

Implémentation logicielle de l'algorithme de composition modulaire de Kedlaya-Umans

Jocelyn Ryckeghem

Université Pierre et Marie Curie (Paris VI)
Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI)

13 septembre 2016

Plan

① Introduction

② Composition modulaire rapide

Substitution de Kronecker inverse

Réduction à l'évaluation multipoint multivariée

③ Évaluation multipoint multivariée rapide

EMM dans \mathbb{F}_p , p un petit nombre premier

EMM dans $\mathbb{Z}/r\mathbb{Z}$

④ Implémentation

Contexte

Techniques

Résultats

Composition modulaire

Problème de la composition modulaire

Soit R un anneau commutatif.

Soient $f(X)$, $g(X)$ et $h(X) \in R[X]$ des polynômes univariés, de degré au plus $n - 1$.

On souhaite calculer $f(g(X)) \bmod h(X)$.

$$f(X) = \sum_{i=0}^{n-1} f_i X^i$$

$$f(g(X)) \bmod h(X) = \sum_{i=0}^{n-1} f_i (g(X)^i \bmod h(X))$$

Complexité : $O(n^2)$.

Méthodes sous-quadratiques

Brent & Kung, 1978

Réduction au produit matriciel $\Rightarrow O(n^{\frac{\omega+1}{2}})$.

$\omega = 3$: $O(n^2)$, équivalent à l'algorithme naïf.

$\omega \geq 2$: $\Omega(n^{1.5})$.

Kedlaya-Umans, 2011

Réduction à l'évaluation multipoint multivariée $\Rightarrow O(n^{1+o(1)})$.

Kedlaya-Umans

K. S. Kedlaya et C. Umans : Fast Polynomial Factorization and Modular Composition, SIAM Journal on Computing, Vol. 40, No. 6, pp. 1767–1802, 2011

Composition modulaire dans $\mathbb{F}_q[X]$ avec \mathbb{F}_q une extension de corps.

Complexité : $\tilde{O}(n \log(q))$ opérations élémentaires.

Applications proposées dans l'article :

- Factorisation de polynôme : $\tilde{O}(n^{1.5} \log(q) + n \log^2(q))$.
- Test d'irréductibilité de polynôme : $\tilde{O}(n \log^2(q))$.
- Calcul du polynôme minimal : $\tilde{O}(n \log(q))$.



Substitution de Kronecker inverse : $\psi_{d,m}$

On pose $n \simeq d^m$.

Exemple pour $n = 1000$:
on choisit $d = 10$, ainsi $m = 3$.

$$\begin{aligned}\psi_{10,3} : X &\mapsto X_0 \\ X^{10} &\mapsto X_1 \\ X^{100} &\mapsto X_2 \\ X^{259} &\mapsto X_2^2 X_1^5 X_0^9\end{aligned}$$

Substitution de Kronecker inverse

$$f(X) = f(X, X^d, X^{d^2}, \dots, X^{d^{m-1}}) = f(X_0, X_1, X_2, \dots, X_{m-1})$$

$$X^{d^i} = X_i$$

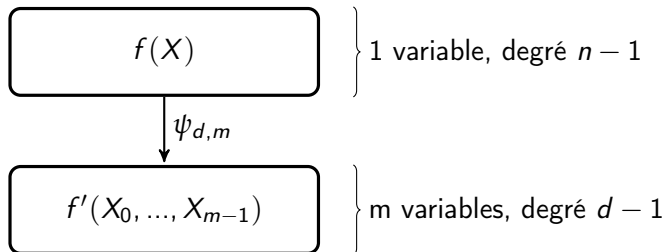
Transformation du problème

$$n \simeq d^m$$

$$\boxed{f(X)} \quad \left. \vphantom{\boxed{f(X)}} \right\} \text{1 variable, degré } n - 1$$

Transformation du problème

$$n \simeq d^m, X_i = X^{d^i}$$



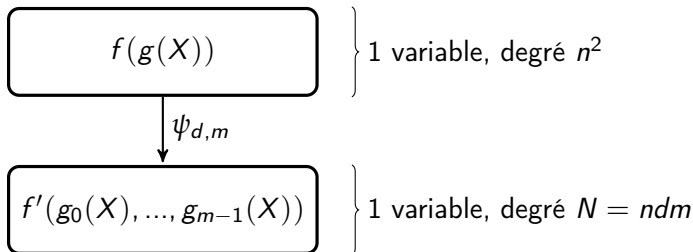
Transformation du problème

$$n \simeq d^m, X_i = X^{d^i}$$

$$\boxed{f(g(X))} \quad \left. \vphantom{\boxed{f(g(X))}} \right\} \text{1 variable, degré } n^2$$

