Bibliothèque C de calcul sur F2 de polynômes de degré inférieur à 64

Ismail Baaj M2MIC

Octobre 2016

1 Introduction

Il est possible de représenter un polynôme sur F2 en C avec un entier. En effet, en écrivant un nombre N en base 2, nous sommes capable de représenter n'importe quel polynôme sur F2 avec pour limite de degré la taille du type de la variable choisie.

Exemple : le polynôme de l'AES : $X^8 + X^4 + X^3 + X + 1$ est représenté par l'entier 283 qui a pour valeur binaire : $b = 100011011$. On lit b de droite (les termes du polynômes de bas degré) vers la gauche : jusqu'au degré du polynôme.

Le polynôme de l'AES peut être stocké dans une variable de type *unsigned short int*. Le plus grand polynôme stockable dans ce type est representé par $65535 = 2^{15} - 1$ qui n'est autre que $X^{14} + X^{13} + X^{12} + X^{11} + X^{10} + X^9 + X^8 + X^7 + X^6 + X^5 + X^4 + X^3 + X^2 + X^1 + X + 1$.

Pour notre bibliothèque nous utilisons le type $f2\_poly\_t = unsigned\ int64$.

1.1 Compilation & Éxécution

Le programme a été testé avec GCC en version 4.8.4 (sur les ordinateurs ubuntu et debian de l'université Paris Diderot) avec les $flags$ suivants : -std=c99 -Wall -pedantic -Wextra -Werror -O3.

Pour les compiler il suffit de lancer dans un terminal :

```
$ make all
```

Il comporte 8 programmes :

- ./arithm_test : des tests des fonctions arithmétiques implémentées
- ./enumerate_irreductible N : afficher dans le terminal (stdout) tous les polynômes irréductibles de degré N sur F2.
- ./enumerate_primitive N : affichre dans le terminal (stdout) tous les polynômes primitifs de degré N sur F2.
- ./count_irreductible : Compter le nombre de polynômes irréductibles par énumération du degré 1 à 63.

- ./count_primitive : Compter le nombre de polynômes primitifs par énumération du degré 1 à 63.
- ./f2_poly_test : Procéder à des tests des fonctions implémentées et donne le nombre de polynômes irréductibles de degré de 1 à 63 et le nombre de polynômes primitifs de 1 à 63.
- ./f2_poly_interface (P) : traitement rapide d'un polynôme en argument (P) ou via l'interface
- ./f2_poly_random (N): analyser des polynômes aléatoires de degré N en argument ou via l'interface

1.2 Fonctions implementées

Dans f2_poly.c :

- `f2_deg_t f2_poly_deg(f2_poly_t pol)` Retourne l'index du MSB (*most significant bit*) de l'entier : le bit "le plus à gauche" du nombre.
- `int f2_poly_print(f2_poly_t f, char c, FILE * fi)` Fonction d'affichage.
- `int f2_poly_div(f2_poly_t *q,f2_poly_t *r, f2_poly_t dividende, f2_poly_t diviseur)` Division du dividende par le diviseur : XOR du dividende avec décalage du diviseur par la différence de degré entre le dividende et le diviseur

$$\begin{array}{r|l} X^2+X+1 & X+1 \\ \cline{2-2} -(X^2+X+0) & X \\ 1 & \end{array}$$

  X^2+X+1 est representé par 111 en base 2 (= 7) et $X+1$ par 11 en binaire (= 3). La différence de polynôme entre les deux nombres est $k=1$. Le premier XOR est donc entre 111 et 110 (diviseur décalé de k). A la seconde itération, dividende = 1 et donc k <0. On sort de la boucle, le *reste* correspond à la valeur du dividende ici égal à 1. A chaque itération, le *quotient* fait l'opération binaire OR avec le monôme X^k.

