Divertirsi con i campi finiti

R. Bernardini

31 ottobre 2007

Indice

Capitolo 1

Introduzione

Perché?

L'importanza dell'algebra lineare nell'ingegneria elettronica è incontestabile. Durante i cinque anni di studio per la laurea in ingegneria lo studente vede principalmente spazi lineari reali o, al più, complessi. Nonostante tali spazi lineari siano quelli che si incontrano più spesso in pratica, esistono diverse applicazioni che richiedono l'uso di spazi lineari definiti su altri tipi di scalari. In particolare, nel campo delle telecomunicazioni sono molto comuni applicazioni che richiedono l'uso di *corpi finiti*, ossia, insiemi finiti su cui sono definite quattro "operazioni" che si comportano come le quattro operazioni tradizionali.

La teoria dei campi finiti è, assieme alla pseudo-inversa e la decomposizione a valori singolari, un "grande assente" dei corsi di studi di ingegneria. Il motivo è forse da ricercarsi nella loro definizione relativamente astratta e forse anche nel fatto che le operazioni su corpi finiti non hanno un legame evidente con le operazioni tradizionali che nascono per operare su "quantità" fisiche.

Nonostante le possibili difficoltà iniziali indotte dall'astrazione necessaria per l'introduzione dei corpi finiti, l'autore considera tale argomento decisamente importante per l'ingegnere elettronico ed in particolare per l'ingegnere che lavora nel campo delle telecomunicazioni.

Struttura di queste note

Lo scopo di queste note è di introdurre i principali risultati sulla teoria dei campi finiti. L'obiettivo finale è di mettere lo studente in grado di lavorare autonomamente con i corpi finiti. La teoria verrà sviluppata in maniera abbastanza rigorosa, saltando però le dimostrazioni più tecniche, irrinunciabili nella formazione di un matematico, ma di interesse secondario per un ingegnere.

Nella seconda parte di queste note descriviamo inoltre alcune applicazioni.

Capitolo 2

La teoria

2.1 Introduzione

Il significato che comunemente si dà alla frase "somma (o moltiplica) a e b modulo N" è "esegui la somma (o il prodotto) di a e b, dividi per N il risultato e prendi il resto." Nonostante tale definizione operativa sia molto intuitiva, risulta particolarmente scomoda quando si tratta di analizzare algoritmi che fanno uso dell'aritmetica modulo N. Esiste un modo alternativo (più astratto) di pensare l'aritmetica modulo N che, nonostante possa presentare qualche difficoltà all'inizio, risulta però molto più comodo da maneggiare algebricamente. Da tale approccio astratto derivano poi altri strumenti matematici che trovano applicazione in molti campi di interesse attuale, quali la crittografia e la trasmissione digitale. Lo scopo di questo capitolo è di presentare (brevemente) tali strumenti algebrici, tipicamente trascurati nei curriculum ingegneristici.

2.2 Le quattro operazioni modulo N

2.2.1 Interi modulo N

Il primo passo verso la definizione di somma e prodotto di interi modulo N è la definizione di *equivalenza modulo N*.

Definzione 1. *Siano $a, N \in \mathbb{Z}$ due interi e siano $q \in \mathbb{Z}$, $r \in \{0, \ldots, N-1\}$ gli unici interi tali che*

$$a = Nq + r. \tag{2.1}$$

Il valore r è il resto della divisione di a per N *e sarà indicato con* $a \bmod N$. *Inoltre, useremo talvolta la notazione* $q = a \operatorname{div} N$.

Commento 2.2.1

Può sembrare eccessiva pignoleria preoccuparsi di definire formalmente il resto della divisione di a per N (in fondo cosa sia il resto di una divisione lo sappiamo dalle elementari...), ma in realtà nell'idea intuitiva di resto c'è un'ambiguità quando $a < 0$; per esempio, se divido -5 per 3 il resto è -2 (poiché $-5 = -2 + (-1) \cdot 3$) o 1 (poiché $-5 = 1 + (-2) \cdot 3$)? Alcuni linguaggi di programmazione (es. `C`) seguono la prima convenzione, altri (es. `Perl`, `Ruby`) seguono la seconda, altri ancora (es. `Matlab`, `Octave`) hanno funzioni diverse per le due convenzioni (`rem` per la prima e `mod` per la seconda).

Secondo la definizione data più sopra, il resto della divisione è sempre un numero non negativo e minore di N (seguiamo quindi la seconda convenzione).

Definzione 2. *Due interi $a, b \in \mathbb{Z}$ si dicono* equivalenti modulo N *se* $a \bmod N = b \bmod N$ *o, equivalentemente, se esiste $k \in \mathbb{Z}$ tale che*

$$a - b = Nk. \tag{2.2}$$

Se a e b sono equivalenti modulo N scriveremo

$$a \equiv b \pmod{N} \tag{2.3}$$

Esempio 2.2.1
Esempio banale, banale: due numeri sono equivalenti modulo 2 se sono entrambi pari o entrambi dispari.

Non è un caso che nella Definizione 2 si sia usata la parola *equivalenti* e il simbolo "≡;" infatti la relazione di equivalenza modulo N è una… relazione di equivalenza, nel senso che è

$$\begin{aligned} a \equiv a \pmod N & & \text{Riflessiva} \\ a \equiv b \pmod N \Rightarrow b \equiv a \pmod N & & \text{Simmetrica} \\ a \equiv b \pmod N, b \equiv c \pmod N \Rightarrow a \equiv c \pmod N & & \text{Transitiva} \end{aligned}$$

come è immediato dimostrare (esercizio). Ne segue che l'insieme degli interi si partiziona in classi di equivalenza, una classe per ogni possibile resto della divisione per N (ci sono quindi N classi). In letteratura esistono diverse notazioni per l'insieme delle classi di equivalenza modulo N; le più comuni sono: $\mathbb{Z}/\mathbb{Z}(N)$, $\mathbb{Z}/N\mathbb{Z}$ e $\mathbb{Z}_N$ (la prima notazione è mutuata dalla teoria unificata, la seconda e la terza sono forse più comuni in letteratura). Noi useremo $\mathbb{Z}_N$.

Esempio 2.2.2
Sempre nello spirito dell'esempio banale presentato più sopra, l'equivalenza modulo 2 divide gli interi in due classi: in una classe ci sono tutti i numeri pari, nell'altra tutti i numeri dispari.

Gli elementi di $\mathbb{Z}_N$ non sono quindi interi, ma *insiemi di interi*. In letteratura solitamente non si distingue notazionalmente tra un intero $n \in \mathbb{Z}$ e la classe di equivalenza modulo N a cui n appartiene. In questo capitolo però, per distinguere chiaramente tra elementi di $\mathbb{Z}$ ed elementi di $\mathbb{Z}_N$, indicheremo la classe a cui n appartiene con $[n]_N$.

Esempio 2.2.3
L'insieme $\mathbb{Z}_4$ ha come elementi le quattro classi di equivalenza

$$[0]_4 = \{\ldots,-8,-4,0,4,8,12,\ldots\} \tag{2.4a}$$
$$[1]_4 = \{\ldots,-7,-3,1,5,9,13,\ldots\} \tag{2.4b}$$
$$[2]_4 = \{\ldots,-6,-2,2,6,10,14,\ldots\} \tag{2.4c}$$
$$[3]_4 = \{\ldots,-5,-2,3,7,11,15,\ldots\} \tag{2.4d}$$

Commento 2.2.2
Si osservi come con la notazione appena introdotta esistano più modi per indicare la stessa classe; per esempio, $[2]_4$ può anche essere scritta $[6]_4$ o $[-2]_4$, poiché 2, 6 e -2 sono tra loro equivalenti modulo 4. Più in generale, si ha

$$[a]_N = [a+Nk]_N \tag{2.5}$$

per ogni $k \in \mathbb{Z}$.

Un concetto che risulta spesso utile è quello di *cella elementare* o *insieme di rappresentanti* di $\mathbb{Z}_N$

Definzione 3. *Un insieme $I \subset \mathbb{Z}$ è una* cella elementare *di $\mathbb{Z}_N$ se ogni classe di $\mathbb{Z}_N$ ha uno ed un solo elemento in comune con I, ossia*

$$\forall [n]_N \in \mathbb{Z}_N \quad |I \cap [n]_N| = 1 \tag{2.6}$$

dove $|\cdot|$ indica la cardinalità di un insieme.

Esempio 2.2.4
Due tipiche celle per $\mathbb{Z}_{256}$ sono l'insieme dei possibili resti della divisione per 256, ossia $\{0,1,\ldots,255\}$, e la cella "simmetrica" $\{-128,-127,\ldots,127\}$. Altre possibili celle sono, per esempio, $\{-10,-9,\ldots,244,245\}$ o $\{-128,\ldots,-1\} \cup \{256+0,\ldots,256+127\}$

Osservazione 1. *Una cella di $\mathbb{Z}_N$ ha sempre N elementi. L'intervallo $\{a, a+1,\ldots,a+N-1\}$ è una cella di $\mathbb{Z}_N$, qualsiasi sia $a \in \mathbb{Z}$.*

2.2.2 Somma in $\mathbb{Z}_N$

Il nostro scopo è introdurre una somma ed un prodotto tra gli elementi di $\mathbb{Z}_N$ che corrispondano all'idea intuitiva di somma e prodotto modulo N richiamata all'inizio di questo capitolo. Qui nasce un piccolo problema: gli elementi di $\mathbb{Z}_N$ sono *insiemi* di interi, come si fa a definire una "somma tra insiemi"? Il trucco che spesso si usa in questi casi è di "procedere per rappresentanti," ossia, per sommare due classi di equivalenza scelgo un elemento a della prima classe, un elemento b della seconda, e *definisco* la somma delle due classi come *l'unica* classe che contiene $a+b$. In formule,

$$[a]_N + [b]_N := [a+b]_N \tag{2.7}$$

dove il simbolo ":=" in (2.7) significa "per definizione;" in altre parole, la metà di destra dell'equazione (2.7) è il significato che diamo alla metà di sinistra.

Commento 2.2.3

Si osservi il differente significato del simbolo "+" nei due lati dell'equazione (2.7): a destra indica la somma *tra interi*, mentre a sinistra indica la somma *tra classi di equivalenza*. Avremmo potuto introdurre un nuovo simbolo per la somma su $\mathbb{Z}_N$ (per esempio, $\oplus$ o $+_N$), ma abbiamo deciso di non appesantire ulteriormente la notazione.

Il problema con la definizione (2.7) è che potrebbe essere mal posta poiché potrebbe capitare che scegliendo due diversi rappresentanti $a' \in [a]_N$ e $b' \in [b]_N$ la classe della somma $[a'+b']_N$ potrebbe non coincidere con $[a+b]_N$. Per verificare che la definizione (2.7) è ben posta, si osservi che, per definizione di equivalenza modulo N, se $a' \in [a]_N$ e $b' \in [b]_N$, allora $a' = a+Nk$ e $b' = b+N\ell$ per qualche $k,\ell \in \mathbb{Z}$. La classe che contiene $a'+b'$ è quindi

$$\left[a'+b'\right]_N = [a+b+N(k+\ell)]_N = [a+b]_N \tag{2.8}$$

Il risultato di (2.7) non dipende quindi dai rappresentanti a e b scelti.

Esempio 2.2.5

La somma in $\mathbb{Z}_2$ può essere descritta dalla seguente tabella

	$[0]_2$	$[1]_2$
$[0]_2$	$[0]_2$	$[1]_2$
$[1]_2$	$[1]_2$	$[0]_2$

Si osservi come la somma in $\mathbb{Z}_2$ coincida con l'or esclusivo tra bit. Si osservi inoltre come le regole della somma modulo due possano essere riformulate in termini di numeri pari e dispari. Per esempio, la regola $[0]_2+[1]_2=[1]_2$ può essere letta "la somma di un numero pari (un elemento di $[0]_2$) con un numero dispari (un elemento di $[1]_2$) è un numero dispari."

Commento 2.2.4

Sia $N=4$ e si osservi che $[0]_4+[1]_4=[1]_4$, $[1]_4+[1]_4=[2]_4$, $[2]_4+[1]_4=[3]_4$ e $[3]_4+[1]_4=[4]_4=[0]_4$, ossia sommando quattro volte $[1]_4$ a $[0]_4$ riotteniamo $[0]_4$. Non è difficile convincersi che questo accade per qualsiasi valore di N, ossia

$$\underbrace{[1]_N+[1]_N+\cdots+[1]_N}_{N \text{ volte}} = [0]_N \tag{2.9}$$

Tale "periodicità" suggerisce la rappresentazione geometrica mostrata in Fig. 2.1a in cui ad ogni classe modulo $N=8$ è associato un punto su una circonferenza. La Fig. 2.1b mostra come la somma in $\mathbb{Z}_N$ possa essere eseguita con l'aiuto di un "regolo circolare."

A causa della sua intrinseca "periodicità" la somma modulo N viene talvolta detta "somma del contachilometri." In effetti, un contachilometri con k cifre mostra il numero di chilometri percorsi modulo 10^k.

Commento 2.2.5 (A volte ritornano... $\mathbb{Z}_N$ e teoria unificata)

Come già detto, una delle possibili notazioni per l'insieme delle classi modulo N è $\mathbb{Z}/\mathbb{Z}(N)$. La somiglianza di tale notazione col dominio per segnali periodici $\mathbb{Z}(1)/\mathbb{Z}(N)$ della teoria unificata non è casuale, poiché, anche se talvolta $\mathbb{Z}(1)/\mathbb{Z}(N)$ viene introdotto in maniera "semplificata" dicendo che si tratta di una notazione comoda per ricordarsi che il segnale è periodico di periodo N, volendo essere precisi, $\mathbb{Z}(1)/\mathbb{Z}(N)$ andrebbe definito come l'insieme di classi di equivalenza modulo N, ossia $\mathbb{Z}_N$ e $\mathbb{Z}(1)/\mathbb{Z}(N)$ *sono lo stesso oggetto*.

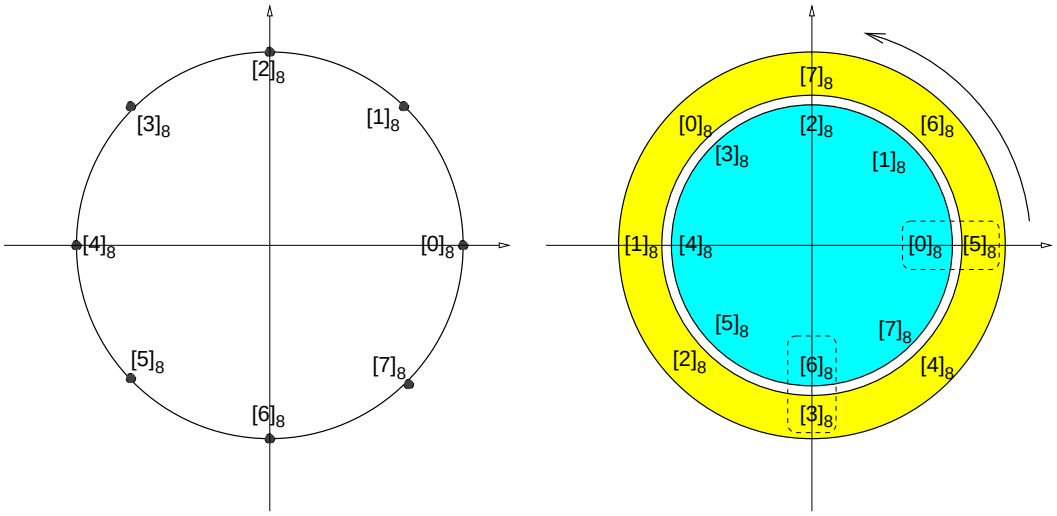

Figura 2.1: (a) Disposizione delle classi di $\mathbb{Z}_8$ lungo una circonferenza (b) Calcolo di $[5]_8 + [6]_8 = [3]_8$ tramite un regolo circolare: si ruota il cerchio esterno fino a portare il punto $[5]_8$ in corrispondenza dello $[0]_8$ del cerchio interno, dopo la rotazione, in corrispondenza del $[6]_8$ interno troviamo il punto $[3]_8$ esterno che corrisponde alla somma di $[5]_8$ e $[6]_8$.

A questo punto potrebbero sorgere dei dubbi: un segnale periodico x a tempo discreto di periodo N viene tipicamente definito come una funzione $x : \mathbb{Z} \to \mathbb{C}$ definita su $\mathbb{Z}$ tale che $x(n) = x(n+N)$ per ogni $n \in \mathbb{Z}$. Cosa vuol dire allora che x è definito su $\mathbb{Z}(1)/\mathbb{Z}(N) = \mathbb{Z}_N$ un insieme finito i cui elementi sono sottoinsiemi di $\mathbb{Z}$? In realtà il segnale x ed il suo corrispondente definito su $\mathbb{Z}_N$ sono *due oggetti diversi*, anche se profondamente legati tra loro. Più precisamente, se se $x : \mathbb{Z} \to \mathbb{C}$ è periodica di periodo N il corrispondente segnale definito su $\mathbb{Z}_N$ è il segnale $\hat{x} : \mathbb{Z}_N \to \mathbb{C}$ definito da

$$\hat{x}([n]_N) := x(n) \tag{2.10}$$

ossia, il valore che $\hat{x}$ associa all'insieme $[n]_N$ è uguale al valore che x associa al rappresentante $n \in [n]_N$. Si osservi che la (2.10) ben posta (ossia, il valore di $\hat{x}([n]_N)$ non dipende dal rappresentante scelto) poiché x è periodico di periodo N.

Scoprire la vera natura di $\mathbb{Z}(1)/\mathbb{Z}(N)$ permette anche di chiarire il vero significato dell'operazione più "bislacca" della teoria unificata: la deperiodicizzazione. Con l'approccio semplificato (in cui $\mathbb{Z}/\mathbb{Z}(N)$ è solo una notazione per ricordarsi che il segnale ha periodo N), la periodicizzazione sembra solo un superficiale cambio di notazione che però ha conseguenze profonde nella definizione della trasformata di Fourier (poiché nel dominio duale si passa da una trasformata definita su un dominio discreto ad una trasformata definita su un dominio continuo con impulsi ideali centrati sulle frequenze multiple di $1/N$). Con l'approccio più rigoroso, invece, la versione deperiodicizzata di $\hat{x} : \mathbb{Z}_N \to \mathbb{C}$ viene definita come il segnale $x : \mathbb{Z} \to \mathbb{C}$ definito da

$$x(n) := \hat{x}([n]_N), \qquad n \in \mathbb{Z}, \tag{2.11}$$

ossia, la deperiodicizzazione è l'operazione inversa di (2.10). La (2.11) può essere così letta: Per calcolare il valore che x assume in $n \in \mathbb{Z}$, mappo n nella classe di equivalenza $[n]_N$ alla quale appartiene e prendo il valore che $\hat{x}$ assume su $[n]_N$. Si osservi che poiché $[n+N]_N = [n]_N$ il segnale così ottenuto è certamente periodico di periodo N.

