In-place polynomial arithmetic

Pascal Giorgi[1] **Bruno Grenet**[1] Daniel S. Roche[2]

JNCF'20 – Luminy – March 2-6, 2020

[1] LIRMM, Université de Montpellier
[2] CS Department, US Naval Academy

Multiplication of polynomials: $\mathsf{M}(n)$

Naive:	$O(n^2)$	
Karatsuba:	$O(n^{\log_2 3}) = O(n^{1.585})$	Karatsuba (1962)
Toom-3:	$O(n^{\log_3 5}) = O(n^{1.465})$	Toom (1963), Cook (1966)
FFT-based algorithms:		
	$O(n \log n)$ with $\omega^{2n} = 1$	Cooley, Tukey (1965)
	$O(n \log n \log \log n)$	Cantor, Kaltofen (1991)
	...	

Computer Algebra 101

Multiplication of polynomials: $\mathsf{M}(n)$

Naive:	$O(n^2)$	
Karatsuba:	$O(n^{\log_2 3}) = O(n^{1.585})$	Karatsuba (1962)
Toom-3:	$O(n^{\log_3 5}) = O(n^{1.465})$	Toom (1963), Cook (1966)
FFT-based algorithms:		
	$O(n \log n)$ with $\omega^{2n} = 1$	Cooley, Tukey (1965)
	$O(n \log n \log \log n)$	Cantor, Kaltofen (1991)
	...	

Other polynomial and power series operations:

Short and middle products	$\mathsf{M}(n) + O(n)$
Inversion, divisions:	$O(\mathsf{M}(n))$
Evaluation & interpolation:	$O(\mathsf{M}(n) \log n)$
GCD:	$O(\mathsf{M}(n) \log n)$

...

Computer Algebra 101

Multiplication of polynomials: $\mathsf{M}(n)$

Naive: $O(n^2)$

Karatsuba: $O(n^{\log_2 3}) = O(n^{1.585})$ — Karatsuba (1962)

Toom-3: $O(n^{\log_3 5}) = O(n^{1.465})$ — Toom (1963), Cook (1966)

FFT-based algorithms:

$O(n \log n)$ with $\omega^{2n} = 1$ — Cooley, Tukey (1965)

$O(n \log n \log \log n)$ — Cantor, Kaltofen (1991)

...

Other polynomial and power series operations:

Short and middle products	$\mathsf{M}(n) + O(n)$
Inversion, divisions:	$O(\mathsf{M}(n))$
Evaluation & interpolation:	$O(\mathsf{M}(n) \log n)$
GCD:	$O(\mathsf{M}(n) \log n)$

...

What about space complexity?

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one ring element

$\rightarrow$ Count *extra* registers used (not input nor output)

Space-complexity models

***Algebraic*-RAM Machine**

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one ring element

$\rightarrow$ Count *extra* registers used (not input nor output)

Read-write permissions

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one ring element

$\rightarrow$ Count *extra* registers used (not input nor output)

Read-write permissions

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication
- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one ring element

$\rightarrow$ Count *extra* registers used (not input nor output)

Read-write permissions

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication
- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end
- ✓ Read-only input / read-write output
  - *Reasonable* from a programmer's viewpoint

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$
Karatsuba's algorithm:		
Original (1962)	$O(n)$	$< 6.5n^{\log 3}$
Thomé (2002)	$n + O(\log n)$	$< 7n^{\log 3}$
Roche (2009)	$O(\log n)$	$< 10n^{\log 3}$

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$
Karatsuba's algorithm:		
Original (1962)	$O(n)$	$< 6.5n^{\log 3}$
Thomé (2002)	$n + O(\log n)$	$< 7n^{\log 3}$
Roche (2009)	$O(\log n)$	$< 10n^{\log 3}$
Toom-Cook algorithms:		
Toom-3 (1963)	$O(n)$	$< \frac{73}{4} n^{\log_3 5}$

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$
Karatsuba's algorithm:		
Original (1962)	$O(n)$	$< 6.5n^{\log 3}$
Thomé (2002)	$n + O(\log n)$	$< 7n^{\log 3}$
Roche (2009)	$O(\log n)$	$< 10n^{\log 3}$
Toom-Cook algorithms:		
Toom-3 (1963)	$O(n)$	$< \frac{73}{4}n^{\log_3 5}$
FFT/TFT-based algorithms (given $\omega^{2n} = 1$):		
Original (1965)	$O(n)$	$\sim 9n\log(2n)$
Roche (2009) if $n = 2^k$	$O(1)$	$\sim 11n\log(2n)$
Harvey, Roche (2010)	$O(1)$	$O(n\log(n))$

Space complexity analyses for other operations

	space	time
Naive algorithms:	$O(1)$	$O(n^2)$

(inversion, division, multipoint evaluation, interpolation)

Space complexity analyses for other operations

	space	time
Naive algorithms:	$O(1)$	$O(n^2)$
(inversion, division, multipoint evaluation, interpolation)		
Inversion and divisions:		Hanrot, Quercia, Zimmermann (2004)
Power series inversion:	$2n$	$2\mathsf{M}(n)$
Power series division:	$2.5n$	$2.5\mathsf{M}(n)$

Space complexity analyses for other operations

	space	time
Naive algorithms:	$O(1)$	$O(n^2)$
(inversion, division, multipoint evaluation, interpolation)		
Inversion and divisions:	Hanrot, Quercia, Zimmermann (2004)	
Power series inversion:	$2n$	$2\mathrm{M}(n)$
Power series division:	$2.5n$	$2.5\mathrm{M}(n)$
Division with remainder:	$\max(2.5m\text{-}n, 3n)$	$2.5\mathrm{M}(m) + \mathrm{M}(n)$
(in size $(m+n-1, n)$)	$4n$	$2\mathrm{M}(m) + 2\mathrm{M}(n)$

