Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Protocolli crittografici innovativi basati sulla teoria dei gruppi

Laureando:
Filippo Bernardello

Relatore:
Ch.mo Prof. Alberto Tonolo

Anno accademico 2010/2011

. . .

Sommario

Breve sommario della tesi.

Di seguito viene presentata una rassegna dei capitoli della tesi e il loro contenuto.

Introduzione : si introduce il lettore alla crittografia moderna, citando alcuni famosi protocolli come RSA e mostrando come essi garantiscono la sicurezza. Dopodiché si espone brevemente l'idea di Shpilrain e Zapata che conduce ad un nuovo protocollo per la cifratura basato sul *word problem*;

Capitolo 1: si espongono i concetti base della teoria dei gruppi, vengono pure riportate le dimostrazioni dei risultati più importanti. Si parte dalla definizione di gruppo e di sottogruppo, passando per il primo teorema d'isomorfismo e si conclude con la definizione di gruppo libero e alcune sue proprietà;

Capitolo 2: si fornisce un primo approccio alla teoria della computabilità. Come modello di calcolo si utilizza la macchina di Turing che viene descritta in tutte le sue varianti. Vengono elencate inoltre, le varie classi di linguaggi: ricorsivi, ricorsivamente enumerabili, P e NP. Gran parte della trattazione segue [1];

Capitolo 3: dopo un elenco dei problemi più studiati della teoria combinatoria dei gruppi ci si concentra su uno di essi: il *word problem*. Questo problema costituisce la "botola" del sistema di cifratura di Shpilrain e Zapata: un intruso deve essere in grado di risolverlo per violare il crittosistema;

Capitolo 4: si descrive dettagliatamente il protocollo ideato da Shpilrain e Zapata [2]. Di esso si approfondisce ogni singolo aspetto: dai teoremi matematici che lo supportano, ai dettagli implementativi. Si presenta inoltre un possibile "quotient attack" a tale sistema di cifratura e si analizzerà la sua efficacia. Nelle ultime sezioni si generalizza il concetto di crittosistema basato su un problema decisionale della teoria combinatoria dei gruppi e si forniscono la descrizione e alcuni risultati sperimentali dell'implementazione del protocollo studiato in linguaggio C;

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli algoritmi

Introduzione

La crittografia [3] moderna[1] è garante della sicurezza informatica sin dalla sua introduzione nel 1976, quando W. Diffie e M. Hellman idearono un modo nuovo e rivoluzionario per trasmettere informazione. L'idea di base è abbastanza semplice: prevede l'utilizzo di una cosiddetta *"funzione unidirezionale"* (one-way function) f per codificare il messaggio [4]. Informalmente una funzione unidirezionale è una funzione per la quale è semplice calcolare il valore di $f(x)$ per ogni argomento x nel dominio di f, ma è molto complicato calcolare il valore della sua inversa[2] $f^{-1}(y)$ per la maggior parte del valori y del codominio di f. La più famosa funzione unidirezionale è quella emersa dal lavoro di Rivest, Shamir e Adleman che costituisce le fondamenta del celeberrimo sistema di cifratura[3] a chiave pubblica chiamato RSA. Tale protocollo ad esempio è impiegato negli internet browser Netscape e Internet Explorer; appare chiaro che RSA svolge un ruolo cruciale, ogni giorno di più, nelle transazioni e nelle comunicazioni che si verificano in Internet.
Questo schema basa la sua efficacia, come molti altri sistemi di cifratura, sulla complessità dei gruppi finiti abeliani (o commutativi). Essi sono particolari esempi di *gruppi finitamente generati*.
Si svolgono ricerche sui gruppi finitamente generati da più di 150 anni e hanno dimostrato una straordinaria complessità. Sebbene la sicurezza di Internet non appaia al momento minacciata, a causa dell'assenza di punti deboli dei protocolli di sicurezza attuali (come RSA), la prudenza invita allo sviluppo di eventuali loro miglioramenti o addirittura di protocolli sostitutivi basati su gruppi finiti abeliani.
La prima parte di questa tesi, dopo i capitoli introduttivi necessari per fornire le conoscenze basilari sui gruppi [4], per definire cosa si intente per "problema" e per "difficoltà",

[1]Con i termini *crittografia moderna, crittografia a chiave pubblica o crittografia asimmetrica* vengono identificati tutti i crittosistemi la cui funzione di decifratura presenta una complessità computazionale di ordine maggiore rispetto a quella di cifratura. Viceversa alla classe *crittografia classica o crittografia simmetrica* appartengono i crittosistemi in cui la funzione di decifratura può essere implementata con una complessità computazionale equivalente a quella di cifratura.

[2]Una funzione $f : X \to Y$ si dice *invertibile* se esiste una funzione $g : Y \to X$ tale che $g(f(x)) = x \ \forall x \in X$, e $f(g(y)) = y \ \forall y \in Y$. Se esiste, g è unica. Si indica con f^{-1}, ed è detta la *funzione inversa* di f. Se f è biiettiva esiste la sua inversa.

[3]Sia $\mathbb{M}$ l'insieme dei messaggi in chiaro e $\mathbb{C}$ l'insieme dei messaggi cifrati. Si definisce *funzione di cifratura* una funzione iniettiva $f : \mathbb{M} \to \mathbb{C}$. Dato che è iniettiva, è possibile definire su $f(\mathbb{M})$ la sua inversa f^{-1}. Un *crittosistema* è una quaterna $(\mathbb{M}, \mathbb{C}, f, f^{-1})$. A tale quaterna sovente si aggiunge un altro insieme $\mathbb{K}$, detto insieme delle *chiavi*. Con il termine *chiave* si indicano i parametri riguardanti le funzioni di cifratura/decifratura.

descriverà alcuni protocolli basati sulla teoria dei gruppi. Questi schemi si basano su alcuni *problemi di ricerca* (search problems) che risultano essere varianti di tradizionali *problemi di decisione* (decision problems) della teoria combinatoria dei gruppi. I protocolli basati sui problemi di ricerca ricadono nel paradigma generale dei crittosistemi a chiave pubblica basati su funzioni unidirezionali. Nella seconda parte della tesi verrà presentato un protocollo proposto da Vladimir Shpilrain e Gabriel Zapata che racchiude al proprio interno un problema di decisione. Tale natura innovativa dello schema interrompe il legame con i paradigmi canonici e permette di costruire protocolli crittografici sicuri anche contro alcuni "attacchi a forza bruta[4]" portati da un avversario con potenza di calcolo illimitata.

[4]Il metodo forza bruta (anche noto come ricerca esaustiva della soluzione) è un algoritmo di risoluzione di un problema che consiste nel verificare tutte le soluzioni teoricamente possibili fino a che si trova quella effettivamente corretta. Il suo principale fattore positivo è che consente teoricamente sempre di trovare la soluzione corretta, ma per contro è sempre la soluzione più lenta o dispendiosa.

Capitolo 1

Richiami teorici

In questa prima parte si espongono i concetti base della teoria dei gruppi. Si parte dalla definizione di gruppo e di sottogruppo, passando per il primo teorema d'isomorfismo e si conclude con la definizione di gruppo libero e alcune sue proprietà.

1.1 Teoria dei gruppi

1.1.1 Definizioni fondamentali

Il concetto di *gruppo* fu per la prima volta formalizzato da Cayley (1821-95) intorno al 1854.

Definizione 1.1. *Un'* OPERAZIONE *su un insieme G è una mappa $\circ : G \times G \to G$. Il valore $\circ(g, h)$ viene spesso indicato con $g \circ h$ o gh.*

Definizione 1.2. *Una coppia $(G, \circ)$ formata da un insieme G e un'operazione $\circ : G \times G \to G$ è detta* GRUPPO *se soddisfa le seguenti tre proprietà:*

- *L'operazione è **associativa**:*

$$s_1 \circ (s_2 \circ s_3) = (s_1 \circ s_2) \circ s_3 \quad \forall s_1, s_2, s_3 \in G$$

- *Esiste un **elemento neutro**, ovvero un elemento $e_G \in G$ tale che*

$$e_G \circ s = s \quad \wedge \quad s \circ e_G = s \quad \forall s \in G$$

  Nel caso sia chiaro il gruppo in questione, si scriverà semplicemente "e"

- *Per ogni $s \in G$ esiste un **inverso**, ovvero un elemento $t \in G$ tale che*

$$s \circ t = e_G \quad \wedge \quad t \circ s = e_G$$

Definizione 1.3. *Un gruppo G è detto* ABELIANO *se $x \circ y = y \circ x \ \forall \ x, y \in G$. Il numero di elementi $|G|$ di G è chiamato* ORDINE *di G o* CARDINALITÀ *di G.*

1.1.2 Sottogruppi e classi laterali

Si dimostra facilmente che in un gruppo G ci può essere un unico elemento neutro, e per ogni $g \in G$ un solo inverso, indicato con g^{-1}. Dal mondo dei numeri si possono attingere i primi semplici esempi di gruppi: si pensi ad esempio a $(\mathbb{Z}, +)$ oppure a $(\mathbb{R}\setminus\{0\}, \times)$. Diversamente $(\mathbb{N}, +)$ non è gruppo: l'operazione è associativa, esiste unico elemento neutro (il numero 0), ma l'elemento 1, ad esempio, non ammette inverso in $\mathbb{N}$: $\nexists x \in \mathbb{N}$ tale che $x + 1 = 0$. Può accadere che un gruppo contenga un sottoinsieme di elementi che, a sua volta, è gruppo con la medesima operazione. Questa considerazione porta alla definizione di sottogruppo.

Definizione 1.4. *Un* SOTTOGRUPPO *di un gruppo $(G, \circ)$ è un sottoinsieme non vuoto $H \subseteq G$ tale che $(H, \circ)$ è gruppo a sua volta. H è sottogruppo di G se e solo se*

- $e_G \in H$
- $x^{-1} \in H \quad \forall\, x \in H$
- $xy \in H \quad \forall\, x, y \in H$

Alcune proprietà di un gruppo sono ereditate dai suoi sottogruppi: un sottogruppo di un gruppo finito è finito, un sottogruppo di un gruppo abeliano è abeliano.

Definizione 1.5. *Sia H un sottogruppo di G e sia $g \in G$. Allora il sottoinsieme*

$$gH = \{gh | h \in H\} \subseteq G$$

è chiamata CLASSE LATERALE SINISTRA *di g rispetto a H. In maniera equivalente il sottoinsieme*

$$Hg = \{hg | h \in H\} \subseteq G$$

è detto CLASSE LATERALE DESTRA *di g rispetto a H. L'insieme delle classi laterali sinistre di H è indicato con G/H. L'insieme delle classi laterali destre è indicato con $H \setminus G$.*

Lemma 1.6. *Sia H un sottogruppo di un gruppo G e siano $x, y \in G$. Allora*

- $x \in xH$
- $xH = yH \iff x^{-1}y \in H$
- *se $xH \neq yH$ allora $xH \cap yH = \emptyset$*
- *La mappa $\varphi : H \to xH$ data da $\varphi(h) = xh$ è biiettiva*

Corollario 1.7. *Sia H un sottogruppo di G. Allora*

$$G = \bigcup_{g \in G} gH$$

e se $g_1H \neq g_2H$ allora $g_1H \cap g_2H = \emptyset$

1.1.3 Sottogruppi normali e gruppi quoziente

Si espone ora un'importante nozione della teoria dei gruppi:

Definizione 1.8. *Un sottogruppo N di un gruppo G è detto* NORMALE *se*

$$gNg^{-1} = \{gng^{-1} | n \in N\} = N$$

per ogni $g \in G$.

Se N è normale in G allora l'insieme G/N delle classi laterali sinistre eredita da G una struttura di gruppo:

Proposizione 1.9. *Sia N un sottogruppo normale del gruppo $(G, \circ)$. Allora la seguente operazione $\star$ tra classi laterali, fa di $(G/N, \star)$ un gruppo:*

$$(g_1 N) \star (g_2 N) = (g_1 \circ g_2)N$$

per ogni $g_1 N, g_2 N \in G/N$.

Definizione 1.10. *Sia N un sottogruppo normale di G. Il gruppo G/N è chiamato* GRUPPO QUOZIENTE.

1.1.4 Morfismi di gruppo

Definizione 1.11. *Siano G e K due gruppi. Una mappa $f : G \to K$ è detta* MORFISMO DI GRUPPO *se $f(xy) = f(x)f(y)\ \forall\ x, y \in G$*

Sia $f : G \to K$ un morfismo di gruppo. Gli elementi di G che vengono mappati nell'elemento neutro di K costituiscono il nucleo (kernel) del morfismo considerato:

Definizione 1.12. *Il* NUCLEO *(kernel) di un morfismo $f : G \to K$ è*

$$\ker(f) = \{g \in G | f(g) = e_K\}$$

L'immagine di f è $f(G) = \{f(g) \mid g \in G\} \subseteq K$. Un morfismo biiettivo è chiamato ISOMORFISMO DI GRUPPO. Si diranno isomorfi due gruppi G e H tra i quali esiste un isomorfismo: in tal caso si scriverà $G \cong H$.
Si prosegue ora con l'enunciato del primo teorema d'isomorfismo:

Teorema 1.13. *Siano G e K due gruppi e $f : G \to K$ un morfismo con nucleo $N = \ker(f)$. Allora*

$$\tilde{f} : G/N \to f(G)$$

dato da $\tilde{f}(gN) = f(g)$ è una mappa ben definita e un isomorfismo di gruppo.

Se $f : G \to K$ è un morfismo suriettivo allora per il teorema 1.13 si ha:

$$\tilde{f} : G/\ker(f) \cong K$$

1.1.5 Gruppi ciclici

In un gruppo $(G, \circ)$, dove l'operazione "$\circ$" sarà chiamata per comodità *moltiplicazione*, si può scegliere un elemento a caso $g \in G$ e moltiplicarlo per se stesso un numero arbitrario di volte: $g,\ gg,\ (gg)g, \ldots$ Risulta quindi utile introdurre una precisa definizione di "potenza di elementi in un gruppo". Sia $g^0 = e,\ g^n = g^{n-1}g$ per $n > 0$ e $g^n = (g^{-1})^{-n}$ per $n < 0$, questo per ogni $g \in G$.
Si ha quindi una mappa ben definita $f_g :\ \mathbb{Z} \to G$ data da $f_g(n) = g^n$.

Proposizione 1.14. *Sia G un gruppo e $g \in G$. La mappa*

$$f_g :\ \mathbb{Z} \to G$$

definita come segue: $f_g(n) = g^n$, è un morfismo di gruppo da $(\mathbb{Z}, +)$ in G.

L'immagine $f_g(\mathbb{Z}) = \{g^n \mid n \in \mathbb{Z}\}$ si indica con $\langle g \rangle$. Essa risulta essere un sottogruppo abeliano di G. Il numero di elementi di $\langle g \rangle$ è chiamato ORDINE di g; la notazione per tale quantità è $\mathrm{ord}(g)$. L'ordine di un elemento o è un numero naturale o è infinito. Nel caso in cui g abbia ordine finito, tale ordine è il più piccolo $n > 0$ tale che $g^n = e$.

Proposizione 1.15. *Sia G un gruppo finito*[1] *e sia g un suo elemento, allora le seguenti affermazioni sono vere.*

(i) L'ordine $\mathrm{ord}(g)$ *di g divide $|G|$*

(ii) $g^{|G|} = e$

(iii) Se $g^n = e$ *per qualche $n > 0$, allora* $\mathrm{ord}(g)$ *divide n*

Definizione 1.16. *Un* GRUPPO CICLICO *è un gruppo G contenente un elemento g tale che $G = \langle g \rangle$. L'elemento g in questione è chiamato* GENERATORE *di G; si dirà inoltre che G è generato da g.*

Seguono due esempi di gruppi ciclicli.

Esempio 1.17. Il gruppo $(\mathbb{Z}, +)$ è ciclico: tale gruppo può essere generato da 1 o da -1. In questo caso l'elemento neutro è lo zero e l'elemento g^n è solitamente indicato con ng.

Esempio 1.18. Il gruppo $(\{1, -1, i, -i\}, \cdot)$, dove $\{1, -1, i, -i\} \subseteq \mathbb{C}$ e l'operazione "$\cdot$" è la moltiplicazione, è ciclico. Esso può essere generato da i o da $-i$ infatti: $1 = i^0$, $i = i^1$, $-1 = i^2$, $-i = i^{-1}$.

Un gruppo ciclico $G = \langle g \rangle$ ha una struttura particolarmente "semplice", infatti esso è univocamente determinato (a meno di isomorfismi) dal suo *ordine*: $|G|$. Se $|G| = \infty$ allora $(G, \circ) \cong (\mathbb{Z}, +)$; se $|G| = n \geq 1$ allora $(G, \circ) \cong (\mathbb{Z}/\mathbb{Z}n, +)$.
Tutti gli elementi di un gruppo ciclico $G = \langle g \rangle$ possono essere espressi mediante potenze di g inoltre, considerando quanto appena detto, la moltiplicazione tra gli elementi del gruppo può essere descritta dalla seguente regola:

[1] Un gruppo G si dice finito se $|G| < \infty$.

$$g^m g^n = g^{m+n}$$

e dalla seguente relazione:

$$g^r = e \quad \text{dove} \quad r = \text{ord}(g) = |G|$$

Esempio 1.19. Sia b il generatore di un gruppo ciclico B, con $|B| = 12$. Allora:

$$b^5 b^9 = b^{14} = b^2 b^{12} = b^2 e = b^2$$

Inoltre dalla relazione $b^{12} = e$ si evince che gli elementi di B possono essere rappresentati nel modo seguente: $e, b, b^2, b^3, b^4, b^5, b^6, b^7, b^8, b^9, b^{10}, b^{11}$. Infatti poiché B è ciclico con generatore b ogni elemento del gruppo può essere espresso nella forma b^n con $n \in \mathbb{Z}$. Usando la divisione euclidea, $n = 12q + s$ dove $0 \leq s < 12$, dunque:

$$b^n = b^{12q+s} = (b^{12})^q b^s = e^q b^s = eb^s = b^s$$

1.2 Gruppi liberi

Si consideri un insieme I. Se $i \in I$ allora i simboli i e i^{-1} sono detti LITERALI. L'insieme di tutti i literali si indica con $I^{\pm 1} = I \cup I^{-1}$.

Definizione 1.20. *Una* PAROLA W *in* I *è una sequenza finita di literali:*

$$f_1, f_2, \ldots, f_{n-1}, f_n \tag{1.1}$$

La sequenza vuota, è detta PAROLA VUOTA *e si indica con 1. La* LUNGHEZZA $L(W)$ *di* W *è l'intero* n *e* $L(1) \triangleq 0$.

Una parola W si dice RIDOTTA (reduced) se o è la parola vuota, oppure non contiene nessuna sottoparola della forma ii^{-1} o $i^{-1}i$ per ogni $i \in I$. Nel caso non fossero chiaro, se si vogliono esibire i simboli coinvolti in W si scriverà $W(a, b, c, \ldots)$. Inoltre è abitudine scrivere la sequenza (1.1) senza le virgole separatrici:

$$f_1 f_2 \ldots f_{n-1} f_n \tag{1.2}$$

Un blocco di n occorrenze consecutive del medesimo carattere ("a" ad esempio), può essere abbreviato con la scrittura a^n; in maniera simile n occorrenze consecutive di a^{-1} si può indicare con a^{-n}.

Esempio 1.21. Sia $I = \{a, b, c, \ldots\}$. La parola $a^3 b^2 b^{-1} a^{-2} c^{-1}$ è identica a $aaabbb^{-1}a^{-1}a^{-1}c^{-1}$, ma è differente da $a^3 b a^{-2} c^{-1}$.

Così aa^{-1} è una parola che vanta lunghezza 2; $a^2ba^{-1}b^{-2}b$ è una parola in a e b di lunghezza 7; b^2c^{-2} è una parola in a, b e c di lunghezza 4, e anche una parola in b e c di lunghezza 4. In più 1 è una parola in a, b e c di lunghezza zero.

