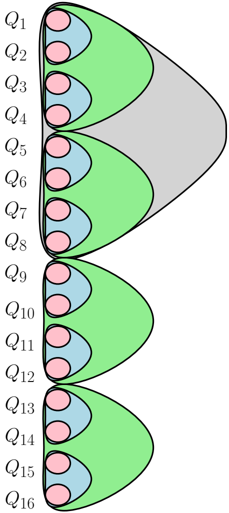
Half-gcd algorithm



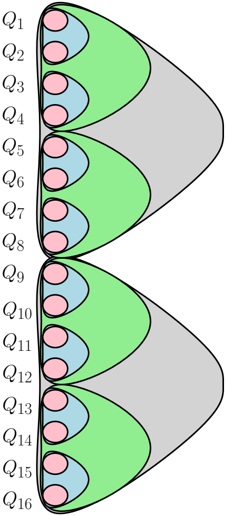
Half-gcd algorithm



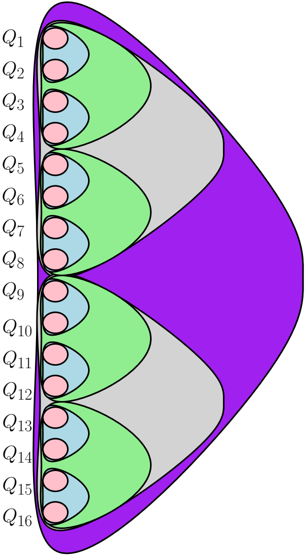
Half-gcd algorithm



Half-gcd algorithm



Half-gcd algorithm



Coefficient in the ring $R = \mathbb{K}[x]/\langle x^k \rangle$

- Pseudo-euclidean division not well-defined

Example: pseudo-euclidean division ill-defined for k=5

$$P = y^{10} + y$$
$$Q = xy + 1$$

$$x^{10}P - (x^9y^9 + \cdots - 1)Q = 1 - x^9$$

A differential equation: $P(y)$ and $Q(x, y)$ monic

- **Definition of the resultant**

$$Res(P, Q) = Q(x, \sigma_1) \cdots Q(x, \sigma_d)$$

- **Logarithmic derivative**

$$\frac{\partial_x Res(P, Q)}{Res(P, Q)} = \frac{\partial_x Q(\sigma_i)}{Q(\sigma_1)} + \cdots + \frac{\partial_x Q(\sigma_d)}{Q(\sigma_d)}$$

- **Assume U and V such that**

$$UP + VQ = 1$$

- **Sum of $V\partial_x Q$ over the polynomial roots of P**

$$\frac{\partial_x Res(P, Q)}{Res(P, Q)} = coeff_{y^{d-1}}(V\partial_x Q \partial_y P \text{ rem}_y \ P)$$

Algorithm in the general case

1. **Compute U, V and t minimal such that**

$$UP + VQ = x^t$$

2. **Compute the first non-zero term of $Res(P, Q)$**
3. **Solve the differential equation**

$$x^t \frac{\partial_x Res}{Res} = \mathrm{coeff}_{y^{d-1}}(V \partial_x Q \partial_y P \text{ rem}_y\ P) + \mathrm{coeff}_{y^{d-1}}(U \partial_x P \partial_y Q \text{ rem}_y\ Q)$$

Theorem: fast computation of the resultant modulo x^k

$$Res(P, Q) \quad \bmod x^k$$

can be computed in $\tilde{O}(dk)$ operations in $\mathbb{K}$.

Steps overview

Solving the differential equation

- Bostan, Chowdhury, Lebreton, Salvy, Schost. 2012.
  Power series solutions of singular (q)-differential equations.

Computing the first non-zero term of $Res(P, Q)$

- Recurrence relation

Computing $UP + VQ = x^t$

- $t = 0$: standard halg-gcd algorithm
- $t \geq 1$: variant of the half-gcd algorithm

First non-zero coefficient

Assume that we can compute in $\tilde{O}(dt)$ operations U, V and t minimal such that:

$$UP + VQ = x^t \bmod x^{t+1}$$

Lemma: recurrence relation on the resultant

If P is normal, we can find in $\tilde{O}(dt)$ operations N normal, M and W such that:

$$MP + NQ = x^t \bmod x^{t+1}$$

and

$$Res(P, Q) = x^{t(d_P - d_N)}(-1)^{d_P d_N} \frac{Lc(P)^{d_N + d_Q}}{Lc(N)^{d_M + d_P}}(1 + xW)Res(N, M)$$

Lemma: first non-zero coefficient

The valuation μ and the first non-zero coefficient of $Res(P, Q)$ can be computed in $\tilde{O}(d\mu)$ operations.