La somma tra elementi di $\mathbb{Z}_N$ eredita diverse proprietà dalla somma tra interi; più precisamente, si ha

$$[m]_N + [n]_N = [m]_N + [n]_N \qquad \text{La somma è commutativa} \tag{2.12a}$$
$$([m]_N + [n]_N) + [\ell]_N = [m]_N + ([n]_N + [\ell]_N) \qquad \text{La somma è associativa} \tag{2.12b}$$
$$[0]_N + [n]_N = [n]_N \qquad [0]_N \text{ è l'elemento neutro} \tag{2.12c}$$
$$[-n]_N + [n]_N = [0]_N \qquad [-n]_N \text{ è l'inverso di } [n]_N \tag{2.12d}$$

Chi apprezza il linguaggio algebrico amerà sapere che le proprietà (2.12) mostrano che $\mathbb{Z}_N$ è un *gruppo commutativo* [1]. Inoltre, poiché ogni elemento di $\mathbb{Z}_N$ può essere ottenuto sommando $[1]_N$ a sé stesso un numero sufficiente di volte, $\mathbb{Z}_N$ è un *gruppo ciclico* [1].

2.2.3 Prodotto in $\mathbb{Z}_N$

Procedendo sulla falsa riga di quanto fatto per la somma, è possibile definire anche un prodotto tra elementi di $\mathbb{Z}_N$. Più precisamente, il prodotto tra due classi $[a]_N, [b]_N \in \mathbb{Z}_N$ si definisce

$$[a]_N \cdot [b]_N = [ab]_N . \tag{2.13}$$

Ancora una volta, bisognerebbe verificare che la definizione (2.13) è ben posta, ossia che il risultato del prodotto non dipende dai due rappresentanti $a \in [a]_N$, $b \in [b]_N$ scelti. Si lascia la dimostrazione come esercizio al lettore.

Esercizio 2.2.1
Se $a, b, P \in \mathbb{R}$ diremo che a e b sono equivalenti modulo P se per qualche $k \in \mathbb{Z}$

$$a - b = Pk \tag{2.14}$$

Si dimostri che

- La relazione di equivalenza modulo P è effettivamente una relazione di equivalenza. Sia $\mathbb{R}/\mathbb{Z}(P)$ l'insieme delle classi di equivalenza.
- Si provi ad estendere la definizione di somma e prodotto all'insieme $\mathbb{R}/\mathbb{Z}(P)$. Si scoprirà che un'operazione può venire definita, ma l'altra no. Cosa va storto?

In maniera analoga a quanto avviene per la somma, il prodotto in $\mathbb{Z}_N$ eredita alcune proprietà dal prodotto tra interi; più precisamente

$$([m]_N [n]_N)[\ell]_N = [m]_N ([n]_N [\ell]_N) \qquad \text{Il prodotto è associativo} \tag{2.15a}$$
$$[m]_N [n]_N = [n]_N [m]_N \qquad \text{Il prodotto è commutativo} \tag{2.15b}$$
$$[1]_N [n]_N = [n]_N \qquad [1]_N \text{ è l'elemento neutro} \tag{2.15c}$$
$$([a]_N + [b]_N)[n]_N = [a]_N [n]_N + [b]_N [n]_N \qquad \text{Il prodotto è distributivo} \tag{2.15d}$$

Le (2.12) con (2.15) mostrano che $\mathbb{Z}_N$ è un *anello commutativo* [1]. Dalle (2.12) e (2.15) discendono altre ben note proprietà

$$[0]_N [n]_N = [0]_N \qquad [0]_N \text{ annulla il prodotto} \tag{2.16a}$$
$$(-[m]_N)[n]_N = -([m]_N [n]_N) \qquad \text{regola del segno} \tag{2.16b}$$

Una proprietà del prodotto tra interi che non viene ereditata dal prodotto su $\mathbb{Z}_N$ è la legge di annullamento del prodotto ($a \neq 0, b \neq 0 \Rightarrow ab \neq 0$), ossia si possono trovare N, $[a]_N \neq 0$ e $[b]_N \neq 0$ tali che $[ab]_N = [0]_N$; per esempio,

$$[6]_{16} [8]_{16} = [48]_{16} = [0]_{16} \tag{2.17}$$

Gli interi modulo N, in generale, non formano quindi un *dominio* (ossia, un anello che gode anche della proprietà di annullamento del prodotto) [1].

Il fatto che le proprietà formali della somma e il prodotto su $\mathbb{Z}_N$ siano uguali alle proprietà formali della somma e prodotto su $\mathbb{Z}$, implica che qualsiasi risultato sugli interi ottenuto sfruttando unicamente le proprietà (2.12) e (2.15) vale anche per gli interi modulo N.

Aritmetica a n bit e $\mathbb{Z}_{2^n}$

Nonostante la definizione di somma e prodotto modulo N possa sembrare molto astratta e di nessun utilizzo pratico, è in realtà molto usata, ma tipicamente non ci si fa caso... In effetti, l'aritmetica intera a n bit implementata nei microprocessori (in cui un eventuale riporto all'$n+1$-simo bit viene ignorato) è aritmetica in $\mathbb{Z}_{2^n}$. In un certo senso, potremmo dire che un valore a n bit all'interno del microprocessore non rappresenta un intero, ma una classe di $\mathbb{Z}_{2^n}$. Tale interpretazione rende ragione di alcuni risultati a prima vista misteriosi riguardanti l'aritmetica complemento a due.

I misteri del complemento a due Tipicamente, la prima volta che lo studente incontra l'aritmetica complemento a due gli viene detto che il bit più significativo rappresenta il segno. Tale convenzione suona ragionevole per cui uno si aspetta (lavorando con 8 bit) una rappresentazione in modulo e segno in cui 0000.0001 (il "." inserito a metà ha l'unico scopo di migliorare la leggibilità del numero) rappresenta il valore "1" e 1000.0001 rappresenta "-1." Invece gli si dice che -1 è rappresentato dalla stringa 1111.1111, ossia 255 in decimale. Perché? Il motivo è molto semplice: -1 e 255 appartengono alla stessa classe modulo 256. Si spiega anche l'algoritmo apparentemente bizzarro usato per trovare l'opposto di un numero (si negano tutti i bit e si somma 1); in realtà tale algoritmo semplicemente calcola $256-n$ a partire da n (negare tutti i bit corrisponde a calcolare $255-n$, sommando 1 si ottiene $256-n$).

Numeri con segno e senza segno Un'altra domanda legata all'aritmetica a complemento a due è come fa il processore a sapere che il programmatore desidera eseguire calcoli con interi con segno o con interi senza segno? Semplice: non lo sa… Il processore semplicemente esegue calcoli in $\mathbb{Z}_{2^n}$ e siamo noi che alla fine interpretiamo la classe risultante associandola al rappresentante in $\{-2^{n-1}, -2^{n-1}+1, \ldots, 2^{n-1}-1\}$ per l'aritmetica con segno o in $\{0, 1, \ldots, 2^n-1\}$ per l'aritmetica senza segno. Addirittura, se sapessi che nel programma che sto scrivendo il risultato si troverà nel range $I = \{-10, -9, \ldots, 245\}$ potrei lavorare in aritmetica a 8 bit ed associando al risultato finale l'unico suo rappresentante in I (un approccio più "intuitivo" avrebbe richiesto l'uso di aritmetica a 16 bit per poter rappresentare tutti gli elementi di I).

Overflow intermedi Un risultato importante, anche se poco intuitivo, di tutta la teoria presentata finora riguarda l'elaborazione dei segnali con aritmetica in virgola fissa. Si supponga che il segnale $x : \mathbb{Z} \to \mathbb{R}$ (ottenuto magari tramite un convertitore analogico/digitale) sia tale che $16 \leq x(n) \leq 32$ per ogni $n \in \mathbb{Z}$ e si supponga di voler filtrare x usando il filtro

$$y(n) = 3x(n) + 4x(n-1) - x(n-2) - 6x(n-3) \tag{2.18}$$

Dalla (2.18) è possibile ricavare il range dei valori assunti da y; si ha infatti

$$y(n) \leq 3 \cdot 32 + 4 \cdot 32 - 16 - 6 \cdot 16 = 7 \cdot 16 = 112 \tag{2.19a}$$

$$y(n) \geq 3 \cdot 16 + 4 \cdot 16 - 32 - 6 \cdot 32 = -7 \cdot 16 = -112 \tag{2.19b}$$

Dalle (2.19b) si vede che i valori di uscita possono essere rappresentati con 8 bit, ma se $y(n)$ viene calcolato col seguente algoritmo

```
acc  = 3*x(n);    // Passo 1
acc += 4*x(n-1);  // Passo 2
acc -=   x(n-2);  // Passo 3
acc -= 6*x(n-3);  // Passo 4
```

al secondo passo può verificarsi un overflow se, per esempio, $x(n) = x(n-1) = 32$ (infatti, $7 \cdot 32 = 214 > 127$). Ne segue forse che `acc` deve essere a 16 bit per proteggersi dagli overflow intermedi? La risposta è **no** poiché se il risultato finale è nell'intervallo $-128 \cdots 127$ *gli overflow intermedi non danno alcun fastidio.* Il motivo di tale risultato apparentemente "magico" è che, poiché il microprocessore esegue i calcoli in $\mathbb{Z}_{256}$, nel registro accumulatore noi troveremo non $y(n)$, ma *la classe* $[y(n)]_{256}$ e questo *indipendentemente dalla presenza di overflow intermedi*. Tale classe verrà interpretata, al momento dell"'utilizzo" di $y(n)$, come numero con segno ad 8 bit associando con $[y(n)]_{256}$ l'unico suo rappresentante nell'intervallo $\{-128, \ldots, 127\}$.

2.2.4 Divisione in $\mathbb{Z}_N$

L'appetito vien mangiando e dopo aver introdotto la somma e il prodotto tra interi modulo N, ci piacerebbe introdurre anche la divisione. È chiaro che il trucco di procedere per rappresentanti non funziona più poiché la divisione di due numeri interi non è sempre un numero intero.

Poiché la divisione è l'operazione inversa della moltiplicazione, per capire come si possa definire la divisione in $\mathbb{Z}_N$ prendiamo spunto dall'operazione inversa della somma: la differenza. Come visto al paragrafo 2.2.2, è possibile definire la differenza di due elementi $[a]_N$ e $[b]_N$ di $\mathbb{Z}_N$ come la somma di $[a]_N$ con l'opposto di $[b]_N$, ossia

$$[a]_N - [b]_N = [a]_N + (-[b]_N) = [a]_N + [-b]_N \tag{2.20}$$

In altre parole, la differenza non è un'operazione indipendente, ma semplicemente la combinazione delle due operazioni più "primitive" somma e opposto. Anzi, nemmeno l'opposto è un concetto indipendente dalla somma poiché l'opposto di n è per definizione quell'elemento che sommato ad n dà l'elemento neutro della somma.

Similmente, la divisione tra a e b (appartenti, per esempio, all'insieme dei numeri reali) non è un'operazione indipendente dal prodotto, ma il prodotto di a per l'inverso moltiplicativo di b, b^{-1}. In realtà neanche l'operazione di "inversa" è un'operazione indipendente dal prodotto poiché l'inverso di un elemento b è definito come quell'elemento b^{-1} tale che $b \cdot b^{-1} = 1$. Ne segue che l'inverso moltiplicativo di una classe $[a]_N$ è la classe $[x]_N$ tale che

$$[x]_N [a]_N = [1]_N \tag{2.21}$$

Dalla (2.21) possiamo dedurre $ax \equiv 1 \pmod{N}$ o, equivalentemente,

$$xa - Nk = 1 \tag{2.22}$$

per qualche $k \in \mathbb{Z}$. L'equazione (2.22) ha due termini noti (a ed N) e due incognite (x e k). Poiché le variabili in (2.22) assumono solo valori interi, l'equazione (2.22) viene detta *diofantea*.

La soluzione di equazioni diofantee è spesso difficile, ma la particolare equazione (2.22) ha una soluzione molto semplice data dalla seguente proprietà.

Teorema 1 (Teorema del resto cinese). *Siano $M, N, c \in \mathbb{Z}$. L'equazione in x e y*

$$Mx + Ny = c \tag{2.23}$$

*ha soluzioni **intere** (ossia, esistono $x, y \in \mathbb{Z}$ che rendono vera l'uguaglianza (2.23))* se e solo se *c è divisibile per il massimo comun divisore di M e N.*

Non daremo qui la dimostrazione del Teorema 1. Ci limiteremo solamente ad osservare che una direzione del teorema (se (2.23) ha soluzioni intere, allora c è divisibile per $\mathrm{MCD}(M,N)$) è ovvia, poiché se M e N sono entrambi multipli di $\mathrm{MCD}(M,N)$, anche qualsiasi loro combinazione lineare intera sarà multipla di $\mathrm{MCD}(M,N)$ (si pensi al caso in cui sia M che N sono pari). La dimostrazione dell'altra implicazione (se c è divisibile per $\mathrm{MCD}(M,N)$, (2.23) ha soluzioni intere) non è così immediata, ma può essere dimostrata in maniera costruttiva facendo ricorso all'algoritmo di Euclide per il calcolo del massimo comun divisore.[1]

Applicando il Teorema 1 all'equazione (2.23) si ha immediatamente il seguente corollario relativo all'esistenza dell'inverso moltiplicativo di una classe di $\mathbb{Z}_N$.

Corollario 1. *La classe $[a]_N$ ha inverso moltiplicativo* se e solo se

$$\mathrm{MCD}(a,N) = 1. \tag{2.24}$$

La condizione (2.24) viene spesso descritta dicendo che a e N sono *primi* **tra loro** o *coprimi*. Si osservi che la condizione di coprimalità *non implica* che a e N siano primi. Per esempio, $a = 12 = 2^2 \cdot 3$ e $N = 35 = 7 \cdot 5$ sono coprimi nonostante nessuno dei due sia un numero primo.

Esempio 2.2.6
Si noti che $3 \cdot 12 = 35 + 1 \equiv 1 \pmod{N}$ da cui segue che

$$[3]_{35} = [12]_{35}^{-1} \tag{2.25}$$

Se vogliamo che la divisione in $\mathbb{Z}_N$ sia definita per ogni elemento non nullo di $\mathbb{Z}_N$ (ossia, se vogliamo che $\mathbb{Z}_N$ sia un *campo*), è necessario che ogni elemento di $\mathbb{Z}_N$ abbia inverso moltiplicativo. Poiché ogni classe di $\mathbb{Z}_N$ ha un rappresentante in $\{0, \ldots, N-1\}$, possiamo dedurre che $\mathbb{Z}_N$ è un campo se e solo se N è coprimo con ogni intero positivo minore di N e questo è vero *se e solo se N è primo*.

×	0	1	2	3	4
0	0	0	0	0	0
1	0	**1**	2	3	4
2	0	2	4	**1**	3
3	0	3	**1**	4	2
4	0	4	3	2	**1**

(a)

×	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	**0**	2	4
3	0	3	**0**	3	**0**	3
4	0	4	2	**0**	4	2
5	0	5	4	3	2	1

(b)

Figura 2.2: (a) Tabellina dei prodotti in $\mathbb{Z}_5$, (b) tabellina dei prodotti in $\mathbb{Z}_6$.

Teorema 2. *L'anello $\mathbb{Z}_N$ è un campo* se e solo se *N è un numero primo.*

Esempio 2.2.7

In Fig. 2.2 è possibile vedere la tabella dei prodotti in $\mathbb{Z}_5$ (Fig. 2.2.a) e $\mathbb{Z}_6$ (Fig. 2.2.b). Si osservi come in $\mathbb{Z}_5$ ogni elemento non nullo abbia inverso moltiplicativo dato che in ogni riga/colonna corrispondente ad un elemento non nullo è presente il valore "1" (mostrato in grasseto Fig. 2.2.a). Si osservi inoltre che dalla Fig. 2.2.b possiamo dedurre che $\mathbb{Z}_6$ non è un campo poiché nelle righe/colonne corrispondenti agli elementi 2, 3 e 4 (non mutuamente primi con 6) sono presenti degli zeri (mostrati in grassetto) e nessun "1." Si osservi inoltre che gli unici elementi invertibili in $\mathbb{Z}_6$ sono 1 e 5 che sono mutualmente primi con 6.

2.3 Altri campi finiti

Vedremo nel Capitolo 3 che la possibilità di avere un campo finito (ossia, un insieme finito sul quale sono definite le "quattro operazioni") ci permette di risolvere alcuni problemi pratici difficilmente risolvibili altrimenti. Nel paragrafo 2.2 abbiamo visto che l'insieme delle classi modulo N è un campo (con N elementi) se e solo se N è un numero primo. Ci si potrebbe chiedere se esistano campi con N elementi, qualsiasi sia N. Per esempio, sarebbe molto utile poter costruire campi con 2^{8n} (2^8, 2^{16}, ...) elementi, poiché un elemento di un tale campo potrebbe essere rappresentato da un numero intero di byte. È chiaro che $\mathbb{Z}_{256}$ non è un campo, poiché 256 è decisamente non primo, ma questo non esclude che possa essere possibile costruire un campo con 256 elementi tramite qualche altra procedura..

Definzione 4. *Il campo finito con N elementi verrà indicato con*[2] *$GF(N)$*

Un risultato fondamentale nella teoria dei campi finiti è il seguente.

Teorema 3. *Esiste (ed è unico a meno di isomorfismo) un campo finito con N elementi* se e solo se *$N = p^k$, con p un numero primo.*

Commento 2.3.1

Unico a meno di isomorfismo vuol dire che dati due campi finiti $\mathrm{GF}(p^k)$ e $\mathrm{GF}'(p^k)$ con p^k elementi esiste una funzione invertibile $\phi : \mathrm{GF}(p^k) \to \mathrm{GF}'(p^k)$ che conserva le operazioni, ossia, tale che

$$\phi(a+b) = \phi(a)+\phi(b) \tag{2.26a}$$
$$\phi(ab) = \phi(a)\phi(b) \tag{2.26b}$$

Si osservi che le operazioni sul lato sinistro delle (2.26) sono operazioni su $\mathrm{GF}(p^k)$ mentre le operazioni sul lato destro sono operazioni su $\mathrm{GF}'(p^k)$.

Informalmente, possiamo dire che due campi isomorfi hanno essenzialmente la stessa struttura e l'unica differenza è che sono stati nomi diversi agli stessi oggetti.