Definizione 1.22. *L'*INVERSO W^{-1} *di una parola W espressa come in* (1.2) *è la parola:*

$$f_n^{-1} f_{n-1}^{-1} \dots f_2^{-1} f_1^{-1} \tag{1.3}$$

dove se f_v è a o a^{-1}, con $a \in I$, allora f_v^{-1} è a^{-1} o a rispettivamente. L'inverso della parola vuota è se stessa.

Esempio 1.23. Sia $I = \{a, b, c, \dots\}$. $(aa^{-1})^{-1} = aa^{-1}$, $(aaba^{-1}b^{-1}b^{-1}b)^{-1} = b^{-1}bbab^{-1}a^{-1}a^{-1}$, $1^{-1} = 1$.

Chiaramente, $L(W) = L(W^{-1})$ e $(W^{-1})^{-1} = W$.

Proposizione 1.24. *Sia F_I l'insieme delle parole in I. Considerando l'operazione di concatenazione con la regola $i^{-1}i = ii^{-1} = 1$ $\forall i \in I$, F_I è gruppo.*

Dimostrazione. L'operazione di concatenazione è ovviamente associativa, la parola vuota è l'unico elemento neutro e ogni parola ammette inverso. □

Definizione 1.25. *Dato un insieme I il gruppo F_I è detto gruppo libero su I.*

Se, come di consueto, G è un gruppo e $S \subseteq G$ allora ogni parola W in S determina un unico elemento in G. Per convenzione la parola vuota identifica l'elemento neutro e di G. La mappa dal gruppo libero F_S in G così definita è un morfismo di gruppo.

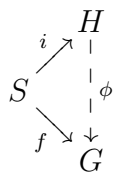
Definizione 1.26. *Un gruppo G è detto* GRUPPO LIBERO *se esiste un sottoinsieme $S \subseteq G$ per il quale la mappa $F_S \to G$ che associa ad ogni parola in S il corrispondente elemento di G è una biiezione.*

Si noti che, in questo caso, la biiezione tra F_S e G è un isomorfismo di gruppo e pertanto si dice che G è un gruppo libero su S.

1.2.1 Proprietà universale

I gruppi liberi godono della seguente *proprietà universale*:

Teorema 1.27. *Sia H un gruppo e $S \subseteq H$. Allora H è libero su S se e solo se la seguente proprietà universale è soddisfatta: per ogni mappa $f : S \to G$ dove G è un gruppo qualsiasi, $\exists!$ un morfismo $\phi : H \to G$ tale che $\phi(s) = f(s)$ $\forall s \in S$. In tal caso si ha $H \cong F_S$.*



Corollario 1.28. *Sia F un gruppo libero su S. Allora 1_F è l'unico morfismo che estende la mappa di inclusione: $i : S \to F$.*

Teorema 1.29. *Siano F_1 e F_2 gruppi liberi rispettivamente su S_1 e S_2. Allora F_1 e F_2 sono isomorfi se e solo se S_1 e S_2 hanno la stessa cardinalità.*

1.3 Presentazione di un gruppo

In matematica, una *presentazione di un gruppo* è una particolare definizione di tale entità, la quale fornisce una descrizione completa dei seguenti insiemi:

- i *generatori* del gruppo, ovvero degli elementi il cui prodotto combinato da origine a tutti gli elementi del gruppo;
- le *relazioni*, ovvero le eguaglianze tra le parole ridotte nell'insieme dei generatori;

Prima della definizione formale, si riporta come esempio una breve descrizione del gruppo di Klein. Esso infatti, si presta particolarmente bene, a introdurre i concetti di "relazione" e di "elementi generatori".

1.3.1 Il 4-gruppo di Klein

L'insieme di simboli $\{e_V, \beta, \delta, \gamma\}$ con la seguente tabella moltiplicativa:

	e_V	β	δ	γ
e_V	e_V	β	δ	γ
β	β	e_V	γ	δ
δ	δ	γ	e_V	β
γ	γ	δ	β	e_V

Tabella 1.1: Tabella moltiplicativa del gruppo di Klein.

è gruppo, solitamente indicato con V ("Vierer", quattro in tedesco) chiamato anche 4-gruppo di Klein in onore di Felix Klein.
Si fornisce una presentazione del gruppo V.
Ogni elemento di V è esprimibile come prodotto di β e δ: $e_V = \beta \circ \beta$ mentre $\gamma = \beta \circ \delta$.
Sia $S = \{b, d\}$, un insieme con due elementi. Si consideri il gruppo libero F_S sull' insieme S. La mappa $S \to V$, definita come segue: $b \mapsto \beta$ e $d \mapsto \delta$ si estende, per il teorema 1.27, a un unico morfismo $\psi : F_S \to V$. Essendo β e δ generatori, si tratta ovviamente di un morfismo suriettivo. Per il teorema d'isomorfismo 1.13, sia ha che V è isomorfo a $F_S/\ker(\psi)$. Gli elementi del nucleo di ψ sono i *relatori* di V. È facile verificare che b^2, d^2, $bdb^{-1}d^{-1}$ appartengono a $\ker(\psi)$. Ovviamente tale insieme contiene infiniti altri elementi. Da quanto appena detto segue che $b^2 = e_V, d^2 = e_V$ e $bd = db$ sono *relazioni* valide in V (si ricordi che, come è stato detto nel primo paragrafo di questa sezione, le relazioni sono eguaglianze tra le

parole ridotte nell'insieme dei generatori). Si può dimostrare che i tre elementi evidenziati: b^2, d^2, $bdb^{-1}d^{-1}$, sono generatori del nucleo di ψ; rimangono altre relazioni algebriche (che coinvolgono solamente l'operazione "composizione") soddisfatte da b e d? Chiaramente la risposta è affermativa, si pensi a: $b^3 = b$, $(bd)^2 = e_V$; tuttavia quest'ultime elencate posso essere derivate dalle tre nominate in precedenza; proseguendo con altri esempi: $bdbd = b(db)d = b(bd)b = bbdd$ segue da $bd = db$; inoltre $bbdd = b(bd)d = b^2d^2 = e_V \cdot e_V = e_V$ segue da $b^2 = e_V$ e da $d^2 = e_V$. Si può dimostrare che ogni relazione algebrica tra b e d può essere derivata dalle relazioni: $b^2 = e$, $d^2 = e$, $bd = db$ (e dalle proprietà generali che valgono in ogni gruppo, come $bb^{-1} = e_V$ e $b \cdot e_V = b$). Sia

$$b^{\kappa_1}d^{\mu_2}\dots b^{\kappa_r}d^{\mu_r} = b^{\omega_1}d^{\sigma_1}\dots b^{\omega_s}d^{\sigma_s} \tag{1.4}$$

una relazione algebrica soddisfatta in V, si ricorda che vale sempre $b = \beta$ e $d = \delta$.
Essa è equivalente a:

$$b^{\kappa_1}d^{\mu_2}\dots b^{\kappa_r}d^{\mu_r-\sigma_s}b^{-\omega_s}\dots d^{-\sigma_1}b^{-\omega_1} = e_V \tag{1.5}$$

Usando $b^2 = e_V$ e $b^2 = e_V$ si può facilmente ottenere $b = (b^2)b^{-1} = e_Vb^{-1} = b^{-1}$ e in maniera del tutto simile $d = d^{-1}$. In questo modo l'equazione (1.5) può riscritta cosicché tutti gli esponenti siano non negativi; inoltre usando $b^2 = e$, $d^2 = e$ essi possono essere ridotti a 0 o a 1. Dato che in ogni gruppo $b^0 = d^0 = e$, eventuali occorrenze di b o d con esponente zero possono essere cancellate. Ancora, usando sempre le tre relazioni "principali": $b^2 = e_V$, $d^2 = e_V$ e $bd = db$, l'equazione (1.5) può ulteriormente essere ridotta alla seguente forma:

$$b^{\epsilon_1}d^{\epsilon_2} = e_V \tag{1.6}$$

dove ϵ_1 e ϵ_2 posso essere 0 o 1. Nel caso ϵ_1 o ϵ_2 o entrambi, fossero pari a 1, la parte sinistra dell'equazione (1.6) diventerebbe b, d o bd e quindi rispettivamente β, δ o γ; tuttavia in V nessuno di essi è l'elemento neutro e_V. Perciò dev'essere: $\epsilon_1 = \epsilon_2 = 0$, dunque (1.6) si riduce a $e_V = e_V$. Ripercorrendo a ritroso quanto detto finora si può derivare l'equazione (1.4) da $b^2 = e_V$, $d^2 = e_V$ e $bd = db$.
Il 4-gruppo di Klein è caratterizzato allora da due elementi generatori b e d che soddisfano le seguenti relazioni: $b^2 = e_V$, $d^2 = e_V$ e $bd = db$, in più tutte le altre relazioni tra b e d possono essere derivate da queste tre appena elencate. Inoltre, ogni altro gruppo avente due generatori f e h che soddisfano le relazioni $f^2 = e$, $h^2 = e$ e $fh = hf$ e dove, ogni altra relazione tra f e h si può evincere dalle tre indicate precedentemente, è isomorfo al 4-gruppo di Klein.

1.3.2 Generatori e relatori

Nella sottosezione 1.3.1 appena conclusa, si è visto in modo più approfondito attraverso un esempio, quanto già detto nell'introduzione della sezione 1.3; ora si formalizza il concetto di presentazione di un gruppo.

Definizione 1.30. *Presentare un gruppo G significa dare un morfismo suriettivo ψ da un gruppo libero F_S in G esplicitando l'insieme S e una famiglia R di elementi di* $\ker(\psi)$ *tale che la chiusura normale del sottogruppo $\langle R\rangle$ generato da R coincida con* $\ker(\psi)$*. Gli elementi di S sono detti simboli generatori di G e gli elementi di R sono detti defining relators di G. La coppia $\langle S \mid R\rangle$ è detta una presentazione di G. Gli elementi del nucleo di ψ sono detti relatori e le espressioni del tipo: $l_1kl_2 = l_1l_2$ con $k \in \ker(\psi)$ e $l_1, l_2 \in F_S$ sono dette relazioni del gruppo G e, tra queste, quelle del tipo: $r = 1$ con $r \in R$ sono dette defining relations associate alla presentazione $\langle S \mid R\rangle$.*

Si osservi che se $\langle S \mid R\rangle$ è una presentazione di un gruppo G e, dati $a, b \in F_S$ il prodotto ab appartiene a R, la defining relation $ab = 1$ potrà essere nel seguito anche scritta nella forma equivalente $a = b^{-1}$.

Una presentazione, a volte, è data usando relazioni al posto di relatori, altre volte ancora è usata una presentazione ibrida dichiarando quindi sia relatori che relazioni:

$$\langle a, b, \dots \mid a^2 = 1, b^2 = 1, ab = ba\rangle,$$

$$\langle a, b, \dots \mid a^2, b^2, ab = ba\rangle,$$

$$\langle a, b, \dots \mid a^2 = 1, b^2 = 1, aba^{-1}b^{-1}\rangle$$

definiscono tutte il medesimo gruppo.

Esempio 1.31. Il gruppo ciclico B con $|B| = 12$, introdotto nell'esempio 1.19, può essere presentato fornendo come unico simbolo generatore b; e un'unica defining realtion: $b^{12} = 1$.

Esempio 1.32. Il 4-gruppo di Klein V, presentato nella sottosezione 1.3.1, può essere presentato mediante un insieme: $\{b, d\}$ di simboli generatori; e un insieme di defining relation: $\{b^2 = 1, d^2 = 1, db = db\}$.

Definizione 1.33. *Siano $P, Q, R, \dots$ alcuni relatori di G. Si dirà che la parola W è* DERIVABILE *da $P, Q, R, \dots$ se le seguenti operazioni, applicate un numero finito di volte, trasformano W nella parola vuota:*

(i) Inserimento di una delle seguenti parole $P, P^{-1}, Q, Q^{-1}, R, R^{-1}, \dots$ oppure di uno dei relatori banali tra due simboli consecutivi qualunque di W, nonché all'inizio di W o alla fine.

(ii) Cancellazione di una delle seguenti parole $P, P^{-1}, Q, Q^{-1}, R, R^{-1}, \dots$ oppure di uno dei relatori banali, nel caso formino un blocco di simboli consecutivi in W.

Questa definizione di derivabilità per i *relatori* induce una nozione di derivabilità anche per le *relazioni*. Formalmente, data l'equazione $W = V$ si dirà che è DERIVABILE DALLE RELAZIONI $P_1 = P_2, Q_1 = Q_2, R_1 = R_2, \dots$ se e solo se la parola WV^{-1} è derivabile a partire dai relatori $P_1P_2^{-1}, Q_1Q_2^{-1}, R_1R_2^{-1} \dots$

1.3.3 Esempi di presentazioni di gruppi

Nella tabella seguente sono riportate alcune presentazioni di gruppi di uso comune; molti di questi gruppi possiedono numerose altre possibili presentazioni che qui non sono riportate.

GRUPPO LIBERO SU S	
$\langle S\|\emptyset\rangle$	Un gruppo libero non è soggetto ad alcuna relazione tra i suoi elementi.
GRUPPO LIBERO ABELIANO SU S	
$\langle S\|R\rangle$	L'insieme R è l'insieme di tutti i commutatori di S.
C_n GRUPPO CICLICO DI ORDINE n	
$\langle a\|a^n\rangle$	Dove n è l'ordine del generatore a del gruppo.
GRUPPO DEI QUATERNIONI Q	
$\langle i,j\|i^4, i^2j^2, ijij^{-1}\rangle$	Il gruppo dei quaternioni è generato da due elementi.

Tabella 1.2: Presentazioni di gruppi di uso comune.

1.3.4 Presentazioni finite e presentazioni ricorsive

Una presentazione $\langle S|R\rangle$ di un gruppo G è *finitamente generata* se l'insieme S dei generatori è finito, *finitamente relazionata* se l'insieme R dei relatori è finito. Una presentazione è detta finita se sia S che R sono insiemi finiti. Un gruppo si dice essere finitamente generato/relazionato/presentato se ha una presentazione che risulta essere finitamente generata/finitamente relazionata/finita.
Se l'insieme delle relazioni è ricorsivamente enumerabile[2] , la presentazione è detta ricorsi-

[2]Un insieme $T \subseteq \mathbb{N}$, si dice ricorsivo se la sua *funzione caratteristica* $f : \mathbb{N} \to \{0,1\}$ con

$$f(t) = \begin{cases} 1 & \text{se } t \in T \\ 0 & \text{se } t \notin T \end{cases}$$

è **calcolabile**.

Un insieme $T \subseteq \mathbb{N}$, si dice ricorsivamente enumerabile se è *semidecidibile* cioè se la sua *funzione semicaratteristica*

$$f(t) = \begin{cases} 1 & \text{se } t \in T \\ n.d. & \text{se } t \notin T \end{cases}$$

è **calcolabile**.

Per quanto riguarda questa nota, il termine **calcolabile** va inteso come l'intuizione suggerisce. Il concetto verrà formalizzato nel capitolo 2.

va[3].

1.3.5 Proprietà delle presentazioni di gruppi

Per le presentazioni di un gruppo valgono le seguenti proprietà:

- ogni gruppo ha una presentazione in termini di generatori e relatori;
- ogni gruppo finito ha una presentazione ricorsiva, mentre non è vero l'inverso;
- ogni gruppo finito ha una presentazione finita;
- in generale esistono delle presentazione per le quali nessun algoritmo è in grado di decidere se una parola è pari all'elemento neutro del gruppo oppure no (*word problem*);

[3]La sovrapposizione dei termini è giustificata dal seguente fatto: data una presentazione con insieme delle relazioni ricorsivamente enumerabile è sempre possibile trovare una presentazione del medesimo gruppo per cui l'insieme delle relazioni è ricorsivo.

Capitolo 2

Macchine di Turing e indecidibilità

In questo capitolo si fornisce un primo approccio alla teoria della computabilità. Come modello di calcolo si utilizza la macchina di Turing che viene descritta in tutte le sue varianti. Vengono elencate inoltre, le varie classi di linguaggi: ricorsivi, ricorsivamente enumerabili, P e NP. Gran parte della trattazione segue [1];

2.1 Modelli di calcolo

2.1.1 Macchine di Turing deterministiche

Le macchine di Turing sono dei dispositivi di calcolo astratti, i quali non sono soggetti ad alcuna limitazione propria dei calcolatori fisicamente realizzati.
Si fornisce ora una presentazione discorsiva delle loro principali caratteristiche:

- non si pone un limite superiore alla capacità di memoria. Questa, in una macchina di Turing, è rappresentata da un *nastro* monodimensionale infinito, costituito da una sequenza lineare di *celle*, che viene considerata infinita in entrambe le direzioni. Ogni casella sul nastro o è vuota o contiene un singolo segno. Benché il nastro venga considerato infinitamente lungo, su di esso deve essere presente un numero finito di segni reali. Al di là di un certo punto in ogni direzione il nastro deve essere completamente vuoto.

- non si pone un limite alla durata dei processi di calcolo e si assume anche che una tale macchina funzioni sempre perfettamente. I processi di calcolo eseguibili da una macchina di Turing sono estremamente elementari: dotata di una *testina* di lettura/ scrittura che può osservare solo una casella per volta, essa può stampare un simbolo appartenente ad un alfabeto finito sulla cella osservata (eventualmente rimpiazzando il carattere già stampato in quest'ultima) o spostare la testina di lettura di una casella a sinistra o a destra rispetto a quella in cui si trova.

- i simboli che una macchina di Turing può utilizzare sono in numero finito, l'insieme di questi simboli costituisce l'*alfabeto* della stessa.

- un altro importante elemento delle macchine di Turing è lo stato interno. Si tratta della configurazione memorizzata dalla macchina calcolata in base agli ingressi (input) presentatisi in istanti precedenti (cioè in base a ciò che ha potuto leggere sul nastro) . Naturalmente, gli stati interni devono essere in un numero finito. Lo stato interno, e tutte le informazioni che permettono alla macchina di evolversi sono memorizzate nel suo centro di elaborazione detto: "*controllo finito*".

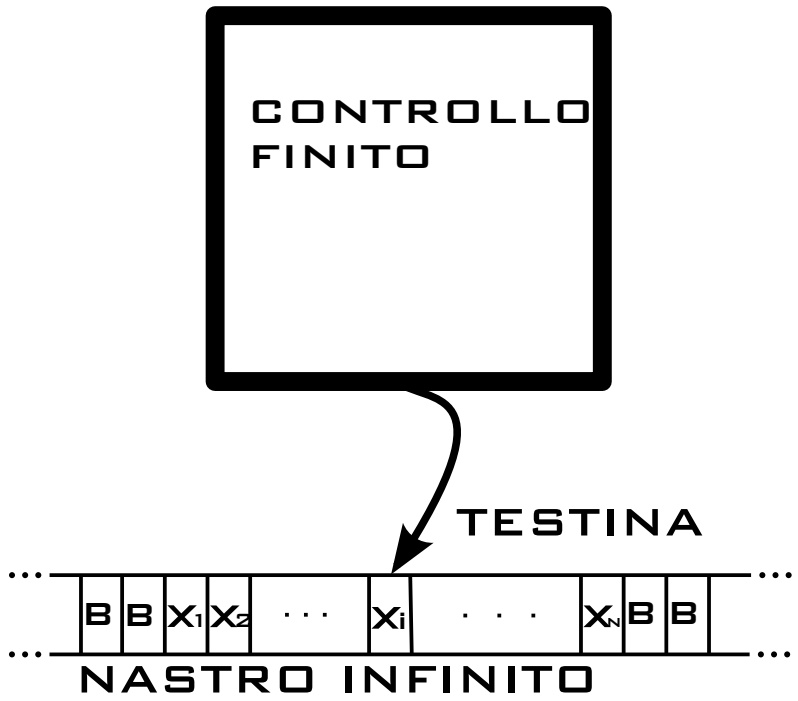


Figura 2.1: Una macchina di Turing.