Transformation du problème

$$n \simeq d^m, X_i = X^{d^i}, g_i(X) = g(X)^{d^i} \bmod h(X)$$



Évaluation multipoint multivariée

Problème de l'Évaluation Multipoint Multivariée (EMM)

Soit $f(X_0, \dots, X_{m-1}) \in R[X_0, \dots, X_{m-1}]$ de degré au plus $d - 1$ pour chaque variable.

Soient $\alpha_0, \dots, \alpha_{N-1} \in R^m$ des points d'évaluations.

On souhaite calculer $f(\alpha_0), \dots, f(\alpha_{N-1})$.

Idée de l'algorithme de composition modulaire de Kedlaya-Umans :
réduction au problème EMM.

Algorithme de composition modulaire

$$(X)$$

$$g(X)$$

$$f(X)$$

$$f(g(X))[h(X)]$$

Algorithme de composition modulaire

$$(X)$$

$$(X_0, \dots, X_{m-1})$$

$$g(X)$$

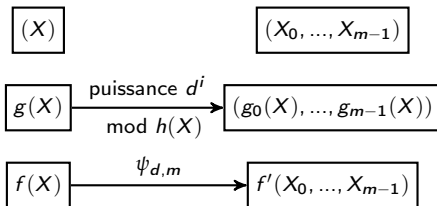
$$f(X)$$

 $\psi_{d,m}$

$$f'(X_0, \dots, X_{m-1})$$

$$f(g(X))[h(X)]$$

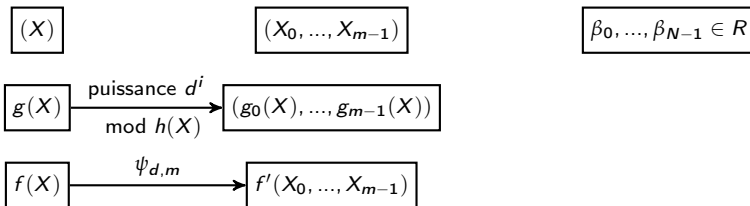
Algorithme de composition modulaire



$$f(g(X))[h(X)]$$

$$g_i(X) = g(X)^{d^i}.$$

Algorithme de composition modulaire

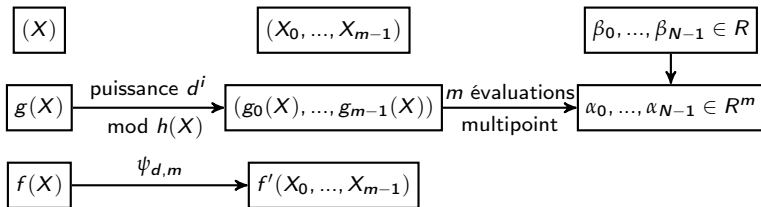


$$f(g(X))[h(X)]$$

$$g_i(X) = g(X)^{d^i}.$$

On choisit les $\beta_0, \dots, \beta_{N-1}$ distincts.

Algorithme de composition modulaire

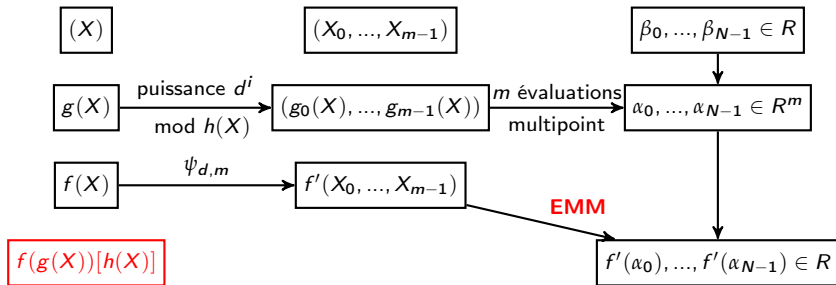


$$f(g(X))[h(X)]$$

$$g_i(X) = g(X)^{d^i}.$$

On choisit les $\beta_0, \dots, \beta_{N-1}$ distincts.