- `f2_poly_t f2_poly_rem(f2_poly_t dividende, f2_poly_t diviseur )` Reste de la division de *dividende* par *diviseur* analogue à l'algorithme de la division ci-dessus.
- `f2_poly_t f2_poly_gcd(f2_poly_t arg1, f2_poly_t arg2)` Application de l'algorithme d'Euclide. Déterminination à l'initialisation du plus grand polynôme donné dans les arguments (pour l'utilisation ensuite de *f2_poly_rem*).
- `f2_poly_t f2_poly_xtimes(f2_poly_t arg1, f2_poly_t arg2)` Calcul de $X * arg1 \equiv arg2$. Décalage de $arg1$ de 1 bit vers la gauche. Utilisation ensuite de *f2_poly_rem*

Algorithm 1 Division

```
procedure DIVISION(quotient, reste, dividende, diviseur)
    k ← deg(dividende) − deg(diviseur)
    while k >= 0 ET dividende ≠ 0 do
        quotient ← quotient|(1 << k)                    ▷ OR
        dividende ← dividende ⊕ (diviseur << k)         ▷ XOR
        k ← deg(dividende) − deg(diviseur)
    end while
    reste ← dividende
end procedure
```

- `f2_poly_t f2_poly_times(f2_poly_t arg1, f2_poly_t arg2, f2_poly_t arg3)` Renvoi $arg1 * arg2 \equiv arg3$. Deux algorithmes en fonction des degrées de $d1 = arg1$ et $d2 = arg2$. Si $d1 + d2$<64 on fait :

Algorithm 2 f2_poly_times 1

```
procedure F2_POLY_TIMES 1(arg1, arg2, arg3)
    d =← deg(arg1)
    r ← 0
    for i = 0; i < d; i = i + 1 do
        if Si arg1 a un coefficient de degré de rang i then
            r ← r ⊕ (arg2 <<i)                          ▷ XOR
        end if
    end for
    retourne f2_poly_rem(r, arg3)
end procedure
```

sinon, cet algorithme :

- `f2_poly_t f2_poly_x2n(f2_deg_t n, f2_poly_t arg2)` Pour calculer $X^{2^n} \equiv arg2$, on part de $r = X$ et on applique f2_poly_times de 0 jusqu'au degré n de cette manière : $r = f2\_poly\_times(r, r, arg2)$.
- `f2_poly_t f2_poly_parity(f2_poly_t arg1)` Retourne le reste de la division par X+1 (xor des bits)
- `f2_poly_t f2_poly_recip(f2_poly_t arg1)` Retourne le polynôme réciproque considéré comme de degré le second argument
- `int f2_poly_irred(f2_poly_t arg1)` Retourne 1 si le polynôme est irréductible, 0 sinon.
  - On regarde si $arg1$ n'est pas une constante (pas irréductible) ou un polynôme de degré 1 ($XetX + 1$ sont irréductibles)
  - On a donc un polynôme de degré n supérieur à 1. On regarde si il n'a pas un seul bit à 1 (qu'il serait de la forme X^n)

Algorithm 3 f2_poly_times 2

procedure F2_POLY_TIMES $2(arg1, arg2, arg3)$
    $arg1 \leftarrow f2\_poly\_rem(arg1, arg3)$
    $arg2 \leftarrow f2\_poly\_rem(arg2, arg3)$
    $r \leftarrow 0$
    while $arg2 \neq 0$ **do**
        if arg2 fini par ..+1 **then**
            $r \leftarrow r \oplus arg1$ ▷ XOR
            $arg2 \leftarrow arg2 <<1$
        else
            $arg1 \leftarrow f2\_poly\_xtimes(arg1, arg3)$ ▷ $X * arg1 \equiv arg3$
            $arg2 \leftarrow arg2 >> 1$
        end if
    end while
end procedure

- avec *__builtin_popcountll*() on compte son nombre de bits à 1, si celui-ci n'est pas impair alors il n'est pas irréductible
- On regarde si $X^{2^n} \equiv X \pmod{arg1}$
- $\forall p$ premier inférieur à 64 (il y en a 18), si $p < n$ et que p divise n on vérifie que $PGCD(X^{2^{n/p}} - X, arg1) = 1$

• `uint64_t f2_poly_irred_count(f2_deg_t n)` On donne le nombre à l'aide de la fonction de mobius μ :

$$M_n(q) = \frac{1}{n} \sum_{d|n} \mu(d) q^{n/d}.$$

On est sur F2 donc $q = 2$ ici.