L'*input*, una stringa[1] di lunghezza finita formata da blocchi scelti dell'*alfabeto di input*, viene inizialmente posta sul nastro. Tutte le altre celle, che si estendono senza limiti a sinistra e a destra, contengono all'inizio un carattere speciale detto *blank*. Il blank è un *simbolo di nastro* ma non di input; oltre ai simboli di input e al blank, ci possono anche essere altri simboli di nastro (che solitamente vengono utilizzati come "marcatori di nastro" si veda, ad esempio la macchina di Turing associata alla definizione 3.14 del capitolo 3). Una *testina* mobile di lettura/scrittura è collocata su una delle celle del nastro. Si dirà infatti che la macchina di Turing *visita* quella cella. All'inizio la testina si trova sulla cella più a sinistra tra quelle che contengono l'input. In una *mossa* la macchina di Turing compie tre azioni:

1. Cambia stato. Lo stato successivo può coincidere con lo stato corrente.

[1]Una stringa è una sequenza finita di elementi appartenenti ad un alfabeto Σ. L'insieme di tutte le stringhe in un alfabeto Σ viene indicato convenzionalmente con Σ^*. Quindi se definiamo Σ^k come tutte le stringhe di lunghezza k, con simboli tratti da Σ si ha che $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$, dove per definizione $\Sigma^0 = \epsilon$. L'insieme delle stringhe non vuote sull'alfabeto Σ si indica con Σ^+, quindi vale: $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

2. Scrive un simbolo di nastro nella cella che visita. Questo simbolo sostituisce quello che già si trovava in precedenza su di essa (può anche essere lo stesso).

3. Muove la testina verso destra o verso sinistra.

Si procede ora con la definizione formale di macchina di Turing.

Definizione 2.1. *Una* MACCHINA DI TURING *(TM, Turing Machine) è definita come la settupla*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

i cui componenti hanno i seguenti significati:

Q*: l'insieme finito degli* STATI *del controllo.*

Σ*: l'insieme finito dei* SIMBOLI DI INPUT.

Γ*: l'insieme completo dei* SIMBOLI DI NASTRO *(finito anch'esso);* Σ *è sempre un sottoinsieme di* Γ.

δ*: la* FUNZIONE DI TRANSIZIONE. *Gli argomenti di* $\delta(q, X)$ *sono uno stato* q *e un simbolo di nastro* X. *Il valore di* $\delta(q, X)$*, se definito, è una tripla* (p, Y, D)*, dove:*

1. p*: elemento di* Q*, è lo stato successivo.*
2. Y*: è il simbolo di* Γ *scritto nella cella visitata, al posto di qualunque simbolo vi fosse.*
3. D*: è una* DIREZIONE*,* L *o* R *(rispettivamente per "left", sinistra o "right", destra) e segnala la direzione in cui si muove la testina.*

q_0*: lo* STATO INIZIALE *del controllo; è un elemento di* Q.

B*: il simbolo detto* BLANK. *Si trova in* Γ *ma non in* Σ*, cioè non è un simbolo di input. Inizialmente compare in tutte le celle, tranne quelle (finite) che contengono i simboli di input.*

F*: l'insieme degli* STATI FINALI *o* ACCETTANTI*, un sottoinsieme di* Q.

2.1.2 Descrizioni istantanee delle macchine di Turing

Per descrivere formalmente come agisce una macchina di Turing si sviluppa una notazione per le configurazioni o *descrizioni istantanee* (ID, *Istantaneous Descriptions*). In una ID si mostrano solo le celle tra il simbolo diverso da blank più a sinistra e quello più a destra (rappresentare tutto il nastro sarebbe impossibile dato che la sua lunghezza è infinita per definizione). Si deve inoltre rappresentare il controllo e la posizione della testina. A tal scopo incorporiamo lo stato nel nastro e lo poniamo immediatamente a sinistra della cella visitata. Si userà dunque la stringa $X_1X_2 \cdots X_{i-1}qX_iX_{i+1} \cdots X_n$ per rappresentare una ID in cui

1. q è lo stato della macchina di Turing
2. la testina visita l'i-esimo simbolo da sinistra
3. $X_1X_2\cdots X_n$ è la porzione di nastro dal simbolo diverso dal blank più a sinistra e quello più a destra

Si possono descrivere le mosse di una macchina di Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ mediante la notazione $\vdash_M$. Quando la TM M è implicita si usa solo $\vdash$ per denotare una mossa, mentre $\overset{\star}{\vdash}$ indica zero, una o più mosse.

Esempio 2.2. Si supponga che per una macchina di Turing M valga $\delta(q, X_i) = (p, Y, L)$, cioè la prossima mossa è verso sinistra. Allora si può scrivere:

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \quad \vdash_M \quad X_1X_2\cdots X_{i-2}pX_{i-1}YX_{i+1}\cdots X_n$$

2.1.3 Il linguaggio di una macchina di Turing

In termini informali si descrive ora il modo in cui una macchina di Turing accetta un linguaggio. La stringa di input viene posta sul nastro e la testina parte dal simbolo di input più a sinistra. Se la TM entra in uno stato accettante, allora l'input è accettato, altrimenti no.
In termini formali, sia $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ una macchina di Turing. Allora $L(M)$ è l'insieme di stringhe $w \in \Sigma^*$ tale che $q_0w \overset{\star}{\vdash} \alpha p\beta$ per uno stato p in F e qualunque stringa di nastro α e β.
I linguaggi che si possono accettare usando una macchina di Turing sono denominati *linguaggi ricorsivamente enumerabili* o linguaggi RE (*Recursively Enumerable*).

Definizione 2.3. *Due macchine di Turing M e M' si dicono equivalenti se $L(M) = L(M')$.*

2.1.4 Macchina di Turing nondeterministiche

Una macchina di Turing *nondeterministica* (NTM, *Nondeterministic Turing Machine*) si distingue da quella deterministica nella funzione δ, che associa a ogni stato q e a ogni simbolo di nastro X un insieme di triple:

$$\delta(q, X) = \{(q_1, Y_1, D_1),\ (q_2, Y_2, D_2),\ \ldots,\ (q_k, Y_k, D_k)\}$$

dove k è un intero finito. A ogni passo una NTM può scegliere una delle triple come mossa.

Teorema 2.4. *Se M_N è una macchina di Turing nondeterministica, esiste una macchina di Turing deterministica M_D tale che $L(M_N) = L(M_D)$.*

In questa tesi, salvo esplicita indicazione, s'intenderanno sempre TM deterministiche.

2.1.5 Enumerazione delle stringhe binarie

Tornerà utile nei capitoli successivi l'esistenza di una corrispondenza biunivoca tra i numeri interi e le stringhe binarie. Se w è una stringa binaria, si considera $1w$ come un intero binario i. Si chiamerà dunque w la "i-esima stringa". Quindi ϵ è la prima stringa, 0 la seconda, 1 la terza 00 la quarta, 01 la quinta e così via. Con questo criterio le stringhe risultano ordinate per lunghezza, con le stringhe di uguale lunghezza ordinate lessicograficamente. Si userà d'ora in avanti riferirsi alla i-esima stringa come la stringa w_i.

2.1.6 Codici per le macchine di Turing

Ogni macchina di Turing, seguendo il procedimento che verrà presentato in questo paragrafo, può essere completamente descritta senza ambiguità mediante una stringa binaria; in questo modo ogni TM con alfabeto di input $\{0, 1\}$ potrà essere considerata come una stringa binaria. Dato che nella sottosezione precedente si è ideato un modo per enumerare le stringhe binarie si potrà parlare della "i-esima macchina di Turing M_i". Il primo passo da compiere è etichettare con dei numeri interi gli stati, i simboli e le direzioni. Una volta scelti gli interi che rappresentano tali entità, si può procedere a codificare la funzione di transizione δ. Si supponga che una regola di transizione sia $\delta(q_i, X_j) = (q_k, X_l, D_m{}^2)$, per certi valori interi i, j, k, l e m. Tale regola verrà codificata per mezzo della stringa $0^i10^j10^k10^l10^m$. Poiché i, j, k, l e m valgono ciascuno almeno 1 nel codice di una transizione non compaiono mai due o più 1 consecutivi.
Un codice per l'intera TM M consiste di tutti i codici delle transizioni in un ordine fissato, separati da coppie di 1:

$$C_1 11 C_2 11 \cdots C_{n-1} 11 C_n$$

dove ognuna delle C_y con $y = 1, \ldots, n$, è il codice di una transizione di M. Nelle sezioni successive verranno codificate coppie costituite da una TM e da una stringa (M, w), la stringa binaria per indicare tale coppia sarà il codice binario di M seguito da 111, seguito a sua volta da w. Poiché nessun codice valido di una TM contiene tre 1 in fila, si può essere certi che la prima occorrenza di 111 separa il codice di M da w.

2.2 Indecidibilità

2.2.1 Diverse classi di linguaggi

La capacità di calcolo della macchina di Turing crea una divisione tra i problemi[3]: quelli che possono essere risolti dalla macchina di Turing e quelli che non possono essere risolti

[2]Si farà riferimento alla direzione L come D_1 e alla direzione R come D_2.

[3]La sovrapposizione dei termini "linguaggio/problema" e "riconoscere/risolvere" è giustificabile: linguaggi e problemi sono in realtà la stessa cosa, scegliere l'uno o l'altro termine dipende dai punti di vista. Se ci si occupa di stringhe prese come tali allora si è portati a pensare all'insieme delle stesse come a un *linguaggio*. Quando si assegna una "semantica" alle stringhe e quindi ci si occupa di cosa viene rappresentato dalla stringa stessa allora si tende a considerare l'insieme delle stringhe un *problema*.

da tale modello di computazione. Si distinguono successivamente i problemi che possono essere risolti da una macchina di Turing in due classi: quelli per cui c'è un *algoritmo* (cioè una macchina di Turing che si arresta, a prescindere dal fatto di accettare o no il suo input) e quelli che vengono risolti solo da macchine di Turing che possono lavorare in eterno, cioè non terminare mai l'esecuzione, su input non accettati. La seconda forma di accettazione è problematica: per quanto a lungo la TM lavori, non è infatti dato sapere se l'input sarà accettato o no.

2.2.2 Linguaggi ricorsivi

Si ricorda che un linguaggio L è detto ricorsivamente enumerabile (RE) se $L = L(M)$ per una TM M.

Definizione 2.5. *Un linguaggio L si dice* RICORSIVO *se $L = L(M)$ per una macchina di Turing M tale che:*

1. *se $w \in L$ allora M accetta (e dunque si arresta[4])*
2. *se $w \notin L$ allora M si arresta pur non entrando in uno stato accettante*

Una TM di questo tipo corrisponde alla nozione informale di "algoritmo": una sequenza ben definita di passi che termina sempre e produce una risposta. Se si considera il linguaggio L come un "problema" allora L è *decidibile* se si tratta di un linguaggio ricorsivo, *indecidibile* se si tratta di un linguaggio non ricorsivo.

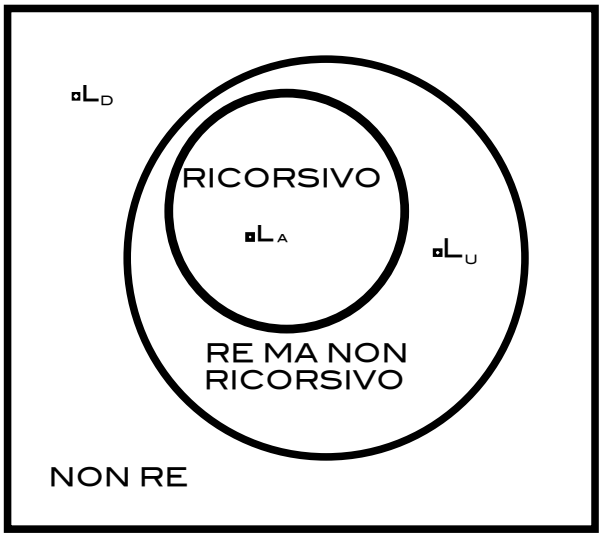


Figura 2.2: Suddivisione dei problemi.

Si fornisce ora un esempio di linguaggio ricorsivo.

[4]Una TM si arresta se entra in uno stato q visitando un simbolo di nastro X e non ci sono mosse possibili in questa situazione, cioè $\delta(q, X)$ è indefinito.

Esempio 2.6. Si costruisce una macchina di Turing che accetta il linguaggio $L_A = \{0^n1^n | n \geq 1\}$ e che si arresti in ogni caso, dimostrando così che L_A è un linguaggio ricorsivo. Inizialmente alla macchina viene data sul nastro una sequenza finita di 0 e 1, preceduta e seguita da blank. La TM cambia in modo alternato uno 0 in X e un 1 in Y finché tutti gli 0 e gli 1 sono abbinati. Più precisamente, partendo dall'estremità sinistra dell'input, cambia uno 0 in X e si muove verso destra su tutti gli 0 e Y finché non arriva a un 1. Cambia l'1 in Y e si muove verso sinistra, sulle Y e gli 0 finché non trova una X. A questo punto cerca uno 0 immediatamente a destra; se ne trova uno, lo cambia in X e ripete il processo, trasformando un 1 corrispondente in Y. Se l'input costituito dai caratteri diversi dal blank non si trova in $\{0^*1^*\}$, la TM prima o poi non ha una mossa successiva e termina senza accettare. Se invece conclude la trasformazione di tutti gli 0 in X nello stesso "giro" in cui cambia l'ultimo 1 in Y allora ha stabilito che l'input è della forma 0^n1^n e accetta (e termina). La specifica formale della TM M_A è:

$$M_A = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

dove δ è data dalla seguente tabella:

	Simbolo				
Stato	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Tabella 2.1: Una macchina di Turing che accetta L_A.

Mentre M svolge la sua computazione, la porzione del nastro su cui è passata la testina è sempre una sequenza di simboli descritta dall'espressione regolare $X^*0^*Y^*1^*$. Lo stato q_0 è lo stato iniziale; M rientra in q_0 ogni volta che ritorna allo 0 residuo più a sinistra. Se M si trova nello stato q_0 e visita uno 0, la regola in alto a sinistra della tabella 2.1 impone di andare nello stato q_1, trasformare lo 0 in X e muoversi verso destra. Nella stato q_1, M continua a muoversi verso destra su tutti gli 0 e le Y in cui si imbatte sul nastro, rimanendo in q_1. Se M legge una X o una B muore. Se invece legge un 1 quando si trova nello stato q_1, lo trasforma in Y, entra nello stato q_2 e comincia a muoversi verso sinistra. Nello stato q_2, M si muove verso sinistra sugli 0 e le Y, rimanendo in q_2. Quando raggiunge la X più a destra, che segna l'estremità destra del blocco 0 già trasformati in X, M ritorna nello stato q_0 e si muove verso destra. Si possono individuare due casi.

1. Se ora M vede uno 0, ripete il ciclo di abbinamento appena descritto.

2. Se M vede una Y, tutti gli 0 sono trasformati in X. Se tutti gli 1 sono stati trasformati in Y, allora l'input era di forma 0^n1^n e M deve accettare. Di conseguenza M entra

nello stato q_3 e comincia a muoversi verso destra, sulle Y. Se il primo simbolo diverso da una Y che M vede è un blank, allora il numero di 0 e di 1 era lo stesso, dunque M entra nello stato q_4 e accetta. D'altra parte, se M incontra un altro 1, allora ci sono troppi 1 e M muore senza accettare. M muore anche se incontra uno 0 (input nella forma sbagliata).

2.2.3 Proprietà linguaggi ricorsivi e RE

Un modo efficace per dimostrare che un linguaggio appartiene all'area delimitata dalle due circonferenze in figura 2.2 è lo studio del suo complemento. Infatti i linguaggi ricorsivi sono chiusi rispetto all'operazione di complemento. Perciò se un linguaggio L è RE, ma il suo complemento $\overline{L}$ non lo è, L non può essere ricorsivo. Infatti se L fosse ricorsivo, anche $\overline{L}$ sarebbe ricorsivo e dunque RE. Si dimostra che, delle nove possibilità di collocare un linguaggio L e il suo complemento $\overline{L}$ nel diagramma di figura 2.2, solo quattro sono consentite.

1. Sia L sia $\overline{L}$ sono ricorsivi, cioè si trovano entrambi nel cerchio più interno.
2. Nè L nè $\overline{L}$ sono RE, cioè sono entrambi al di fuori della circonferenza esterna.
3. L è RE ma non ricorsivo, e $\overline{L}$ non è RE; uno si trova nell'anello delimitato dalle due circonferenze, l'altro al di fuori della circonferenza esterna.
4. $\overline{L}$ è RE ma non ricorsivo, e L non è RE; il caso è analogo al punto (3), ma L e $\overline{L}$ sono scambiati.

I linguaggi ricorsivi sono inoltre chiusi rispetto all'operazione di unione e di intersezione, come dimostrano i seguenti due teoremi.

Teorema 2.7. *Siano L_1 e L_2 due linguaggi ricorsivi, allora anche il linguaggio $L_1 \cup L_2$ lo è.*

Dimostrazione. Per controllare se una stringa w appartiene all'unione dei due linguaggi ricorsivi L_1 e L_2 si costruisce una TM che copia il proprio input w in un secondo nastro ausiliario. Sul primo nastro simula la TM che accetta L_1 e sul secondo nastro quella che accetta L_2 (entrambe le computazioni terminano perché i due linguaggi sono ricorsivi). Se almeno una simulazione delle due macchine di Turing riconosce la stringa allora la si accetta; altrimenti la TM costruita si ferma senza accettare. □

Teorema 2.8. *Siano L_1 e L_2 due linguaggi ricorsivi, allora anche il linguaggio $L_1 \cap L_2$ lo è.*

Dimostrazione. Per controllare se una stringa w appartiene all'intersezione dei due linguaggi ricorsivi L_1 e L_2 si costruisce una TM che copia il proprio input w in un secondo nastro ausiliario. Sul primo nastro simula la TM che accetta L_1 e sul secondo nastro quella che accetta L_2 (entrambe le computazioni terminano perché i due linguaggi sono ricorsivi). Solo se entrambe le simulazioni delle due macchine di Turing riconoscono la stringa allora essa viene accettata; altrimenti la TM costruita si ferma senza accettare. □

2.2.4 Linguaggio di diagonalizzazione

In questa sottosezione verrà presentato un linguaggio che non può essere riconosciuto da nessuna macchina di Turing appartiene dunque alla classe non RE. Si ricorda inoltre che si dispone di una nozione concreta di M_i, la "i-esima macchina di Turing": la TM M il cui codice è w_i, l'i-esima stringa binaria.

Definizione 2.9. *Il linguaggio L_d, detto* LINGUAGGIO DI DIAGONALIZZAZIONE*, è l'insieme delle stringhe w_i tali che w_i non è in $L(M_i)$.*

In altre parole L_d consiste di tutte le stringhe w tali che la TM M con codice w non accetta quando riceve w come input.

Teorema 2.10. *L_d non è un linguaggio ricorsivamente enumerabile; quindi non esiste alcuna macchina di Turing che accetta L_d.*

Dimostrazione. Si supponga $L_d = L(M)$ per una TM M. Poichè L_d è un linguaggio sull'alfabeto $\{0, 1\}$, M rientra nell'elenco delle macchina di Turing che possono essere descritte mediante una stringa binaria. Di conseguenza esiste almeno un codice[5] per M, si ponga $M = M_i$.
Ora può essere che $w_i \in L_d$?

- Se $w_i \in L_d$, allora M_i accetta w_i. Pertanto, per la definizione di L_d, w_i non è in L_d, poichè L_d contiene solo le stringhe w_j tali che M_j non accetta w_j.
- Analogamente, se w_i non è in L_d, allora M_i non accetta w_i. Di conseguenza, per la definizione di L_d, w_i è in L_d.

Dato che w_i non può allo stesso tempo essere in L_d e non essere in L_d, si ha una contraddizione all'ipotesi che M esista. In altre parole L_d non è un linguaggio ricorsivamente enumerabile. □

2.2.5 Il linguaggio universale

Si definisce L_u, il *linguaggio universale* come l'insieme delle stringhe binarie che codificano una coppia (M, w), dove M è una macchina di Turing con alfabeto di input binario, e w è una stringa in $(0 + 1)^*$ tale che w sia in $L(M)$[6].