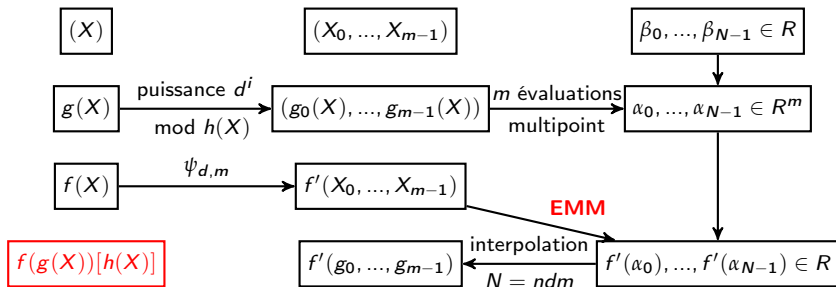
Algorithme de composition modulaire



$$g_i(X) = g(X)^{d^i}.$$

On choisit les $\beta_0, \dots, \beta_{N-1}$ distincts.

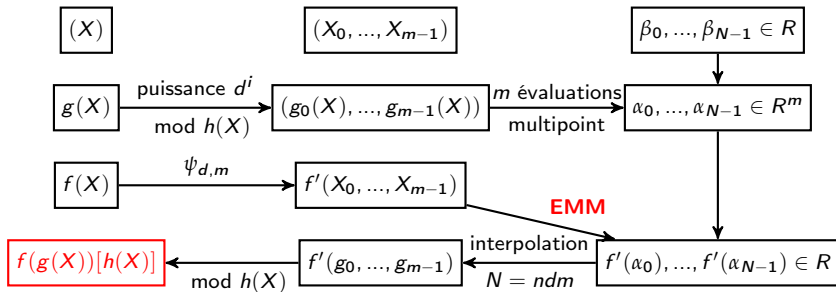
Algorithme de composition modulaire



$$g_i(X) = g(X)^{d^i}.$$

On choisit les $\beta_0, \dots, \beta_{N-1}$ distincts.

Algorithme de composition modulaire



$$g_i(X) = g(X)^{d^i}.$$

On choisit les $\beta_0, \dots, \beta_{N-1}$ distincts.

Évaluation d'un polynôme univarié en tous les points de \mathbb{F}_p

R. Fateman : The (finite field) Fast Fourier Transform (FFT)

Transformée de Fourier Discrète (DFT)

Soit $f = f_0, \dots, f_{n-1}$ une liste de n éléments de R .

Soit ω une racine primitive n -ème de l'unité.

La DFT de f est la liste des valeurs $x_j = \sum_{k=0}^{n-1} f_k (\omega^j)^k$ pour j allant de 0 à $n-1$.

C'est une évaluation d'un polynôme en les racines n -èmes de l'unité.

Dans \mathbb{F}_p^* , tout élément est racine $(p-1)$ -ème de l'unité.

\Rightarrow La DFT est une évaluation d'un polynôme en tout point de \mathbb{F}_p^* .

EMM par la FFT multidimensionnelle

Transformation de Fourier discrète multidimensionnelle

Il existe une transformation de Fourier discrète multidimensionnelle, et elle peut se voir comme l'évaluation d'un polynôme multivarié en les racines $(p-1)$ -èmes de l'unité pour chaque variable.

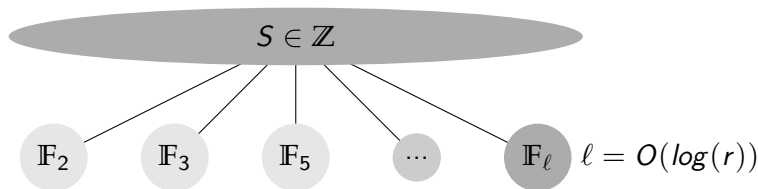
Elle peut se calculer comme suit :

- $f = \sum_{i=0}^{p-1} X_{m-1}^i f_i(X_0, \dots, X_{m-2})$.
- p appels récurrents, une pour chaque $f_i(X_0, \dots, X_{m-2}) \Rightarrow p^{m-1}$ points.
- On applique p^{m-1} FFT, une pour chaque f intermédiaire.
- Le résultat obtenu est une liste de p^m éléments de \mathbb{F}_p .