Space complexity analyses for other operations

	space	time	
Naive algorithms:	$O(1)$	$O(n^2)$	
(inversion, division, multipoint evaluation, interpolation)			
Inversion and divisions:		Hanrot, Quercia, Zimmermann (2004)	
Power series inversion:	$2n$	$2\mathrm{M}(n)$	
Power series division:	$2.5n$	$2.5\mathrm{M}(n)$	
Division with remainder:	$\max(2.5m\text{-}n, 3n)$	$2.5\mathrm{M}(m) + \mathrm{M}(n)$	
(in size $(m+n-1, n)$)	$4n$	$2\mathrm{M}(m) + 2\mathrm{M}(n)$	
Evaluation & interpolation:			
Bostan, Lecerf, Schost	$n \log n$	$1.5\mathrm{M}(n) \log n$	(eval)
(2003)	$n \log n$	$2.5\mathrm{M}(n) \log n$	(interp)

Space complexity analyses for other operations

	space	time	
Naive algorithms:	$O(1)$	$O(n^2)$	
(inversion, division, multipoint evaluation, interpolation)			
Inversion and divisions:	Hanrot, Quercia, Zimmermann (2004)		
Power series inversion:	$2n$	$2\mathsf{M}(n)$	
Power series division:	$2.5n$	$2.5\mathsf{M}(n)$	
Division with remainder:	$\max(2.5m\text{-}n, 3n)$	$2.5\mathsf{M}(m) + \mathsf{M}(n)$	
(in size $(m+n-1, n)$)	$4n$	$2\mathsf{M}(m) + 2\mathsf{M}(n)$	
Evaluation & interpolation:			
Bostan, Lecerf, Schost	$n \log n$	$1.5\mathsf{M}(n) \log n$	(eval)
(2003)	$n \log n$	$2.5\mathsf{M}(n) \log n$	(interp)
von zur Gathen, Shoup	n	$3.5\mathsf{M}(n) \log n$	(eval)
(1992)	$3n$	$5.5\mathsf{M}(n) \log n$	(interp)

Space complexity analyses for other operations

	space	time	
Naive algorithms:	$O(1)$	$O(n^2)$	
(inversion, division, multipoint evaluation, interpolation)			
Inversion and divisions:		Hanrot, Quercia, Zimmermann (2004)	
Power series inversion:	$2n$	$2\mathsf{M}(n)$	
Power series division:	$2.5n$	$2.5\mathsf{M}(n)$	
Division with remainder:	$\max(2.5m\text{-}n, 3n)$	$2.5\mathsf{M}(m) + \mathsf{M}(n)$	
(in size $(m+n-1, n)$)	$4n$	$2\mathsf{M}(m) + 2\mathsf{M}(n)$	
Evaluation & interpolation:			
Bostan, Lecerf, Schost	$n \log n$	$1.5\mathsf{M}(n) \log n$	(eval)
(2003)	$n \log n$	$2.5\mathsf{M}(n) \log n$	(interp)
von zur Gathen, Shoup	n	$3.5\mathsf{M}(n) \log n$	(eval)
(1992)	$3n$	$5.5\mathsf{M}(n) \log n$	(interp)
Giorgi, Grenet, Roche (2020)	$2n$	$5\ \mathsf{M}(n) \log n$	(interp)

Our problematic

Arithmetic on polynomials without extra memory?

Our problematic

Arithmetic on polynomials without extra memory?

- Polynomial multiplications
  - Karatsuba? Toom-Cook?
  - FFT/TFT without $\omega^{2^n} = 1$?
  - Other products (short and middle)?

Our problematic

Arithmetic on polynomials without extra memory?

- Polynomial multiplications
  - Karatsuba? Toom-Cook?
  - FFT/TFT without $\omega^{2^n} = 1$?
  - Other products (short and middle)?
- Other operations
  - Inversions and divisions
  - Evaluation & interpolation
  - GCD, ...

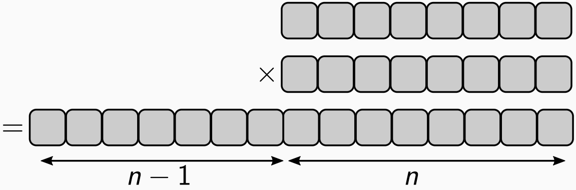
Our problematic

Arithmetic on polynomials without extra memory?

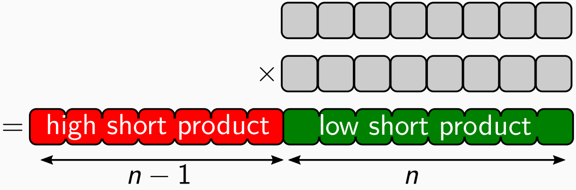
- Polynomial multiplications
  - ✓ Karatsuba? Toom-Cook?
  - ✓ FFT/TFT without $\omega^{2^n} = 1$?
  - ✓ Other products (short and middle)? (almost)
- Other operations
  - ✓ Inversions and divisions (almost)
  - ✓ Evaluation & interpolation
  - ? GCD, ...