Teorema 2.11. *Il linguaggio L_u è RE.*

[5]Dato che per ogni TM M si possono assegnare interi ai suoi stati e ai suoi simboli di nastro in molti modi diversi, per una TM ci sarà più di una codifica. Si tratta comunque di un elemento irrilevante in quanto verrà dimostrato che nessuna codifica può rappresentare un TM M tale che $L(M) = L_d$.

[6]La scrittura $(0 + 1)^*$ indica tutte le stringhe di lunghezza arbitraria, purché finita, con caratteri appartenenti all'alfabeto $\{0, 1\}$.

Dimostrazione. Si procede con la descrizione di una macchina di Turing che accetta tutte e sole le stringhe appartenenti a L_u. Per semplicità la TM U che accetterà tale linguaggio sarà multinastro, nello spirito della figura 2.3:

- il primo nastro contiene l'input di U, ovvero il codice di M e la stringa w;
- il secondo nastro è usato per contenere il nastro simulato di M;
- il terzo nastro di U contiene lo stato di M;

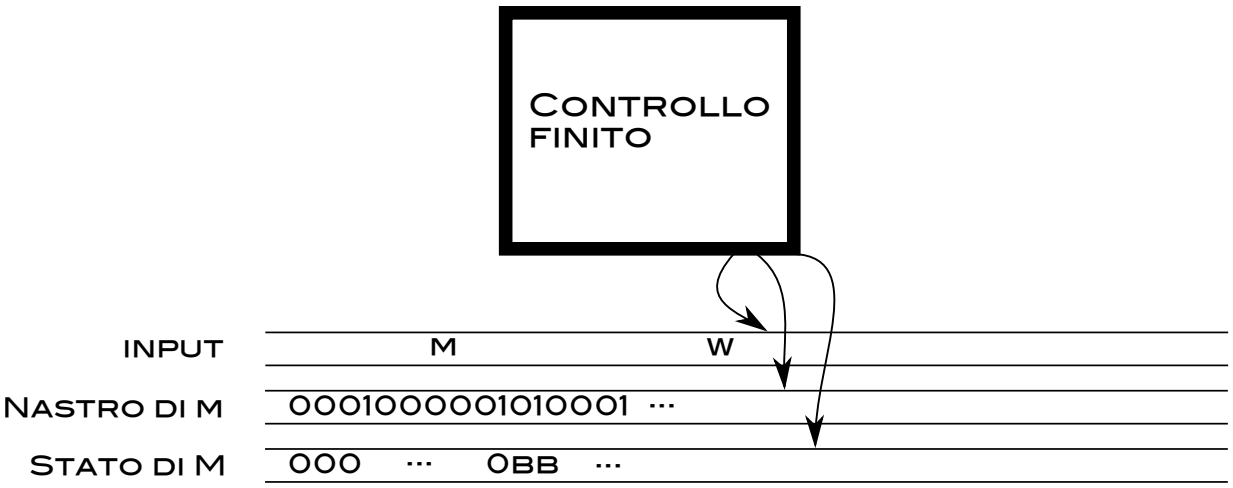


Figura 2.3: Struttura di una macchina di Turing universale.

Si descrive ora come opera U.

1. Esamina l'input per assicurarsi che il codice di M sia un codice legittimo per una TM. Se non è così allora U si arresta senza accettare. Poiché si presuppone che i codici non validi rappresentino la TM senza mosse, che non accetta nessun input, si tratta di un'azione corretta.

2. Dato che il secondo nastro serve a simulare il nastro di M, su di esso viene ricopiata la stringa w.

3. Scrive lo stato iniziale di M sul terzo nastro, e muove la testina del secondo nastro verso destra in corrispondenza del primo simbolo di w.

4. Per simulare una mossa di M, U percorre il primo nastro in corrispondenza di una transizione $\delta(q_i, X_j) = (q_k, X_h, D)$ tale che q_i sia lo stato attuale della macchina simulata (annotato sul terzo nastro), e X_j è il simbolo di nastro indicato dalla testina nel nastro 2. Questa è la transizione di M utilizzerebbe per determinare lo stato successivo. U deve fare tre cose:

(a) trasformare il contenuto del nastro 3 scrivendo lo stato q_k, ossia simulare il cambiamento di stato di M;

(b) sul nastro 2 la testina cambia il simbolo visitato X_j in X_h;

(c) muove la testina del nastro 2 in maniera opportuna, a seconda del valore di D;

5. se M non ha transizioni per lo stato simulato e il simbolo di nastro indicato, non ci sarà alcuna transizione in (4). M si arresta nella configurazione simulata e U fa altrettanto.

6. se M entra nel suo stato accettante, allora U accetta.

In questo modo U simula M su w. U accetta la coppia codificata (M, w) se e solo se M accetta w. □

2.2.6 Indecidibilità del linguaggio universale

Nella sottosezione 2.2.5 si è creata una TM che accetta L_u dunque esso è RE. Il seguente teorema dimostra che il linguaggio universale è RE ma non ricorsivo.

Teorema 2.12. *L_u è RE ma non ricorsivo.*

Dimostrazione. Si supponga (per assurdo) che L_u sia ricorsivo. Per quanto descritto del paragrafo 2.2.3 anche $\overline{L}_u$ dovrebbe esserlo. Se si avesse una TM M che accetta $\overline{L}_u$, si potrebbe costruire una TM che accetti L_d (come verrà mostrato tra poche righe). Poiché è stato dimostrato precedentemente che L_d non è RE, si ricava una contraddizione con l'ipotesi che L_u sia ricorsivo.
Si supponga dunque $L(M) = \overline{L}_u$. Come suggerito dalla figura 2.4 si può modificare la TM M in una TM M' che accetta L_d.

1. Data una stringa w in input, M' la trasforma in $w111w$ [7] . Se si desidera che M' sia a nastro singolo allora, per compiere la suddetta trasformazione, si potrebbe in primo luogo usare un secondo nastro ausiliario per copiare w (ciò consente di creare la stringa $w111w$ più facilmente) e successivamente effettuare la conversione della TM con due nastri in una a nastro singolo [8].

[7] Si ricorda quanto detto nel paragrafo 2.1.6, ovvero la stringa binaria per indicare la coppia (M, w) sarà il codice binario di M seguito da 111, seguito a sua volta da w. Poiché nessun codice valido di una TM contiene tre 1 in fila, si può essere certi che la prima occorrenza di 111 separa il codice di M da w.

[8] Si ricorda il fatto che le macchine di Turing multinastro e a nastro singolo sono esattamente equivalenti infatti si hanno i seguenti teoremi:

(a) Ogni linguaggio accettato da una TM multinastro è ricorsivamente enumerabile.

(b) Il tempo impiegato da un'opportuna macchina di Turing N mononastro per simulare n mosse di una macchina di Turing M con k nastri è $O(n^2)$.

2. M' simula M sul nuovo input. Se nella numerazione consueta w è w_i, allora M' determina se M_i accetta w_i. Poiche M accetta $\overline{L}_u$ accetterà se e solo se M_i non accetta w_i; quindi w_i è in L_d.

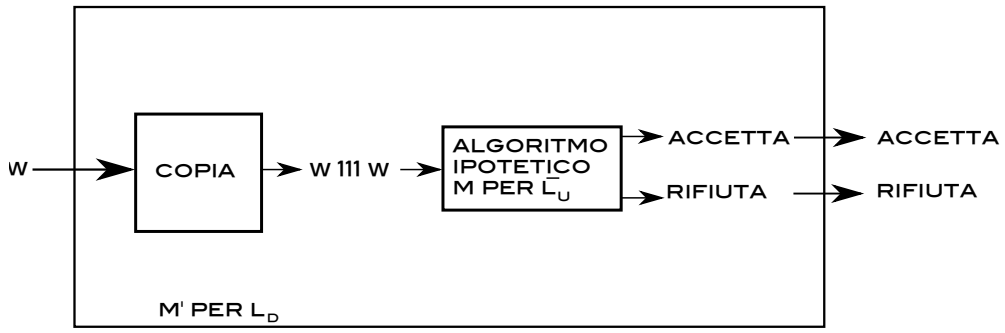


Figura 2.4: Riduzione di L_d a $\overline{L}_u$.

Di conseguenza M' accetta w se e solo se w è in L_d. Poiche è stato dimostrato che per il teorema 2.10 M' non può esistere. Si conclude quindi che L_u non è ricorsivo. □

2.3 Le macchine di Turing e i computer

In questa sezione si confronta la capacità di calcolo della macchina di Turing con quella di un computer ordinario. Per quanto possano sembrare modelli diversi essi riconoscono esattamente gli stessi linguaggi, ossia i linguaggi ricorsivamente enumerabili. Si può dunque affermare che:

1. Un computer può simulare una macchina di Turing.

2. Una macchina di Turing può simulare un computer in maniera efficiente.

2.3.1 Simulazione di una macchina di Turing da parte di un computer

É possibile simulare una TM per mezzo di un computer reale se si accetta il fatto che esiste una riserva potenzialmente infinita di dispositivi di memoria mobili, come i dischi, per simulare la porzione non bianca del nastro della TM. Per quanto riguarda il controllo finito della TM esso non causa problemi: esistono infatti solo un numero finito di stati e un numero finito di regole di transizione, il programma eseguito sul computer reale può codificare gli stati come stringhe di caratteri e usare una tabella di transizione che viene consultata per stabilire ciascuna mossa. Analogamente anche i simboli di nastro possono essere codificati come stringhe di caratteri di lunghezza finita, visto che sono presenti in numero finito.

2.3.2 Simulazione di un computer da parte di una macchina di Turing

Una TM può simulare la memoria e il controllo di un computer reale impiegando un nastro che memorizza tutte le locazioni e il loro contenuto: i registri, la memoria centrale e altri dispositivi di memoria. Di conseguenza si può essere certi che un compito non eseguibile da una TM non può essere eseguito neppure da un computer reale.

2.3.3 Confronto dei tempi di esecuzione dei computer e delle macchine di Turing

In questa sezione verrà preso in considerazione il tempo di esecuzione di una macchina di Turing che simula un computer. Si possono fare infatti tre importanti osservazioni:

- Le TM risultano utili non solo per stabilire cosa sia computabile e cosa no, ma anche per stabilire cosa sia risolvibile efficientemente e cosa richiederebbe un tempo di calcolo inaccettabile.
- In genere si ritiene che lo spartiacque tra i problemi *trattabili*, ossia quelli che possono essere risolti in tempi rapidi, e *intrattabili*, ossia quelli che possono essere risolti ma non abbastanza velocemente da rendere fruibile la soluzione, si collochi tra quanto può essere computato in tempo polinomiale nelle dimensione dell'input e quanto richiede un tempo di esecuzione superiore a qualsiasi polinomio.
- Si necessita di una garanzia che assicuri la risoluzione in tempo polinomiale di un problema da parte di un tipico computer se tale soluzione può essere fornita in tempo polinomiale da una macchina di Turing.

Si hanno dunque i seguenti teoremi:

Teorema 2.13. *Se un computer:*

1. *ha solo istruzioni che aumentano la lunghezza massima delle parole di non più di una unità.*
2. *ha solo istruzioni che una TM multinastro può eseguire su parole di lunghezza k in $O(k^2)$ passi o meno.*

allora un'opportuna macchina di Turing può simulare n passi del computer in $O(n^3)$ dei suoi passi.

Convertendo semplicemente la TM multinastro ad una mononastro e considerando i tempi di tale trasformazione si ha il seguente teorema:

Teorema 2.14. *Un computer del tipo descritto nel teorema 2.13 può essere simulato per n passi da una macchina di Turing a nastro unico impiegando $O(n^6)$ passi della stessa.*

2.4 Problemi P e NP

2.4.1 Problemi decisionali e problemi di ricerca

Sia Σ un alfabeto finito e Σ^* l'insieme di tutte le parole nell'alfabeto Σ[9]. Un problema decisionale definito su un sottoinsieme $L \subseteq \Sigma^*$ potrebbe essere espresso come:

> esiste un algoritmo tale che, per una parola $w \in \Sigma^*$, determina se $w \in L$ oppure $w \notin L$?

Se tale algoritmo esiste, allora esso si dice *algoritmo decisionale* per L e in questo caso il problema di decisione per L, come pure il linguaggio stesso, si dice *decidibile*. Viceversa se non ci sono algoritmi decisionali per tale problema esso si dice *indecidibile*. Si riporta ora la definizione formale:

Definizione 2.15. *Un* PROBLEMA DECISIONALE *è dato dalla coppia* $D = (L, \Sigma^*)$ *con* $L \subseteq \Sigma^*$. *Le parole in* Σ^* *si dicono* ISTANZE *del problema decisionale* D. *Dato un input* $x \in \Sigma^*$ *si deve dire se* $x \in L$ *o meno. L'insieme* L *è la parte* positiva *o la* "Yes-part" *del problema decisionale* D. *Il complemento:* $\overline{L}(D) = \Sigma^* \setminus L$ *è la parte* negativa *o la* "No-part" *di* D.

A volte i problemi decisionali occorrono in una forma più generale data dalla coppia $D = (L, U)$ dove $L \subseteq U$ e U è un sottoinsieme del prodotto cartesiano $\Sigma^* \times \cdots \times \Sigma^*$ di $k \geq 1$ copie di Σ^*.
Si passa ora ai problemi di ricerca, essi richiedono qualcosa di più di una semplice risposta binaria; un problema di questa classe è della seguente natura: data una proprietà P e la garanzia che esistano oggetti che possano vantare tale proprietà, trovare almeno un particolare elemento che goda di P. Formalmente:

Definizione 2.16. *Un* PROBLEMA COMPUTAZIONALE DI RICERCA *può essere descritto mediante un predicato binario* $R(x, y) \subseteq \Gamma^* \times \Sigma^*$, *dove* Γ *e* Σ *sono alfabeti finiti. In questo caso dato un input* $x \in \Gamma^*$ *si deve trovare un* $y \in \Sigma^*$ *tale che il predicato binario* $R = (x, y)$ *sia vero.*

In generale si possono distingue due differenti forme di problemi di ricerca:

- il primo richiede, dato un input $x \in \Gamma^*$, di dimostrare prima l'esistenza di un potenziale $y \in \Sigma^*$ tale che il predicato $R(x, y)$ sia vero; solamente dopo trovare un particolare elemento che lo soddisfi.
- nella seconda variante si assume in principio che dato l'input x, il corrispondente elemento y esista sempre; la questione dunque si restringe solamente a trovare y. In questo caso il problema può essere descritto mediante il predicato binario $R(x, y) \subseteq U \times \Sigma^*$, dove U è la "Yes-part" del problema decisionale: "$\exists y$ tale che $R(x, y)$ sia soddisfatto".

Molto spesso i problemi di ricerca vengono detti "problemi decisionali con testimone".

[9] A volte, ci si riferirà ai sottoinsiemi di Σ^*, con il termine "*linguaggi*".

2.4.2 Funzioni di complessità

In questa sezione si pone l'accento non su cosa sia calcolabile in termini assoluti e cosa no, ma solamente su quali questioni siano risolvibili in "poco" tempo. Quindi ci si restringe solamente ai problemi *decidibili* e si cerca di capire quali possono essere risolti da macchine di Turing che impiegano un tempo polinomiale nella dimensione dell'input.

Definizione 2.17. *Sia $f : \mathbb{N} \to \mathbb{N}$ una funzione. Si dice che f è una* FUNZIONE DI COMPLESSITÀ *ben definita se le seguenti condizioni sono soddisfatte:*

- *f è una funzione non decrescente: $\forall n \in \mathbb{N}\ f(n+1) \geq f(n)$*
- *esiste una TM multinastro M_f tale che per ogni n e per ogni input x di lunghezza n, M_f computa una stringa di $0^{f(|x|)}$ in tempo $T_{M_f} = O(n+f(n))$ e spazio $S_M = O(f(n))$*

La classe delle funzioni di complessità ben definite, è molto vasta. In particolare essa include tutte le funzioni polinomiali, le funzioni $\sqrt{n}$, $n!$ e $\log n$. Inoltre se due funzioni f e g appartengono a tale classe, allora pure le seguenti sono funzioni di complessità ben definite: $f + g$, $f \cdot g$ e 2^f.
Per una funzione di complessità ben definita f si definisce $\mathbf{TIME}(f)$ come la classe contenente tutti i problemi decisionali che una macchina di Turing deterministica risolve (e si arresta) dopo aver fatto al massimo $f(n)$ mosse[10] a fronte di un input di lunghezza n. Sia $\mathbf{NTIME}(f)$ la classe contenente tutti i problemi decisionali che una macchina di Turing nondeterministica risolve (e si arresta) dopo aver fatto al massimo $f(n)$ operazioni a fronte di un input di lunghezza n.

2.4.3 Problemi P

Nella teoria della complessità computazionale la classe P identifica l'insieme dei problemi risolvibili da una macchina di Turing deterministica in un numero polinomiale di mosse, rispetto alla dimensione dell'input. Formalmente:

$$P = \bigcup_{k=1}^{\infty} \mathbf{TIME}(n^k)$$

2.4.4 Problemi NP

Nella teoria della complessità computazionale la classe NP identifica l'insieme dei problemi risolvibili da una macchina di Turing nondeterministica in un numero polinomiale di mosse, rispetto alla dimensione dell'input. Formalmente:

$$NP = \bigcup_{k=1}^{\infty} \mathbf{NTIME}(n^k)$$

[10] A volte si potranno usare espressioni del tipo "in tempo $f(n)$", oppure "complessità in tempo $f(n)$".

2.4.5 Complessità della conversione DTM - NTM

Il teorema 2.4 afferma che se M_N è una macchina di Turing nondeterministica, esiste una macchina di Turing deterministica M_D tale che $L(M_N) = L(M_D)$. La seguente domanda sorge spontanea:

> La classe P e la classe NP sono veramente diverse?

Se si conoscesse una procedura che, in tempo polinomiale, riuscisse a convertire M_N in M_D con $L(M_N) = L(M_D)$ allora si avrebbe che $P = NP$. Naturalmente al momento non si sa se questa "traduzione veloce" dal nondeterminismo al determinismo sia possibile, tuttavia non è neppure stato dimostrato il contrario lasciando quindi la questione ancora aperta. Si procede ora con la descrizione di M_D e con l'analisi temporale della simulazione di M_N che essa compie; le righe seguenti forniscono inoltre una dimostrazione del teorema 2.4. Si costruirà M_D come una macchina multinastro:

- il primo nastro di M_D contiene una sequenza di ID di M_N che comprendono lo stato di M_N. Una ID di M_N è contrassegnata[11] come "corrente" nel senso che le ID successive sono in corso di elaborazione. Tutte le ID a sinistra della ID corrente sono già state elaborate e si possono ignorare.

- il secondo è un nastro ausiliario che semplifica le operazioni che M_D dovrà compiere.

Per elaborare la ID corrente M_D compie quattro operazioni:

1. Esamina lo stato e il simbolo visitato dalla ID corrente. Nel controllo di M_D sono incorporate le scelte di mossa di M_N per ogni stato e simbolo. Se lo stato nella ID corrente è accettante, M_D accetta e smette di simulare M_N.

2. Se lo stato è non accettante, e la combinazione stato-simbolo permette k mosse, M_D copia la ID sul secondo nastro e fa k copie della ID in coda alla sequenza di ID sul nastro 1.

3. M_D modifica ognuna delle k ID secondo una delle k scelte di mossa che M_N può fare dalla ID corrente.

4. M_D torna alla ID corrente contrassegnata in precedenza, cancella il contrassegno e lo sposta alla successiva ID a destra. Il ciclo ricomincia dal passo (1).