[1] Per completezza di esposizione, l'algoritmo di Euclide è riportato in Appendice B.

[2] GF=*Galois Field*, dal nome del matematico francese Evariste Galois.

Secondo il Teorema 3 possiamo costruire campi con $256 = 2^8$ elementi, ma non campi con $42 = 2 \cdot 3 \cdot 7$ elementi. Il trucco per costruire campi con p^k elementi è di partire da un campo con p elementi (che sappiamo costruire, grazie alla teoria presentata nel paragrafo 2.2) e di "estenderlo" a p^k elementi usando una tecnica simile a quella usata per costruire l'insieme dei complessi a partire dai numeri reali.

Esempio 2.3.1

Come si costruiscono i numeri complessi? Semplice, poiché non esiste nessun numero reale il cui quadrato sia -1, introduciamo un nuovo oggetto (che chiamiamo j) il cui quadrato, per definizione, è -1; "buttiamo" j in mezzo ai numeri reali e prendiamo tutte le possibili combinazioni (somme e prodotti) di j e numeri reali. La ben nota regola per il prodotto dei numeri complessi la si ottiene calcolando il prodotto in maniera simbolica e ricordando che $j^2 = -1$

$$\begin{aligned}(a+jb)(c+jd) &= ac + jbc + jad + j^2bd \\ &= ac + j(bc+ad) - bd \\ &= (ac-bd) + j(bc+ad)\end{aligned} \tag{2.27}$$

Lo stesso "trucco" funziona anche con (alcuni) campi finiti. Per esempio, in $\mathrm{GF}(3) = \mathbb{Z}_3$ non c'è nessun elemento il cui quadrato sia $[-1]_3 = [2]_3$ (infatti, $[2]_3^2 = [1]_3^2 = 1$), per cui possiamo introdurre un nuovo elemento, che chiameremo x e dire che per definizione $x^2 = [2]_3$. Prendiamo tutte le possibili combinazioni di x con gli elementi di $\mathbb{Z}_3$ otteniamo oggetti del $a + bx$, dove $a, b \in \mathbb{Z}_3$ la cui somma e prodotto è definita in maniera analoga ai numeri complessi

$$(a+bx) + (c+dx) = (a+c) + (b+d)x \tag{2.28a}$$

$$(a+bx)(c+dx) = (ac + [2]_3\, bd) + (cb+ad)x = (ac-bd) + (cb+ad)x \tag{2.28b}$$

È chiaro che l'insieme di polinomi che otteniamo ha 9 elementi. Con un po' di pazienza possiamo ottenere la tabellina mostrata in Fig. 2.3 da cui si deduce che l'insieme dei polinomi così ottenuto è effettivamente un campo poiché ogni riga/colonna corrispondente ad un elemento non nullo ha un "1" (mostrato in grassetto). L'insieme di polinomi (a coefficienti in $\mathbb{Z}_N$) che abbiamo costruito è quindi $\mathrm{GF}(9)$.

Un modo alternativo per verificare che ogni polinomio della forma $a + xb$ ha inverso moltiplicativo è osservare che

$$\begin{aligned}\frac{1}{a+bx} &= \frac{a-bx}{a^2 - b^2x^2} \\ &= \frac{a-bx}{a^2+b^2}\end{aligned} \tag{2.29}$$

Rimane da verificare che il termine a denominatore in (2.29) ha inverso moltiplicativo non appena $a + bx \neq 0$. Consideriamo separatamente i due casi $a = 0$ e $a \neq 0$

- Se $a = 0$ deve essere $b \neq 0$ da cui $a^2 + b^2 = b^2 \neq 0$
- se $a \neq 0$ possiamo scrivere
$$a^2 + b^2 = a^2(1 + (a^{-1}b)^2) \tag{2.30}$$
che è diverso da zero poiché $a \neq 0$ per ipotesi e $1 + (a^{-1}b)^2 \neq 0$ poiché *in* $\mathbb{Z}_3$ *l'equazione* $1 + x^2 = 0$ *non ha soluzione*.

Si osservi come il fatto che in $\mathbb{Z}_3$ $[-1]_3$ non abbia una radice quadrata sia *fondamentale* nella dimostrazione che ogni polinomio $a + bx$ ha inverso moltiplicativo.

Il trucco usato nell'Esempio 2.3.1 di estendere $\mathbb{Z}_3$ aggiungendo una radice quadrata di -1 funziona unicamente se -1 non ha radici quadrate. Per esempio, il trucco dell'Esempio 2.3.1 funzionerebbe per $\mathbb{Z}_7$, ma non per $\mathbb{Z}_5$ poiché $[2]_5^2 = [4]_5 = [-1]_5$. Può quindi nascere spontanea la domanda se sia possibile generalizzare il trucco ad altri casi.

2.3.1 Polinomi modulo $P(x)$

La generalizzazione del trucco usato per $\mathbb{Z}_3$ passa attraverso l'estensione del concetto di somma e prodotto modulo N ai polinomi. Il percorso che seguiremo sarà praticamente identico a quello seguito per gli interi. La prima cosa che ci serve è un po' di notazione.

$\times$	0	1	2	x	$1+x$	$2+x$	$2x$	$1+2x$	$2+2x$
0	0	0	0	0	0	0	0	0	0
1	0	**1**	2	x	$1+x$	$2+x$	$2x$	$1+2x$	$2+2x$
2	0	2	**1**	$2x$	$2+2x$	$1+2x$	x	$2+x$	$1+x$
x	0	x	$2x$	2	$2+x$	$2+2x$	**1**	$1+x$	$1+2x$
$1+x$	0	$1+x$	$2+2x$	$2+x$	$2x$	**1**	$1+2x$	2	x
$2+x$	0	$2+x$	$1+2x$	$2+2x$	**1**	x	$1+x$	$2x$	2
$2x$	0	$2x$	x	**1**	$1+2x$	$1+x$	2	$2+2x$	$2+x$
$1+2x$	0	$1+2x$	$2+x$	$1+x$	2	$2x$	$2+2x$	x	**1**
$2+2x$	0	$2+2x$	$1+x$	$1+2x$	x	2	$2+x$	**1**	$2x$

Figura 2.3: Tabellina dei prodotti in $\mathbb{Z}_3$ esteso tramite $x^2 = [2]_3$

Definzione 5. *Se $\mathbb{F}$ è un campo (per esempio, $\mathbb{C}$, $\mathbb{R}$, $\mathbb{Q}$ o $\mathbb{Z}_N$ con N primo), indicheremo con $\mathbb{F}[x]$ l'insieme dei polinomi a coefficienti in $\mathbb{F}$.*

Anche per i polinomi a coefficienti in un campo è possibile definire il resto della divisione "intera."

Definzione 6. *Se $\mathbb{F}$ è un campo e $A(x), B(x) \in \mathbb{F}[x]$ sono polinomi a coefficienti in $\mathbb{F}$, esistono (e sono unici) due polinomi $Q(x), R(x)$ tali che*

$$A(x) = B(x)Q(x) + R(x) \tag{2.31}$$

con $\deg R(x) < \deg B(x)$. *Il polinomio $R(x)$ è* il resto della divisione di $A(x)$ per $B(x)$ *e sarà indicato con* $A(x) \bmod B(x)$.

Esempio 2.3.2

Hmmm... ho un vago ricordo dalle superiori dell'algoritmo di divisione tra polinomi, ma come si calcola il resto di una divisione tra oggetti strani quali polinomi i cui coefficienti sono *insiemi di interi*? Semplice, *esattamente nello stesso modo* usato per i polinomi a coefficienti reali, *sostituendo somme, prodotti e divisioni* tra numeri reali con le corrispondenti operazioni tra interi modulo N.

Calcoliamo, per esempio, la divisione tra i polinomi in $\mathbb{Z}_5[x]$

$$A(x) := [2]_5 x^4 + [3]_5 x^3 + [4]_5 x^2 + [2]_5 \quad ; \quad B(x) := [3]_5 x^2 + [2]_5 x + [1]_5 \tag{2.32}$$

$$\begin{array}{l|l} [2]_5 x^4 + [3]_5 x^3 + [4]_5 x^2 \quad + [2]_5 & [3]_5 x^2 + [2]_5 x + [1]_5 \\ \hline & \end{array}$$

Il primo passo è dividere il termine di grado più alto di $A(x)$ (ossia $[2]_5 x^4$) per il termine di grado più alto di $B(x)$: otteniamo (con l'aiuto della tabellina in Fig. 2.2) $[2]_5 x^4\, [3]_5^{-1} x^{-2} = [2]_5\, [2]_5 x^2 = [4]_5 x^2$

$$\begin{array}{l|l} [2]_5 x^4 + [3]_5 x^3 + [4]_5 x^2 \quad + [2]_5 & [3]_5 x^2 + [2]_5 x + [1]_5 \\ \hline & [4]_5 x^2 \end{array}$$

Ora moltiplichiamo $[4]_5 x^2$ per $B(x)$ e sottraiamo il risultato da $A(x)$ ottenendo $A(x) - ([2]_5 x^4 + [4]_5 x^3 + [4]_5 x^2) = [4]_5 x^3 + [2]_5$

$$\begin{array}{l|l} \begin{array}{l} [2]_5 x^4 + [3]_5 x^3 + [4]_5 x^2 \quad + [2]_5 \\ - [2]_5 x^4 - [4]_5 x^3 - [4]_5 x^2 \\ \hline [4]_5 x^3 \quad + [2]_5 \end{array} & \begin{array}{l} [3]_5 x^2 + [2]_5 x + [1]_5 \\ \hline [4]_5 x^2 \end{array} \end{array}$$

Dividiamo ora $[4]_5 x^3$ per $[3]_5 x^2$ ottenendo $[8]_5 x = [3]_5 x$

Moltiplichiamo $[3]_5 x$ per $B(x)$ e sottraiamo il risultato da $[4]_5 x^3 + [2]_5$

$$\begin{array}{rl|l} [2]_5x^4+[3]_5x^3+[4]_5x^2 & +[2]_5 & [3]_5x^2+[2]_5x+[1]_5 \\ \cline{3-3} -[2]_5x^4-[4]_5x^3-[4]_5x^2 & & [4]_5x^2+[3]_5x \\ \cline{1-2} [4]_5x^3 & +[2]_5 & \end{array}$$

e così via ...

$$\begin{array}{rl|l} [2]_5x^4+[3]_5x^3+[4]_5x^2 & +[2]_5 & [3]_5x^2+[2]_5x+[1]_5 \\ \cline{3-3} -[2]_5x^4-[4]_5x^3-[4]_5x^2 & & [4]_5x^2+[3]_5x \\ \cline{1-2} [4]_5x^3 & +[2]_5 & \\ -[4]_5x^3-[1]_5x^2-[3]_5x & & \\ \cline{1-2} [4]_5x^2+[2]_5x+[2]_5 & & \end{array}$$

e così via ...

$$\begin{array}{rl|l} [2]_5x^4+[3]_5x^3+[4]_5x^2 & +[2]_5 & [3]_5x^2+[2]_5x+[1]_5 \\ \cline{3-3} -[2]_5x^4-[4]_5x^3-[4]_5x^2 & & [4]_5x^2+[3]_5x+[3]_5 \\ \cline{1-2} [4]_5x^3 & +[2]_5 & \\ -[4]_5x^3-[1]_5x^2-[3]_5x & & \\ \cline{1-2} [4]_5x^2+[2]_5x+[2]_5 & & \\ -[4]_5x^2-[1]_5x-[3]_5 & & \\ \cline{1-2} [1]_5x+[4]_5 & & \end{array}$$

Il resto della divisione $A(x)/B(x)$ è quindi $R(x) = [1]_5 x + [4]_5$.

Definzione 7. *diremo che due polinomi $A(x), B(x) \in \mathbb{F}[x]$ (a coefficienti in un campo) sono* equivalenti modulo $P(x) \in \mathbb{F}[x]$ *(e scriveremo $A(x) \equiv B(x) \pmod{P(x)}$) se danno lo stesso resto quando divisi per $P(x)$, ossia se*

$$A(x) \bmod P(x) = B(x) \bmod P(x) \tag{2.33}$$

o, equivalentemente, $A(x)$ e $B(x)$ sono equivalenti modulo $P(x)$ se differiscono per un multiplo di $P(x)$, ossia, esiste un polinomio $R(x)$ tale che

$$A(x) - B(x) = P(x)R(x). \tag{2.34}$$

2.3.2 Somma e prodotto tra polinomi modulo P

In maniera analoga a quanto fatto con gli interi, è possibile dimostrare che l'equivalenza modulo $P(x)$ è una relazione di equivalenza che divide l'insieme dei polinomi $\mathbb{F}[x]$ in classi di equivalenza. L'insieme delle classi di equivalenza verrà indicato con $\mathbb{F}[x]/P(x)$.

Esempio 2.3.3

Quali e quanti sono gli elementi di $\mathbb{Z}_2[x]/x^3+x+1$, l'insieme delle classi di equivalenza dei polinomi a cofficienti $\mathbb{Z}_2$ modulo il polinomio x^3+x+1? È facile convincersi che la classe a cui appartiene $A(x) \in \mathbb{Z}_2[x]$ è univocamente determinata da $R(x) = A(x) \bmod x^3+x+1$, il resto della divisione di $A(x)$ per x^3+x+1. Ne segue che le classi di $\mathbb{Z}_2[x]/x^3+x+1$ sono in corrispondenza biunivoca con i possibili resti della divisione per x^3+x+1 e poiché il resto della divisione per x^3+x+1 può avere grado al più due, ne deduciamo che esiste una corrispondenza biunivoca tra le classi di $\mathbb{Z}_2[x]/x^3+x+1$ ed i polinomi di $\mathbb{Z}_2[x]$ di grado non superiore a due. Il problema di enumerare le classi di $\mathbb{Z}_2[x]/x^3+x+1$ si riconduce quindi al problema di enumerare i polinomi di $\mathbb{Z}_2[x]$ della forma ax^2+bx+c, con $a,b,c \in \mathbb{Z}_2$.

Poiché $\mathbb{Z}_2$ ha due elementi ne segue che esistono 8 possibili polinomi a coefficienti in $\mathbb{Z}_2$ di grado non superiore a 2. Le classi di $\mathbb{Z}_2[x]/x^3+x+1$ sono quindi

$$\begin{array}{cccc} [0]_{x^3+x+1} & [1]_{x^3+x+1} & [x]_{x^3+x+1} & [x+1]_{x^3+x+1} \\ [x^2]_{x^3+x+1} & [x^2+1]_{x^3+x+1} & [x^2+x]_{x^3+x+1} & [x^2+x+1]_{x^3+x+1} \end{array} \tag{2.35}$$

Si osservi infine che ogni elemento di $\mathbb{Z}_2[x]/x^3+x+1$ ha una rappresentazione "naturale" come parola di tre bit $b_2b_1b_0$ considerando il bit b_i come il coefficiente di x^i.

Su $\mathbb{R}[x]/P(x)$ si possono introdurre la somma e il prodotto procedendo "per rappresentanti," in maniera analoga a quanto fatto con i numeri interi, ossia

$$[A]_{P(x)} + [B]_{P(x)} := [A+B]_{P(x)} \tag{2.36a}$$
$$[A]_{P(x)} \cdot [B]_{P(x)} := [A \cdot B]_{P(x)} \tag{2.36b}$$

Si lascia al lettore la verifica che le definizioni (2.36) sono ben poste.

Esempio 2.3.4

Calcoliamo il prodotto tra le classi $\left[x^2+1\right]_{x^3+x+1}$ e $\left[x^2+x\right]_{x^3+x+1}$. Per prima cosa, procediamo per rappresentanti ottenendo

$$\left[x^2+1\right]_{x^3+x+1}\left[x^2+x\right]_{x^3+x+1} = \left[x^4+x^3+x^2+x\right]_{x^3+x+1} \tag{2.37}$$

Già la (2.37) potrebbe essere una possibile risposta poiché sappiamo che possiamo rappresentare una classe di equivalenza usando un qualsiasi rappresentante. Poiché però è spesso più conveniente ricondursi sempre ad un insieme di rappresentanti "standard" (per esempio, (2.35)) vediamo come ottenere il rappresentante standard di $x^4+x^3+x^2+x$.

Un possibile approccio è dividere $x^4+x^3+x^2+x$ per x^3+x+1 e prendere il resto. Una tecnica leggermente più semplice è la seguente: noi vogliamo ridurre $x^4+x^3+x^2+x$ ad un polinomio di grado al più due, dobbiamo quindi eliminare i termini in x^4 e x^3. Poiché possiamo sommare a $x^4+x^3+x^2+x$ un qualsiasi multiplo di x^3+x+1 senza cambiarne la classe, sommiamo a $x^4+x^3+x^2+x$ il polinomio $x(x^3+x+1)$ onde "cancellare" il termine in x^4

$$\begin{aligned}\left[x^4+x^3+x^2+x\right]_{x^3+x+1} &= \left[x^4+x^3+x^2+x+x(x^3+x+1)\right]_{x^3+x+1}\\ &= \left[x^4+x^3+x^2+x+(x^4+x^2+x)\right]_{x^3+x+1}\\ &= \left[x^3\right]_{x^3+x+1}\end{aligned} \tag{2.38}$$

(si ricordi che i coefficienti sono elementi di $\mathbb{Z}_2$ per cui $1+1=0$!). Applichiamo ora lo stesso trucco a $\left[x^3\right]_{x^3+x+1}$ sommando a x^3 il polinomio x^3+x+1; si ottiene

$$\left[x^3\right]_{x^3+x+1} = \left[x^3+(x^3+x+1)\right]_{x^3+x+1} = [x+1]_{x^3+x+1} \tag{2.39}$$

Ne segue che possiamo scrivere

$$\left[x^2+1\right]_{x^3+x+1}\left[x^2+x\right]_{x^3+x+1} = [x+1]_{x^3+x+1} \tag{2.40}$$

o, facendo "cadere" la notazione $[\cdot]_{x^3+x+1}$,

$$(x^2+1)(x^2+x) = x+1 \pmod{x^3+x+1} \tag{2.41}$$

Commento 2.3.2

A questo punto il lettore potrà essere un po' perplesso poiché noi abbiamo cominciato a parlare di polinomi modulo P perché volevamo estendere il trucco usato per la definizione dei numeri complessi ad un caso più generale, ma non è chiaro quale sia il legame tra i polinomi modulo P ed i numeri complessi...