Given 2 polynomials in $\mathbb{C}[y]$

$$P = G \cdot A$$
$$Q = G \cdot B$$

Compute the extended gcd

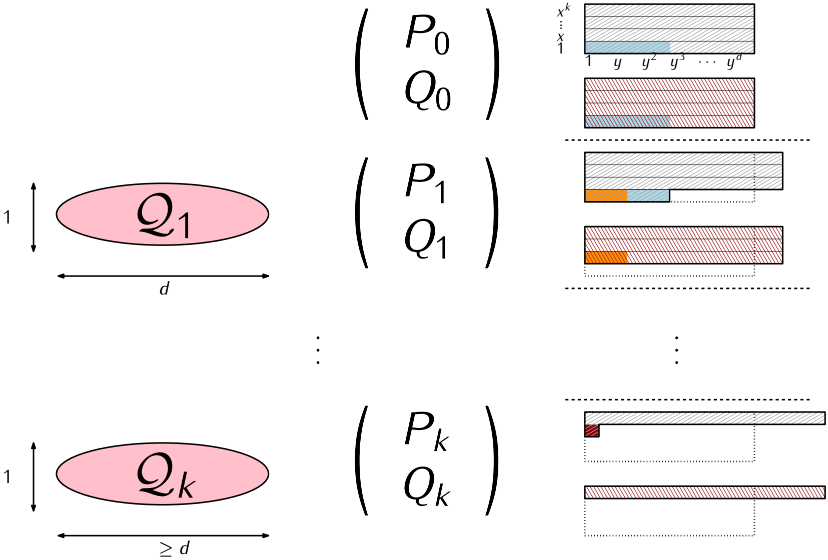
$$U \cdot A + V \cdot B = 1$$

Define the matrix $\mathcal{Q}$

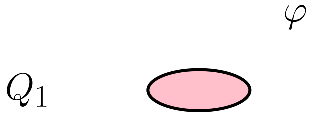
$$\begin{pmatrix} P \\ Q \end{pmatrix}$$

$$\mathcal{Q} := \begin{pmatrix} U & V \\ B & -A \end{pmatrix} \begin{pmatrix} G \\ 0 \end{pmatrix}$$