La fedeltà della simulazione dovrebbe risultare evidente: M_D accetta se e solo se scopre che M_N può raggiungere una iD accettante. Si deve però essere certi che se M_N entra in una ID accettante dopo una sequenza di n mosse, quella ID diventa prima o poi la ID corrente di M_D, che quindi accetta. Sia m il massimo numero di scelte di M_N nelle sue configurazioni. Allora in M_N ci sono una ID iniziale, non più di m ID raggiungibili in una

[11] Per contrassegnare una ID come corrente, dato che essa è memorizzata sul nastro, basta giustapporre un simbolo speciale come " $\star$ " o " $\diamond$ " alla codifica della ID.

mossa, non più di m^2 ID raggiungibili in due mosse e così via. Dopo n mosse, M_N può aver raggiunto al massimo $1 + m + m^2 + \cdots + m^n$ ID. Questo numero non supera nm^n. M_D esplora le ID nell'ordine detto "in ampiezza": prima tutte le ID raggiungibili in 0 mosse (la ID iniziale), poi quelle raggiungibili in una mossa, poi quelle in due mosse e così via. In particolare M_D fa diventare corrente ogni ID raggiungibile in non più di n mosse prima di elaborarne una raggiungibile in più di n mosse.
Di conseguenza la ID accettante di M_N viene considerata da M_D fra le prime nm^n. L'importante è che M_D elabori questa ID dopo un tempo finito; il limite calcolato garantisce che prima o poi quella ID sarà elaborata. Perciò se M_N accetta, accetta anche M_D. Poiché si è già fatto notare che M_D accetta solo se M_N accetta allora si conclude che $L(M_N) = L(M_D)$.
Si osservi che la TM deterministica così costruita può impiegare un tempo esponenzialmente più alto della TM nondeterministica per elaborare l'input, non si sa ancora se questo rallentamento è inevitabile. Quindi lo stesso input che viene accettato, impiegando un tempo polinomiale, da una una TM nondeterministica potrebbe essere accettato in tempo esponenziale da un macchina di Turing deterministica (o da un sistema di calcolo realizzato in pratica) che la simula.

2.4.6 Problemi FP e FNP

Le classi di problemi P e NP sono definite solo per problemi decisionali, questo crea qualche perplessità quando si sta cercando di classificare un problema che non prevede semplicemente una risposta binaria. Il problema della fattorizzazione di un intero n, per esempio, prevede come soluzione la sequenza di fattori primi che moltiplicati danno come risultato n appunto. Per questo motivo sono state definite le classi FP e FNP, esse non sono altro che le estensioni di P e NP rispettivamente, per i problemi di ricerca.
In teoria della complessità computazionale, la categoria FP rappresenta l'insieme dei problemi di ricerca (a volte chiamati anche "function problems" questo giustifica tale notazione) che possono essere risolti utilizzando una TM deterministica in tempo polinomiale sulla taglia dell'input, come già detto altro non è che la versione "per problemi di ricerca" della classe di complessità "per problemi decisionali" P. Detto ancora più semplicemente è l'insieme contenente tutte le funzioni che possono essere calcolate efficientemente da un algoritmo deterministico eseguito su un computer classico. Segue ora la definizione formale:

Definizione 2.18. *Un predicato binario $R(x, y) \subseteq \Gamma^* \times \Sigma^*$, dove Γ e Σ sono alfabeti finiti è in* FP *se e solo se esiste una TM deterministica la quale, fornito x sul nastro, calcola un y tale che $R(x, y)$ sia vero in tempo polinomiale (nella taglia di x). Se tale y non esiste essa si arresta (sempre in tempo polinomiale) con esito negativo.*

In teoria della complessità computazionale, la categoria FNP rappresenta l'insieme dei problemi di ricerca che possono essere risolti utilizzando una TM nondeterministica in tempo polinomiale sulla taglia dell'input. Esiste un linguaggio NP associato ad ogni relazione FNP, esso è detto problema decisionale *indotto dal* o *corrispondente al* predicato binario $R(x, y) \in FNP$. Esso è il linguaggio formato da tutte gli "x" che per un y fissato

rendono la relazione nominata poco fa vera. Segue ora la definizione formale della classa FNP:

Definizione 2.19. *Un predicato binario $R(x, y) \subseteq \Gamma^* \times \Sigma^*$, dove Γ e Σ sono alfabeti finiti è in* FNP *se e solo se esiste una TM nondeterministica la quale, fornito x sul nastro, calcola un y tale che $R(x, y)$ sia vero in tempo polinomiale (nella taglia di x). Se tale y non esiste essa si arresta (sempre in tempo polinomiale) con esito negativo.*

Si dimostra che $FP = FNP$ se e solo se $P = NP$.

2.4.7 La garanzia di sicurezza di RSA

In questa sottosezione si descrive il legame tra RSA e il problema della fattorizzazione di un intero. Prima di procedere, si riporta qui di seguito, lo schema di tale protocollo.
Ogni utente deve eseguire le seguenti operazioni:

1. generare due numeri primi p e q distinti e molto grandi ($\approx 10^{300}$)
2. calcolare $n = pq$
3. calcolare $\varphi(n)$[12] $= (p-1)(q-1)$
4. scegliere a caso un numero intero e tale che $mcd(e, \varphi(n)) = 1$ (ovvero siano coprimi)
5. determinare d con $mcd(d, \varphi(n)) = 1$, tale che $de \equiv 1 \pmod{\varphi(n)}$[13]

Per un generico utente X si ha che:

- chiave pubblica: $\mathfrak{K}_{E_X} = (n_X, e_X)$
- chiave privata: $\mathfrak{K}_{D_X} = (d_X)$
- le quantità p_X, q_X, $\varphi(n_X)$ devono essere mantenute segrete, altrimenti un intruso potrebbe violare RSA in tempo polinomiale

La sicurezza di questo crittosistema dipende in modo essenziale dalla difficoltà di scomporre n_X nei suoi fattori primi. Se un intruso fosse in grado di fattorizzarlo velocemente, allora sarebbe in grado di calcolare facilmente d_X e decifrare tutti i messaggi destinati all'utente X. Si osservi inoltre che la conoscenza di n_X, e_X e d_X consente di fattorizzare efficientemente n_X mediante il seguente algoritmo probabilistico. Sia $k = d_X e_X - 1 = l\varphi(n)$ con $l \in \mathbb{Z}$; ora $\varphi(n)$ è pari quindi lo è anche il suo multiplo k. Si può dunque esprime k nel

[12] Con la notazione $\mathbb{Z}/n$ si indica l'insieme $\{z \in \mathbb{Z}$ *tale che* $0 \leq z < n\}$. Mentre con $(\mathbb{Z}/n)^*$ si indica l'insieme $\{z \in \mathbb{Z}/n$ tali che $mcd(z, n) = 1\}$. La famosa "funzione-φ di Eulero" è così definita: $\varphi(n) = |(\mathbb{Z}/n)^*|$.

[13] Controllando, nel passo (4), che $\varphi(n)$ ed e siano coprimi, mediante l'algoritmo di Euclide esteso si ottiene anche l'inverso di e modulo $\varphi(n)$, quindi un numero intero d tale che $de \equiv 1 \pmod{\varphi(n)}$. Si ricorda inoltre che l'algoritmo di Euclide esteso ha complessità computazionale polinomiale (quadratica).

modo seguente: $k = 2^t r$ dove r non è un multiplo di 2. Per il teorema di Eulero-Fermat[14] si ha che $a^k \equiv 1 \pmod{n_X}\ \forall\ a \in (\mathbb{Z}/n_X)^*$. Quindi $a^{k/2}$ e una radice di 1 modulo n_X. Dato che n_X è prodotto di due primi distinti utilizzando il teorema cinese del resto si può facilmente dimostrare che possono esserci solo quattro radici dell'unità in $(\mathbb{Z}/n_X)$, due radici sono banali, le altre due non banali: $\{1, -1, s_1, s_2\}$ (si ribadisce il fatto che questi quattro numeri sono intesi "modulo n_X"). Sia ora $x \in \{1, -1, s_1, s_2\}$ si controlla se uno dei seguenti risultati $mcd(x-1, n_X)$ o $mcd(x+1, n_X)$, produce un fattore non banale di n_X. Nel caso non dia il risultato sperato si può fare la seguente supposizione: se anche $a^{k/2}$ fosse pari a 1 modulo n_X allora $a^{k/4}$ sarebbe una radice di 1 modulo n_X, e si ripete quanto detto prima.
Riscrivendo la procedura del precedente capoverso in maniera più schematica si ottiene il seguente algoritmo:

Algoritmo 2.1 Algoritmo probabilistico per fattorizzare n_X conoscendo e_X e d_X

FattProb(n_X, e_X, d_X)
1 sia k una variabile intera
2 $k \leftarrow d_X e_X - 1$
3 esprime k come $2^t r$ con r dispari
4 si generi un numero casuale a coprimo con n_X
5 generare la lista $\mathcal{L} = \{a^{k/2}, a^{k/4}, \cdots, a^{k/2^t}\}$
6 **for each** $x \in \mathcal{L}$ **do**
7 **if** ($(1 < mcd(x-1, n_X) < n_X)$) **then**
8 **return** $mcd(x-1, n_X)$
9 **if** ($(1 < mcd(x+1, n_X) < n_X)$) **then**
10 **return** $mcd(x+1, n_X)$
11 **return** "nessun fattore trovato, provare con un a differente"

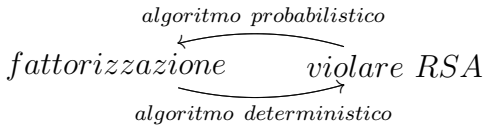
Si dimostra che operando una scelta casuale di $a \in (\mathbb{Z}/n_X)^*$ la probabilità che uno degli elementi $\{a^{k/2}, a^{k/4}, \cdots, a^{k/2^t}\}$ fornisca una radice non banale è maggiore o uguale a $1/2$. Ripetendo u volte la scelta casuale di a si ottiene che un algoritmo di fattorizzazione che termina con successo con probabilità maggiore o uguale a $1 - 2^{-u}$. Tutte le potenze modulari $\{a^{k/2}, a^{k/4}, \cdots, a^{k/2^t}\}$ possono essere calcolate in $O((\log n_X)^3)$ operazioni elementari[15],

[14] Il teorema di Eulero-Fermat afferma che, dati un numero intero a e un numero naturale n coprimi tra loro, allora

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

[15] Con "operazione elementare" si intende la somma di tre bit.

quindi u iterazioni comportano $O(u(\log n_X)^3)$ operazioni elementari.



2.4.8 La fattorizzazione di un intero

La sicurezza del sistema di cifratura RSA dipende dalla mancanza di un metodo generale, per scomporre in fattori primi un numero intero, avente complessità temporale polinomiale rispetto al numero di cifre dell'input.
Nelle righe che seguono si individuerà la classe di complessità a cui il problema della fattorizzazione appartiene, in modo tale da avere una "misura" precisa della difficoltà di tale questione e di conseguenza saggiare la bontà del crittosistema RSA. Per incasellare correttamente il quesito è necessario distinguere tra la versione decisionale e quella di ricerca dello stesso:

- *problema di ricerca*: dato un intero N, trovare un intero d nell'intervallo $[2, N-1]$ che divide N (oppure concludere che N è primo). Questo problema appartiene alla classe FNP e attualmente non è stata fornita alcuna dimostrazione che conferma o smentisce la sua appartenenza alla classe FP. Moltissimi algoritmi sono stati ideati per risolvere questa versione "di ricerca", i migliori hanno raggiunto complessità subesponenziale[16]: per citarne uno, si stima che la complessità del crivello quadratico di Pomerance abbia complessità inferiore a $exp((1+o(1))(\log N \log\log N)^{\frac{1}{2}})$ [4].

- *problema decisionale*: dato un intero N e un intero M con $1 \leq M \leq N$ dire se esiste un fattore di N nell'intervallo $[2, M-1]$. Come appare subito evidente è la naturale traduzione della versione precedente in "ambiente decisionale"; combinato con la ricerca binaria può risolvere il problema di ricerca associato con un numero di iterazioni logaritmico.

Non è chiaro al momento quale classe di complessità contenga la scrittura decisionale del problema. Esso, è stato dimostrato, appartiene sia a NP che a $co-NP$[17]. Inoltre, siccome il teorema fondamentale dell'aritmetica afferma che ogni numero intero ammette un'unica scomposizione in fattori primi, il problema appartiene sia alla classe UP che a $co-UP$.[18]

Va menzionato inoltre un altro problema decisionale associato a quello appena descritto: "N è un numero primo?" (o, in maniera equivalente, "N è un numero composto?").

[16] Si dirà che un algoritmo ha complessità subesponenziale se per ogni $\epsilon > 0$ il suo costo è $O(exp(\epsilon \log n))$, dove n è naturalmente il valore dell'input.

[17] Un linguaggio è in $co-NP$ se il suo complemento è in NP.

[18] In teoria della complessità UP (Unambiguous non-deterministic Polynomial-time) è la classe che contiene tutti e soli i problemi decisionali risolvibili da una TM nondeterministica con al massimo un "certificato di accettazione" per ogni input.

Nel 2004 Manindra AGRAWAL, Neeraj KAYAL e Nitin SAXENA dell' "*Indian Institute of Technology Kanpur*" hanno dimostrato che la verifica di primalità appartiene alla classe P. Si fornisce ora un percorso che, a partire dal teorema che ha costituito l'ispirazione per i tre ricercatori indiani, guida il lettore fino allo schema dell'algoritmo AKS (acronimo dei tre cognomi degli inventori).

Teorema 2.20. *La congruenza*

$$(x+b)^n \equiv x^n + b \pmod{n}$$

vale per tutti i $b \in \mathbb{Z}$ *se e solo se* $n \geq 2$ *è primo.*

Dimostrazione. "⇒" Per ipotesi si ha dunque che $n \geq 2$ è primo. Per prima cosa si deve dimostrare che se n è un numero primo allora n divide $\binom{n}{k}$ per $k = 1, \ldots, n-1$.
Si ha che $\binom{n}{k} \overset{\text{def}}{=} \frac{n!}{k!(n-k)!} = \frac{n(n-1)!}{k!(n-k)!}$, siccome è un coefficiente binomiale $\frac{n(n-1)!}{k!(n-k)!} \in \mathbb{Z}$. Poiché $mcd(n, k!(n-k)!) = 1$ allora deve essere che $k!(n-k)!$ divide $(n-1)!$.
Quindi n divide $\binom{n}{k}$.
Detto questo si nota subito che nella parte sinistra della congruenza[19] sopravvivono solo i termini $x^n + b^n$. Per il teorema di Eulero-Fermat si ha che $x^n + b^n \equiv x^n + b \pmod{n}$.
"⇐" Si ipotizza, per assurdo, n composto. Se così fosse esisterebbe un numero primo p tale che p divide n con $1 < p < n$. Sia α la massima potenza di p che divide n, ovvero p^α divide n ma $p^{\alpha+1}$ non divide n. Viene riportata ora la parte di sinistra della congruenza, espandendola secondo la formula di Newton:

$$x^n + \binom{n}{1}x^{n-1}b + \binom{n}{2}x^{n-2}b^2 + \cdots + \binom{n}{p}x^{n-p}b^p + \cdots \binom{n}{n-2}x^2b^{n-2} + \binom{n}{n-1}xb^{n-1} + b^n$$

Si focalizzi l'attenzione sul termine

$$\binom{n}{p} = \frac{n \overbrace{(n-1)\cdots(n-p+1)}^{\text{nessuno, tra questi fattori, è diviso da } p}}{p(p-1)\cdots 1}$$

Questo suggerisce una semplificazione tra il "p" a sinistra, al denominatore e l' "n" a sinistra, al numeratore. Se, come dovrebbe essere, $\binom{n}{p} \equiv 0 \pmod{n}$ allora dato che p^α divide n, si avrà anche $\binom{n}{p} \equiv 0 \pmod{p^\alpha}$ il che è impossibile perché proprio a causa della semplificazione messa in evidenza poche righe più, la massima potenza di p che divide $\binom{n}{p}$ è $p^{\alpha-1}$.
Da ciò segue che $\binom{n}{p} \not\equiv 0 \pmod{n}$, e si ha una contraddizione dell'ipotesi, dunque n è primo. □

[19] In particolare si pensi al suo sviluppo secondo formula di Newton per la potenza di un binomio: $(x+b)^n = \sum_{j=0}^{n} \binom{n}{j} b^j x^{n-j}$.

Naturalmente questo teorema non è di alcuna utilità pratica come test di primalità: si dovrebbe ripetere il test $\forall\ b \in \mathbb{Z}$, in realtà dato che si lavora modulo n ci si può limitare a $\mathbb{Z}/n$. Tuttavia anche con questo accorgimento restano comunque da valutare n congruenze (in ognuna delle quali si devono calcolare 3 elevamenti a potenza), il costo di tale procedimento appare subito estremamente oneroso, impossibile per un impegno reale. Il teorema di Agrawal, Kayal, Saxena perfeziona questo teorema, limitando il grado dei polinomi che verranno valutati nelle varie congruenze e limitando pure il numero dei "b" che devono essere testati; tale limite risulterà polinomiale in n.

Teorema 2.21 (Agrawal, Kayal, Saxena)**.** *Sia $n \geq 4$ un numero intero e sia $r < n$ un numero positivo tale che n abbia ordine $d > \log_2^2 n$ in $(\mathbb{Z}/r)^*$. Allora n è primo se e solo se*

(i) n non è una potenza perfetta;

(ii) n non ha fattori primi $\leq r$;

(iii) $(x+b)^n \equiv x^n + b \pmod{x^r - 1, n}$ [20] per ogni b intero, $1 \leq b \leq \sqrt{r}\log_2 n$;

Una possibile schematizzazione dell'algoritmo di Agrawal-Kayal-Saxena è:

Algoritmo 2.2 Algoritmo di Agrawal-Kayal-Saxena come test di primalità

AKS(n)
1 **if** ($n = \alpha^\beta$ per $\alpha, \beta \in \mathbb{N}$) **then**
2 **return** Il numero n è COMPOSTO.
3 determinare il minimo r per cui n abbia ordine $d > \lceil \log_2^2 n \rceil$ in $(\mathbb{Z}/r)^*$
4 **if** (1 <mcd(b,n)< n per qualche $b \leq r$) **then**
5 **return** Il numero n è COMPOSTO.
6 **if** ($n \leq r$) **then**
7 **return** Il numero n è PRIMO.
8 **for each** b intero, $1 \leq b \leq \sqrt{r}\log_2 n$ **do**
9 **if** $((x+b)^n \not\equiv x^n + b \pmod{x^r - 1, n})$ **then**
10 **return** Il numero n è COMPOSTO.
11 **return** Il numero n è PRIMO

[20] Tale scrittura sta a significare che i polinomi protagonisti della congruenza sono elementi di $\frac{F_n[X]}{\langle x^r - 1\rangle}$. In maniera molto informale significa che, preso un polinomio qualsiasi p(x), è ammessa la seguente scrittura: $p(x) = q(x)(x^r - 1) + m(x)$ dove $m(x)$ ha grado strettamente minore di r; inoltre si può affermare che $p(x) \equiv m(x) \pmod{x^r - 1}$. Non è tutto, anche i coefficienti del polinomio sono ridotti modulo n: $m(x) = \sum_{j=0}^{r-1} m_j x^j \equiv \sum_{j=0}^{r-1} (m_j \bmod n) x^j \pmod{n}$. Ecco quindi che è stato ridotto sia il grado dei polinomi che si devono manipolare, sia i b da testare, rendendo quindi utilizzabile questo teorema come tets di primalità in pratica.

L'algorimo AKS è corretto e polinomiale[21]. Con un'analisi poco approfondita si ottiene una complessità computazionale $O(\log^{\frac{33}{2}} n)$, tuttavia Lenstra e Pomerance hanno elaborato una variante di AKS avente complessità $O(\log^{6+\epsilon} n)$.

Quest'ultima sottosezione, riassumendo, garantisce che RSA sia un sistema di cifratura sicuro: il problema della fattorizzazione di un intero (versione "di ricerca") ovvero la "botola" del crittosistema nominato è risolvibile in tempo polinomiale solo da una TM nondeterministica. Se si volesse implementare tale procedura in un computer classico (che abbiamo visto essere equivalente, a meno di un ritardo di tempo polinomiale, ad una TM deterministica) l'esecuzione subirà un ritardo di tempo esponenziale.

[21]Si dimostra abbastanza facilmente che tutte le operazioni coinvolte nell'algoritmo descritto si possono eseguire in tempo polinomiale nella lunghezza del'input. La parte più difficile sta nel garantite che r si riesca a calcolare facilmente, è stato comunque dimostrato che esiste almeno un $r \leq \lceil \log_2^5 n \rceil$ tale che n abbia ordine $d > \log_2^2 n$ in $(\mathbb{Z}/r)^*$.