Il legame in realtà è molto semplice: *i numeri complessi possono essere considerati come polinomi a coefficienti reali modulo* x^2+1. Per convincersene si osservi che la somma delle classi $[a+bx]_{1+x^2}$ e $[c+dx]_{1+x^2}$ si scrive

$$[a+bx]_{1+x^2} + [c+dx]_{1+x^2} = [(a+c)+(b+d)x]_{1+x^2} \tag{2.42}$$

mentre il prodotto è

$$[a+bx]_{1+x^2}[c+dx]_{1+x^2} = [(a+bx)(c+dx)]_{1+x^2} \qquad \text{Procedo per rappresentanti} \tag{2.43a}$$
$$= \left[ac+bcx+adx+bdx^2\right]_{1+x^2} \tag{2.43b}$$
$$= \left[ac+(bc+ad)x+bdx^2-bd(1+x^2)\right]_{1+x^2} \qquad \text{"Cancello" } bdx^2 \tag{2.43c}$$
$$= [(ac-bd)+(bc+ad)x]_{1+x^2} \tag{2.43d}$$

Si osservi che per passare dalla (2.43b) alla (2.43c) abbiamo aggiunto tra parentesi quadre il polinomio $-bd(1+x^2)$; ciò è lecito poiché aggiungere un multiplo del modulo $1+x^2$ non cambia la classe. È facile riconoscere in (2.42) e (2.43) le regole per la somma ed il prodotto dei numeri complessi.

Commento 2.3.3

Intuitivamente, dire che $A(x)$ e $B(x)$ sono equivalenti modulo $P(x)$ corrisponde a dire "$A(x)$ e $B(x)$ sono uguali, a parte un multiplo di $P(x)$ (che non mi interessa)." In un certo senso, possiamo dire che per i nostri scopi possiamo assimilare $P(x)$ a zero. Considerare i polinomi modulo $1+x^2$ vuol dire quindi assimilare x^2+1 a zero o, equivalentemente, x^2 a -1.

Esercizio 2.3.1

Si consideri l'insieme $\mathbb{Z}_2[x]/P(x)$ dove $P(x) \in \mathbb{Z}_2[x]$ ha grado n.

1. Estendere la rappresentazione "naturale" introdotta alla fine dell'Esempio 2.3.3 all'insieme $\mathbb{Z}_2[x]/P(x)$. Quanti bit servono per rappresentare un elemento di $\mathbb{Z}_2[x]/P(x)$?
2. Come si esegue la somma tra elementi di $\mathbb{Z}_2[x]/P(x)$ operando sui rappresentanti binari?
3. * (un po' più difficile, ma non troppo) Disegnare lo schema di un circuito che moltiplica un elemento $\mathbb{Z}_2[x]/P(x)$ per x (operando, ovviamente, sui rappresentanti binari)

2.3.3 Divisione tra polinomi modulo P

Sempre in maniera analoga a quanto fatto per gli interi, la divisione tra polinomi modulo P può essere definita come il prodotto di una classe per l'inverso dell'altra, dove l'inverso di un generico elemento $[A]_P$ di $\mathbb{F}[x]/P(x)$ è la classe $\left[\hat{A}\right]_P$ tale che

$$[A]_P \left[\hat{A}\right]_P = [1]_P \tag{2.44}$$

Riscrivendo esplicitamente la (2.44) si ha che deve essere

$$A(x)\hat{A}(x) - P(x)Q(x) = 1 \tag{2.45}$$

per un qualche polinomio (incognito) $Q(x)$. La (2.45) è il corrispondente per i polinomi dell'equazione diofantea (2.22) e come per gli interi, la sua soluzione si appoggia sul corrispondente del teorema del resto cinese per i polinomi

Teorema 4 (Teorema del resto cinese per polinomi). *Siano $A(x), B(x), c(x) \in \mathbb{F}[x]$. L'equazione in $G(x)$ e $H(x)$*

$$A(x)G(x) + B(x)H(x) = c(x) \tag{2.46}$$

ha soluzioni in $\mathbb{F}[x]$ (ossia, esistono $G(x), H(x) \in \mathbb{F}[x]$ che rendono vera l'uguaglianza (2.46)) se e solo se *$c(x)$ è divisibile per il massimo comun divisore di $A(x)$ e $B(x)$.*

Anche la dimostrazione del Teorema 4 si appoggia sull'algoritmo di Euclide per il calcolo del massimo comun divisore.

Proseguendo sulla falsa riga di quanto fatto per gli interi modulo N possiamo quindi dire che $[A]_P \in \mathbb{F}[x]/P(x)$ ha inverso moltiplicativo *se e solo se* $\mathrm{MCD}(A,P) = 1$, ossia A e P sono coprimi. Ne segue che se vogliamo che *ogni* classe di $\mathbb{F}[x]/P(x)$ abbia un inverso moltiplicativo, $P(x)$ non deve essere fattorizzabile come prodotto di polinomi di grado minore, ossia $P(x)$ deve essere *primo*. Riassumendo, si ha

Teorema 5. *Sia $\mathbb{F}$ un campo generico. L'insieme $\mathbb{F}[x]/P(x)$ è un campo* se e solo se *$P(x)$ è primo, ossia non esistono due polinomi $A(x)$ e $B(x)$ entrambi di grado maggiore di zero tali che $P(x) = A(x)B(x)$.*

Commento 2.3.4

Dal Corollario 5 segue che l'insieme dei numeri complessi è un campo poiché $1+x^2 \in \mathbb{R}[x]$ non è fattorizzabile come prodotto di polinomi a coefficienti reali (ma questo lo sapevamo già...)

Esempio 2.3.5

Qualche tempo fa, l'autore si è scontrato con la necessità di eseguire calcoli *esatti*[3] con numeri del tipo $a+\rho b$,

[3] Il motivo per cui non era possibile accontentarsi dell'aritmetica in virgola mobile è che i calcoli prevedevano l'uso di termini del tipo ρ^{-n}, con n grande, che sarebbero diventati rapidamente molto piccoli e si sarebbero "persi" nelle nebbie del processore...

dove a e b erano numeri razionali e $\rho = (1+\sqrt{5})/2$. I numeri razionali possono essere rappresentati in maniera esatta all'interno del calcolatore, ma tentare di rappresentare la radice di cinque con un numero in virgola mobile avrebbe introdotto certamente delle approssimazioni. Poiché ρ è una radice di x^2-x-1 (e quindi $\rho^2 = \rho+1$), la soluzione a questo problema fu di lavorare con i *polinomi a coefficienti razionali modulo* x^2-x-1. Si osservi che poiché x^2-x-1 non è scrivibile come prodotto di polinomi *a coefficienti razionali*, x^2-x-1 è primo e possiamo quindi dedurre che $\mathbb{Q}[x]/x^2-x-1$ è un campo.

Più nel dettaglio, il numero $a+b\rho$, $a,b \in \mathbb{Q}$, veniva rappresentato dalla classe $[a+bx]_{x^2-x-1} \in \mathbb{Q}[x]/x^2-x-1$ (a sua volta rappresentata all'interno del calcolatore da un array di due numeri razionali); la regola per la somma era

$$[a+bx]_{x^2-x-1} + [c+dx]_{x^2-x-1} = [(a+c)+(b+d)x]_{x^2-x-1} \tag{2.47}$$

mentre la regola per il prodotto era

$$\begin{aligned} [a+bx]_{x^2-x-1} \cdot [c+dx]_{x^2-x-1} &= \left[ac+(bc+ad)x+bdx^2\right]_{x^2-x-1} \\ &= \left[ac+(bc+ad)x+bdx^2-bd(x^2-x-1)\right]_{x^2-x-1} \\ &= [(ac+bd)+(bc+ad+bd)x]_{x^2-x-1} \end{aligned} \tag{2.48}$$

Infine, per calcolare l'inverso $[\alpha+\beta x]_{x^2-x-1}$ di $[a+bx]_{x^2-x-1}$ è sufficiente osservare dalla (2.48) che deve essere

$$a\alpha + b\beta = 1 \tag{2.49a}$$
$$b\alpha + (a+d)\beta = 0 \tag{2.49b}$$

Le equazioni (2.49) formano un sistema lineare la cui soluzione è

$$\alpha = \frac{a+b}{a^2+ab-b^2} \tag{2.50a}$$
$$\beta = \frac{-b}{a^2+ab-b^2} \tag{2.50b}$$

Possiamo quindi scrivere

$$[a+bx]^{-1}_{x^2-x-1} = \left[\frac{a+b}{a^2+ab-b^2} + \frac{-b}{a^2+ab-b^2}x\right]_{x^2-x-1} \tag{2.51}$$

che rappresenta la regola per il calcolo dell'inverso.

Esercizio 2.3.2

Si estenda l'approccio dell'Esempio 2.3.5 per dire come si possa lavorare in aritmetica esatta con numeri del tipo $a+b\phi$, dove $a,b \in \mathbb{Q}$ e $\phi = (5+\sqrt{17})/2$, sapendo che ϕ è una delle radici dell'equazione $x^2-5x+2=0$.

2.3.4 Costruzione di campi finiti

Abbiamo finalmente tutti gli strumenti per costruire campi finiti con un numero di elementi non primo. Più precisamente, se p è un numero primo e $P(x) \in \mathbb{Z}_p[x]$ è un polinomio non fattorizzabile di grado n, il Teorema 5 ci garantisce che $\mathbb{Z}_p[x]/P(x)$ è un campo. Seguendo un ragionamento analogo a quello usato nell'Esempio 2.3.3 è facile verificare che le classi di $\mathbb{Z}_p[x]/P(x)$ sono in corrispondenza biunivoca con i polinomi di $\mathbb{Z}_p[x]$ di grado minore di n. Poiché un tale polinomio è univocamente determinato da n coefficienti ed ogni coefficiente può assumere p valori ne deduciamo che $\mathbb{Z}_p[x]/P(x)$ *è un campo finito con* p^n *elementi*, ossia $\mathrm{GF}(p^n)$.

Il ragionamento appena visto ci mostra come costruire *alcuni* campi finiti con p^n elementi, dove p è primo. Due domande sorgono spontanee

1. La costruzione funziona per ogni p e n? Ossia, dato p primo e n intero, posso sempre trovare un polinomio $P(x) \in \mathbb{Z}_p[x]$ di grado n e primo? La risposta è **sì** (ma non vediamo la dimostrazione)

2. È possibile trovare un campo con un numero di elementi che non sia pari ad una potenza di un numero primo (magari usando altre costruzioni)? La risposta è **no** (ma non vediamo la dimostrazione).

Riassumendo, abbiamo il seguente importante risultato

Teorema 6. *Esiste un campo finito $GF(N)$ con N elementi se e solo se $N = p^n$, con p primo. Ogni campo finito $GF(p^n)$ può essere costruito come $\mathbb{Z}_p[x]/P(x)$ dove $P(x) \in \mathbb{Z}_p[x]$ è un polinomio non fattorizzabile di grado n.*

Esempio 2.3.6

Costruiamo $\mathrm{GF}(16) = \mathrm{GF}(2^4)$. Come prima cosa, abbiamo bisogno di un polinomio $P(x)$ primo, di grado 4 e a coefficienti in $\mathbb{Z}_2$. Si può facilmente controllare che $P(x) = x^4 + x + 1$ non è fattorizzabile. Ne segue che gli elementi di $\mathrm{GF}(16)$ sono classi di equivalenza del tipo

$$\left[a_0 + a_1 x + a_2 x^2 + a_3 x^3\right]_{P(x)} \tag{2.52}$$

dove $a_i \in \mathbb{Z}_2$. Per esempio, il prodotto di $\left[1+x+x^2\right]_{P(x)}$ per $\left[x+x^3\right]_{P(x)}$ si scrive

$$\begin{aligned}
\left[1+x+x^2\right]_{P(x)}\left[x+x^3\right]_{P(x)} &= \left[(1+x+x^2)(x+x^3)\right]_{P(x)} \\
&= \left[x+\cancel{x^3}+x^2+x^4+\cancel{x^3}+x^5\right]_{P(x)} \\
&= \left[x+x^2+x^4(1+x)\right]_{P(x)} \\
&= \left[x+x^2+(1+x)(1+x)\right]_{P(x)} \qquad \text{Sommando } P(x)(x+1) = (x^4+x+1)(1+x) \\
&= \left[x+\cancel{x^2}+1+\cancel{x^2}\right]_{P(x)} \\
&= [1+x]_{P(x)}
\end{aligned}$$

2.3.5 I logaritmi

Ebbene sì, non ci accontentiamo di definire le quattro operazioni, ma vogliamo anche i logaritmi... La possibilità di definire i logaritmi su un campo finito discende dal seguente risultato (che non dimostriamo)

Teorema 7. *Per ogni campo finito $GF(p^n)$ esiste almeno un elemento g con la proprietà che per ogni elemento non nullo $x \in GF(p^n)$ esiste k tale che*

$$g^k = x \tag{2.53}$$

Il valore k in (2.53) può ovviamente essere interpretato come *il logaritmo di x in base g*.

Commento 2.3.5

Osservando che $\mathrm{GF}(p^n)$ ha $p^n - 1$ elementi non nulli e che la sequenza $g^0, g^1, g^2, \ldots$ dovrà prima o poi ripetersi, non è difficile convincersi che deve essere $g^{p^n-1} = 1$. A causa di tale periodicità è possibile considerare k in (2.53) come un elemento di $\mathbb{Z}_{p^n-1}$; più precisamente, possiamo definire l'elevamento di un elemento $x \in \mathrm{GF}(p^n)$ alla potenza $[k]_{p^n-1} \in \mathbb{Z}_{p^n-1}$ come

$$x^{[k]_{p^n-1}} := x^k = \underbrace{x \cdot x \cdot \cdots \cdot x}_{k \text{ volte}} \tag{2.54}$$

Si lascia al lettore la verifica che la definizione (2.54) è ben posta.

Sfruttando i logaritmi è possibile ricondurre il calcolo del prodotto e della divisione in $\mathrm{GF}(p^n)$ al calcolo di somma e differenza in $\mathbb{Z}_{p^n-1}$. Il problema è che il calcolo del logaritmo in un campo finito è un problema "difficile," al punto tale che esistono alcuni tecniche crittografiche (es. lo scambio di chiavi di Diffie-Hellman) la cui forza dipende proprio dalla difficoltà del calcolo del logaritmo complesso.

Commento 2.3.6

Nella tecnica di Diffie-Hellman per lo scambio di chiavi, due persone (Alice e Bob) si accordano (anche pubblicamente) su un numero primo N (molto grande) ed un generatore $g \in \mathbb{Z}_N$. Successivamente, quando Alice e Bob vogliono comunicare in maniera protetta, seguono il seguente algoritmo

- Alice, nel segreto della sua stanzetta, genera un numero aleatorio $e_A \in \mathbb{Z}_{N-1}$, calcola $t_A = g^{e_A}$ e trasmette t_A a Bob su un canale insicuro

- Bob, nel segreto della sua stanzetta, genera un numero aleatorio $e_B \in \mathbb{Z}_{N-1}$, calcola $t_B = g^{e_B}$ e trasmette t_B ad Alice su un canale insicuro
- Alice, usando il valore t_B ricevuto da Bob, calcola $k_A = t_B^{e_A} = g^{e_B e_A}$
- Bob, usando il valore t_A ricevuto da Alice, calcola $k_B = t_A^{e_B} = g^{e_B e_A} = k_A$
- Alice e Bob usano il valore $k_A = k_B$ come chiave

Un generico attaccante può mettersi in ascolto sul canale insicuro ed intercettare i valori di t_A e t_B. Se fosse facile calcolare il logaritmo discreto, l'attaccante potrebbe ottenere i valori di e_A e e_B tramite le equazioni

$$e_A = \log_g t_A \tag{2.55a}$$
$$e_B = \log_g t_B \tag{2.55b}$$

Se il campo però non è troppo grande, è sufficiente crearsi (elevando g a potenze via via più grandi) due tabelle, una che mappa ogni $x \neq 0$ nel suo logaritmo $\log_g x$ e l'altra che esegue l'associazione opposta (quest'ultima tabella non è in realtà strettamente necessaria poiché il calcolo di g^k può essere svolto usando un algoritmo efficiente che richiede circa $\log_2 k$ operazioni). Il calcolo del prodotto e dell'inverso in $\mathrm{GF}(p^n)$ si riconduce quindi a 3 accessi in memoria (2 per il calcolo dell'inverso) ed una somma.

2.4 Algoritmi per $\mathrm{GF}(2^n)$

Il caso forse più interessante di campo finito è quello con 2^n elementi poiché i suoi elementi possono essere indicizzati con parole a n bit. In particolare, se $n = 8k$, ogni elemento di $\mathrm{GF}(2^{8k})$ può essere rappresentato da un numero intero di byte. Una possibile obiezione che potrebbe nascere dopo aver visto la costruzione teorica presentata nei paragrafi precedenti è che calcolare in $\mathrm{GF}(2^n)$ può essere molto costoso da un punto di vista computazionale, poiché gli elementi di $\mathrm{GF}(2^n)$ sono polinomi e l'implementazione di operazioni in $\mathrm{GF}(2^n)$ richiede l'uso di funzioni per il calcolo con polinomi. In particolare, il prodotto in $\mathrm{GF}(2^n)$ richiederebbe di eseguire prima un prodotto tra polinomi seguito da una divisione di cui teniamo il resto.

In realtà, vedremo in questo capitolo che i calcoli in $\mathrm{GF}(2^n)$ possono essere svolti in maniera molto efficiente senza bisogno di usare strumenti di calcolo simbolico (in alcuni contesti applicativi le operazioni in $\mathrm{GF}(2^n)$ possono arrivare a costare meno delle corrispondenti operazioni sugli interi!). In questo paragrafo vedremo

- come rappresentazione degli elementi di $\mathrm{GF}(2^n)$ all'interno del calcolatore;
- come eseguire la somma in $\mathrm{GF}(2^n)$;
- come eseguire il prodotto in $\mathrm{GF}(2^n)$;

2.4.1 Rappresentazione degli elementi di $\mathrm{GF}(2^n)$

Come visto in § 2.3.4, $\mathrm{GF}(2^n)$ può essere costruito scegliendo un polinomio primo $P(x) \in \mathbb{Z}_2[x]$ di grado n e considerando l'insieme delle classi di equivalenza modulo $P(x)$, $\mathbb{Z}_2[x]/P(x)$. Seguendo un ragionamento analogo a quello fatto nell'Esempio 2.3.3, è possibile vedere che ogni classe di $\mathbb{Z}_2[x]/P(x)$ contiene uno ed uno solo polinomio di $\mathbb{Z}_2$ di grado minore di n. Ogni classe di $\mathbb{Z}_2[x]/P(x)$ può quindi essere rappresentata da un polinomio

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \tag{2.56}$$

dove ogni $a_k \in \mathbb{Z}_2$. Poiché ogni a_k può assumere unicamente i valori 0 e 1 (dovremmo dire $[0]_2$ e $[1]_2$ poiché gli a_k appartengono a $\mathbb{Z}_2$, ma credo che a questo punto non ci sia più possibilità di confusione...) possiamo rappresentare il polinomio (2.56) con la parola a n bit

a_{n-1}	a_{n-2}	$\cdots$	a_1	a_0

(2.57)

Per semplicità notazionale, da ora in poi indicheremo le parole del tipo (2.57) con la notazione

$$\left[a_{n-1}; a_{n-2}; \cdots; a_1; a_0\right] \tag{2.58}$$

Chiameremo la parola (2.58) il *rappresentante binario di* $A(x) = \sum_{k=0}^{n-1} a_k x^k$.