Décomposition modulo des facteurs premiers

Stratégie de calcul

- Passer de $\mathbb{Z}/r\mathbb{Z}$ à \mathbb{Z} .
- Décomposer le problème modulo de petits facteurs premiers.
- Reconstruire la solution S dans \mathbb{Z} avec le CRT.
- Réduire modulo r pour revenir dans $\mathbb{Z}/r\mathbb{Z}$.

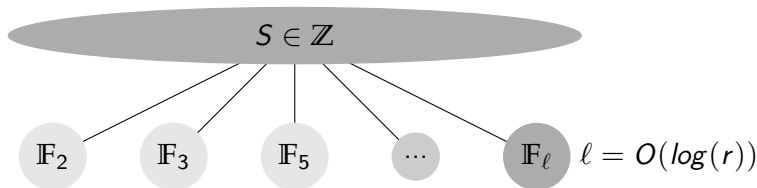


Unicité lors de la reconstruction par CRT

Majoration de S :

$$\text{EMM dans } \mathbb{Z} \Rightarrow S \leq d^m(r-1)^{dm}.$$

Il faut donc $2 \times 3 \times 5 \times \dots \times \ell > d^m(r-1)^{dm}$.



Résolution pour les petits facteurs premiers

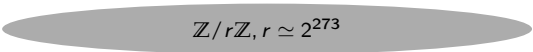
Utilisation de l'algorithme EMM avec FFT :

- Coût mémoire : ℓ^m .
- Pour $r \simeq 2^{273}$ et $m = 4$, on a $\ell = 7069 \Rightarrow \ell^m \simeq 2.5 \times 10^{15}$.
- Table de taille 20 Po !

Solution : appliquer récursivement l'algorithme Multimodulaire.

Exemple : $n = 10000$, $d = 10$, $m = 4$

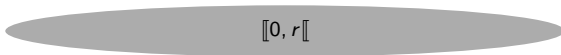
$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



$$\mathbb{Z}/r\mathbb{Z}, r \simeq 2^{273}$$

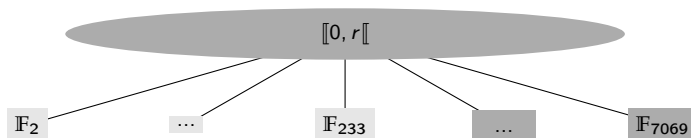
Exemple : $n = 10000$, $d = 10$, $m = 4$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



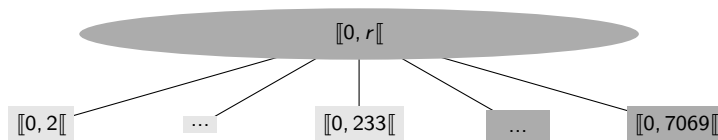
Exemple : $n = 10000$, $d = 10$, $m = 4$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



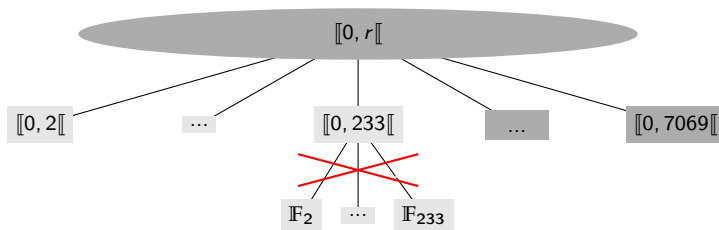
Exemple : $n = 10000$, $d = 10$, $m = 4$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



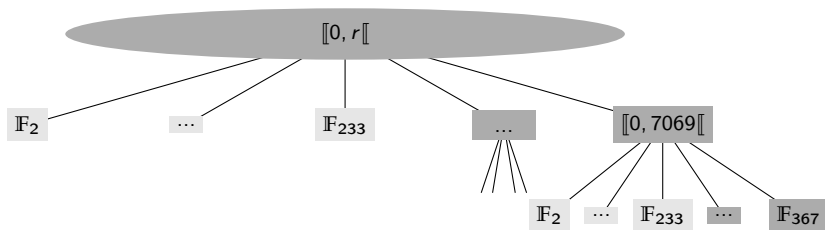
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