Space-efficient polynomial products

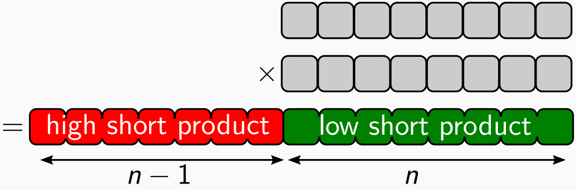
Short product



Short product



Short product

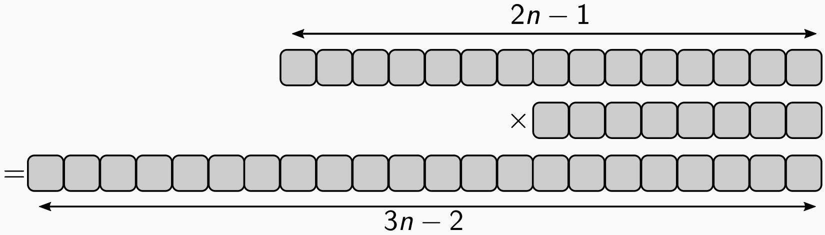


Formal definition

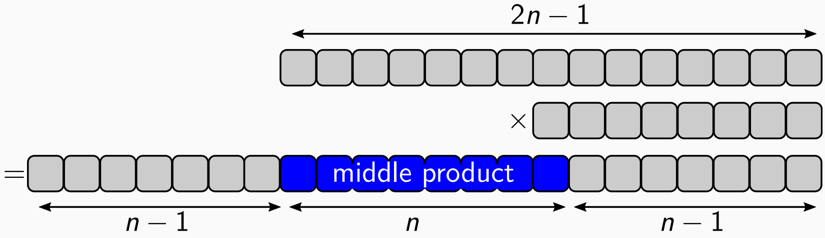
- $\mathrm{SP}_{\mathrm{lo}}(f,g) = f \cdot g \bmod X^n$
- $\mathrm{SP}_{\mathrm{hi}}(f,g) = f \cdot g \operatorname{div} X^n$

Example of use
Product of truncated power series

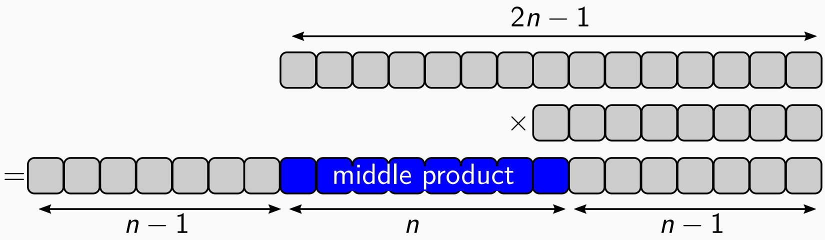
Middle product



Middle product



Middle product



Formal definition
$\mathrm{MP}(f, g) = \left(f \cdot g \operatorname{div} X^{n-1}\right) \bmod X^n$

Example of use
Newton iteration (division, square root, ...)

Multiplications as linear maps

Example:

$$f = 3X^2 + 2X + 1$$
$$g = X^2 + 2X + 4$$
$$fg = 3X^4 + 8X^3 + 17X^2 + 10X + 4$$

Multiplications as linear maps

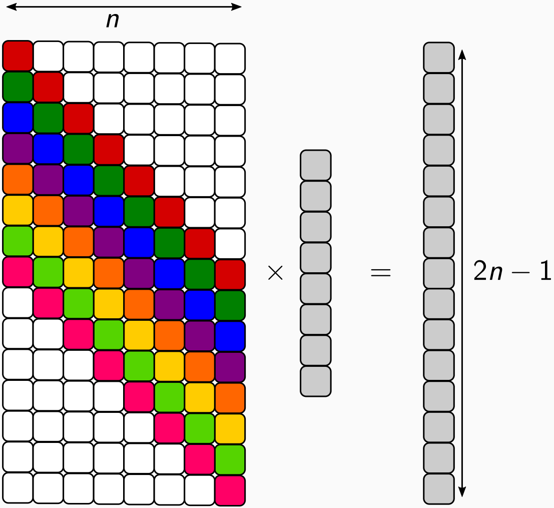
Example:

$$f = 3X^2 + 2X + 1$$
$$g = X^2 + 2X + 4$$
$$fg = 3X^4 + 8X^3 + 17X^2 + 10X + 4$$

$$\begin{bmatrix} 1 & & \\ 2 & 1 & \\ 3 & 2 & 1 \\ & 3 & 2 \\ & & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 10 \\ 17 \\ 8 \\ 3 \end{bmatrix}$$

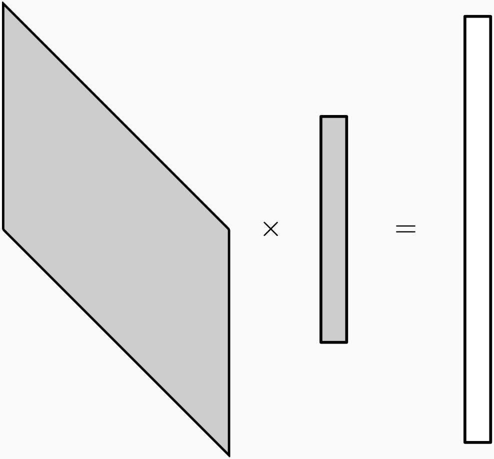
Multiplications as linear maps

Full product:



Multiplications as linear maps

Full product:



Framework

Reduction from out-of-place algorithms to in-place algorithms

- Oblivious of the actual out-of-place algorithm
- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Framework

Reduction from out-of-place algorithms to in-place algorithms

- Oblivious of the actual out-of-place algorithm
- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Goal

- Space complexity: $O(1)$
- Time complexity: closest to the out-of-place algorithm