Capitolo 3

Il Word-Problem

In questo capitolo dopo un elenco dei problemi più studiati della teoria combinatoria dei gruppi ci si concentra su uno di essi: il *word problem*. Questo problema costituisce la "botola" del sistema di cifratura di Shpilrain e Zapata: un intruso deve essere in grado di risolverlo per violare il crittosistema.

3.1 Problemi definiti sui gruppi

La teoria combinatoria dei gruppi ha saputo definire molti problemi di ricerca e di decisione su di essi. Tali problemi risultano risolvibili o meno a seconda del gruppo preso in considerazione.

3.1.1 Il problema del coniugato

Versione problema decisionale:

> *Problema del coniugato* (conjugacy problem): data una presentazione ricorsiva di un gruppo G e due elementi $g, h \in G$, dire se esiste o meno un elemento $x \in G$ tale che $x^{-1}gx = h$.

Come tutti i problemi decisionali esso è composto dalla "Yes-part" (coppie di elementi in cui il primo è il coniugato del secondo) e dalla "No-part" (coppie di elementi in cui le due parti non hanno tale relazione). Si noti che la "Yes-part" è sempre ricorsiva in quanto, dato un elemento, l'insieme di tutti i suoi coniugati è ricorsivamente enumerabile.

Versione problema di ricerca:

> *Problema di ricerca del coniugato*: data una presentazione ricorsiva di un gruppo G e due elementi coniugati $g, h \in G$, trovare quel particolare elemento $x \in G$ tale che $x^{-1}gx = h$.

Il problema di ricerca del coniugato ha sempre una soluzione ricorsiva in quanto, come già detto l'insieme dei coniugati di un elemento fissato è ricorsivamente enumerabile. Ecco quindi che una possibile soluzione potrebbe essere quella di generare tutti i coniugati di g fino a trovare la soluzione del problema. Naturalmente questo tipo di soluzione è estremamente inefficiente.

3.1.2 Il problema della scomposizione e della fattorizzazione

Segue ora una variante del problema del coniugato.

Versione problema decisionale:

> *Problema della scomposizione* (decomposition problem): data una presentazione ricorsiva di un gruppo G, due sottogruppi $A, B \leq G$, e due elementi $g, h \in G$ dire se esistono due elementi $x \in A$ e $y \in B$ tali che $x \cdot g \cdot y = h$.

Versione problema di ricerca:

> *Problema di ricerca della scomposizione*: data una presentazione ricorsiva di un gruppo G, due sottogruppi $A, B \leq G$, e due elementi $g, h \in G$ trovare due elementi $x \in A$ e $y \in B$ tali che $x \cdot g \cdot y = h$.

Innanzitutto si noti che ponendo $x = e_G$ e $y = g^{-1}h$ l'equazione $x \cdot g \cdot y = h$ è verificata, tuttavia non è detto che $y \in B$. Nel caso in cui $A = B$ allora ci si riferirà a questo problema come il *double coset problem*.

Un caso particolare del problema della scomposizione si ottiene ponendo $g = e_G$.

Versione problema decisionale:

> *Problema della fattorizzazione* (factorization problem): data una presentazione ricorsiva di un gruppo G, due sottogruppi $A, B \leq G$ e un elemento $w \in G$ dire se esistono o meno due elementi $a \in A$ e $b \in B$ tali che $a \cdot b = w$.

Versione problema di ricerca:

> *Problema di ricerca della fattorizzazione*: data una presentazione ricorsiva di un gruppo G, due sottogruppi $A, B \leq G$ e un elemento $w \in G$ esibire due elementi: $a \in A$ e $b \in B$ tali che $a \cdot b = w$.

3.1.3 Il problema dell'appartenenza

Versione problema decisionale:

> *Problema dell'appartenenza* (membership problem): data una presentazione ricorsiva di un gruppo G, un sottogruppo $H \leq G$ generato da $h_1, \ldots, h_k$, e un elemento $g \in G$, dire se $g \in H$ oppure no.

Il problema dell'appartenenza è composto dalla "Yes-part" e dalla "No-part"; ancora una volta la "Yes-part" è sempre ricorsiva in quanto l'insieme di tutti gli elementi esprimibili attraverso un prodotto degli elementi $h_1, \ldots, h_k$ (con numero di fattori finito) è sempre ricorsivamente enumerabile.

Versione problema di ricerca:

> Il *problema di ricerca dell'appartenenza*: data una presentazione ricorsiva di un gruppo G, un sottogruppo $H \leq G$ generato da $h_1, \ldots, h_k$, e un elemento $g \in G$, esprimere g in un prodotto con fattori in $\{h_1, \ldots, h_k\}$.

3.1.4 Il problema dell'isomorfismo

Versione problema decisionale:

> *Problema dell'isomorfismo* (isomorphism problem): date due presentazioni finite di due gruppi G_1 e G_2, dire se essi sono, o meno, isomorfi.

Le trasformazioni di Tietze (descritte in 4.1.4) sono un metodo per, a partire da un gruppo finitamente presentato, generare tutti i gruppi a esso isomorfi. Questo fatto implica che la "Yes-part" è ricorsiva.

Versione problema di ricerca:

> *Problema di ricerca dell'isomorfismo*: date due presentazioni finite di due gruppi ismorfi G_1 e G_2, calcolare la sequenza di trasformazioni di Tietze che permette di passare dalla prima presentazione alla seconda.

3.2 Word-Problem

Il word problem verrà descritto più approfonditamente rispetto ai problemi nominati in precedenza. In particolare viene fornita la dimostrazione completa del teorema di Novikov-Boone-Britton che garantisce l'esistenza di gruppi in cui il word problem è non risolvibile.

3.2.1 Definizione del problema

Versione problema decisionale:

> *Word problem* (problema "della parola" o "dell'unità"): data una presentazione ricorsiva di un gruppo G e un elemento $g \in G$ dire se $g = e$ in G.

Dato che la presentazione del gruppo è ricorsiva si ha la seguente proposizione:

Proposizione 3.1. *Sia $\langle X, R\rangle$ una presentazione ricorsiva di un gruppo G. Allora l'insieme di tutte le parole $w \in X^{*1}$ tali che $w = e$ in G è ricorsivamente enumerabile.*

Naturalmente esiste anche una versione di ricerca del problema nominato:

> *Word search problem* (WSP): data una presentazione ricorsiva di un gruppo $G = \langle X \mid R\rangle$ e un elemento $g = e_G \in G$ esprimere g mediante prodotto di relatori in R, dei loro inversi o dei loro coniugati, ovvero:
>
> $$u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_t r_t^{\epsilon_t} u_t^{-1} = g \quad \text{dove } u_i \in X^*, r_i \in R \text{ e } \epsilon_i \in \{\pm 1\}$$

Si mette in risalto il fatto che il word search problem ha sempre una soluzione ricorsiva dato che è possibile enumerare tutti i vari prodotti tra relatori, i loro inversi e i loro coniugati: idealmente, una volta generata tale lista (o almeno una parte sufficiente dato che essa è infinita) si può procedere confrontando l'istanza del problema con ciascun elemento della sequenza calcolata.
Naturalmente questo procedimento non è affatto efficiente in quanto esistono gruppi in cui, data una parola w di lunghezza n uguale a e, il numero di fattori in una qualsiasi sua scomposizione in relatori cresce esponenzialmente con n.

3.2.2 Primo approccio al word problem

Novikov, Boone, e Britton dimostrarono, indipendentemente, che esiste un gruppo $\mathfrak{B} = \langle X, R\rangle$ finitamente presentato per il quale nessun calcolatore potrà in generale decidere se una parola casuale $w \in X^*$ sia pari a $e_{\mathfrak{B}}$.
Nel breve paragrafo che segue si richiamano le definizioni di algoritmo e di problema risolvibile.
Sia data una lista $\mathfrak{L}$ (non necessariamente finita) di domande. Un *processo di decisione* (o *algoritmo*)per $\mathfrak{L}$ è un'insieme di istruzioni uniformi e non ambigue, che se seguite conducono, per ogni domanda presente nella suddetta lista, ad una risposta corretta "si" o "no" in un numero finito di passi.
Sia data la presentazione ricorsiva di un gruppo finitamente generato:

$$G = \langle x_1, \dots, x_n | r_j = 1, j \geq 1\rangle$$

ogni parola w su $X = \{x_1, \dots, x_n\}$, non necessariamente ridotta, determina un elemento in G[2].
Si dirà che G ha WORD PROBLEM RISOLVIBILE se esite un processo di decisione per l'insieme $\mathfrak{L} = \{\text{“}w = e_g?\text{”} : w \in X^*\}$
É possibile dimostrare che se G è finitamente generato e il word problem ammette soluzione

[1] Si definisce X^k come l'insieme di tutte le stringhe di lunghezza k, con simboli tratti da X; si ha che $X^* = X^0 \cup X^1 \cup X^2 \cup \cdots$, dove per definizione $X^0 = \epsilon$.

[2] Vedere sezione 1.3.

per *una* delle sue presentazioni, allora esso è risolvibile su *ogni* presentazione del medesimo gruppo avente numero di generatori finito.
Si ricorda che la lunghezza di una parola di gruppo $w = x_1^{\tau_1} \dots x_m^{\tau_m}$ dove $\tau_i = \pm 1 \ \forall i = 1, \dots, m$ (non necessariamente ridotta) è pari a m. A esempio, la parola vuota ha lunghezza zero, ma la parola xx^{-1} ha lunghezza 2.
Inoltre si possono ordinare le parole, usando un metodo simile a quello utilizzato per le stringhe binarie nel paragrafo 2.1.5, in questo modo: al primo posto si mette la parola vuota, dopodiché tutte le parole di lunghezza 1 come segue $x_1, x_1^{-1}, \dots, x_n, x_n^{-1}$, proseguendo si elencano le parole di lunghezza 2 in ordine lessicografico (come in un normale dizionario): $x_1x_1 < x_1x_1^{-1} < \dots < x_n^{-1}x_n^{-1}$ poi ancora le parole di lunghezza 3 e così via.
Ricapitolando ϵ è la prima stringa, x_1 la seconda, x_2 la terza e così via, in questo modo è lecito parlare della "k-esima parola in X".
Si può ora definire la lista $\mathfrak{L}$ più precisamente, la k-esima domanda sarà infatti: $w_k = e$ in G?

3.2.3 I gruppi liberi hanno word problem risolvibile

Sia dato un gruppo libero:

$$G = \langle x_1, x_2, \dots x_n | \emptyset \rangle$$

esso ha word problem risolvibile.

Si riporta di seguito l'algoritmo per risolverlo, che conferma intuitivamente quanto appena affermato.

Algoritmo 3.1 Algoritmo per risolvere il word problem in un gruppo libero

WP-FREEGROUP($\{x_1, x_2, \dots x_n\}, w_k$)

```
1  while (length(w_k)^a ≥ 2) do
2        evidenzia la prima coppia di lettere adiacenti della forma x_i x_i^{-1} o x_i^{-1} x_i
3        if (una coppia di lettere è stata evidenziata) then
4              elimina i due caratteri sottolineati
5  if ( length(w_k)= 0 ) then
6        return w_k = e_G
7  else
8        return w_k ≠ e_G
```

[a] La funzione "**int** length(z)" riceve come parametro una parola $z = \alpha_1 \dots \alpha_t$ sull'alfabeto $X = \{x_1, x_2, \dots x_n\}$ e restituisce la sua lunghezza t.

Anche senza una trattazione formale, si evince facilmente che l'algoritmo descrive in maniera precisa e senza ambiguità un procedimento che, in un numero finito di passi (infatti riduce monotonicamente la lunghezza della parola ad ogni iterazione) approda alla

risposta corretta.

3.2.4 Richiami e definizioni utili

La dimostrazione del teorema di Novikov-Boon-Britton può essere divisa in due parti. La parte iniziale è solo una questione logico-matematica: un teorema provato indipendentemente da Markov e Post mostra che esiste un semigruppo[3] S finitamente presentato con word problem non risolvibile.
La seconda parte, più impegnativa, consiste nel costruire un gruppo $\mathfrak{B}$ finitamente presentato e mostrare che se tale gruppo avesse word problem risolvibile allora anche S potrebbe vantare tale proprietà, il che contraddice chiaramente il teorema di Markov-Post.
Prima di approdare ai risultati nominati, viene fornita la definizione di semigruppo e vengono riportate le definizioni più importanti riguardanti le macchine di Turing. Sono inoltre riportati alcuni esempi di linguaggi RE sottoinsiemi dei numeri naturali, utili nelle dimostrazioni dei risultati nominati all'inizio del paragrafo.

Definizione 3.2. *Una coppia* $(S, \bullet)$ *formata da un insieme* S *e un'operazione* $\bullet : S \times S \to S$ *è detta* SEMIGRUPPO *se soddisfa la seguente proprietà:*

L'operazione " $\bullet$ *" è* ***associativa****:*

$$s_1 \bullet (s_2 \bullet s_3) = (s_1 \bullet s_2) \bullet s_3 \quad \forall s_1, s_2, s_3 \in S$$

A differenza della definizione di gruppo, come si può facilmente notare, non è richiesta l'esistenza dell'elemento neutro, né l'esistenza dell'inverso per nessun elemento di S.
Si incontrano molti esempi di semigruppi, finiti e infiniti. Eccone alcuni facilmente definibili.

- L'insieme vuoto.
- L'insieme dei numeri interi positivi munito dell'addizione (operazione notoriamente associativa).
- L'insieme dei numeri interi naturali munito della moltiplicazione (anche questa operazione è associativa).
- L'insieme, numerabile, di tutte le stringhe su un dato alfabeto finito. In questo caso l'operazione "$\bullet$" sarà la giustapposizione di stringhe.

Una macchina di Turing (in seguito denotata con TM) è definita come la settupla $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ i cui componenti hanno i seguenti significati: Q è l'insieme finito degli stati del controllo, Σ è l'insieme finito dei simboli di input, Γ è l'insieme completo dei simboli di nastro (finito anch'esso), δ è la funzione di transizione, q_0 è lo stato iniziale del

[3] La definizione di *semigruppo* è data in 3.2.4.

controllo, B è il simbolo detto blank, F è l'insieme degli stati accettanti.
Essa è una macchina di calcolo astratta con la stessa potenza dei computer reali per quanto riguarda ciò che può essere calcolato.
La TM compie mosse basate sul suo stato corrente e sul simbolo di nastro presente nella cella visitata dalla testina. In una mossa la TM cambia stato, sovrascrive la cella corrente con un simbolo di nastro e muove la testina di una cella verso sinistra o verso destra. La macchina di Turing parte con l'input, una stringa di lunghezza finita di simboli, sul nastro; il resto del nastro contiene il simbolo blank in ogni cella. La TM accetta il suo input se entra in uno stato accettante.
Un linguaggio di dice ricorsivamente enumerabile (RE) se esiste una macchina di Turing in grado di accettare tutte e sole le stringhe che vi appartengono. I linguaggi RE accettati da una TM che si arresta sempre (anche quando l'input non appartiene al linguaggio della TM considerata) sono detti ricorsivi. I linguaggi ricorsivi sono chiusi rispetto al complemento, inoltre se un linguaggio e il suo complemento sono entrambi RE, allora entrambi sono ricorsivi. Di conseguenza il complemento di un linguaggio RE ma non ricorsivo non può essere RE.
Si può descrivere la configurazione corrente di una macchina di Turing mediante una stringa di lunghezza finita che include tutte le celle del nastro, dal simbolo diverso da blank più a sinistra a quello più a destra. La posizione della testina e lo stato del controllo si indicano inserendo quest'ultimo all'interno della sequenza dei simboli di nastro, a sinistra della cella visitata.

3.2.5 Sottoinsiemi di $\mathbb{N}$

Richiamando la definizione 2.1.3 del capitolo 2 si ha che un insieme I è ricorsivamente enumerabile se esiste una macchina di Turing T per la quale il linguaggio $L(T)$ possa essere identificato con l'insieme I. Ad esempio l'insieme dei numeri naturali $\mathbb{N}$ è ricorsivamente enumerabile: infatti se T è una macchina di Turing così definita ($Q = \{q_f, q_0\}, \Sigma = \{X\}, \Gamma = \{X, B\}, \delta, q_0, B, F = \{q_f\}$) dove la funzione di transizione δ è definita dalla seguente tabella:

	Simboli	
Stato	B	X
q_0	(q_f, B, R)	(q_o, X, R)
q_f	-	-

Tabella 3.1: Funzione di transizione di una macchina di Turing che accetta $\mathbb{N}$.

allora $L(T) = \{X^n : n \in \mathbb{N}\}$ e posto $X^{n+1} = n$ si ha $L(T) = \mathbb{N}$, quindi l'insieme dei numeri naturali $\mathbb{N}$ è RE.
In generale, dato $X \in \Sigma$ ogni $n \in \mathbb{N}$ può essere rappresentato dalla parola X^{n+1}, non è restrittivo pertanto supporre che i numeri naturali appartengano al linguaggio della generica macchina di Turing T.

In questo modo $E \subseteq \mathbb{N}$ è un SOTTOINSIEME RE DI $\mathbb{N}$ se esiste una macchina di Turing T tale che i numeri naturali associati alle stringhe che essa accetta sono tutti e soli gli elementi di E. Nel corso di questa sezione si dimostrerà che esistono sottoinsiemi RE dei naturali che non sono ricorsivi e sottoinsiemi di $\mathbb{N}$ che non sono RE.
Sia T la macchina di Turing definita dalla settupla $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$. Si rappresentano rispettivamente con 0 e 1 le due direzioni R e L che la testina può intraprendere; gli stati del controllo $Q = \{q_0, q_1, \ldots, q_k\}$ si rappresentano con i numeri pari:

$$q_1 \mapsto 4;\ q_2 \mapsto 6;\ \ldots\ q_k \mapsto 2k + 2$$

Associamo inoltre al simbolo di nastro B il numero 3; ai simboli di nastro $X_1, X_2, \ldots, X_M$ che stanno nell'alfabeto Σ i numeri dispari:

$$X_1 \mapsto 5;\ X_2 \mapsto 7;\ \ldots\ X_M \mapsto 2M + 3$$

Ai rimanenti simboli di nastro si associano i numeri dispari successivi. La funzione di transizione δ per una macchina di Turing è così definita: $\delta(q, X) \to (p, Y, D)$[4]; si può pensare di riscrivere ciascuna di queste regole mediante una stringa di cinque caratteri (quindi senza "δ", le parentesi, le virgole e la freccia) usando una notazione posizionale: i primi due simboli rappresentano, in ordine, lo stato attuale e il simbolo presente sul nastro letto dalla testina; gli ultimi tre invece rappresentano il nuovo stato, il simbolo di nastro che andrà a sovrascrivere quello presente e la direzione verso la quale si sposterà la testina.

Esempio 3.3. Alla luce di quanto detto, la scrittura $\delta(q, X) \to (p, Y, D)$ viene trasformata in $qXpYD$.

Se T è una TM il cui comportamento è descritto da m quintuple, allora esse si possono giustapporre in ordine casuale, per formare una parola $w(T)$ di lunghezza $5m$.

Definizione 3.4. *Il numero*

$$G(T) = \prod_{i=1}^{5m} p_i^{e_i}$$

dove p_i è l'i-esimo numero primo mentre e_i è il numero naturale assegnato come descritto sopra all'i-esima lettera in $w(T)$, è detto NUMERO DI GÖDEL ASSOCIATO A T.