• `f2_poly_t f2_poly_xn(f2_poly_t arg1, f2_poly_t arg2)` Retourne $X^{arg1} \equiv arg2$ (décalage et calcul du reste)

• `f2_poly_t f2_poly_random_inf( f2_deg_t n)` Retourne un polynôme tiré au hasard parmi les polynômes de degré ¡ n. Utilisation de $/dev/urandom$. Calcul d'un polynôme aléatoire basé sur un nombre aléatoire de nombre de bits à 1. (pour un degré n il y a beaucoup plus de polynômes de degré X^{n-1} que de polynômes de degré 1....)

• `f2_poly_t f2_poly_random(f2_deg_t n)` Analogue à f2_poly_random_inf.

• `int f2_poly_primitive(f2_poly_t arg1)` Vérifie si le polynôme arg1 est primitif

- On regarde si arg1 est irréductible sinon il n'est pas primitif
- $d \leftarrow degre(arg1)$

- Si d est un exposant d'un nombre premier de Mersenne inférieur à 64 (il y en a 9) alors $arg1$ est primitif.
- $n \leftarrow 2^d - 1$
- $diviseurs \leftarrow tous\ les\ diviseurs\ de\ n\ different\ de\ 1\ et\ n$
- Pour chaque diviseur noté q si ($X^q \equiv 1 \pmod{arg1}$ ou $X^{n/q} \equiv 1 \pmod{arg1}$) alors $arg1$ n'est pas primitif
- sinon renvoyer $VRAI$

- `uint64_t f2_poly_irred_order(f2_poly_t polP)` Analogue à f2_poly_primitive. On est obligé de parcourir de i de 2 à $2^n - 1$ et de faire un seul modulo (X^i).
- `f2_poly_t f2_poly_irred_random( f2_deg_t arg1)` Tant qu'il n'est pas irréductible tirer un nouveau polynôme au hasard...
- `f2_poly_t f2_poly_primitive_random (f2_deg_t arg1)` Analogue à f2_poly_irred_random.
- `uint64_t f2_poly_primitive_count( f2_deg_t n)`

  Si n est un exposant premier de Mersenne inférieur à 64 renvoyer le nombre de polynômes irréductibles de degré n.

  Sinon calculer $\phi(2^n - 1)/n$ avec ϕ l'indicatrice d'Euler.

Dans arithm.c :

- `uint64_t pp_diviseur_premier(uint64_t n)` Retourne le plus petit diviseur premier de n.
- `int64_t f_exp(int64_t a, uint64_t b)` Implémentation de l'exponentiation rapide.
- `int8_t mobius(int d)` Implémentation de la fonction de Mobiüs.
- `uint64_t euler(uint64_t n)` Indicatrice d'Euler.
- `Diviseurs* get_all_divisors( uint64_t n, Diviseurs *d)` Retourne l'ensemble des diviseurs de l'entier n.
- `void divisors_memory_management( Diviseurs *d, uint64_t *suiteListe, int taille)` Fonction de réallocation mémoire pour trouver les diviseurs d'un nombre.

2 Résultats & Conclusion

Il est assez long de calculer un polynôme primitif au dessus du degré 30, il faut en effet trouver tous les diviseurs d'un nombre supérieur à 10^9. L'énumération des polynômes irréductibles est par compte très rapide. Il est facile de faire évoluer notre programme : en C il existe le type *unsigned int128* ainsi, nous pourrions faire des opérations sur des polynômes jusqu'au degré 128.

3 Références

- Livre : A Course in Computational Algebraic Number Theory d'Henri Cohen
- publication : Optimal Irreducible Polynomials for GF(2m) Arithmetic par Michael Scott
- article : Bit Twiddling Hacks par Sean Eron Anderson