Framework

Reduction from out-of-place algorithms to in-place algorithms

- Oblivious of the actual out-of-place algorithm
- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Goal

- Space complexity: $O(1)$
- Time complexity: closest to the out-of-place algorithm

Techniques

- Oracle calls in smaller size
- *Fake* padding of inputs (*cf.* strides in lin. alg.)
- **Tail** recursive call (avoid $O(\log n)$ stack)

Framework

Reduction from out-of-place algorithms to in-place algorithms

- Oblivious of the actual out-of-place algorithm
- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Goal

- Space complexity: $O(1)$
- Time complexity: closest to the out-of-place algorithm

Techniques

- Oracle calls in smaller size
- *Fake* padding of inputs (*cf.* strides in lin. alg.)
- **Tail** recursive call (avoid $O(\log n)$ stack)

Similar approach for matrix mul.: Boyer, Dumas, Pernet, Zhou (2009)

Our results

Theorem

- In-place (half-additive) full product in time $(2c+7)\mathsf{M}(n)$
- In-place short product in time $(2c+5)\mathsf{M}(n)$
- In-place middle product in time $O(\mathsf{M}(n)\log n)$

  (or $O(\mathsf{M}(n))$ if $\mathsf{M}(n) = \Omega(n^{1+\delta})$)

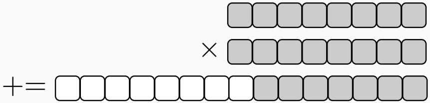
Our results

Theorem

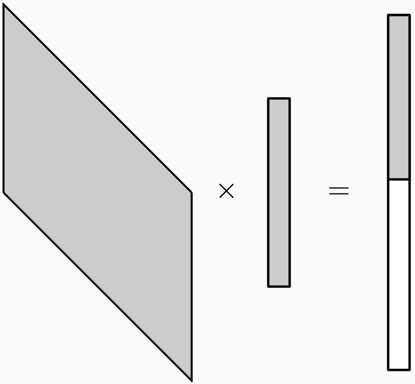
- In-place (half-additive) full product in time $(2c+7)\mathsf{M}(n)$
- In-place short product in time $(2c+5)\mathsf{M}(n)$
- In-place middle product in time $O(\mathsf{M}(n)\log n)$
  (or $O(\mathsf{M}(n))$ if $\mathsf{M}(n) = \Omega(n^{1+\delta})$)

Half-additive full product:

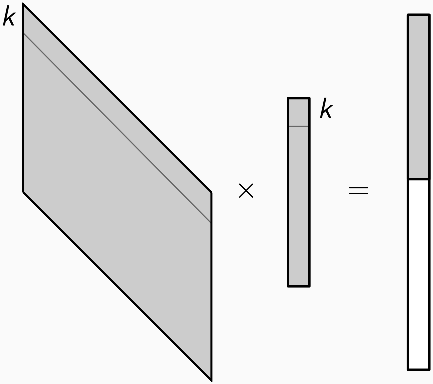
$$h \leftarrow h + f \cdot g \text{ where } \deg(h) < \deg(f), \deg(g)$$



In-place FP$^+$ from out-of-place FP

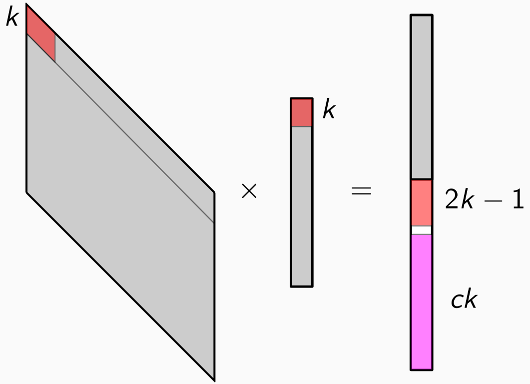


In-place FP$^+$ from out-of-place FP



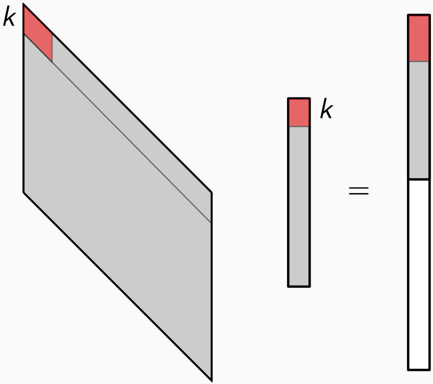
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



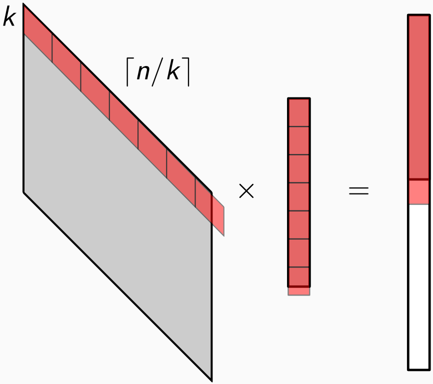
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



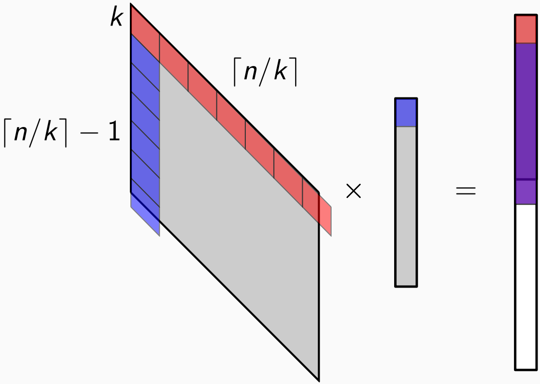
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