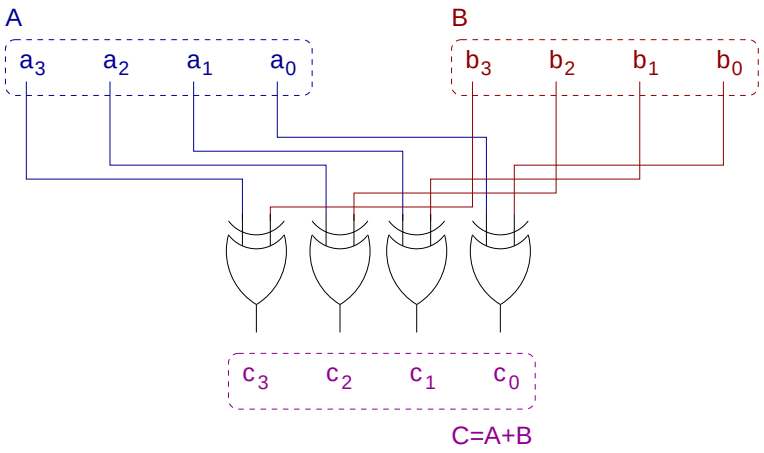

Figura 2.4: Circuito per la somma in GF(16)

2.4.2 Somma in GF(2^n)

La somma di due elementi $A = \sum_{k=0}^{n-1} a_k x^k$ e $B = \sum_{k=0}^{n-1} b_k x^k$ di $\mathbb{Z}_2[x]/P(x)$ (anche qui dovremmo usare la notazione $[\cdot]_P$, ma ormai siamo cresciuti...) si scrive

$$A(x) + B(x) = \sum_{k=0}^{n-1} (a_k + b_k) x^k \tag{2.59}$$

Dalla (2.59) è facile vedere che la parola di n bit che rappresenta $A(x) + B(x)$ è

$$[a_{n-1} + b_{n-1}; a_{n-2} + b_{n-2}; \cdots; a_1 + b_1; a_0 + b_0] \tag{2.60}$$

Ricordando che le somme in (2.60) sono somme in $\mathbb{Z}_2$ si ottiene

Risultato 1. *Il rappresentante binario di $A(x) + B(x)$ è l'or esclusivo dei rappresentanti binari di $A(x)$ e $B(x)$.*

Uno schema circuitale per la somma in GF(16) è mostrato in Fig. 2.4.

Costo computazionale Se la somma GF(16) è implementata in software (es. in C), il costo è pari al costo di un or esclusivo che certamente non è superiore al costo di una somma intera. Se la somma in GF(16) è implementata in hardware, dalla Fig. 2.4 è immediato riconoscere che il tempo di propagazione attraverso il sommatore in GF(16) è pari al tempo di propagazione attraverso un or esclusivo, certamente inferiore al tempo di propagazione attraverso un sommatore a quattro bit. È anche facile vedere che il tempo di propagazione attraverso un sommatore in GF(2^n) non dipende da n, mentre il tempo di propagazione attraverso un sommatore ad n bit cresce linearmente con n.

2.4.3 Prodotto in GF(2^n)

Prodotto per x in GF(2^n)

Come passo preliminare per lo sviluppo di un algoritmo efficiente per il prodotto in GF(2^n), è utile vedere come moltiplicare $A(x) = \sum_{k=0}^{n-1} a_k x^k$ per x. Si ha

$$\begin{aligned} x \cdot A(x) &= \sum_{k=0}^{n-1} a_k x^{k+1} \\ &= a_{n-1} x^n + a_{n-2} x^{n-1} + a_{n-3} x^{n-2} + \cdots a_1 x^2 + a_0 x \end{aligned} \tag{2.61}$$

Dalla (2.61) si vede facilmente che

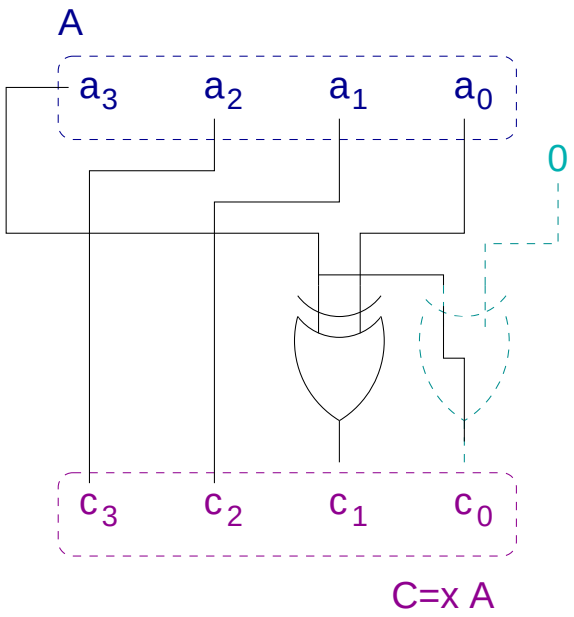

Figura 2.5: Circuito per il prodotto per x in $\mathbb{Z}_2[x]/x^4+x+1$

- Il coefficiente di x^0 in $x \cdot A$ è nullo
- Tutti i coefficienti traslano di un passo verso sinistra
- Il grado di $x \cdot A$ non è più grande di n ed è uguale a n se e solo se il $a_{n-1} = 1$

Come già detto più volte, quando si lavora modulo P, possiamo usare il rappresentante che più ci aggrada. Nonostante ciò, è spesso conveniente limitarsi a considerare i rappresentanti "standard" di grado minore di n. Per vedere come normalizzare (2.61) si osservi che

- Se $a_{n-1} = 0$, $x \cdot A$ è già normalizzato
- Se $a_{n-1} = 0$, si sommi $P(x) = x^n + \hat{P}(x)$ a $x \cdot A = x^n + \hat{A}(x)$ ottenendo

$$x \cdot A + P(x) = [x^n + \hat{A}(x)] + [x^n + \hat{P}(x)] = \hat{A}(x) + \hat{P}(x) \tag{2.62}$$

Riassumendo, l'algoritmo per moltiplicare A per x è

1. Trasla a sinistra di una posizione la parola rappresentativa di A, ponendo a zero il bit meno significativo del risultato e salvando a_{n-1} come "carry"
2. Se il carry è zero, hai finito, altrimenti esegui l'or esclusivo tra il risultato del passo precedente e la parola binaria che rappresenta $\hat{P}(x)$

Un circuito per lo shift di $A \in \mathrm{GF}(16) = \mathbb{Z}_2[x]/x^4+x+1$ è visibile in Fig. 2.5. Si osservi che poiché in questo caso $\hat{P}(x) = x+1$, se $a_3 = 1$ bisogna fare l'or esclusivo tra la parola ottenuta con lo shift $[a_2; a_1; a_0; 0]$ e la parola corrispondente a $\hat{P}$, ossia, $[0;0;1;1]$. In pratica, questo vuol dire che dopo aver fatto lo shift dobbiamo eseguire l'or esclusivo tra a_3 e i bit in posizione 0 e 1 della parola traslata, come mostrato in Fig. 2.5. Si osservi che la porta XOR disegnata con linea a tratti è "concettualmente" presente, ma praticamente inutile poiché il bit meno significativo della parola traslata è sempre zero.

Prodotto tra elementi di GF(2^n)

Il prodotto tra due elementi $A = \sum_{k=0}^{n-1} a_k x^k$ e $B = \sum_{k=0}^{n-1} b_k x^k$ di $\mathbb{Z}_2[x]/P(x)$ si scrive

$$A(x) \cdot B(x) = \left[\sum_{k=0}^{n-1} a_k x^k \right] B(x) = \sum_{k=0}^{n-1} a_k \cdot \left[x^k B(x) \right] = \sum_{k=0}^{n-1} a_k B_k(x) \tag{2.63}$$

dove, ovviamente, $B_k(x) := x^k B(x)$. A partire dall'equazione (2.63) possiamo ricavare il seguente algoritmo per il prodotto $A(x)B(x)$

1. Inizializza un accumulatore $C(x)$ a zero e $B_0(x) = B(x)$
2. Con k che va da 0 a $n-1$
   (a) Se $a_k = 1$, poni $C(x) = C(x) + B_k(x)$ (o, equivalentemente, poni $C(x) = C(x) + a_k B_k(x)$)
   (b) Calcola $B_{k+1}(x) = xB_k(x)$
3. $C(x)$ è il risultato del prodotto $A(x)B(x)$

Si osservi che l'algoritmo appena descritto esegue il prodotto $A(x)B(x)$ in n iterazioni e che è praticamente identico all'algoritmo per il prodotto di due interi in notazione binaria, con la differenza che la traslazione a sinistra viene sostituita dal prodotto per x al passo 2.b. Tale prodotto può essere calcolato usando l'algoritmo già visto.

La Fig. 2.6 mostra uno schema per il prodotto di due elementi di $\mathbb{Z}_2[x]/x^4+x+1$. I blocchi etichettati con "Shift" sono blocchi che eseguono il prodotto per x ed hanno la struttura di Fig. 2.5. I blocchi etichettati con "+" eseguono la somma in $\mathbb{Z}_2[x]/x^4+x+1$ ed hanno la struttura mostrata in Fig. 2.4. Si osservi la struttura ad albero dei sommatori che permette di avere il tempo di propagazione associato alle somme che cresce con $\log_2 n$ invece che con n. Si osservi inoltre che la stessa struttura potrebbe essere usata per il prodotto di interi a 4 bit, sostituendo gli shift in $\mathbb{Z}_2[x]/x^4+x+1$ con shift "semplici" e i blocchi etichettati con "+" con sommatori a $2n$ bit (sì, $2n$ e non n poiché il prodotto di due numeri a n bit ha in generale $2n$ bit).

Costo computazionale Se l'algoritmo per il prodotto viene implementato in software costa circa n shift e n or esclusivi. Se l'algoritmo viene implementato in hardware secondo lo schema di Fig. 2.6 si ha un tempo di propagazione che cresce con $n + \log_2 n$ associato al percorso che attraversa tutti gli shift e il tratto di albero di sommatori che va dall'uscita dell'ultimo shift all'uscita dello schema (freccia a tratti in Fig. 2.6). Se vogliamo confrontare tale ritardo di propagazione con quello di un moltiplicatore tra interi a n bit, possiamo osservare che nel moltiplicatore tra interi lo shift introduce un ritardo trascurabile, mentre i sommatori (a $2n$ bit) introducono un ritardo proporzionale a $2n$. Il ritardo complessivo in un moltiplicatore a n bit è quindi proporzionale a $2n \log_2 n$, ossia circa $\log_2 n$ volte più grande del ritardo associato al prodotto in GF(2^n).

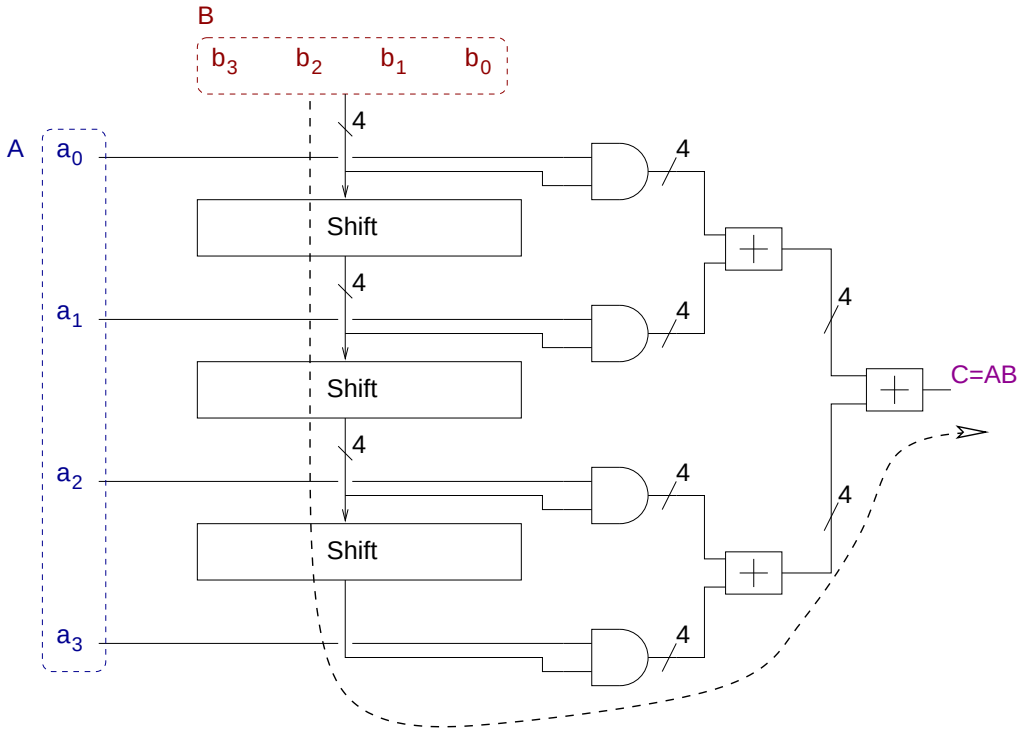

Figura 2.6: Schema per il prodotto di due elementi in $\mathbb{Z}_2[x]/x^4+x+1$

Capitolo 3

La pratica

Dopo una "buzzata" di algebra astratta, è giunto il momento di vedere qualche applicazione molto interessante dei campi finiti.

3.1 Condivisione di segreti

3.1.1 Il problema

La cosa migliore per introdurre il problema della condivisione di segreti è fare un esempio. Supponiamo che in una base militare la procedura per il lancio di un missile nucleare preveda l'utilizzo di un codice segreto di 33 cifre. Vista la gravità delle conseguenze derivanti da un lancio effettuato per errore, decidiamo che almeno tre degli M ufficiali di servizio alla base debbano essere presenti perché sia possibile lanciare il missile. Non possiamo quindi dare il codice segreto ad un singolo ufficiale, ma dobbiamo poter "dividere" il segreto in modo tale che

1. due soli ufficiali non possano dedurre *assolutamente nulla* sul codice, anche se mettono insieme tutte i dati che hanno a disposizione
2. qualsiasi gruppo di tre ufficiali riesce a ricostruire il segreto.

Una prima soluzione (sbagliata) potrebbe essere dividere il codice in tre porzioni (di 11 cifre ciascuna) e dare una porzione a ciascun ufficiale. Tale soluzione però non soddisfa la prima condizione poiché in questo modo due ufficiali "complici" possono arrivare a conoscere i due terzi del codice (per esempio, le prime 22 cifre). La soluzione proposta non soddisfa neanche la seconda condizione poiché in alcuni casi la ricostruzione del segreto potrebbe non essere possibile (per esempio, se nel gruppo di tre ufficiali ce ne sono due che hanno la stessa porzione del codice).

Più in generale il problema della "condivisione di segreti" è il seguente:

Problema 1. *Si vuole condividere un "segreto" X (il codice per il lancio del missile, una chiave crittografica, un codice di accesso ad una stanza protetta, ecc...) tra M persone in modo tale*

- *che qualsiasi gruppo di K persone sia in grado di ricostruire il segreto, ma*
- *qualsiasi gruppo di $K-1$ persone non possa dedurre assolutamente nulla sul valore di X (nell'esempio della base era $K=3$).*

Parleremo di problema di condivisione (M,K)

Vedremo in questi appunti come l'uso dei campi finiti permetta di risolvere questo problema, apparentemente insolubile, in modo molto semplice ed elegante.

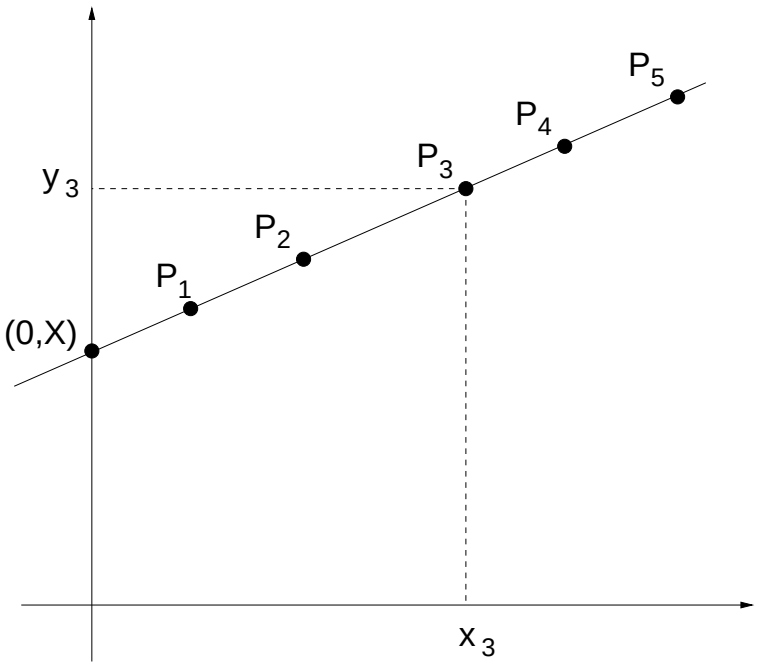

Figura 3.1: Costruzione per la condivisione di un segreto tra 5 persone.

3.1.2 Una soluzione "geometrica"

Per illustrare come il problema della condivisione di segreti possa essere risolto, conviene considerare inizialmente il caso $K = 2$. Supponiamo, senza perdita di generalità, che il segreto sia un numero[1] X.

Un sistema molto semplice per risolvere il problema $(M,2)$ è il seguente: si consideri il piano di Fig. 3.1 e si consideri il punto $P_0 = (0,X)$ dell'asse y. Si scelga una *qualsiasi* retta r non verticale passante per P_0, su questa retta si scelgano M punti a caso $P_1,\ldots,P_M$ (nell'esempio di figura $M = 5$) e si comunichino le coordinate di P_ℓ alla persona numero ℓ.

Vediamo come funziona la condivisione: supponiamo che la persona numero 2 e la numero 5 vogliano determinare X. A tale scopo è sufficiente che traccino la retta che passa per P_2 e P_5: l'intersezione di detta retta con l'asse y fornisce X (vedi Fig. 3.2.a).