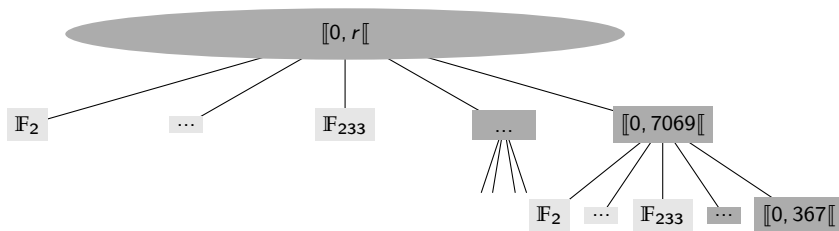
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



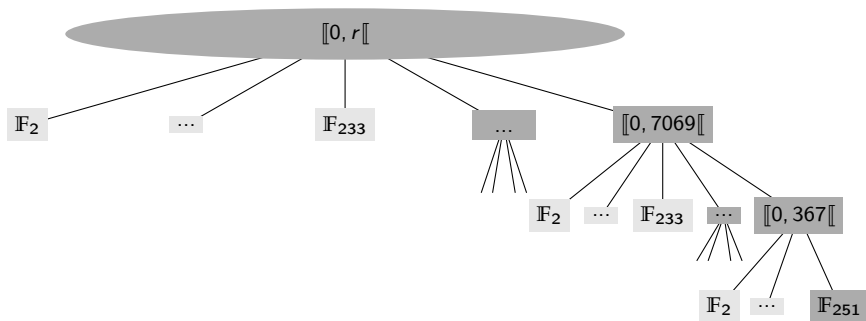
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



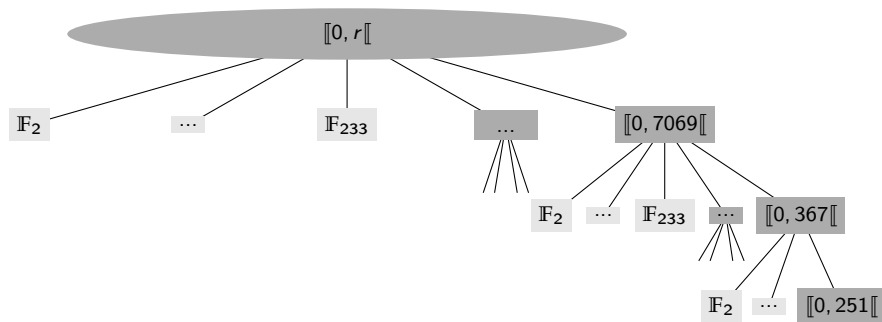
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



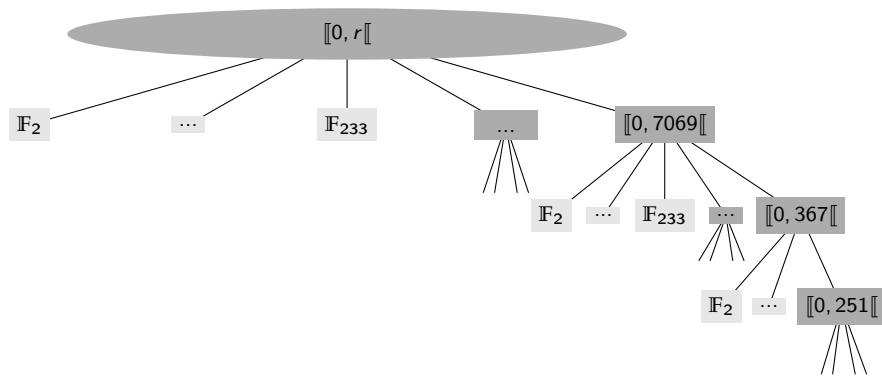
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



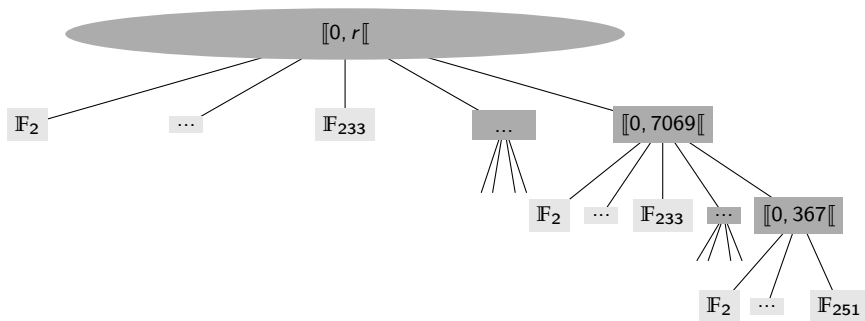
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