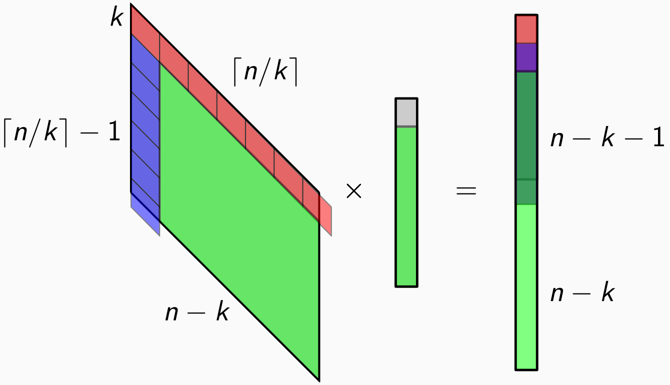
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



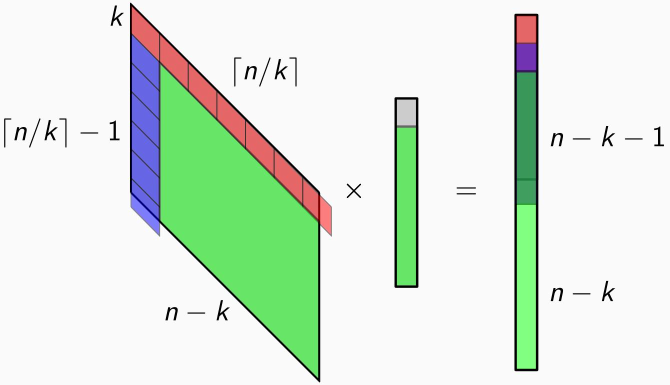
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



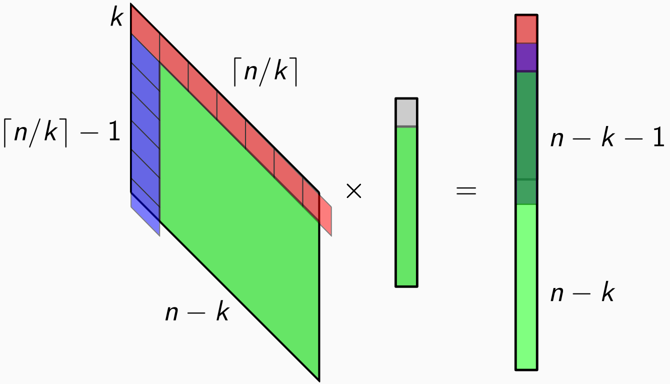
$$(f_0 + X^k\hat{f}) \cdot (g_0 + X^k\hat{g}) = f_0 g_0 + X^k(f_0\hat{g} + \hat{f} g_0) + X^{2k}\hat{f}\hat{g}$$

In-place FP$^+$ from out-of-place FP



- $ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}$
- $T(n) = (2\lceil n/k \rceil - 1)(\mathrm{M}(k) + 2k - 1) + T(n-k)$

In-place FP^+ from out-of-place FP



- $ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}$
- $T(n) = (2\lceil n/k \rceil - 1)(\mathsf{M}(k) + 2k - 1) + T(n-k)$

$$T(n) \leq (2c+7)\mathsf{M}(n) + o(\mathsf{M}(n))$$

Newton iteration: inversion and divisions

Standard Newton iteration for inversion

Lemma
If $G_k = F^{-1} \bmod X^k$, $G_k + (1 - G_k F) G_k = F^{-1} \bmod X^{2k}$

Standard Newton iteration for inversion

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..2k[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..2k[}, G_{[0..k[}), G_{[0..k[})$$

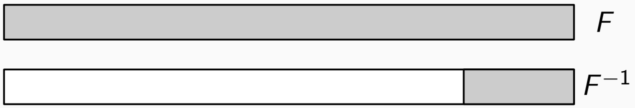
then $G_{[0..2k[}$ contains $F^{-1} \bmod X^{2k}$

Standard Newton iteration for inversion

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..2k[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..2k[}, G_{[0..k[}), G_{[0..k[})$$

then $G_{[0..2k[}$ contains $F^{-1} \bmod X^{2k}$

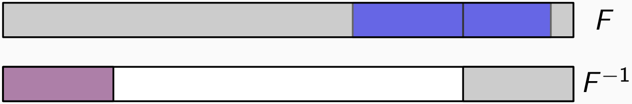


Standard Newton iteration for inversion

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..2k[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..2k[}, G_{[0..k[}), G_{[0..k[})$$

then $G_{[0..2k[}$ contains $F^{-1} \bmod X^{2k}$

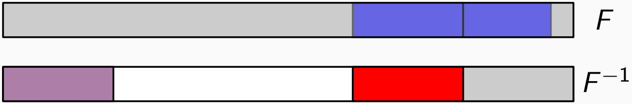


Standard Newton iteration for inversion

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..2k[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..2k[}, G_{[0..k[}), G_{[0..k[})$$

then $G_{[0..2k[}$ contains $F^{-1} \bmod X^{2k}$

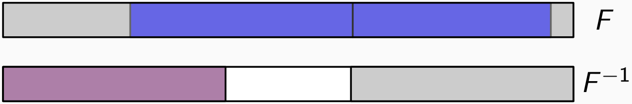


Standard Newton iteration for inversion

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..2k[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..2k[}, G_{[0..k[}), G_{[0..k[})$$

then $G_{[0..2k[}$ contains $F^{-1} \bmod X^{2k}$

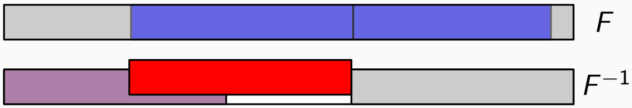


Standard Newton iteration for inversion

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..2k[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..2k[}, G_{[0..k[}), G_{[0..k[})$$

then $G_{[0..2k[}$ contains $F^{-1} \bmod X^{2k}$



In-place algorithm

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..k+\ell[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..k+\ell[}, G_{[0..k[}), G_{[0..\ell[})$$