[4]Gli argomenti di $\delta(q, X)$ sono uno stato q e un simbolo di nastro X. Il valore di $\delta(q, X)$, se definito, è una tripla (p, Y, D), dove:

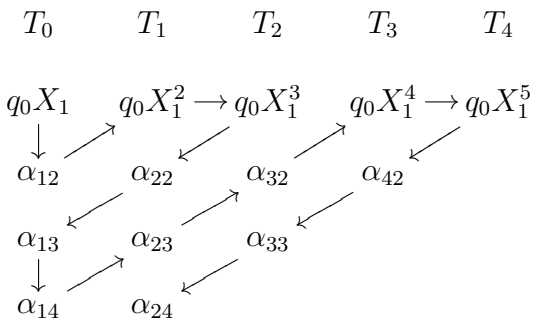
1. p: elemento di Q, è lo stato successivo.
2. Y: è il simbolo di nastro scritto nella cella visitata, al posto di qualunque simbolo vi fosse.
3. D: è una direzione, L o R (rispettivamente per "left", sinistra o "right", destra) e segnala la direzione in cui si muove la testina.

Per una descrizione più dettagliata si veda l'inizio del capito 2.

Si osservi come il numero di Gödel associato a T non dipenda dall'ordine in cui le stringhe, che descrivono il comportamento della stessa, vengono giustapposte per andare a formare la parola $w(T)$, ma solo dall'insieme delle quintuple che definisce la macchina T. Il teorema fondamentale dell'aritmetica[5] assicura che macchine di Turing non equivalenti[6] abbiano numero di Gödel differente.

Teorema 3.5. *Esiste un sottoinsieme RE dei numeri naturali* $\mathbb{N}$ *che non è ricorsivo. In particolare il complementare di tale sottoinsieme non è RE.*

Dimostrazione. Procedendo come descritto poco fa si enumerano le TM a seconda del numero di Gödel associato: $T_0, T_1, \ldots T_n$, dove T precede T' se $G(T) < G(T')$.
Si consideri ora il sottoinsieme dei naturali $E = \{n \in N \text{ tali che } n \in L(T_n)\}$ in questo modo $n \in E$ se e solo se la $(n+1)$-esima macchina di Turing T_n accetta n.
L'insieme E è ricorsivamente enumerabile, come dimostrato dalla seguente figura:



L'n-esima colonna rappresenta le mosse della n-esima macchina di Turing T_n, avente ID iniziale $q_0X_1^{n+1}$. Per "popolare" l'insieme E si seguono le frecce della figura sopra, e si inserisce n in E non appena una macchina di Turing raggiunge la descrizione istantanea accettante α_{ni} in colonna n [7]. Una macchina di Turing T^* che esegua queste semplici istruzioni può essere definita senza difficoltà. In questo modo si è dimostrato che E è un sottoinsieme RE di $\mathbb{N}$.
Per dimostrare che tale sottoinsieme non è ricorsivo risulta sufficiente dimostrare che il suo complemento:

$$\overline{E} = \{n \in \mathbb{N} : n \notin E\} = \{n \in \mathbb{N} : X_1^{n+1} \text{ non appartiene a } L(T_n)\}$$

è un sottoinsieme non-RE di $\mathbb{N}$[8].
Si supponga l'esistenza di una TM T' tale che $L(T') = \overline{E}$; poiché tutte le macchine di Turing sono state elencate nella tabella, $T' = T_m$ per qualche $m \in \mathbb{N}$.

[5] Il teorema fondamentale dell'aritmetica afferma che: ogni numero naturale maggiore di 1 o è un numero primo o si può esprimere come prodotto di numeri primi. Tale rappresentazione è unica, se si prescinde dall'ordine in cui compaiono i fattori.

[6] Per la definizione di MT equivalenti vedere la definizione 2.3 del capitolo 2.

[7] In questa dimostrazione non si usa la definizione di insieme RE riportata nella sottosezione 2.1.3 del capitolo 3; bensì un'altra ad essa equivalente, riportata qui di seguito: " Un insieme S si dice RE se esiste un algoritmo (o equivalentemente una macchina di Turing) che genera in output tutti e soli gli elementi di S ".

[8] Si veda la sottosezione 2.2.3.

- se $m \in \overline{E} = L(T') = L(T_m)$ quindi T_m accetta X_1^{m+1}, dunque $m \in E$; contraddizione.
- se $m \notin \overline{E}$ allora $m \in E$ dunque T_m accetta X_1^{m+1} (per come è stato definito E), ne consegue che $m \in L(T_m) = L(T') = \overline{E}$; contraddizione.

Perciò, $\overline{E}$ è un insieme non-RE e quindi E è RE-non-ricorsivo. □

3.2.6 Teorema di Markov-Post

Prima di procedere è necessario dare la definizione di "parola positiva".

Definizione 3.6. *Una parola W nell'alfabeto $X = \{x_1, x_2, \ldots, x_n\}$ si dice* POSITIVA *se è vuota oppure se è formata da una sequenza di simboli finita*

$$f_1, f_2, \ldots, f_{n-1}, f_n$$

dove ciascuno degli f_v con $v = 1, \ldots, n$ è uno dei seguenti simboli:

$$x_1, x_2, \ldots, x_n$$

L'alfabeto X all'interno del quale si fa vivere la parola W è fondamentale per poter stabilire la sua "positività": la medesima parola può essere positiva o meno a seconda dell'alfabeto di appartenenza scelto, come mostra il seguente esempio.

Esempio 3.7. La sequenza $x_1x_1^{-1}x_2x_3^{-1}$ risulta essere una parola positiva considerando l'alfabeto $\{x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}\}$, mentre non lo è più prendendola all'interno di $\{x_1, \ldots, x_n\}$.

Sia Ξ un semigruppo con generatori $X = \{x_1, \ldots, x_n\}$ ed eventuali relazioni; e sia Ω l'insieme delle parole positive in X. Si dice che IL SEMIGRUPPO Ξ HA WORD PROBLEM RISOLVIBILE se esiste un algoritmo in grado di determinare, data una coppia di parole arbitrarie $\omega, \omega' \in \Omega$, se $\omega = \omega'$ in Ξ oppure no.
La definizione (informale) appena data, ne fornisce una precisa di "non risolvibilità":

Definizione 3.8. *Sia Ξ un semigruppo con generatori $X = \{x_1, \ldots, x_n\}$ ed eventuali relazioni; e sia Ω l'insieme delle parole positive in X.* IL SEMIGRUPPO Ξ HA WORD PROBLEM NON RISOLVIBILE *se esiste una parola $\omega_0 \in \Omega$ tale che $\{\omega \in \Omega : \omega = \omega_0$ in $\Xi\}$ sia non-ricorsivo.*

Ora che è stato delineato per i semigruppi, si procede con la definizione word problem risolvibile per i gruppi.

Definizione 3.9. *Sia G un* gruppo *con presentazione $\langle x_1, \ldots, x_n \mid \Delta \rangle$ e sia Ω l'insieme di tutte le parole in $\{x_1, \ldots, x_n\}$ (visto come l'insieme di tutte le parole positive in $\{x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}\}$). Allora* IL GRUPPO G HA WORD PROBLEM RISOLVIBILE *se $\{\omega \in \Omega : \omega = e_G$ in $G\}$ è ricorsivo.*

La distinzione tra insiemi RE e insiemi ricorsivi persiste anche all'interno della teoria dei gruppi.

Teorema 3.10. *Sia G un gruppo finitamente presentato:*

$$G = \langle x_1, \ldots, x_n \mid r_1, \ldots, r_m \rangle$$

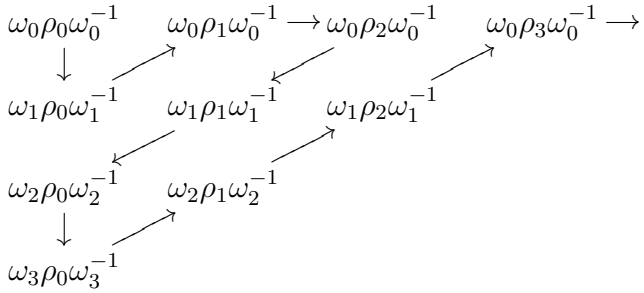
se Ω rappresenta l'insieme di tutte le parole in $\{x_1, \ldots, x_n\}$ allora $E = \{\omega \in \Omega : \omega = e_G \in G\}$ è RE.

Dimostrazione. Per dimostrare che l'insieme E è ricorsivamente enumerabile bisogna descrivere una macchina di Turing M_E tale che, data una parola w nell'alfabeto $\{x_1, \ldots, x_n\}$ pari all'elemento neutro e_G, stabilisca che $w \in L(M_E)$ in un numero finito di transizioni; nel caso in cui $w \neq e_G$ la macchina di Turing M_E potrebbe non terminare mai i suoi calcoli. Si prenda una parola in $\Omega = \{x_1, \ldots, x_n\}$ pari a e_G, allora essa deve essere della seguente forma $w = \varsigma_1\varsigma_2 \ldots \varsigma_n$ per un qualche n finito. Ciascun ς_i è una parola in Ω del seguente tipo: $\omega_j\rho_k\omega_j^{-1}$, dove ρ_k è una parola in $\{r_1, \ldots, r_m\}$ mentre i simboli della stringa ω_j sono scelti a partire dall'alfabeto $\{x_1, \ldots, x_n\}$ (è banale verificare che, una siffatta parola, rappresenta l'elemento neutro di G).

> Si noti che è stato scritto $w = \varsigma_1\varsigma_2 \ldots \varsigma_n$ e non $w \equiv \varsigma_1\varsigma_2 \ldots \varsigma_n$ (vedere la definizione 3.12) infatti tra la varie parole giustapposte per formare la stringa completa w possono avvenire delle cancellazioni che possono modificare, anche in maniera sostanziale, la struttura di w.
>
> **Esempio 3.11.** Si consideri la seguente presentazione $\langle a, b, c, d, e \mid abc^{-1},\ bcd \rangle$ di un gruppo G. La parola $[eb^{-1}](abc^{-1})[be^{-1}][a^{-1}](bcd)[a][c^{-1}a](bcd)[a^{-1}c]$ è banalmente pari a e_G: le parentesi evidenziano chiaramente la struttura del tipo $\omega_j\rho_k\omega_j^{-1}\omega_y\rho_w\omega_y^{-1} \ldots$. Tuttavia la parola $[a^{-1}](abc^{-1})[a][a^{-1}d](bcd)[d^{-1}a] = e_G$ è pari a $bc^{-1}dbca$. Nell'ultima scrittura la struttura del tipo $\omega_j\rho_k\omega_j^{-1}\omega_y\rho_w\omega_y^{-1} \ldots$ è scomparsa.

Il procedimento di massima è il seguente: la parola in ingresso viene ridotta, dopodiché si generano tutti i possibili "prodotti" (ridotti anche loro) aventi come "fattori" parole del tipo $\omega_j\rho_k\omega_j^{-1}$; essi vengono confrontati di volta in volta con la parola da testare: nel caso un prodotto ridotto risultasse equivalente alla stringa in ingresso allora l'input è pari a e_G. Prima o poi, se l'input è pari all'elemento neutro di G, si troverà una corrispondenza, in caso contrario M_E continuerà i suoi calcoli all'infinito.
Bisogna ora descrivere un metodo che permetta di comporre tutti i possibili prodotti aventi come aventi come fattori parole del tipo $\omega_j\rho_k\omega_j^{-1}$. Si immagini di dover enumerare tutte le possibili parole composte da 3 elementi della forma $\omega_j\rho_k\omega_j^{-1}$, usando la notazione introdotta in precedenza, bisogna produrre quindi tutte le stringhe del tipo $w = \varsigma_1\varsigma_2\varsigma_3$; per ciascuna ς_i ci sono infinite scelte. Si procede in questo modo: la tripletta (x, y, z) dove $x, y, z \in \mathbb{N}$ viene interpretata come un sistema di riferimento cartesiano per uno spazio tridimensionale avente come origine (0,0,0). La DISTANZA di un punto (x',y',z') dall'origine è data da $x' + y' + z'$ e si indica con $d(x', y', z')$. Una volta fissata una distanza d' esiste solamente un numero finito di triplette che distano d' dall'origine. Si generano dunque tutte le triplette

che distano 0 dall'origine, poi tutte le triplette che distano 1, e così via. Naturalmente a ciascuna delle tre componenti del sistema di riferimento bisogna poi assegnare un elemento del tipo $\omega_j \rho_k \omega_j^{-1}$: si comincia elencando le parole $\omega_0, \omega_1, \ldots$ in Ω come fatto nella sottosezione 4.1.1: prima la parola vuota, dopodichè le parole con lunghezza 1 nell'ordine $x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}$ seguite dalle parole che vantano lunghezza 2 in ordine lessicografico, a loro volta seguite dalle parola di aventi lunghezza 3, sempre in ordine lessicografico e così via. In maniera del tutto simile, si elencano tutte le parole in $\{r_1, \ldots, r_m\} : \rho_0, \rho_1, \ldots$.
Seguendo lo stesso ragionamento del teorema 3.5, percorrendo il tracciato indicato dalle frecce nella figura sottostante, si riesce ad assegnare un numero intero (il numero dei suoi predecessori nel percorso) ad ogni elemento della forma $\omega_j \rho_k \omega_j^{-1}$.



Quindi, ad esempio, alla tripletta $(9, 1, 5)$ verrà assegnato l'elemento: $\omega_1\rho_2\omega_1^{-1}\omega_0\rho_0\omega_0^{-1}\omega_1\rho_1\omega_1^{-1}$; alla terna $(7, 6, 2)$ la parola $\omega_3\rho_0\omega_3^{-1}\omega_2\rho_0\omega_2^{-1}\omega_1\rho_0\omega_1^{-1}$ e così via.
Ci si può ora chiedere, quanti elementi del tipo (x, y, z) distano (ad esempio) 4 da (0,0,0)?

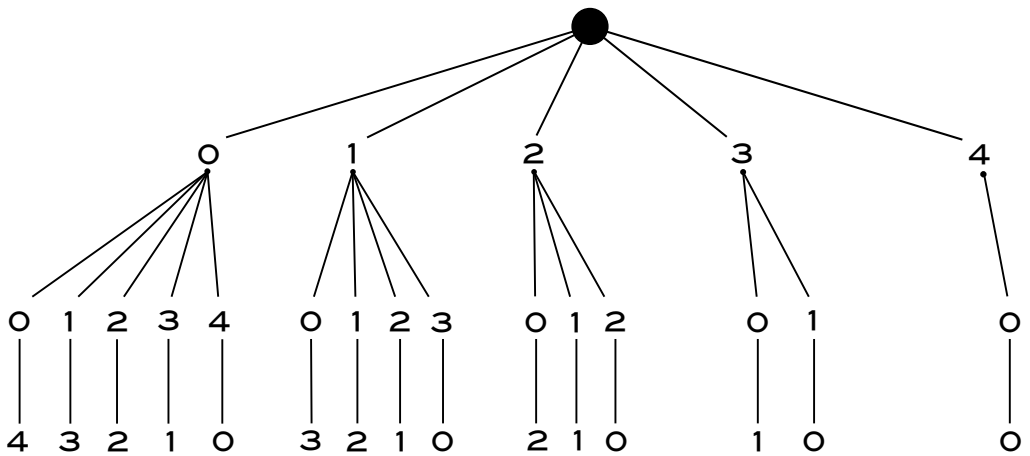
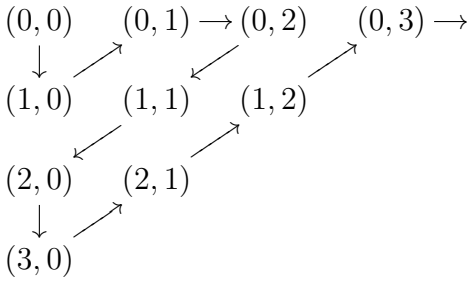


Figura 3.1: Albero che organizza tutte le possibili coordinate tridimensionali che distano 4 dall'origine $(0, 0, 0)$.

L'albero si spiega in questo modo: il nodo radice è un nodo fittizio, i nodi aventi profondità 1 corrispondono a tutte le scelte possibili per la prima dimensione, a seconda di ciò che è stato deciso, i numeri disponibili per y cambiano di conseguenza. Infine la scelta per z (in generale, per l'ultima dimensione) è obbligata (la distanza *deve* essere 4). Quindi

data una "dimensione" dim e una "distanza" d' al livello 1 ci saranno $d'+1$ nodi, al livello 2 compariranno $\sum_{i=1}^{d'+1} i$ nodi, al livello 3 saranno presenti $\sum_{j=1}^{d'+1}\sum_{i=1}^{j} i$ nodi, al livello 4 ci saranno $\sum_{k=1}^{d'+1}\sum_{j=1}^{k}\sum_{i=1}^{j} i$ e così via fino all'ultimo livello (che avrà altezza $dim-1$) il quale avrà un numero di nodi pari al livello $dim-2$.
La dimostrazione non è ancora conclusa: la macchina M_E non può fossilizzarsi su prodotti di 3 fattori, deve affondare la sua ricerca sia nella "distanza" sia nelle "dimensioni". Essa può organizzare il suo lavoro in questo modo:



La macchina di Turing M_E scorre tutte le (infinite) coppie; quando essa esamina il generico elemento $(\tilde{dim}, \tilde{d}')$ controllerà tutte le parole associate ai punti che distano $\tilde{d}'$ dall'origine di uno spazio $\tilde{dim}$-dimensionale. □

Ne consegue che un gruppo finitamente presentato G ha word problem *risolvibile* se e solo se $\{\omega \in \Omega : \omega \neq e_G \text{ in } G\}$ è anch'esso RE[9].

Definizione 3.12. *Se ω e ω' sono due parole nell'alfabeto X, allora si scriverà:*

$$\omega \equiv \omega'$$

se ω e ω' sono composte da esattamente *la stessa sequenza di lettere. Si dirà che ω e ω' sono* EQUIVALENTI.

Esempio 3.13. Si consideri il gruppo libero in $\{a, b, c, d\}$, le parole $adcac$, $adaa^{-1}cabb^{-1}c$, $ac^{-1}cdcdd^{-1}ac$ sono chiaramente tutte uguali, tuttavia non sono tra di loro equivalenti.

Si vuole far risaltare il fatto che "due parole sono *equivalenti* nel gruppo libero in X" è un'affermazione molto più forte rispetto a "due parole sono *uguali* nel gruppo libero in X".
Si consideri ora un semigruppo Ξ avente la seguente presentazione:

$$\Xi = \langle X \mid \boldsymbol{\alpha_j} = \boldsymbol{\beta_j}, j \in J \rangle$$

Siano ω e ω' parole positive in X. In tal caso è facile mostrare che $\omega = \omega'$ in Ξ se e solo se esiste una sequenza finita

$$\omega \equiv \omega_1 \to \omega_2 \to \cdots \to \omega_t \equiv \omega'$$

dove $\omega_i \to \omega_{i+1}$ sta ad indicare che una delle seguenti condizioni è soddisfatta:

[9] Si veda la sottosezione 2.2.3.

(i) $\omega_i \equiv \sigma\boldsymbol{\alpha_j}\tau$ e $\omega_{i+1} \equiv \sigma\boldsymbol{\beta_j}\tau$

(ii) $\omega_i \equiv \sigma\boldsymbol{\beta_j}\tau$ e $\omega_{i+1} \equiv \sigma\boldsymbol{\alpha_j}\tau$

La scrittura $\omega_i \to \omega_{i+1}$ è chiamata anche OPERAZIONE ELEMENTARE.

Si noti la differenza, alla luce di quanto appena detto, tra $\omega = \omega'$ e $\omega \equiv \omega'$ in Ξ. Indicando con " $\bullet$ " l'operazione del semigruppo Ξ ed espandendo le due parole nel modo seguente $\omega = x_1x_2\ldots x_k$ e $\omega' = x'_1x'_2\ldots x'_v$, l'equazione $\omega = \omega'$ significa:

$$x_1 \bullet x_2 \bullet \cdots \bullet x_k \;=\; x'_1 \bullet x'_2 \bullet \cdots \bullet x'_v \;\text{ in }\; \Xi$$

Mentre la scrittura $\omega \equiv \omega'$ garantisce che:

(i) $k = v$

(ii) $x_i = x'_i \;\forall i = 1, 2, \ldots, k(= v)$

Segue banalmente che

$$\omega \equiv \omega' \Rightarrow \omega = \omega'$$

ma non vale il viceversa:

$$\omega \equiv \omega' \not\Leftarrow \omega = \omega'$$

Si procede ora associando un semigruppo ad ogni macchina di Turing $T = (Q, \Sigma, \Gamma, \delta, q_1, s_0, F)$ dove

(i) $Q = \{q_0, q_1, \ldots, q_N\}$ è l'insieme finito degli stati del controllo.