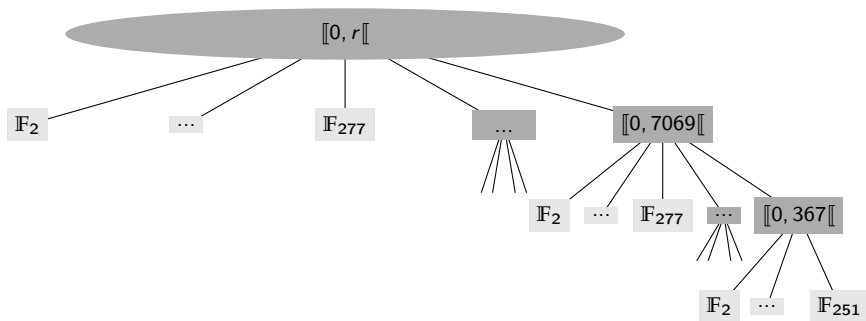
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



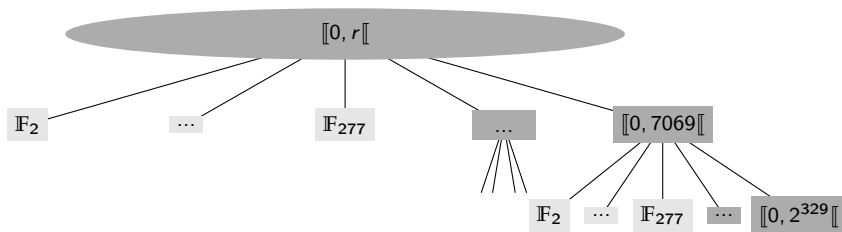
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



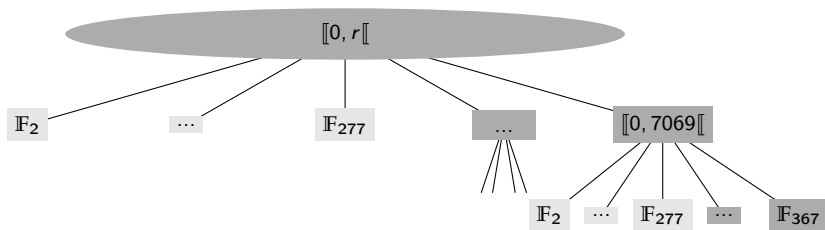
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



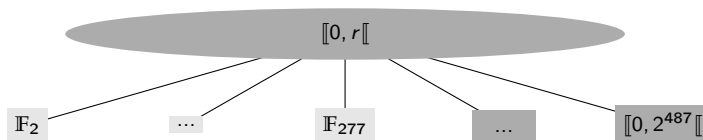
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



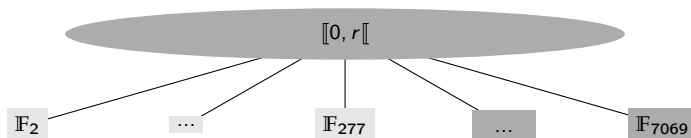
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



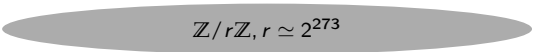
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$

$\llbracket 0, 2^{10079} \llbracket$

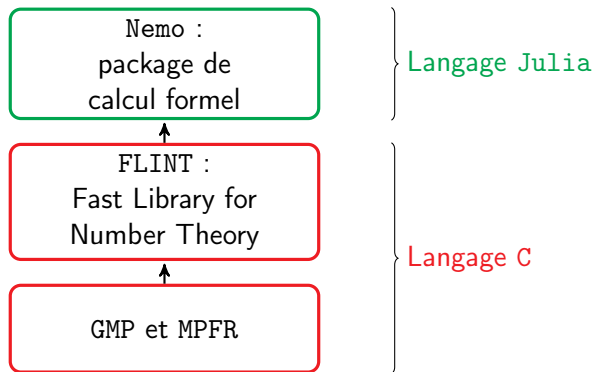
Exemple : $n = 10000$, $d = 10$, $m = 4 \Rightarrow L_{\max} = 233$

$r = 7799922041683461553249199106329813876687996789903550945093032474868511536164700810$



$$\mathbb{Z}/r\mathbb{Z}, r \simeq 2^{273}$$

Langages choisis



Julia \Rightarrow Just-In-Time compilation, proche de la performance du **C**.