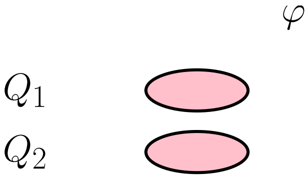
Computing $UP + VQ = x^t \bmod x^{t+1}$



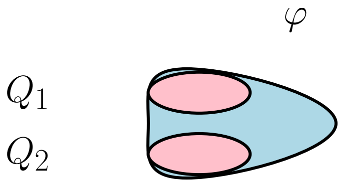
Half-gcd variant



Half-gcd variant



Half-gcd variant

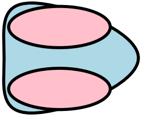


Half-gcd variant

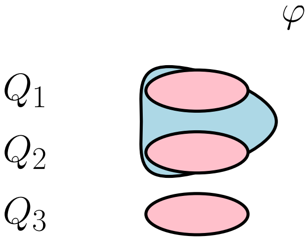
φ

Q_1

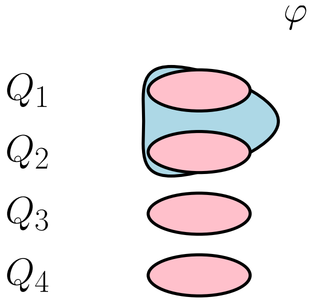
Q_2



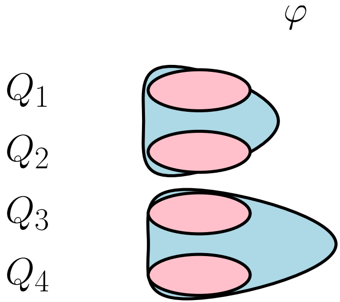
Half-gcd variant



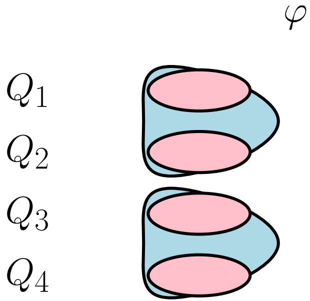
Half-gcd variant



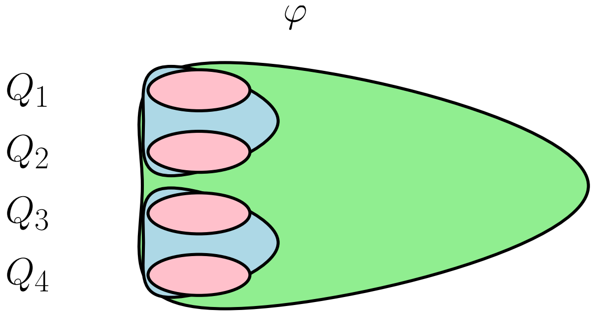
Half-gcd variant



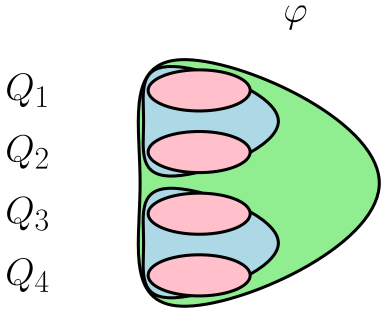
Half-gcd variant



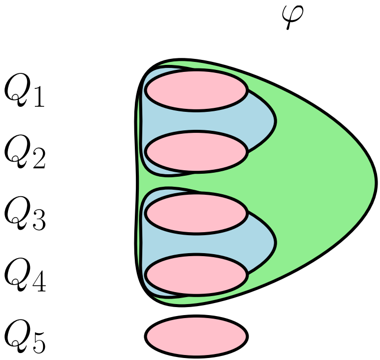
Half-gcd variant



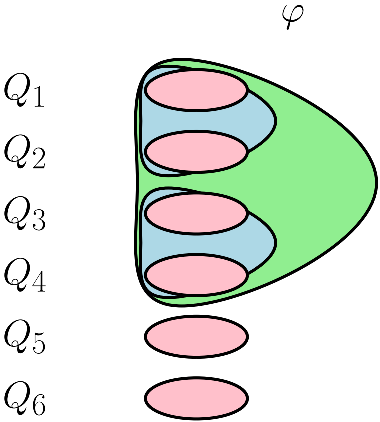
Half-gcd variant



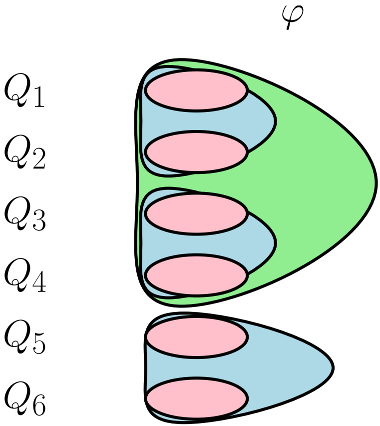
Half-gcd variant



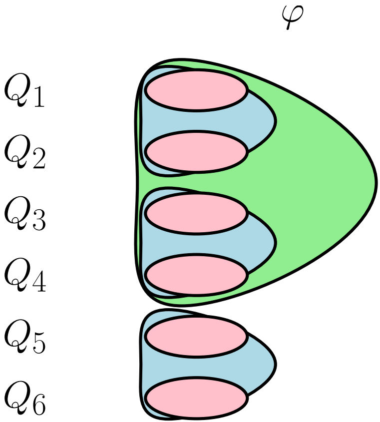
Half-gcd variant



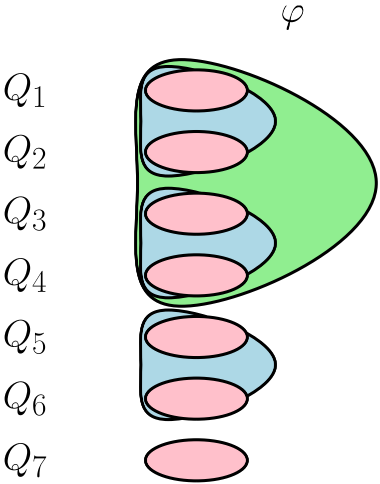
Half-gcd variant



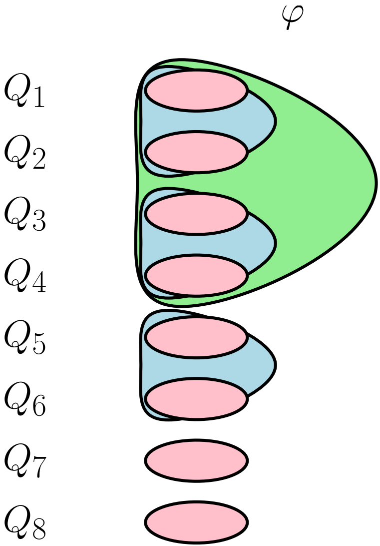
Half-gcd variant



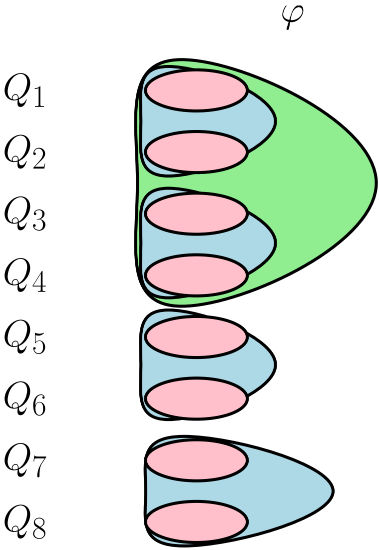
Half-gcd variant



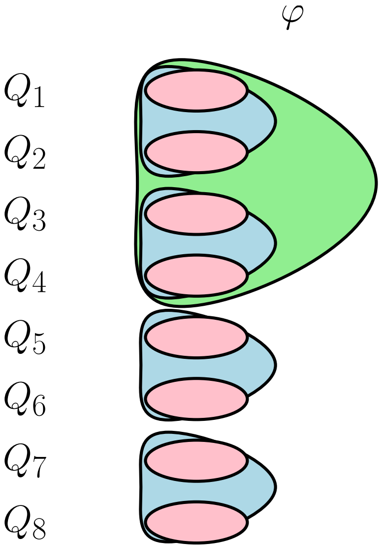
Half-gcd variant