Cosa può dedurre un singono individuo (diciamo la persona numero 4) a proposito del segreto a partire dalle informazioni in suo possesso? Assolutamente nulla; infatti, tutto ciò che la persona numero 4 sa della retta r è che passa per P_4, ma dato che per ogni X c'è una retta che passa per $(0,X)$ e P_4, la quarta persona non può sfruttare le informazioni in suo possesso per ridurre lo spazio di ricerca (vedi Fig. 3.2.b).

Nel caso volessimo $K = 3$, possiamo usare un approccio simile a quello appena visto con la sola differenza che invece di scegliere una retta scegliamo una *parabola* che passa per $(0,X)$. Dato che per tre punti passa una ed una sola parabola, si deduce facilmente che tre persone possono ricostruire il valore di X, ma due persone (per esempio la persona numero 3 e la numero 4) non possono dedurre nulla, dato che per ogni punto A dell'asse y esiste una parabola che passa per P_3, P_4 ed A.

Più in generale, per risolvere il problema (M,K)

1. Si sceglie *in maniera casuale* un polinomio $p(x) = \sum_{n=0}^{K-1} a_n x^n$ di grado $K-1$ tale che $p(0) = a_0 = X$.
2. Scegliamo M valori x_n, $n = 1,\ldots,M$, $x_n \neq 0$ e calcoliamo $y_n = p(x_n)$
3. Diamo la coppia $P_n = (x_n,y_n)$ all'n-simo partecipante.

[1] Qualsiasi informazione, codificata in binario, può essere interpretata come un numero intero. In alternativa, qualora la dimensione del segreto fosse talmente grande da rendere difficile la gestione pratica del numero associato (per esempio, se il segreto fosse un'immagine o uno spreadsheet), si può pensare di cifrare il documento e condividere la chiave.

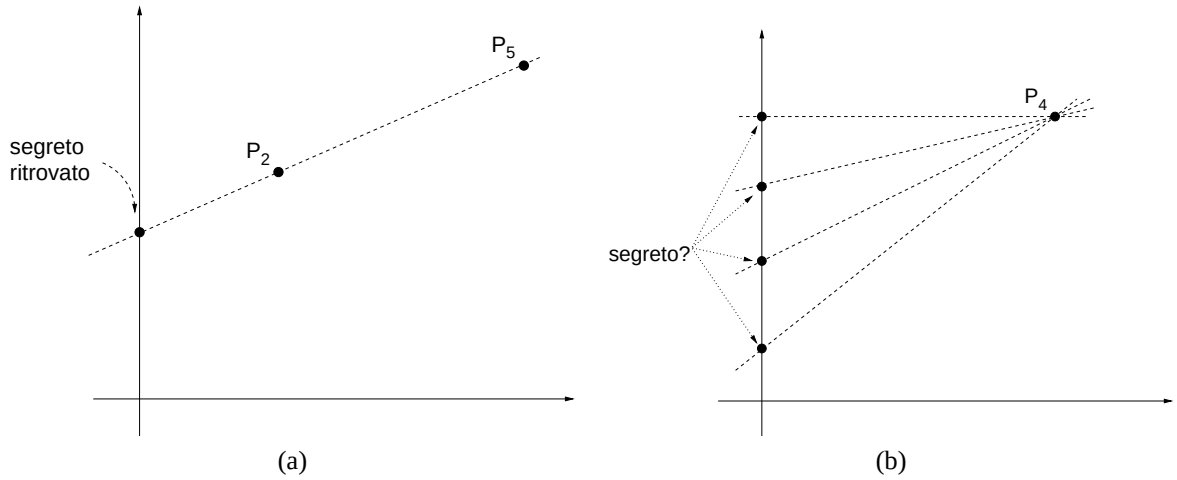

Figura 3.2: (a) Ricostruzione di un segreto condiviso. (b) Una singola persona non è in grado di ricostruire il segreto.

Se K partecipanti vogliono determinare X,

1. Determinano $p(x)$ sapendo che è l'unico polinomio di grado $K-1$ che passa per i punti P_n noti
2. Calcolano X come $X = p(0)$.

Si osservi che questo schema permette anche di avere individui "più importanti" di altri. Per esempio, tornando al caso della base militare, potremmo decidere che per lanciare il missile servano tre ufficiali "normali" o un (qualsiasi) ufficiale ed un generale. A tale fine è sufficiente dare ad ogni ufficiale un punto ed al generale *due punti* della parabola scelta per condividere il segreto.

Vale la pena mostrare esplicitamente come si possano calcolare i coefficienti a_n del polinomio $p(x) = \sum_{n=0}^{K-1} a_n x^n$ passante per i punti $(x_1,y_1),\ldots,(x_K,y_K)$.

Per determinare i coefficienti a_n dobbiamo risolvere il sistema di equazioni

$$\begin{aligned} y_1 &= a_{K-1}x_1^{K-1} + a_{K-2}x_1^{K-2} + \cdots a_1 x_1 + a_0 && \text{(3.1a)} \\ y_2 &= a_{K-1}x_2^{K-1} + a_{K-2}x_2^{K-2} + \cdots a_1 x_2 + a_0 && \text{(3.1b)} \\ &\vdots \\ y_K &= a_{K-1}x_K^{K-1} + a_{K-2}x_K^{K-2} + \cdots a_1 x_K + a_0 && \text{(3.1c)} \end{aligned}$$

dove, ricordiamo, i valori x_n e y_n, $n = 1,\ldots,K$ sono noti, mentre i coefficienti a_n sono incogniti. Riscrivendo il sistema (3.1) in forma matriciale otteniamo

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{K-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{K-1} \\ \vdots & \vdots & & & \\ 1 & x_K & x_K^2 & \cdots & x_K^{K-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{K-1} \end{bmatrix} = \mathbf{Ma} \qquad (3.2)$$

Il sistema (3.2) è chiaramente un sistema lineare e la matrice $\mathbf{M}$ in (3.2) è una matrice di Wandermonde e, come ben noto, il suo determinante vale

$$\det(\mathbf{M}) = \prod_{n=1}^{K-1} \prod_{m=n+1}^{K} (x_m - x_n) \qquad (3.3)$$

Dalla (3.3) è immediato riconoscere che il determinante di $\mathbf{M}$ è non nullo se e solo se tutti gli x_n sono diversi tra loro (ossia, stiamo usando K punti diversi). In tale caso la matrice $\mathbf{M}$ in (3.2) è invertibile e possiamo ricavare il vettore dei coefficienti $\mathbf{a}$.

3.1.3 Un algoritmo pratico

Tutto ciò che abbiamo detto nella sezione precedente va benissimo fintanto che si resta sulla carta, ma se proviamo ad implementare l'algoritmo in pratica ci scontriamo con alcuni problemi di ordine numerico.

È ben noto infatti che il calcolatore esegue i calcoli in precisione finita e per questo motivo dobbiamo aspettarci che il vettore dei coefficienti che otteniamo invertendo la matrice in (3.2) soddisfi il sistema solo in modo approssimato.

Se il risultato rappresentasse un valore "analogico" (per esempio, una temperatura), una lieve imprecisione sui risultati potrebbe essere accettabile. Purtroppo nel nostro caso il risultato finale rappresenta spesso un dato "discreto" e questo rende inaccettabile la benchè minima imprecisione. Se, per esempio, X fosse una chiave crittografica, anche un solo bit sbagliato renderebbe inutilizzabile il risultato. Tra l'altro le matrici di Wandermonde, come quella in (3.2), sono note per essere delle "brutte bestie" da un punto di vista numerico.

Probabilmente sarebbe possibile aggirare questi problemi numerici procedendo con un po' di cautela, ma esiste una soluzione più semplice: lavorare in un campo finito. Verifichiamo che ciò si possa fare

1. Il segreto prima era un numero intero (supponiamo a n bit) X, ora noi consideriamo come segreto la classe l'elemento di $\mathrm{GF}(2^n)$ il cui rappresentante binario è X Se S è il numero massimo di possibili segreti è chiaro che deve essere $2^n > S$ in modo che $\mathrm{GF}(2^n)$ abbia abbastanza elementi per rappresentare ogni possibile segreto.

2. Il secondo passo era trovare un polinomio

$$p(x) = a_{K-1}x^{K-1} + a_{K-2}x^{K-2} + \cdots + a_1 x + a_0 \tag{3.4}$$

   tale per cui $p(0) = X$ (il che implica $a_0 = X$). Si osservi che l'equazione (3.4) continua ad avere senso anche se decidiamo di lavorare su un campo finito; l'unica differenza è che i coefficienti a_n e la variabile x appartengono ad $\mathrm{GF}(2^n)$ e che le somme e prodotti in (3.4) devono essere considerati in $\mathrm{GF}(2^n)$. Poiché il polinomio deve essere scelto a caso, sceglieremo i coefficienti $a_1, \ldots, a_{K-1}$ a caso in $\{0, 1, \ldots, 2^n - 1\}$.

3. Il terzo passo consisteva nello scegliere M punti sul grafico del polinomio $p(x)$. Anche questo non costituisce un problema: si scelgono M elementi $x_1, \ldots, x_M$ di $\mathrm{GF}(2^n)$ e si calcolano i corrispondenti $y_n = p(x_n)$ applicando la (3.4).

4. L'ultimo passo richiede di poter determinare i coefficienti a_n a partire da K coppie di coordinate (x_n, y_n). Questo ci porta a riscrivere il sistema (3.1) (solo che ora somme e prodotti sono $\mathrm{GF}(2^n)$) che possiamo riscrivere in forma matriciale come in (3.2). L'ultimo passo, quello cruciale, è dimostrare se i valori x_n sono tutti diversi tra loro, allora la matrice $\mathbf{M}$ (che ora contiene interi modulo N) è invertibile. Questo discende dal fatto che, anche nel caso di $\mathrm{GF}(2^n)$, il determinante della matrice $\mathbf{M}$ in (3.2) può essere scritto come in (3.3) e questo perché la prova di (3.3) usa solo le "quattro operazioni" sui numeri reali (somma/differenza, moltiplicazione/divisione). La differenza fondamentale ora è che, dato che lavoriamo con numeri interi, non sono più presenti errori di approssimazione.

3.1.4 Variazioni sul tema

Protezione di database "sensibili"

Si supponga di avere un database di dati "sensibili" (es. dati clinici o finanziari) che possa essere un bersaglio "goloso" per qualche malintenzionato. In particolare temiamo che qualcuno possa penetrare nella macchina con il database e scaricarsi il database da esaminare poi con calma. Una possibile contromisura è condividire il database tra M macchine in modo che sia necessario contattare almeno K macchine per poter determinare i dati richiesti. In questo modo un attaccante deve penetrare almeno K macchine per poter accedere al database (e si spera che nel frattempo ci si sia accorti dell'attacco sulla prima macchina). Si osservi che questa soluzione ha anche il vantaggio di ridondare il database: è infatti sufficiente che almeno K su M macchine siano disponibili perché il database sia accessibile.

Minor occupazione di memoria

Un difetto dello schema per la condivisione di segreti è l'aumento di memoria richiesta. In effetti, per ogni byte da (M,K)-condividere dobbiamo distribuire $M \geq K$ byte. L'occupazione di memoria totale è quindi almeno K volte più grande. Per ridurre l'occupazione totale di memoria si può codificare più di un valore per volta. Più precisamente, si ricordi che il polinomio $P(x) = \sum_k p_k x^k$ usato nello schema viene scelto con l'unico vincolo che p_0 sia uguale al valore X da condividere. Se i valori da condividere sono due (diciamo, X_1 e X_2) possiamo sia ripetere la procedura due volte, sia porre $p_0 = X_1$, $p_1 = X_2$ e scegliere casualmente i rimanenti coefficienti. In questo modo codifichiamo due valori con K valori, per un aumento di occupazione di memoria pari a $K/2$.

Il problema del codificare due valori con un solo polinomio è una riduzione della sicurezza. Supponiamo, per esempio, di lavorare con $\mathrm{GF}(256)$, di voler (M,K)-condividere due valori X_1 e X_2 e che $K-1$ malintenzionati raccolgano tutti i dati a loro disposizione. Quante coppie (X_1,X_2) sono compatibili con i dati raccolti dai $K-1$ partecipanti?

- Se i due valori sono stati codificati separatamente, servono **due** elementi di $\mathrm{GF}(256)$ per ricavare X_1 e X_2. Non è difficile convincersi che le possibili coppie (X_1,X_2) compatibili con i dati raccolti sono $256^2 = 2^{16}$, ossia, le $K-1$ persone non possono dedurre nulla su X_1 e X_2
- Se i due valori sono stati codificati in un singolo polinomio, basta **un singolo** elemento di $\mathrm{GF}(256)$ per ricavare X_1 e X_2. Ne segue che solo 256 coppie (X_1,X_2) su 65.536 sono compatibili con i dati raccolti.

Peer-to-peer over ADSL

Un'altra applicazione della tecnica per la condivisione dei segreti è legata alla trasmissione di video *live* tramite architetture peer-to-peer (P2P). L'applicazione di riferimento che di solito si considera è quella in cui un server vuole trasmettere materiale video *live* (es. una partita di calcio) ad un numero relativamente elevato di utenti. La soluzione più diretta (il server manda "personalmente" il video a tutti gli utenti) ha il difetto molto serio che la banda necessaria al server cresce linearmente col numero degli utenti. Per ridurre il problema della banda necessaria al server si stanno studiando alcune soluzioni basate su architetture P2P in cui ogni utente ritrasmette ad altri utenti il contenuto video ricevuto.

La soluzione P2P è interessante, ma è afflitta da due problemi molto seri: la "volatilità" dei nodi intermedi (ossia, un nodo casalingo può abbandonare la rete da un momento all'altro senza preavviso) e l'asimmetria nella banda a disposizione degli utenti casalinghi (tipicamente collegati tramite ADSL). In particolare, per quanto riguarda l'asimmetria, nonostante il singolo utente abba abbastanza banda (dell'ordine dei Mbyte) per ricevere video di qualità, la banda di upload (256 Kbyte) non è sufficiente per agire come server.

Una possibile soluzione al problema dell'asimmetria di banda è di dividere il file da scaricare in blocchi (detti "chuck") e scaricare da ogni nodo una porzione diversa del file. Nonostante tale approccio funzioni bene per file "normali" che non hanno vincoli temporali, la sua utilità risulta limitata nel caso della trasmissione video poiché i *chunk* futuri non esistono ancora, mentre quelli passati sono ormai inutili. Una possibilità per usare l'approccio a chunk nella trasmissione video potrebbe essere l'uso di un buffer in cui tenere un certo numero di chunk passati, ma tale soluzione ha l'incoveniente di richiedere tempi lunghi di startup.

Un altro problema dell'approccio basato su chunk è la necessità di far conoscere ad ogni partecipante quali porzioni di quali file sono possedute da un certo nodo. L'overhead richiesto dalla diffusione di tale informazione può risultare non trascurabile.

Una soluzione alternativa, attualmente esplorata, alla divisione in chunk (*chunkless*) è dividere ogni pacchetto dello stream video in "porzioni" in modo tale che il pacchetto possa essere ricostruito non appena il nodo riceve K porzioni. Una volta ricostruito il pacchetto, il nodo crea una nuova porzione da inoltrare ad altri nodi. Il meccanismo di creazione delle porzioni dovrebbe essere tale che

- ogni nodo decide autonomamente (ossia, senza consultarsi con gli altri nodi della rete) come creare la porzione

- da qualsiasi combinazione di K porzioni diverse deve essere possibile ricreare il pacchetto originale.

Una possibile soluzione al partizionamento dei pacchetti, ispirata alla soluzione per la condivisione di segreti, è data dal seguente algoritmo

1. Ogni nodo sceglie casualmente un valore non nullo $b \in \mathrm{GF}(2^n)$ (con $n = 8$ o $n = 16$) e forma il vettore riga $\mathbf{r}_b = [1, b, \ldots, b^{K-1}]$ (chiamato in seguito *vettore di riduzione*)

2. Interpreta la sequenza di byte che compongono un pacchetto come un vettore $\mathbf{p} = [p_1, \ldots, p_L]$ ad elementi in $\mathrm{GF}(2^n)$. Il vettore $\mathbf{p}$ viene eventualmente allungato con zeri in modo da rendere la sua lunghezza pari ad un multiplo di K

3. Riorganizza i valori di $\mathbf{p}$ come una matrice $\mathbf{P}$ con K righe e L/K colonne

$$\mathbf{P} := \begin{bmatrix} p_1 & p_{K+1} & \cdots & p_{L-K+1} \\ p_2 & p_{K+2} & \cdots & p_{L-K+2} \\ \vdots & \vdots & & \vdots \\ p_K & p_{2K} & \cdots & p_L \end{bmatrix} \tag{3.5}$$

   e moltiplica a sinistra la matrice $\mathbf{P}$ per il vettore $\mathbf{r}$ per ottenere il vettore riga $\mathbf{c}_b = \mathbf{r}_b\mathbf{P}$ che rappresenta la porzione del pacchetto. Si osservi che poiché i ogni colonna di $\mathbf{P}$ viene "collassata" in un unico valore, la banda richiesta per trasmettere $\mathbf{c}_b$ è K volte minore della banda richiesta per trasmettere $\mathbf{p}$.

Non appena un nodo riceve K diverse porzioni $\mathbf{c}_{b_k}$, $k = 1, \ldots, K$, può ricostruire la matrice $\mathbf{P}$ (e quindi il pacchetto $\mathbf{p}$) risolvendo il sistema lineare

$$\underbrace{\begin{bmatrix} \mathbf{c}_{b_1} \\ \mathbf{c}_{b_2} \\ \vdots \\ \mathbf{c}_{b_K} \end{bmatrix}}_{\mathbf{C}} = \begin{bmatrix} \mathbf{r}_{b_1} \\ \mathbf{r}_{b_2} \\ \vdots \\ \mathbf{r}_{b_K} \end{bmatrix} \mathbf{P} = \underbrace{\begin{bmatrix} 1 & b_1 & b_1^2 & \cdots & b_1^{K-1} \\ 1 & b_2 & b_2^2 & \cdots & b_2^{K-1} \\ \vdots & \vdots & & & \vdots \\ 1 & b_K & b_K^2 & \cdots & b_K^{K-1} \end{bmatrix}}_{\mathbf{R}} \mathbf{P} \tag{3.6}$$

Più precisamente, si ha

$$\mathbf{P} = \mathbf{R}^{-1}\mathbf{C} \tag{3.7}$$

e poiché $\mathbf{R}$ è di Wandermonde è certamente invertibile non appena i valori b_k sono tutti diversi tra loro. Si osservi inoltre che poiché la matrice $\mathbf{R}$ dipende unicamente dai valori b_k e questi ultimi sono scelti una volta per tutte dai singoli nodi, la matrice $\mathbf{R}^{-1}$ non cambia durante la trasmissione e può essere calcolata una volta sola.