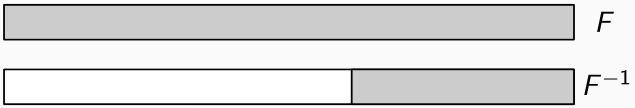
then $G_{[0..k+\ell[}$ contains $F^{-1} \bmod X^{k+\ell}$, where $\ell \leq k$

In-place algorithm

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..k+\ell[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..k+\ell[}, G_{[0..k[}), G_{[0..\ell[})$$

then $G_{[0..k+\ell[}$ contains $F^{-1} \bmod X^{k+\ell}$, where $\ell \leq k$



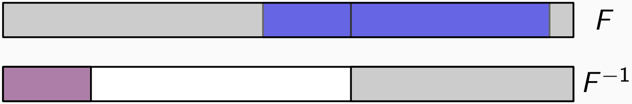
- Compute less and less coefficients at each step
- *Accelerating* and *decelerating* phases

In-place algorithm

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..k+\ell[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..k+\ell[}, G_{[0..k[}), G_{[0..\ell[})$$

then $G_{[0..k+\ell[}$ contains $F^{-1} \bmod X^{k+\ell}$, where $\ell \leq k$



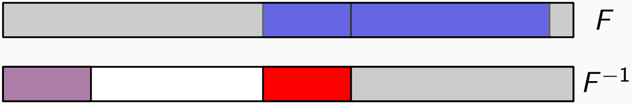
- Compute less and less coefficients at each step
- *Accelerating* and *decelerating* phases

In-place algorithm

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..k+\ell[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..k+\ell[}, G_{[0..k[}), G_{[0..\ell[})$$

then $G_{[0..k+\ell[}$ contains $F^{-1} \bmod X^{k+\ell}$, where $\ell \leq k$



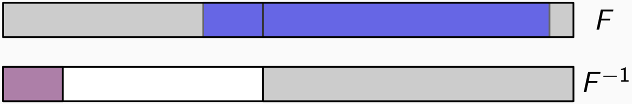
- Compute less and less coefficients at each step
- *Accelerating* and *decelerating* phases

In-place algorithm

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..k+\ell[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..k+\ell[}, G_{[0..k[}), G_{[0..\ell[})$$

then $G_{[0..k+\ell[}$ contains $F^{-1} \bmod X^{k+\ell}$, where $\ell \leq k$



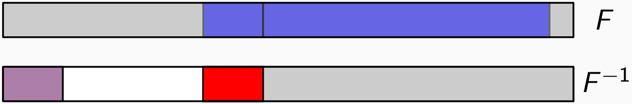
- Compute less and less coefficients at each step
- *Accelerating* and *decelerating* phases

In-place algorithm

Lemma
Given $F^{-1} \bmod X^k$ in $G_{[0..k[}$, after

$$G_{[k..k+\ell[} \leftarrow -\mathrm{SP}(\mathrm{MP}(F_{[1..k+\ell[}, G_{[0..k[}), G_{[0..\ell[})$$

then $G_{[0..k+\ell[}$ contains $F^{-1} \bmod X^{k+\ell}$, where $\ell \leq k$



- Compute less and less coefficients at each step
- *Accelerating* and *decelerating* phases

Results

Theorem

- Given F at precision n, one can compute $F^{-1} \bmod X^n$ in time $O(\mathsf{M}(n) \log n)$ without extra space.
- Given F and G at precision n, one can compute $F/G \bmod X^n$ in time $O(\mathsf{M}(n) \log n)$ without extra space.

Results

Theorem

- Given F at precision n, one can compute $F^{-1} \bmod X^n$ in time $O(\mathsf{M}(n) \log n)$ without extra space.
- Given F and G at precision n, one can compute $F/G \bmod X^n$ in time $O(\mathsf{M}(n) \log n)$ without extra space.
- Given F and G at precision n, one can compute $F/G \bmod X^n$ in time $O(\mathsf{M}(n))$ without extra space *if F can be erased*.

Results

Theorem

- Given F at precision n, one can compute $F^{-1} \bmod X^n$ in time $O(\mathsf{M}(n)\log n)$ without extra space.
- Given F and G at precision n, one can compute $F/G \bmod X^n$ in time $O(\mathsf{M}(n)\log n)$ without extra space.
- Given F and G at precision n, one can compute $F/G \bmod X^n$ in time $O(\mathsf{M}(n))$ without extra space *if F can be erased*.