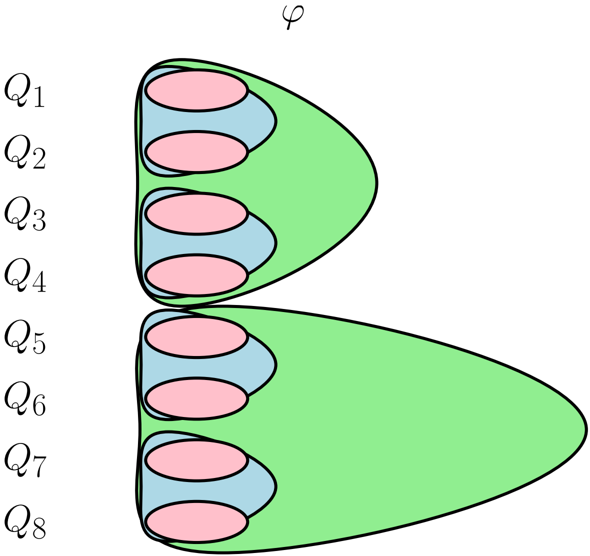
Half-gcd variant



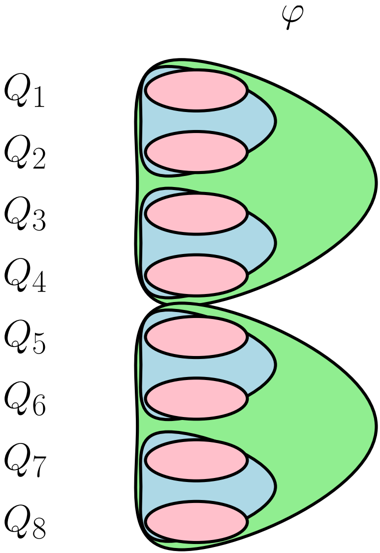
Half-gcd variant



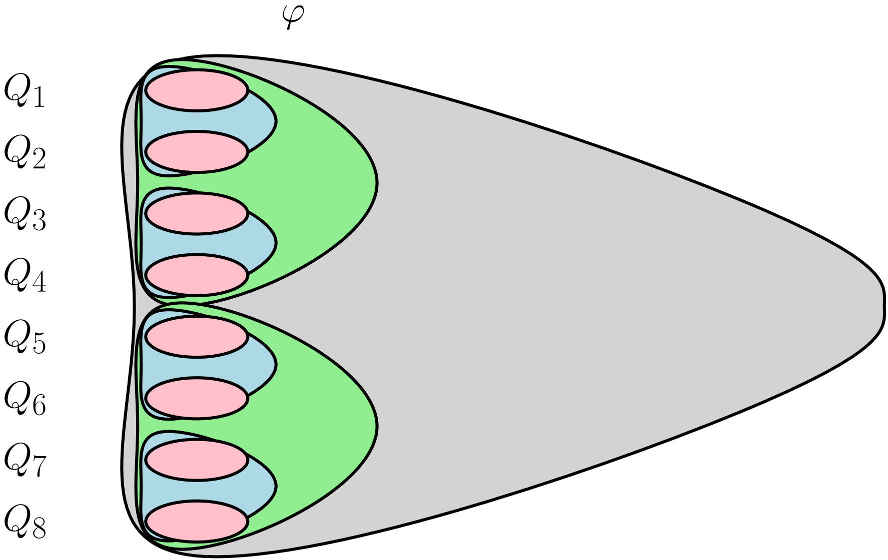
Half-gcd variant



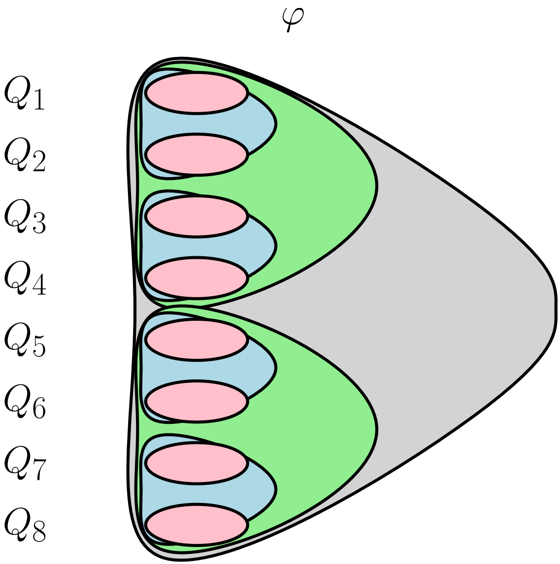
Half-gcd variant



Half-gcd variant



Half-gcd variant



Comparison with classical half-gcd

```
def GenericHalfgcd(n,s):
  s := projection(s,delay(n)+1)
  m := v(s) + n
  if n <= 0 or s=(0,0) then
    return Id
  R := GenericHalfgcd(n/2, s)
  u := R . s
  Q := conditionalQ(u, m - v(u))
  t := Q . u
  S := GenericHalfgcd(m - v(t), t)
  return phi(S . Q . R, s, n)
```

	Classical	Our case
projection	Highest coefficients in y	Smallest coefficients in x
v	Opposite of the degree in y	Valuation in x
conditionalQ	Euclidian quotient	Extended gcd
phi$\left(\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right), (P, Q), n\right)$	$\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right)$	$\begin{pmatrix} A \operatorname{rem}_y Q & B \operatorname{rem}_y P \\ C \operatorname{rem}_y Q & D \operatorname{rem}_y P \end{pmatrix} \bmod x^{n+1}$

Pseudo-inverse

Theorem: pseudo-inverse

Given a pair $s = (P, Q)$, the **GenericHalfgcd** algorithm returns U, V and the smallest t such that:

$$UP + VQ = x^t \bmod x^{t+1}$$

in $\tilde{O}(dt)$ operations.

Conclusion

Result

- Resultant modulo x^k **in** $\tilde{O}(dk)$
- **State-of-the-art** algorithm for $k = 1$ and $k = 2d^2$
- **Unlucky number** handled within $\tilde{O}(dk)$

Extensions in progress

- **Multivariate coefficients** up to degree k : by interpolation
- **First subresultant** : by formula

Open problems

- **Full subresultant chain**
- p**-adic coefficients** up to precision k (question by Caruso)

Conclusion

Result

- Resultant modulo x^k **in** $\tilde{O}(dk)$
- **State-of-the-art** algorithm for $k = 1$ and $k = 2d^2$
- **Unlucky number** handled within $\tilde{O}(dk)$

Extensions in progress

- **Multivariate coefficients** up to degree k : by interpolation
- **First subresultant** : by formula

Open problems

- **Full subresultant chain**
- ***p*-adic coefficients** up to precision k (question by Caruso)

Thank you