3.2 Just say no to twiddle!

Una delle applicazioni più importanti della FFT è il calcolo veloce della convoluzione. Come ben noto, il prodotto di DFT corrisponde nel tempo ad una convoluzione circolare che viene poi "trasformata" in una convoluzione lineare tramite gli algoritmi di *overlap-and-save* e *overlap-and-add* [2]. Nella ricerca di algoritmi sempre più efficienti, ci si potrebbe chiedere se non esista una trasformata più efficiente della FFT per calcolare la convoluzione circolare perché se così fosse tale vantaggio verrebbe immediatamente "ereditato" dagli algoritmi di *overlap-and-save* e *overlap-and-add*.

Per capire quali proprietà dovrebbe avere tale trasformata veloce, rivediamo brevemente i passaggi che dimostrano che il prodotto di due DFT corrisponde alla convoluzione circolare. Dato che ormai abbiamo preso dimestichezza con $\mathbb{Z}_N$, considereremo i segnali di cui calcoliamo le DFT come definiti su $\mathbb{Z}_N$. Osserviamo inoltre che poiché

$$W_N^{Nk} = 1 \tag{3.8}$$

possiamo considerare l'esponente di W_N come una classe di $\mathbb{Z}_N$. La definizione di DFT di $x : \mathbb{Z}_N \to \mathbb{C}$ diventa quindi la funzione $X : \mathbb{Z}_N \to \mathbb{C}$ definita da

$$X(k) = \sum_{n \in \mathbb{Z}_N} x(n) W_N^{nk} \qquad k \in \mathbb{Z}_N \tag{3.9}$$

mentre il prodotto delle DFT di $x : \mathbb{Z}_N \to \mathbb{C}$ e $y : \mathbb{Z}_N \to \mathbb{C}$ diventa

$$\begin{aligned}
X(k)Y(k) &= \left[\sum_{n \in \mathbb{Z}_N} x(n) W_N^{kn}\right] \left[\sum_{m \in \mathbb{Z}_N} y(m) W_N^{km}\right] && && \text{(3.10a)}\\
&= \sum_{m,n \in \mathbb{Z}_N} x(n) y(m) W_N^{k(n+m)} && \text{Sfrutto } W_N^{kn} W_N^{km} = W_N^{k(n+m)} && \text{(3.10b)}\\
&= \sum_{m,\ell \in \mathbb{Z}_N} x(\ell - m) y(m) W_N^{k\ell} && \text{Sostituzione } \ell = n + m && \text{(3.10c)}\\
&= \sum_{\ell \in \mathbb{Z}_N} W_N^{k\ell} \left[\sum_{m \in \mathbb{Z}_N} x(\ell - m) y(m)\right] && && \text{(3.10d)}\\
&= \sum_{\ell \in \mathbb{Z}_N} W_N^{k\ell} x * y(\ell) && && \text{(3.10e)}
\end{aligned}$$

dove la convoluzione in (3.10e) è (ovviamente) da intendersi circolare. Non è difficile convincersi che i due "ingredienti" principali su cui si basa la dimostrazione sono

- Il "nucleo" della DFT ha una forma "esponenziale" W_N^{nk} che permette in (3.10b) di trasformare il prodotto delle due funzioni W_N^{kn} e W_N^{km} nella singola funzione $W_N^{k(n+m)}$ in cui compare la somma degli indici temporali. (Si osservi che tutte le trasformate che godono della proprietà di trasformare convoluzioni in prodotti [Fourier, Laplace, Zeta, DFT, ...] hanno un nucleo di tipo esponenziale)
- La "base" W_N del nucleo della DFT è una radice N-sima dell'unità, ossia soddisfa la (3.8). È per tale motivo, infatti, che possiamo considerare l'esponente di W_N un elemento di $\mathbb{Z}_N$.

Si osservi in particolare che da nessuna parte è stata sfruttata l'ipotesi che W_N sia uguale a $\exp(-j2\pi/N)$. Quanto visto suggerisce che se si riuscisse a trovare una radice N-sima dell'unità ω_N tale che il prodotto per ω_N sia facile da calcolare, si potrebbe definire una nuova trasformata

$$\widetilde{X}(k) := \sum_{n \in \mathbb{Z}_N} x(n) \omega_N^{kn} \tag{3.11}$$

che potrebbe prendere il posto della DFT nel calcolo della convoluzione circolare. Per esempio, perché non usare per ω_N una potenza di due, in modo che i prodotti per ω_N^k diventino delle semplici traslazioni di bit? Come possiamo rendere 2 una radice dell'unità? Semplice... basta lavorare in $\mathbb{Z}_{2^\nu - 1}$ (infatti $2^\nu \equiv 1 \pmod{2^\nu - 1}$). Si osservi che poiché l'algoritmo per la convoluzione circolare (DFT diretta, prodotto, DFT inversa) richiede l'uso solamente di somme e prodotti, se si lavora in $\mathbb{Z}_{2^\nu - 1}$ il risultato dell'algoritmo sarà pari alla convoluzione circolare con i campioni presi modulo $2^\nu - 1$. Se ν viene scelto abbastanza grande da avere il più grande valore della convoluzione circolare minore in modulo di $(2^\nu - 1)/2$, la convoluzione circolare calcolata passando per la trasformata (3.11) coinciderà con la convoluzione circolare "vera."

Volendo essere più precisi, supponiamo di avere un segnale $x : \mathbb{Z}_N \to \mathbb{Z}$ tale che $|x(n)| < (2^\nu - 1)/2$, in modo da poter considerare i campioni di x come elementi di $\mathbb{Z}_{2^\nu - 1}$ senza perdita di informazione. Per ogni $k \in \mathbb{Z}$ definiamo $\omega_{2^k} \in \mathbb{Z}_{2^\nu - 1}$ come

$$\omega_{2^k} := 2^{\nu/2^k} \tag{3.12}$$

ed osserviamo che ω_{2^k} è una radice 2^k-sima dell'unità poiché $\omega_{2^k}^{2^k} = (2^{\nu/2^k})^{2^k} = 2^\nu = 1$. Se N è una potenza di due definiamo la PFT (*Pseudo-Fourier Transform*) di $x : \mathbb{Z}_N \to \mathbb{Z}_{2^\nu - 1}$ su N punti come il segnale $X : \mathbb{Z}_N \to \mathbb{Z}_{2^\nu - 1}$ definito da

$$X(k) = \mathrm{PFT}_N[x](k) := \sum_{n \in \mathbb{Z}_N} x(n) \omega_N^{nk} \tag{3.13}$$

Dovrebbe essere ormai ovvio che la PFT gode della proprietà della convoluzione

$$\mathrm{PFT}_N[x]\,\mathrm{PFT}_N[y] = \mathrm{PFT}_N[x * y] \tag{3.14}$$

Due domande rimangono al momento senza risposta: esiste un algoritmo veloce per il calcolo della PFT? La PFT può essere invertita?

3.2.1 PFT inversa

Prendendo spunto dalla DFT, proviamo a vedere cosa succede cambiando segno all'esponente della definizione della DFT

$$\hat{x}(\ell) = \sum_{k\in\mathbb{Z}_N} \widetilde{X}(k)\omega_N^{-\ell k} = \sum_{k\in\mathbb{Z}_N} \omega_N^{-\ell k} \sum_{n\in\mathbb{Z}_N} x(n)\omega_N^{nk} \tag{3.15a}$$

$$= \sum_{k,n\in\mathbb{Z}_N} \omega_N^{k(n-\ell)} x(n) \tag{3.15b}$$

$$= \sum_{k\in\mathbb{Z}_N} \omega_N^{k(n-\ell)} \sum_{n\in\mathbb{Z}_N} x(n) \tag{3.15c}$$

Si osservi ora che se $n = \ell$ si ha $\sum_{k\in\mathbb{Z}_N} \omega_N^{k(n-\ell)} = N$, mentre se $n \neq \ell$ si ha

$$\left[\sum_{k\in\mathbb{Z}_N} \omega_N^{k(n-\ell)}\right](1-\omega_N^{n-\ell}) = \sum_{k\in\mathbb{Z}_N} \omega_N^{k(n-\ell)} - \sum_{m\in\mathbb{Z}_N} \omega_N^{(m+1)(n-\ell)} \tag{3.16a}$$

$$= \sum_{k\in\mathbb{Z}_N} \omega_N^{k(n-\ell)} - \sum_{r\in\mathbb{Z}_N} \omega_N^{r(n-\ell)} \qquad \text{Sostituendo } m+1 \text{ con } r \tag{3.16b}$$

$$= 0 \tag{3.16c}$$

Poiché $n \neq \ell$, $\omega_N^{n-\ell} \neq 1$ e la (3.16) implica $\sum_{k\in\mathbb{Z}_N} \omega_N^{k(n-\ell)} = 0$. Riassumendo, si ha

$$\sum_{k\in\mathbb{Z}_N} \omega_N^{k(n-\ell)} = N\delta(n-\ell) \tag{3.17}$$

che usata nella (3.15) permette di dedurre

$$\sum_{k\in\mathbb{Z}_N} \widetilde{X}(k)\omega_N^{-\ell k} = Nx(\ell) \pmod{2^\nu - 1} \tag{3.18}$$

Poiché per ipotesi N è una potenza di due, esiste l'inverso moltiplicativo di N modulo $2^\nu - 1$ e dalla (3.18) possiamo ricavare l'espressione della PFT inversa

$$x(n) = N^{-1} \sum_{k\in\mathbb{Z}_N} \widetilde{X}(k)\omega_N^{-nk} \pmod{2^\nu - 1} \tag{3.19}$$

3.2.2 Calcolo veloce della PFT

Il calcolo veloce della PFT può essere svolto usando un algoritmo simile a quello di Cooley-Tukey, ma con i prodotti per i twiddle factor sostituiti da shift. Più precisamente, ponendo $k = 2a + b$ e $n = (N/2)\ell + m$ in

(3.13) si ottiene

$$X(2a+b) = \sum_{\ell=0}^{1}\sum_{m=0}^{N/2-1} x((N/2)\ell+m)\omega_N^{(2a+b)((N/2)\ell+m)} \tag{3.20a}$$

$$= \sum_{\ell=0}^{1}\sum_{m=0}^{N/2-1} x((N/2)\ell+m)\omega_N^{Na\ell}\omega_N^{(N/2)b\ell}\omega_N^{2am}\omega_N^{bm} \tag{3.20b}$$

$$= \sum_{\ell=0}^{1}\sum_{m=0}^{N/2-1} x((N/2)\ell+m)\omega_2^{b\ell}\omega_{N/2}^{am}\omega_N^{bm} \tag{3.20c}$$

$$= \sum_{m=0}^{N/2-1} \omega_{N/2}^{am}\left[\omega_N^{bm}\sum_{\ell=0}^{1} x((N/2)\ell+m)\omega_2^{b\ell}\right] \tag{3.20d}$$

Ricordando che $\omega_2 = -1$, l'algoritmo (3.20d) può essere scritto

$$x_+(n) = x(n) + x(n+\frac{N}{2}\ell) \qquad \text{Somma interna in (3.20d) con } b=0 \tag{3.21a}$$

$$x_-(n) = \omega_N^{bm}[x(n) - x(n+\frac{N}{2}\ell)] \qquad \text{Somma interna in (3.20d) con } b=1 \tag{3.21b}$$

$$X_+(2a) = \mathrm{PFT}_{N/2}[x_+](a) \tag{3.21c}$$

$$X_+(2a+1) = \mathrm{PFT}_{N/2}[x_-](a) \tag{3.21d}$$

È facile riconoscere nella (3.21) il primo stadio di decomposizione della FFT di Cooley-Tukey, ma con i twiddle factor sostituiti da $\omega_N^{bm} = (2^{\nu/N})^{bm}$.

3.2.3 Implementazione dell'aritmetica modulo $2^\nu - 1$

Somma modulo $2^\nu - 1$ Si supponga di scegliere $I := \{0,\dots,2^\nu-2\} \subset \mathbb{Z}$ come insieme di rappresentanti standard delle classi di $\mathbb{Z}_{2^\nu-1}$. Poiché la somma di due elementi di I non può mai essere maggiore di $2(2^\nu-2) = 2^{\nu+1}-4 < 2^{\nu+1}$ è sufficiente calcolare una somma a ν bit con un bit di riporto. Poiché il bit di riporto ha valore 2^ν e $2^\nu = 1 \pmod{2^\nu-1}$, è sufficiente correggere il risultato della somma addizionando al risultato il bit di riporto.

Shift a sinistra Lo shift a sinistra (ossia, il prodotto per 2) modulo $2^\nu-1$ può essere implementato eseguendo uno shift normale e correggendo quindi col carry. Poiché il carry dopo lo shift risulta uguale al bit più significativo del valore shiftato, è facile vedere che uno shift a sinistra modulo $2^\nu-1$ è equivalente ad uno shift *circolare*.

Prodotto modulo $2^\nu - 1$ Si ricordi che il prodotto di due numeri a ν bit è un numero a 2ν bit e si osservi che ogni numero a 2ν bit può sempre essere scritto

$$2^\nu H + L \tag{3.22}$$

dove $H, L \in \{0,\dots,2^\nu-1\}$. Poiché $2^\nu = 1 \pmod{2^\nu-1}$ il rappresentante standard di (3.22) è $H+L$, eventualmente corretto addizionandogli il carry.

Alternativamente, il prodotto può essere implementato tramite il solito algoritmo per il prodotto, ma usando shift circolari invece di shift normali.

Appendice A

Esempi

A.1 Un esempio di condivisione di segreti

Per capire esattamente il funzionamento dello schema di condivisione di segreti presentato può essere utile fare un esempio. Supponiamo che il numero massimo di segreti sia $L = 256$ (non è un gran numero, lo so, ma per questo esempio va bene) e supponiamo di volerlo dividere tra $M = 6$ persone in modo tale che siano necessarie almeno $K = 4$ persone per ricostruire il segreto. Supponiamo che il segreto sia $X = 42$.

Procediamo per passi:

- Il primo passo è scegliere la dimensione N del campo finito $\mathbb{Z}_N$. Per quanto detto prima N deve essere (a) maggiore di L e (b) un numero primo. Il più piccolo numero primo maggiore di 256 è $N = 257$. Lavoreremo quindi con gli interi modulo 257.

- Ora dobbiamo scegliere un polinomio $p(x)$ di grado $K - 1 = 3$ tale che

$$p(0) = X = 42 \tag{A.1}$$

  Come già osservato, la condizione (A.1) fissa il valore di $a_0 = 42$. Gli altri tre coefficienti possono essere scelti a piacere. Possiamo farci aiutare da Matlab

```
>> b=rand(1,3)
b =
    0.9501    0.2311    0.6068
>> a=ceil(257*b)
a =
   245    60   156
```

  Nella prima riga estraggo una matrice 1×3 di numeri casuali compresi tra 0 e 1 usando la funzione `rand`;[1] nella seconda riga moltiplico tale risultato per 257 e prendo la parte intera, ottenendo così tre numeri interi a caso tra 0 e 256 (estremi inclusi). Il polinomio che andiamo ad utilizzare è quindi

$$p(x) = 156x^3 + 60x^2 + 245x + 42. \tag{A.2}$$

  Si noti che i primi tre coefficienti sono quelli ottenuti tramite Matlab, mentre il quarto è il segreto che vogliamo condividere.

- Ora dobbiamo trovare $M = 6$ diversi valori per la variabile x in modo da calcolare i sei punti da consegnare ai singoli individui. Dato che la scelta delle coordinate x non influenza la sicurezza dello schema (finanto che restano segreti i corrisponendenti valori di y), possiamo scegliere, per comodità, $x_n = n$, con $n = 1, \ldots, 6$.

[1] Per avere maggiori informazioni su questa e su le altre funzioni di Matlab che useremo in seguito potete usare il comando `help` di matlab; per esempio: `help rand`

Per calcolare i valori di y possiamo, ancora una volta, farci aiutare da Matlab ricordando che le operazioni tra due classi (a e b) di $\mathbb{Z}_{257}$ si fanno scegliendo un elemento (un "rappresentante") per ogni classe, combinando i rappresentanti e prendendo la classe del risultato. Questo vuol dire che noi, fintanto che si tratta di fare somme e moltiplicazioni, possiamo "dimenticarci" che stiamo lavorando modulo N e portar avanti le operazioni come normali operazioni tra interi, salvo ricordarci di interpretare correttamente il risultato.

In particolare, per quanto riguarda il calcolo dei valori $p(x_n)$ possiamo usare la funzione `polyval` di Matlab che vuole come parametri il vettore dei coefficienti e l'insieme di punti a cui calcolare $p(x)$

```
>> p=[156 60 245 42]  % Vettore dei coefficienti
p =
   156    60   245    42
>> x=1:6  %  Dove calcolare il polinomio
x =
     1     2     3     4     5     6
>> y=polyval(p,x)
y =
       503     2020     5529    11966    22267    37368
```

Col primo comando carichiamo in `p` i coefficienti del polinomio; col secondo carichiamo in `x` i valori della x in cui vogliamo calcolare il polinomio; col terzo comando, infine, calcoliamo $p(x)$ per ogni valore nel vettore `x`. Si osservi, incidentalmente, l'uso del "`%`" per introdurre i commenti in Matlab e l'uso dell'espressione "`n:m`" per creare un vettore di interi da n ad m.

Un problema che riscontriamo è che l'ultimo comando restituisce valori molto alti. Questo in teoria non sarebbe un problema, dato che possiamo usare, per una data classe di $\mathbb{Z}_{257}$, qualsiasi "rappresentante" della classe stessa, anche numeri molto alti.