Update step (Generalized Karp-Markstein's trick)**:**

$$Q_{[k..k+\ell[} \leftarrow \mathrm{SP}(G^{-1}_{[0..\ell[}, F_{[k..k+\ell[} - \mathrm{MP}(G_{[1..k+\ell[}, Q_{[0..k[}))$$

- Since $F_{[0..k[}$ not needed anymore, can serve as work space

Euclidean division

Theorem

Given size-$(2n-1)$ polynomial A and size-n polynomial B, one can compute

- $(A \operatorname{div} B, A \bmod B)$ in time $\simeq 6.29\mathsf{M}(n)$ without extra space [1]
- $A \operatorname{div} B$ in time $O(\mathsf{M}(m)\log m)$ without extra space

1. $4\mathsf{M}(n)$ without space restrictions

Euclidean division

Theorem
Given size-$(2n-1)$ polynomial A and size-n polynomial B, one can compute

- $(A \operatorname{div} B, A \bmod B)$ in time $\simeq 6.29\mathsf{M}(n)$ without extra space [1]
- $A \operatorname{div} B$ in time $O(\mathsf{M}(m) \log m)$ without extra space

Remark
The best known algorithm for computing $A \bmod B$ only, in-place, requires $O(n^2)$ operations

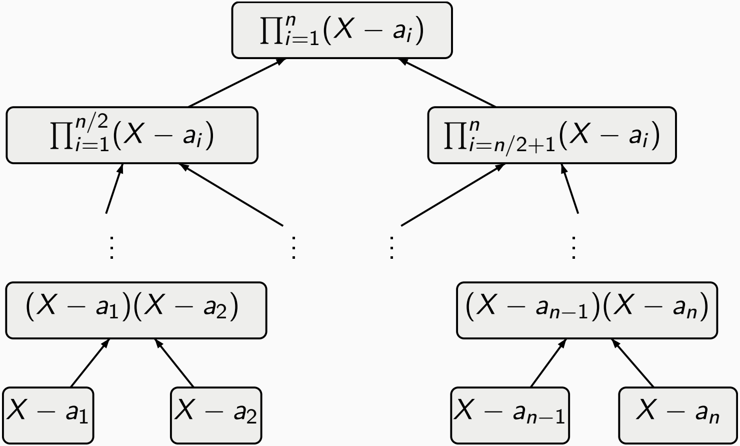
1. $4\mathsf{M}(n)$ without space restrictions

Multipoint evaluation and interpolation

Multipoint evaluation

Evaluate a size-n polynomial F on $(a_1, \ldots, a_n)$

- Classical algorithm computes the *subproduct tree*: size $n \log n$



adapted from: Bostan et al. Algorithmes Efficaces en Calcul Formel, 2017

Multipoint evaluation

Evaluate a size-n polynomial F on $(a_1, \ldots, a_n)$

- Classical algorithm computes the *subproduct tree*: size $n \log n$
- von zur Gathen, Shoup (1992):
  - evaluate by groups of $({}^{n}/_{\log n})$ points
  - space: $O(({}^{n}/_{\log n}) \log({}^{n}/_{\log n})) = O(n)$
  - time: $O(\log n \times \mathsf{M}({}^{n}/_{\log n}) \log({}^{n}/_{\log n})) = O(\mathsf{M}(n) \log n)$

Multipoint evaluation

Evaluate a size-n polynomial F on $(a_1, \ldots, a_n)$

- Classical algorithm computes the *subproduct tree*: size $n \log n$
- von zur Gathen, Shoup (1992):
  - evaluate by groups of $({}^{n}/_{\log n})$ points
  - space: $O(({}^{n}/_{\log n}) \log({}^{n}/_{\log n})) = O(n)$
  - time: $O(\log n \times \mathsf{M}({}^{n}/_{\log n}) \log({}^{n}/_{\log n})) = O(\mathsf{M}(n) \log n)$
- Our technique:
  - evaluate by smaller and smaller groups of points
  - space complexity $O(1)$ using free output space as work space
  - Still time $O(\mathsf{M}(n) \log n)$

Interpolation

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

- Classical algorithm
  - Compute $M = \prod_i (X - a_i)$ and its derivative M'
  - Compute $F/M = \sum_i \frac{y_i}{M'(a_i)} \frac{1}{X - a_i}$ using a D&C alg.
  - Time $O(\mathsf{M}(n)\log n)$; space $n \log n$ for the evaluation of $M'(a_i)$

Interpolation

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

- Classical algorithm
  - Compute $M = \prod_i (X - a_i)$ and its derivative M'
  - Compute $F/M = \sum_i \frac{y_i}{M'(a_i)} \frac{1}{X - a_i}$ using a D&C alg.
  - Time $O(\mathsf{M}(n) \log n)$; space $n \log n$ for the evaluation of $M'(a_i)$
- Using space-$O(n)$ evaluation: $O(n)$ space with time $O(\mathsf{M}(n) \log n)$

Interpolation

Given $(a_1, y_1), \dots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

- Classical algorithm
  - Compute $M = \prod_i (X - a_i)$ and its derivative M'
  - Compute $F/M = \sum_i \frac{y_i}{M'(a_i)} \frac{1}{X - a_i}$ using a D&C alg.
  - Time $O(\mathsf{M}(n) \log n)$; space $n \log n$ for the evaluation of $M'(a_i)$
- Using space-$O(n)$ evaluation: $O(n)$ space with time $O(\mathsf{M}(n) \log n)$
- Using space-$O(1)$ evaluation: still $O(n)$ space…

Our approach

Given (a_1, y_1), ..., (a_n, y_n), compute a size-n poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space

Our approach

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space

$$F(X) = M(X) \sum_{j=1}^{n} \frac{y_j}{M'(a_j)} \frac{1}{(X - a_j)}$$

Our approach

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space

$$F(X) = M(X) \sum_{j=1}^{n} \frac{y_j}{M'(a_j)} \frac{1}{(X - a_j)}$$

$$= M(X) \sum_{i=1}^{n/k} \sum_{j=1+k(i-1)}^{ki} \frac{y_j}{M'(a_j)} \frac{1}{(X - a_j)}$$