Exemple de code FLINT

```

1  //  $g^k \bmod p == h$  ?
2  slong verif_dlp(fmpz_t g, mp_limb_t k, fmpz_t h, fmpz_t p)
3  {
4      fmpz_t gk; // Déclaration d'un entier multiprécision
5      fmpz_init(gk); // Allocation mémoire
6
7      fmpz_powm_ui(gk, g, k, p); //  $gk = g^k \bmod p$ 
8      slong res = fmpz_equal(gk, h); //  $gk == h$ 
9
10     fmpz_clear(gk); // Libération
11     return res;
12 }
```

Exemple de code Julia

```

1  using Nemo # Appel au package Nemo
2
3  function verif_dlp(g,k,h,p)
4      return h == powmod(g,k,p) #  $h == g^k \bmod p$ 
5  end
6
7  ### Exemple d'exécution
8  g = fmpz(2)
9  k = 3
10 h = fmpz(1)
11 p = fmpz(7)
12 res = verif_dlp(g,k,h,p)
13 println(res) # Affiche true

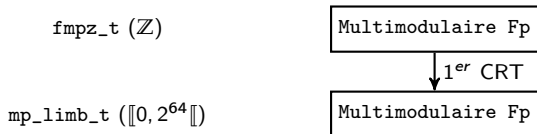
```

Dépendances entre les algorithmes Multimodulaire

fmpz_t (\mathbb{Z})

Multimodulaire Fp

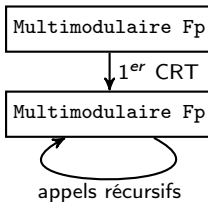
Dépendances entre les algorithmes Multimodulaire



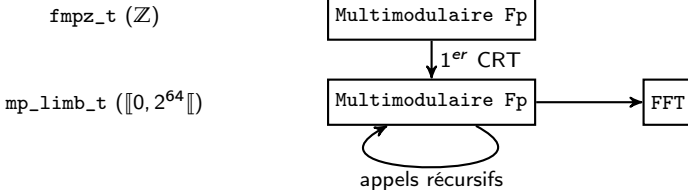
Dépendances entre les algorithmes Multimodulaire

fmpz_t (\mathbb{Z})

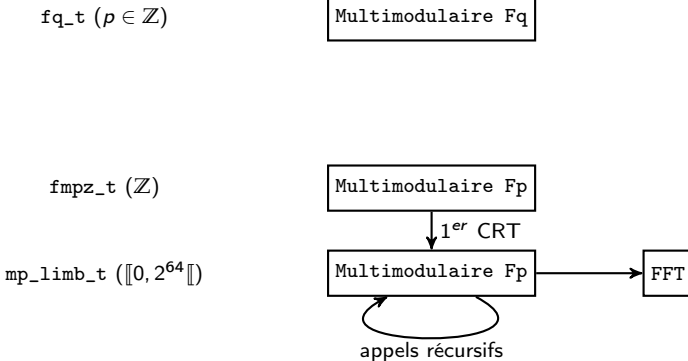
mp_limb_t ($\llbracket 0, 2^{64} \rrbracket$)



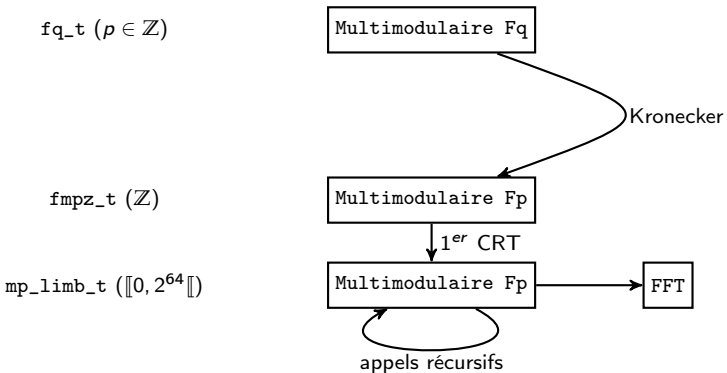
Dépendances entre les algorithmes Multimodulaire