In pratica se i numeri con cui lavoriamo sono troppo alti c'è il rischio che Matlab passi dalla rappresentazione intera a quella in virgola mobile che ha un range più grande ma introduce errori di approssimazione. Onde prevenire tale eventualità, conviene sostituire ogni valore `y(n)` ottenuto in precedenza l'unico intero compreso tra 0 e 256 che appartiene alla stessa classe di `y(n)`. Dato che tale valore altro non è che il resto della divisione di `y(n)` per 256, possiamo sfruttarre la funzione `mod` di Matlab

```
>> y=mod(polyval(p,x), 257)
y =
  246   221   132   144   165   103
```

I punti da consegnare ai 6 partecipanti sono quindi

$$P_1 = (x_1, y_1) = (1, 246) \qquad P_2 = (x_2, y_2) = (2, 221) \qquad P_3 = (x_3, y_3) = (3, 132)$$
$$P_4 = (x_4, y_4) = (4, 144) \qquad P_5 = (x_5, y_5) = (5, 165) \qquad P_6 = (x_6, y_6) = (6, 103)$$

- Infine, vediamo cose si ricostruisce il segreto. Supponiamo che le prime quattro persone decidano di determinare X. Ciò che devono fare è scrivere un sistema simile a (3.2) a partire dalle coordinate dei punti $P_1, \ldots, P_4$, ossia

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 246 \\ 221 \\ 132 \\ 144 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ 1 & x_4 & x_4^2 & x_4^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{A.3}$$

L'unica cosa che manca è l'inversa della matrice in (A.3) ricordando che *gli elementi della matrice sono classi di* $\mathbb{Z}_{257}$*!*. Tale inversa può essere calcolata, per esempio, usando la regola di Cramer che ci dice che l'inversa è l'aggiunta divisa per il determinante oppure, in modo più efficiente, usando il ben noto algoritmo di eliminazione di Gauss, dato che tale algoritmo si basa solo sulle quattro operazioni.

In realtà, visto lo scopo dimostrativo di questi appunti decidiamo di scegliere la strada meno efficiente, ma più facile da fare con Matlab: calcoleremo l'inversa della matrice in (A.3) calcolandone l'aggiunta e dividendola per il determinante.

Per calcolare l'aggiunta possiamo ancora sfruttare Matlab. L'idea è che gli elementi dell'aggiunta di una matrice **M** si ottengono moltiplicando e sommando gli elementi di **M** stessa, ne deduciamo quindi che, ancora una volta, possiamo calcolare l'aggiunta di **M** modulo 257 semplicemente calcolando l'aggiunta di **M** tramite l'aritmetica "normale," avendo cura di interpretare il risultato finale modulo 257.

Nonostante matlab non abbia un comando per il calcolo dell'aggiunta possiamo sopperire calcolando `det(M)*inv(M)`

```
> M=[ 1 1 1 1 ; 1 2 4 8; 1 3 9 27 ; 1 4 16 64]
M =
     1     1     1     1
     1     2     4     8
     1     3     9    27
     1     4    16    64
>> Madj=inv(M)*det(M)
Madj =
   48.0000  -72.0000   48.0000  -12.0000
  -52.0000  114.0000  -84.0000   22.0000
   18.0000  -48.0000   42.0000  -12.0000
   -2.0000    6.0000   -6.0000    2.0000
```

I comandi non dovrebbero aver bisogno di spiegazioni; si noti solamente l'uso del "`;`" nella prima riga per separare tra loro le righe della matrice `M`. Si osservi inoltre che l'aggiunta che otteniamo è sì fatta di numeri interi, ma rappresentati in virgola mobile, come si deduce dal fatto che ogni valore termina per `.0000`. Questo è dovuto al fatto che Matlab calcola l'inversa di una matrice usando la virgola mobile. Il rischio che si corre con questo approccio è che gli elementi dell'aggiunta siano una versione approssimata di quelli "veri," anche se ciò in questo caso sembra poco probabile. Decidiamo di prenderli per buoni e controlleremo alla fine se il risultato è corretto.

Ancora una volta decidiamo, per comodità nostra, di "normalizzare" i valori di `Madj` usando la funzione `mod`

```
>> Madj=mod(round(Madj), 257)
Madj =
    48   185    48   245
   205   114   173    22
    18   209    42   245
   255     6   251     2
```

Si noti l'uso della funzione `round` per forzare gli elementi di `Madj` a valori interi.

Ci serve anche sapere il determinante di `M` (sempre modulo 257!)

```
>> detM=mod(det(M), 257)
detM =
    12
```

Ne deduciamo quindi che l'inversa della matrice in (A.3) può essere scritta

$$\mathbf{M}^{-1} = 12^{-1} \begin{bmatrix} 48 & 185 & 48 & 245 \\ 205 & 114 & 173 & 22 \\ 18 & 209 & 42 & 245 \\ 255 & 6 & 251 & 2 \end{bmatrix} \tag{A.4}$$

dove, ancora una volta, tutti prodotti e (soprattutto!) l'inverso di 12 vanno considerati modulo 257. L'ultima cosa che ci manca per calcolare la (A.4) è l'inverso di 12 modulo 257, ossia quel numero x tale che $12x = 1 \pmod{257}$ che equivale a richiedere che sia

$$12x = 1 + 257m \tag{A.5}$$

per qualche intero m. È facile verificare che

$$12 \cdot 150 = 1800 = 1 + 257 \cdot 7 \tag{A.6}$$

da cui $12^{-1} = 150 \pmod{257}$. Usando tale risultato nella (A.4) si ottiene

```
>> Minv=mod(Madj*150, 257)
Minv =
     4   251     4   256
   167   138   250   216
   130   253   132   256
   214   129   128    43
```

Verifichiamo se questa sia proprio l'inversa desiderata moltiplicando (modulo 257) per `M` e verificando che si ottenga la matrice identità

```
>> Minv*M
ans =
         515        1542        5140       18504
         771        2057        6425       21845
         771        2056        6426       22102
         514        1028        2570        7454
```

Che brutta! Questa non è la matrice identità! Dove abbiamo sbagliato? Da nessuna parte... Si ricordi che il risultato deve essere interpretato modulo 257 e che ogni valore ottenuto non rappresenta un numero intero, ma una classe di $\mathbb{Z}_{257}$. Per verificare che quest'ultima matrice rappresenti l'identità modulo 257 conviene "rinormalizzare" il risultato usando la funzione `mod`

```
>> mod(Minv*M, 257)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Ora si ragiona...

- Ultimo passo: calcoliamo il vettore dei coefficienti moltiplicando l'inversa appena trovata (con tanta fatica!) per il vettore delle y. Si ottiene

```
>> y=[246 ; 221; 132; 144]
y =
   246
   221
   132
   144
>> mod(Minv*y, 257)
ans =
    42
   245
    60
   156
```

da cui

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 42 \\ 245 \\ 60 \\ 156 \end{bmatrix} \tag{A.7}$$

Ricordando che il segreto coincide con il valore di a_0 otteniamo, correttamente, $X = a_0 = 42$.

Appendice B

Algoritmo di Euclide per il calcolo del MCD

In questo capitolo descriviamo brevemente (usando il linguaggio Ada[1]) l'algoritmo di Euclide per il calcolo del massimo comun divisore, citato a proposito della dimostrazione dei Teoremi 1 e 4.

Nei capitoli precedenti abbiamo parlato di un algoritmo di Euclide per gli interi ed un algoritmo per i polinomi. In realtà l'algoritmo di Euclide è uno solo e le due versioni (per i polinomi e gli interi) possono essere considerate diverse "istanze" dell'algoritmo "base."

L'algoritmo di Euclide richiede che l'insieme R degli oggetti su cui andiamo ad operare sia un *dominio euclideo* [1], ossia

- devono essere definite una somma ed un prodotto (commutativo) che godono delle proprietà della somma ed il prodotto tra interi (esistenza degli elementi neutri, commutatività della somma, esistenza dell'opposto, distributività e associatività di somma e prodotto, ...) (R è un *anello*)
- vale la legge di annullamento del prodotto, ossia se $ab = 0$, allora deve essere $a = 0$ o $b = 0$ (R è quindi un *dominio*)
- esiste una funzione $\deg : R \to \mathbb{N}$ tale che per ogni $A, B \in R$ esistono $P, Q \in R$ tali che

$$A = BQ + P \tag{B.1}$$

  con $\deg(P) < \deg(B)$. (È quest'ultima proprietà che rende il dominio R *euclideo*). Nel caso degli interi si può scegliere $\deg(n) := |n|$.

Si osservi la somiglianza della (B.1) con la definizione di resto per la divisione tra interi o polinomi. Vedremo che la funzione deg gioca un ruolo fondamentale nella terminazione dell'algoritmo di Euclide.

Specifiche

Cominciamo col dare le "specifiche" Ada (simile ai "prototipi" C o C++) della funzione generica per il calcolo dell'MCD.

```
generic                                                  -- 1
   type Ring is private;                                 -- 2
   Zero : Ring;                                          -- 3
   One  : Ring;                                          -- 4
   with function"+"(X, Y: Ring) return Ring is <>;       -- 5
```

[1] Perché Ada? Per diverse ragioni... La prima è che l'autore aveva già a disposizione una procedura generica per il calcolo del MCD in Ada, la seconda è che Ada è un linguaggio sufficientemente espressivo da poter essere compreso anche da chi non lo ha studiato

```
   with function"-"(X   : Ring) return Ring is <>;     -- 6 "meno unario"
   with function"*"(X, Y: Ring) return Ring is <>;     -- 7
   with function"/"(X, Y: Ring) return Ring is <>;     -- 8
   with function Degree(X: Ring) return Natural;       -- 9
--
-- L'operatore "/" e la funzione Degree devono essere tali che
--
--    Degree(X-Y*(X/Y)) < Degree(Y)
--
-- per ogni X e Y di tipo Ring
--


--
-- Restituisce in MCD il Massimo Comun Divisore tra X e Y
-- e in X_coeff e Y_coeff i valori tali che
--
--          MCD = X*X_coeff + Y*Y_coeff
--
procedure Generic_MCD(X, Y             : in     Ring;  -- 10
                      MCD              :    out Ring;
                      X_coeff, Y_coeff :    out Ring);
```

Vale la pena di commentare brevemente il codice.

- I commenti sono introdotto da "`--`"
- La prima riga con la parola chiave `generic` introduce una dichiarazione di *unità di compilazione*[2] *generica* (in questo caso una procedura generica). Le unità generiche in `Ada` sono simili ai *template* del `C++`, ma sono più potenti e hanno una sintassi più "umana." In pratica, una dichiarazione di procedura generica non dichiara una procedura effettivamente chiamabile, ma uno "stampo" usato per creare procedure specifiche. Per esempio, la procedura per il calcolo tra interi potrebbe essere creata con

  ```
  procedure Integer_MCD is
     new Generic_MCD(Ring   => Integer,
                     Zero   => 0,
                     One    => 1
                     Degree => Abs);
  ```

- Le linee dalla 2 alla 9 (incluse) descrivono i "parametri" che devono essere specificati in fase di creazione di una nuova istanza della procedura

Linea 2 `Ring` è il tipo per il quale vogliamo creare la procedura per il calcolo dell'MCD. La parola chiave `private` significa che non imponiamo alcun vincolo sulla struttura del tipo `Ring`. (Usando altre parole chiave si può richiedere, per esempio, che `Ring` sia un tipo discreto, in virgola mobile, derivato da un altro tipo, ecc...)

Linee 3 e 4 In un anello esistono sempre l'elemento neutro della somma (lo zero) e l'elemento neutro del prodotto (l'uno). Poiché il compilatore non sa quali elementi di `Ring` siano da considerare "zero" e "uno," dobbiamo essere noi a specificarli.

Linee 5–8 Queste linee dichiarano che su `Ring` devono essere definite le quattro operazioni. Poiché la dichiarazione di ogni operatore termina con "`is <>`," se l'utente non dichiara questi operatori in fase di istanziazione, vengono presi per default eventuali operatori già definiti su `Ring`. Si osservi che la linea 6 non dichiara la sottrazione, ma il meno unario, ossia, la funzione che restituisce l'opposto di `X`.

[2] Un'unità di compilazione (*compilation unit*) in `Ada` può essere definita informalmente come una pezzo di software compilabile separatamente. Le unità di compilazione in `Ada` sono: funzioni, procedure e package

Linea 9 Dichiarazione della funzione deg. Si noti il legame che deve sussistere tra la funzione `Degree` e l'operatore di divisione.

Linea 10 Dichiarazione del prototipo della procedura generica `Generic_MCD`

Corpo

Ora possiamo passare a descrivere l'algoritmo per il calcolo del MCD. Se X e Y sono i due valori di cui vogliamo calcolare il massimo comun divisore, la procedura che vedremo costruisce inizialmente una matrice

$$\left[\begin{array}{cc|c} 1 & 0 & Y \\ 0 & 1 & X \end{array}\right] \tag{B.2}$$

che viene successivamente moltiplicata a sinistra per matrici 2×2 invertibili ad elementi in R fino a portarla nella forma

$$\left[\begin{array}{cc|c} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & M \end{array}\right] \tag{B.3}$$

dove M è l'MCD di X e Y. È facile verificare che

$$M = a_{21}Y + a_{22}X \tag{B.4}$$

```
procedure Generic_MCD(X, Y             : in     Ring;
                      MCD              :    out Ring;
                      X_coeff, Y_coeff :    out Ring)
 is
   Mtx      : array(1..2, 1..3) of Ring;
   Quotient : Ring;

   -- Dichiarazione della sotto-procedura (visibile solo da
   -- Generic_MCD) Swap_Rows
   procedure Swap_rows(Mtx : array(1..2, 1..3) of Ring)
   is
      Tmp      : Ring;
   begin
      for Col in Mtx'Range(2) loop
          Tmp := Mtx(1,Col);
          Mtx(1, Col) := Mtx(1, Col);
          Mtx(2, Col) := Tmp;
       end loop;
   end Swap_rows;

   -- Dichiarazione della sotto-procedura Combine_Rows che somma
   -- alla seconda riga di Mtx la prima riga moltiplicata per Q
   procedure Combine_Rows (Mtx : array(1..2, 1..3) of Ring;
                           Q   : Ring)
   is
   begin
      for Col in Mtx'Range(2) loop
          Mtx(2, Col) := Mtx(2, Col) + Q*Mtx(1, Col);
       end loop;
   end Swap_rows;
begin
   -- Per prima cosa, ci sbarazziamo dei casi particolari in cui
   -- X o Y sono zero.
   if (X=Zero) then
      MCD := Y;
```

```
   X_Coeff := 1;
   Y_Coeff := 1;
   return;
end if;

if (Y=Zero) then
   MCD := X;
   X_Coeff := 1;
   Y_Coeff := 1;
   return;
end if;

-- Inizializza la matrice 2x3 usata nell'algoritmo.

Mtx(1,1) := One;   -- Blocco 2x2 inizializzato
Mtx(1,2) := Zero;  -- con la matrice identita'
Mtx(2,1) := Zero;
Mtx(2,2) := One;

Mtx(1,3) := Y;     -- Vettore colonna inizializzato
Mtx(2,3) := X;     -- con i parametri di ingresso

-- Qui Mtx e' pari a
--
--    [         Y ]
--    [ eye(2)    ]
--    [         X ]

--
-- Per nostra comodita' vogliamo  il termine di grado minore
-- in posizione (1,3).  Se necessario, scambiamo le righe
--
if (Degree(Mtx(1,3)) > Degree(Mtx(2,3))) then
    Swap_rows(Mtx);
end if;

--
-- Ora iniziamo il ciclo di elaborazione di Mtx. Usciremo dal ciclo
-- quando Mtx avra' la forma
--
--        [ a_11    a_12   Zero ]
--  Mtx = [                     ]
--        [ a_21    a_22    MCD ]
--
while (Mtx(1, 3) /= Zero) loop
   -- Invariante: qui si ha sempre
   --   1) Degree(Mtx(1,3)) <= Degree(Mtx(2,3))
   --   2) Mtx(1..2, 1..2)*[Y; X] = Mtx(1..2, 3)
   --   3) Mtx(1, 3) /= Zero

   --
   -- L'idea e' di sommare alla seconda riga la prima moltiplicata
   -- per un'opportuna costante Q in modo da sostituire Mtx(2,3) con
   -- il suo resto della divisione per Mtx(1,3).  Si osservi che
```

```
   -- tale operazione sulle righe corrisponde a moltiplicare Mtx a
   -- sinistra per
   --
   --    [ One    Zero ]
   --    [ Q      One  ]
   --
   --
   --
   -- Visto il legame tra "/" e la funzione Degree, la scelta
   -- piu‘ ovvia per Q e‘ l’opposto di Mtx(2,3)/Mtx(1,3);
   --
   Combine_Rows(Mtx, -Mtx(2,3)/Mtx(1,3));

   --
   -- In realta‘, visto che vogliamo il termine di grado piu‘
   -- piccolo sempre in posizione (1,3), dopo aver eseguito la
   -- combinazione delle righe, dobbiamo anche scambiare le righe
   -- tra loro.
   --

   Swap_rows(Mtx);

   --
   -- Se indichiamo con Mtx_Old il valore di Mtx all’inizio del
   -- ciclo, e‘ abbastanza facile verificare che
   --
   --    Mtx(1,3) = Mtx_Old(2,3) mod Mtx_Old(1,3)
   --    Mtx(2,3) = Mtx_Old(1,3)
   --
   -- E‘ chiaro che si ha
   --
   --    Degree(Mtx(1,3)) < Degree(Mtx(2,3))       [1]
   --    Degree(Mtx(1,3)) < Degree(Mtx_Old(1,3))   [2]
   --
   -- La condizione [1] e‘ l’invariante che si rinnova, mentre la
   -- [2] garantisce che il ciclo terminera‘ sempre poiche’ i gradi
   -- dell’elemento in posizione (1,3) formano una successione
   -- decrescente di numeri non negativi e tale successione non
   -- puo‘ andare avanti per sempre.
   --
end loop;

--
-- Quando arrivo qui
--
--        [ a_11    a_12   Zero ]
--  Mtx = [                     ]
--        [ a_21    a_22   MCD  ]
--
-- dove MCD e‘ il massimo comun divisore di X e Y e a_21 e a_22
-- sono tali che
--
--         MCD = Y*a_21 + X*a_22
--
```

```
   Mcd := Mtx(2,3);
   Y_Coeff := Mtx(2,1);
   X_Coeff := Mtx(2,2);
end Generic_MCD;
```

Esempio di utilizzo

```
with Text_IO;  -- Funzioni di I/O

procedure Prova is
  procedure Integer_MCD is
    new Generic_MCD(Ring   => Integer,
                    Zero   => 0,
                    One    => 1
                    Degree => Abs);
  MCD  : Integer;
  X, Y : Integer;
  X_Coeff, Y_Coeff : Integer;
begin
  X := 12;
  Y := 20;

  Integer_MCD(X, Y, MCD, X_Coeff, Y_Coeff);
  Text_IO.Put_Line("Il MCD di "
                   & Integer.Image(X) & " e "
                   & Integer.Image(Y) & " e‘ "
                   & Integer.Image(MCD) & "="
                   & Integer.Image(X_Coeff) & "*" & Integer.Image(X)
                   & "+"
                   & Integer.Image(Y_Coeff) & "*" & Integer.Image(Y) );
end Prova;
```

Bibliografia

[1] N. Jacobson, *Basic Algebra I*. New York, NY: W.H. Freeman, 1985.

[2] A. Oppenheim and R. S. with I.T. Young, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.