Our approach

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space

$$F(X) = M(X) \sum_{j=1}^{n} \frac{y_j}{M'(a_j)} \frac{1}{(X - a_j)}$$

$$= M(X) \sum_{i=1}^{n/k} \sum_{j=1+k(i-1)}^{ki} \frac{y_j}{M'(a_j)} \frac{1}{(X - a_j)}$$

$$= M(X) \sum_{i=1}^{n/k} \frac{N_i(X)}{T_i(X)} = \sum_{i=1}^{n/k} N_i(X) S_i(X)$$

$$\text{where } T_i = \prod_{j=1+k(i-1)}^{ki} (X - a_j) \text{ and } S_i = \frac{M}{T_i}$$

Our approach

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space

$$F \bmod X^k = \sum_{i=1}^{n/k} N_i (S_i \bmod X^k) \mod X^k \quad \text{where } S_i = M/T_i$$

Our approach

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space

$$F \bmod X^k = \sum_{i=1}^{n/k} N_i (S_i \bmod X^k) \mod X^k \quad \text{where } S_i = M/T_i$$

- Compute each N_i using interpolation
- Compute each T_i using a D&C approach
- Deduce each $S_i \bmod X^k = \prod_{j \neq i} T_j \mod X^k$

Our approach

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space
2. Given k, $F \bmod X^\ell$, compute $F \bmod X^{\ell+k}$ using $O(k)$ space

Our approach

Given (a_1, y_1), ..., (a_n, y_n), compute a size-n poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space
2. Given k, $F \bmod X^\ell$, compute $F \bmod X^{\ell+k}$ using $O(k)$ space
    - Interpolate $(F \operatorname{div} X^\ell) \bmod X^k$

$$y_i \rightsquigarrow \frac{y_i - (F \bmod X^k)(a_i)}{a_i^\ell}$$

    - Use of multipoint evaluation

Our approach

Given (a_1, y_1), ..., (a_n, y_n), compute a size-n poly. F s.t. $F(a_i) = y_i$

1. Given k, compute F mod X^k using $O(k)$ space
2. Given k, F mod X^ℓ, compute F mod $X^{\ell+k}$ using $O(k)$ space
3. Compute smaller and smaller chunks of F

Our approach

Given $(a_1, y_1), \ldots, (a_n, y_n)$, compute a size-$n$ poly. F s.t. $F(a_i) = y_i$

1. Given k, compute $F \bmod X^k$ using $O(k)$ space
2. Given k, $F \bmod X^\ell$, compute $F \bmod X^{\ell+k}$ using $O(k)$ space
3. Compute smaller and smaller chunks of F

Theorem
Multipoint evaluation and interpolation can be computed in time $O(\mathsf{M}(n) \log n)$ without extra space

Summary of the results

	space	time
Polynomial multiplication		
Full product	$O(1)$	$(2c+7)\mathsf{M}(n)$
Short product	$O(1)$	$(2c+5)\mathsf{M}(n))$
Middle product	$O(1)$	$\mathsf{M}(n)\log_{\frac{c+2}{c+1}}(n)^{\star}$
Inversion and divisions:		
Power series inversion:	$O(1)$	$3.81\mathsf{M}(n)\log(n)^{\star}$
Power series division:	$O(1)$	$4.50\mathsf{M}(n)\log(n)^{\star}$
Division with remainder:	$O(1)$	$6.29\mathsf{M}(n)$
Evaluation & interpolation:		
Evaluation	$O(1)$	$11.61\mathsf{M}(n)\log n$
Interpolation	$O(1)$	$105\mathsf{M}(n)\log n$

$^{\star}O(\mathsf{M}(n))$ if $\mathsf{M}(n) = \Omega(n^{1+\delta})$

Conclusion

- Fine analysis of space-time complexities of polynomial arith.
- In-place algorithms with (often) same asymptotic complexity for
  - Polynomial products (full, middle, short)
  - Power series inversion & division, Euclidean division
  - Multipoint evaluation, interpolation

Conclusion

- Fine analysis of space-time complexities of polynomial arith.
- In-place algorithms with (often) same asymptotic complexity for
  - Polynomial products (full, middle, short)
  - Power series inversion & division, Euclidean division
  - Multipoint evaluation, interpolation
- Space-aware reductions between polynomial products
  - Different situation than without space restriction
  - Questions related to the transposition principle

Conclusion

- Fine analysis of space-time complexities of polynomial arith.
- In-place algorithms with (often) same asymptotic complexity for
  - Polynomial products (full, middle, short)
  - Power series inversion & division, Euclidean division
  - Multipoint evaluation, interpolation
- Space-aware reductions between polynomial products
  - Different situation than without space restriction
  - Questions related to the transposition principle

Main open problems

- Remove $\log(n)$ factor for the middle product & inversion
- Other operations (GCD, ...); general characterization
- Case of integer arithmetic
- Practical issues

Conclusion

- Fine analysis of space-time complexities of polynomial arith.
- In-place algorithms with (often) same asymptotic complexity for
  - Polynomial products (full, middle, short)
  - Power series inversion & division, Euclidean division
  - Multipoint evaluation, interpolation
- Space-aware reductions between polynomial products
  - Different situation than without space restriction
  - Questions related to the transposition principle

Main open problems

- Remove $\log(n)$ factor for the middle product & inversion
- Other operations (GCD, . . .); general characterization
- Case of integer arithmetic
- Practical issues

Thank you!