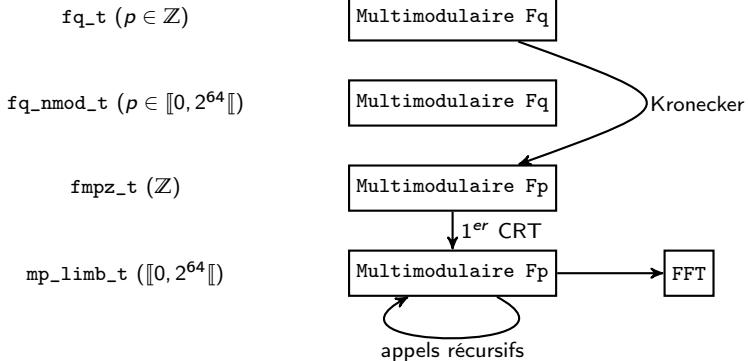
Dépendances entre les algorithmes Multimodulaire



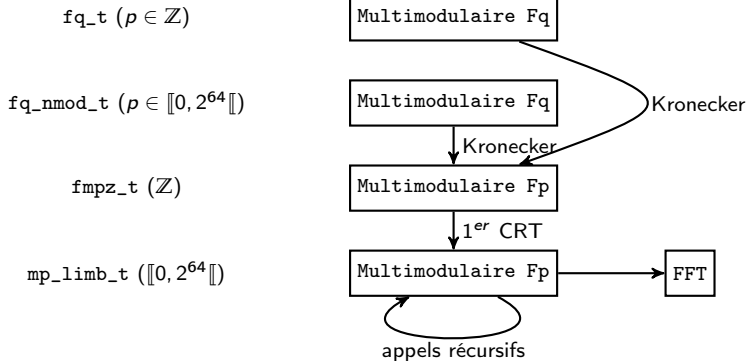
Dépendances entre les algorithmes Multimodulaire



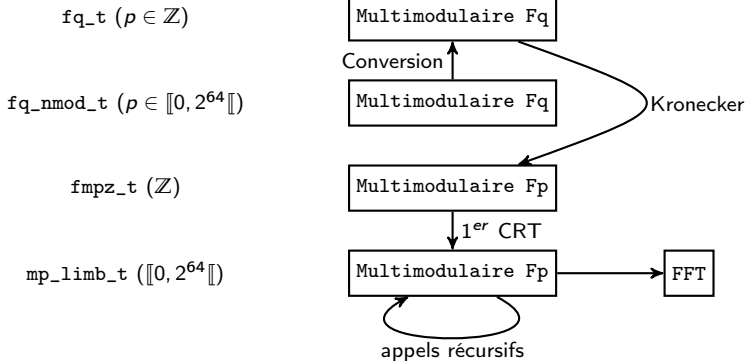
Dépendances entre les algorithmes Multimodulaire



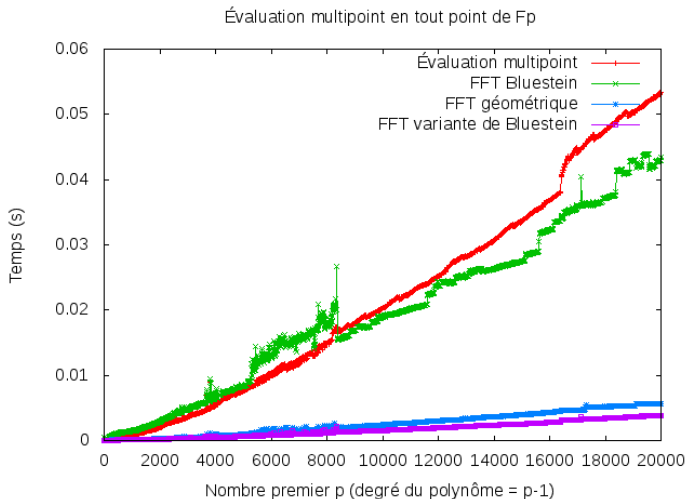
Dépendances entre les algorithmes Multimodulaire



Dépendances entre les algorithmes Multimodulaire



Comparaison des différentes méthodes de FFT



Composition modulaire ($\tilde{O}(n^{1+\frac{2}{m}})$), $p = 2^{200} + 235$