(ii) $\Sigma = \{s_1, \ldots, s_M\}$ è l'insieme finito dei simboli di input.

(iii) $\Gamma = \Sigma \cup s_0 \cup \{q, h\}$ è l'insieme completo dei simboli di nastro. Si fa notare che q e h sono simboli speciali, la cui utilità verra evidenziata più avanti; inoltre nessuno tra s_0, q e h appartiene a Σ.

(iv) δ: la FUNZIONE DI TRANSIZIONE. Gli argomenti di $\delta(q_i, \gamma^\star)$ sono uno stato q_i e un simbolo di nastro $\gamma^\star$. Il valore di $\delta(q_i, \gamma^\star)$, se definito, è una tripla $(q_l, \gamma^\diamond, D)$, dove

  (a) q_l: elemento di Q, è lo stato successivo (può coincidere con q_i).

  (b) $\gamma^\diamond$: il simbolo di Γ scritto nella cella visitata, al posto di $\gamma^\star$ (può essere che $\gamma^\star = \gamma^\diamond$).

  (c) D: una DIREZIONE, L o R (rispettivamente per "left", sinistra o "right", destra); segnala la direzione in cui si muove la testina.

Inoltre se $\delta(q_i, \gamma^\star) \to (q_l, \gamma^\diamond, D)$ è una regola di T, si scriverà $q_i\gamma^\star q_l\gamma^\diamond D \in T$.

(v) q_1 è lo STATO INIZIALE del controllo.

(vi) s_0: il simbolo detto BLANK. Si trova in Γ ma non in Σ, cioè non è un simbolo di input. Inizialmente compare in tutte le celle, tranne quelle (finite) che contengono i simboli di input.

(vii) $F = \{q_0\}$ è l'insieme degli STATI FINALI o ACCETTANTI.

Definizione 3.14. *Sia T una macchina di Turing con stato accettante q_0, inoltre sia $E = L(T)$; il semigruppo che vanta la seguente presentazione:*

$$\Xi(T) = \langle q, h, s_0, s_1, \ldots, s_M, q_0, q_1, \ldots, q_N \mid R(T)\rangle \tag{3.1}$$

dove le relazioni $R(T)$ sono, $\forall\ \beta = 0, 1, \ldots, M$:

$$q_i s_j s_\beta = s_k q_l s_\beta \quad se \quad q_i s_j q_l s_k R \in T \tag{3.2}$$

$$q_i s_j h = s_k q_l s_0 h \quad se \quad q_i s_j q_l s_k R \in T \tag{3.3}$$

$$s_\beta q_i s_j = q_l s_\beta s_k \quad se \quad q_i s_j q_l s_k L \in T \tag{3.4}$$

$$h q_i s_j = h q_l s_0 s_k \quad se \quad q_i s_j q_l s_k L \in T \tag{3.5}$$

$$q_0 s_\beta = q_0 \tag{3.6}$$

$$s_\beta q_0 h = q_0 h \tag{3.7}$$

$$h q_0 h = q \tag{3.8}$$

è detto SEMIGRUPPO ASSOCIATO $\Xi(T)$.

Le relazioni 3.2 e 3.4 descritte nella definizione 3.14 sono abbastanza perspicue, infatti esse stabiliscono semplicemente un uguaglianza tra le porzioni locali del nastro ove la TM agisce compiendo una mossa elementare; rispettivamente muovendo la testina verso destra e verso sinistra.

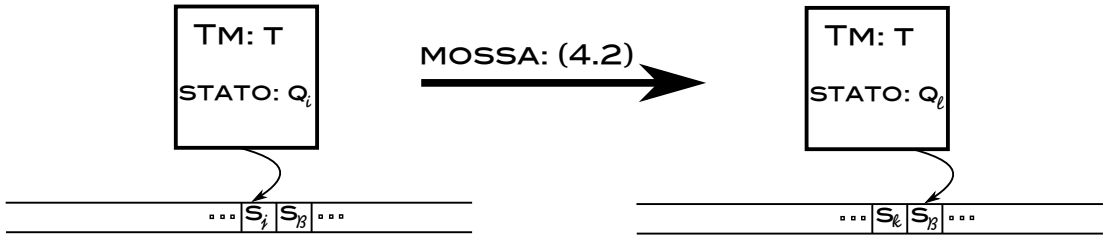


Figura 3.2: La macchina di Turing T che compie la mossa $\delta(q_i, s_j) \to (q_l, s_k, R)$ "in mezzo" al nastro.

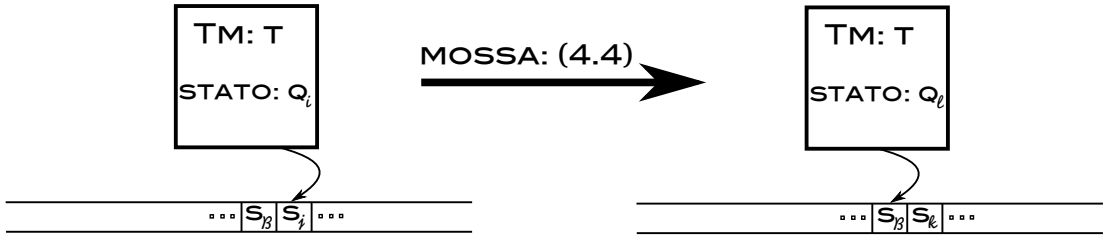


Figura 3.3: La macchina di Turing T che compie la mossa $\delta(q_i, s_j) \to (q_l, s_k, L)$ "in mezzo" al nastro.

Qualche chiarimento in più meritano le relazioni 3.3 e 3.5. Il carattere speciale h introdotto, lo si può interpretare come la *fine* del nastro. Quanto appena detto può causare qualche perplessità, poiché esso è per definizione infinito[10]; tuttavia si può procedere in questo modo: se con la prossima mossa, la testina raggiunge un'estremità del nastro (ovvero h), allora si crea una casella "cuscinetto" contenente il simbolo di blank s_0. Così facendo la TM non incontrerà nessun tipo di limitazione durante la sua elaborazione dell'input.

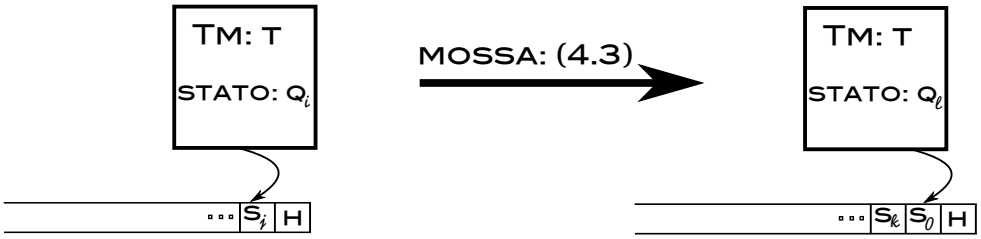


Figura 3.4: La macchina di Turing T che compie la mossa $\delta(q_i, s_j) \to (q_l, s_k, R)$ all'estremità destra del nastro.

[10]Si veda la sottosezione 2.1.1 del capitolo 2.

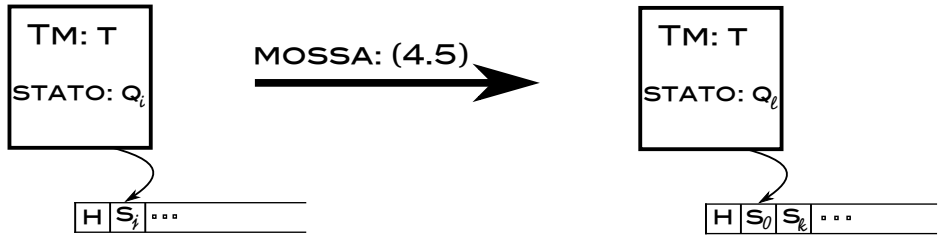


Figura 3.5: La macchina di Turing T che compie la mossa $\delta(q_i, s_j) \to (q_l, s_k, L)$ all'estremità sinistra del nastro.

Le relazioni 3.6, 3.7, 3.8 mettono in relazione il nastro di T, una volta che essa abbia raggiunto il suo stato accettante q_0, con l'altro simbolo introdotto: "q". Infatti tali uguaglianze rendono possibili una serie di "semplificazioni" nel nastro: la 3.6 permette di "cancellare" tutti i simboli a destra rispetto a quello puntato dalla testina fino a raggiungere l'estremità del nastro. Una volta raggiunta, la relazione 3.7 autorizza l'eliminazione di tutti i caratteri verso sinistra. Una volta raggiunta la "forma base" hq_0h la 3.8 permette di eguagliarla a q. Schematicamente, sia:

$$h \dots s_{i-2}s_{i-1}q_0s_is_{i+1}s_{i+2} \dots h$$

la fotografia del nastro di T non appena essa abbia raggiunto lo stato accettante q_0. Grazie alla 3.6 le seguenti uguaglianze sono vere:

$$h \dots s_{i-2}s_{i-1}q_0s_is_{i+1}s_{i+2} \dots h \; =$$

$$= \; h \dots s_{i-2}s_{i-1}q_0s_{i+1}s_{i+2} \dots h \; =$$

$$= \; h \dots s_{i-2}s_{i-1}q_0s_{i+2} \dots h \; = \; \dots \; =$$

$$= \; h \dots s_{i-2}s_{i-1}q_0h$$

Proseguendo, utilizzando ora la relazione 3.7 si ottiene:

$$h \dots s_{i-3}s_{i-2}s_{i-1}q_0h \; = \; h \dots s_{i-3}s_{i-2}q_0h \; =$$

$$= \; h \dots s_{i-3}q_0h \; = \dots \; =$$

$$= \; hq_0h$$

Infine utilizzando la 3.8, si conclude:

$$hq_0h = q$$

Considerando h come marcatore della fine del nastro, è possibile evidenziare alcune parole particolari.

Definizione 3.15. *Una parola si dice* H-SPECIALE *se è nella forma: $h\alpha h$, dove α è una descrizione istantanea di T.*

Si ricorda quanto è spiegato in maniera più esaustiva nella sottosezione 2.1.2, ovvero che una ID fotografa l'intera configurazione della macchina di Turing: il nastro lo stato attuale e la posizione della testina.
Tuttavia, la mera definizione di ID data in 2.1.2 necessita di qualche aggiustamento per adattarsi al caso specifico di T. In particolare, la modalità di rappresentazione dello stato e della testina rimangono invariate; invece la porzione di nastro rappresentata, non sarà dal simbolo diverso da blank più a sinistra fino a quello diverso da blank più a destra, bensì dal simbolo diverso da h più a sinistra fino a quello diverso da h più a destra.
Il capoverso appena concluso dovrebbe far apparire più chiara la definizione 3.15.

Definizione 3.16. *Sia α una ID della macchina di Turing T. Allora essa si dice* ID TERMINALE *se non esiste un'altra descrizione istantanea β tale che $\alpha \vdash_T \beta$*[11]*. Inoltre si assume che la TM T abbia sempre mosse valide per l'ID "$\ldots q_i s_j s_\beta \ldots$" $\forall\ q_i \in Q \setminus F$ e $\forall s_j, s_\beta \in \{s_0, s_1, \ldots, s_M\}$. Detto questo è lecito affermare che q_0 è (l'unico)* STATO TERMINALE *(o "stopping state", in inglese) di T.*

Poiché l'unico stato terminale di T è q_0, ogni $h\alpha h$ con α terminale è della forma: $h\sigma q_0 \tau h$ dove σ e τ sono s-parole e τ è non vuota. Inoltre le ultime tre relazioni permettono di concludere che $h\alpha h = q$ in $\Xi(T)$ ogniqualvolta α è terminale.

Lemma 3.17. *Sia T la macchina di Turing con stato accettante (e terminale) q_0 e*

$$\Xi(T) = \langle q, h, s_0, s_1, \ldots, s_M, q_0, q_1, \ldots, q_N \mid R(T) \rangle$$

il suo semigruppo associato. Allora le seguente affermazioni sono vere:

(i) Siano ω e ω' parole in $\{s_0, s_1, \ldots, s_M, q_0, q_1, \ldots, q_N\}$ con $\omega \not\equiv q$ e $\omega' \not\equiv q$. Se $\omega \to \omega'$, è un'operazione elementare compresa tra 3.2-3.8 allora ω è h-speciale se e solo se lo è anche ω'.

(ii) Se $\omega = h\alpha h$ è h-speciale, $\omega' \not\equiv q$ e $\omega \to \omega'$ è un'operazione tra 3.2, 3.3, 3.4 o 3.5 allora $\omega' \equiv h\beta h$ dove $\alpha \vdash_T \beta$ o $\beta \vdash_T \alpha$

Dimostrazione. (i) L'affermazione è vera poiché l'unica relazione che crea o distrugge h è la 3.8. Dato che per ipotesi $\omega \not\equiv q$ e $\omega' \not\equiv q$ essa non si può utilizzare.

(ii) Grazie alla prima parte, si può affermare che anche ω' è h-speciale, ad esempio $\omega' = h\beta h$. Ora, un'operazione elementare nel semigruppo consiste in una "sostituzione" di lettere, usando un'uguaglianza presente tra le defining relations. Una relazione di questo tipo in $\Xi(T)$ è una compresa tra 3.2-3.5 quindi esiste sempre una corrispondenza con un quintupla di T, che a sua volta rappresenta una mossa elementare della TM in questione. Quindi $\alpha \vdash_T \beta$ o $\beta \vdash_T \alpha$.

□

[11] Per la definizione di "$\vdash$" si veda la sottosezione 2.1.2 del capitolo 2.

Lemma 3.18. *Sia T una TM con stato accettante (e terminale) q_0, e sia Ω l'insieme di tutte le parole positive in Σ (l'insieme finito dei simboli di input di T). Se $\omega \in \Omega$ allora:*

$$\omega \in L(T) \quad \textit{se e solo se} \quad hq_1\omega h = q \quad \textit{in} \quad \Xi(T)$$

Dimostrazione. "$\Rightarrow$" Se $\omega \in L(T)$ allora si può scrivere la seguente catena di ID: $\alpha_1 = q_1\omega, \alpha_2, \ldots, \alpha_t$ dove $\alpha_i \vdash_T \alpha_{i+1}$ e lo stato accettante q_0 compare nell'ultima descrizione istantanea α_t. Usando le operazioni elementari 3.2-3.5 in $\Xi(T)$, si ottiene che $hq_1\omega h = h\alpha_t h$ in $\Xi(T)$. Proseguendo, utilizzando le relazioni 3.6-3.8 risulta $h\alpha_t h = q$ in $\Xi(T)$.
"$\Leftarrow$" La dimostrazione della "sufficienza" è di natura differente e leggermente più complessa: questo perché un'uguaglianza in $\Xi(T)$ è naturalmente una relazione simmetrica, viceversa una mossa elementare valida $\alpha \vdash_T \beta$ per la macchina di Turing T non implica necessariamente che anche $\beta \vdash_T \alpha$ lo sia.
Se $hq_1\omega h = q$ in $\Xi(T)$ allora si possono definire le parole: $\omega_1, \ldots, \omega_t$ sull'alfabeto $\{h, s_0, s_1, \ldots, s_M, q_0, \ldots, q_n\}$ tali che è possibile scrivere la seguente catena di operazioni elementari:

$$hq_1\omega h \equiv \omega_1 \to \omega_2 \to \cdots \to \omega_t \equiv hq_0 h \to q$$

Per il lemma 3.17(i), ogni ω_i è h-speciale: $\omega_i \equiv h\alpha_i h$ dove α_i è una ID di T. Sempre per il lemma 3.17(ii) vale che, o $\alpha_i \vdash_T \alpha_{i+1}$ altrimenti $\alpha_{i+1} \vdash_T \alpha_i$. Si proverà, per assurdo, che sono presenti solo occorrenze del primo tipo vale a dire: "$\alpha_i \vdash_T \alpha_{i+1}$". Seguirà che $q_1\omega \vdash_T \alpha_2 \vdash_T \cdots \vdash_T \alpha_t$ è una sequenza di mosse elementari, con α_t terminale (e quindi lo stato terminale q_0 è presente in α_t); quindi T riconosce ω e si può affermare che $\omega \in L(T)$.
Si procede ora con la dimostrazione di quanto anticipato nelle righe precedenti.
È sempre vero che $\alpha_{t-1} \vdash_T \alpha_t$ poiché α_t è terminale dunque, per definizione, non può essere che $\alpha_t \vdash_T \alpha_{t-1}$. In particolare questo mostra che quanto affermato è vero quando $t = 2$. Si supponga ora che $t > 2$ e che ci siano alcune mosse della TM T che vanno "da destra verso sinistra"[12]. Poiché, come si è detto prima, l'ultima mossa va da sinistra verso destra: $\alpha_{t-1} \vdash_T \alpha_t$, ripercorrendo la sequenza all'indietro si arriverà ad un certo punto a:

$$\alpha_{i-1} \dashv_T \alpha_i \vdash_T \alpha_{i+1}$$

Tuttavia non ci deve mai essere ambiguità circa la prossima mossa che T dovrà intraprendere: per l'ID α_i sono disponibili due mosse ma T è una macchina di Turing deterministica, perciò non ci si potrà mai imbattere in una situazione come quella descritta. Si conclude allora che tutte le "$\vdash_T$" vanno da sinistra a destra. □

Teorema 3.19 (Markov-Post, 1947)**.** *Le seguenti due proposizioni sono vere:*

(i) Esiste un semigruppo finitamente presentato:

$$\Psi = \langle q, h, s_0, s_1, \ldots, s_M, q_0, q_1, \ldots, q_N \mid R \rangle$$

con word problem non risolvibile.

[12]Con questa espressione informale s'intende, ovviamente, tutte mosse del tipo $\alpha_{i+1} \vdash_T \alpha_i$; viceversa la frase "mosse da sinistra verso destra" indica elaborazioni da parte di T del tipo $\alpha_i \vdash_T \alpha_{i+1}$.

(ii) Non esiste alcun algoritmo (o processo di decisione), avente come ingresso una parola h-speciale qualsiasi $h\alpha h$, che riesca a determinare se $h\alpha h = q$ in Ψ oppure no.

Dimostrazione. (i) Sia T una macchina di Turing con stato accettante (e terminale) q_0 e con alfabeto di input $A = \{s_1, \ldots, s_M\}$, l'insieme Ω indica la collezione di tutte le parole positive su A, dunque $L(T) = E \subset \Omega$. Sia ora $\tilde{\Omega}$ l'insieme di tutte le parole positive su $A \cup \{q, h, s_0, q_0, q_1, \ldots, q_N\}$, dove q_0, q_1 (stato iniziale), $\ldots, q_N$ sono q-lettere presenti nelle quintuple di T e rappresentano gli stati del controllo finito della TM in questione, q e h sono dei caratteri speciali mentre s_0 è, al solito, il simbolo detto "blank". In altre parole $\tilde{\Omega}$ contiene tutte le parole positive nell'alfabeto di nastro di T.
Viene definito inoltre:

$$\tilde{E} = \{\tilde{\omega} \in \tilde{\Omega} : \tilde{\omega} = q \text{ in } \Xi(T)\}$$

Sia $\varphi : \Omega \to \tilde{\Omega}$ la mappa data da $\omega \mapsto hq_1\omega h$, e si identifichi Ω con la sua immagine $\Omega_1 \subset \tilde{\Omega}$; di conseguenza il sottoinsieme E di Ω viene rappresentato da:

$$E_1 = \{hq_1\omega h : \omega \in E\}$$

Ricapitolando:

- Ω_1 è l'insieme di tutte le possibili parole che T può ricevere in ingresso, sulle quali però è stata effettuata un'operazione di "orlatura" per renderle compatibili con la definizione di $\Xi(T)$; infatti all'inizio e alla fine di ciascuna parola è presente il carattere speciale h, mentre lo *stato iniziale* q_1 è stato inserito in seconda posizione a partire da sinistra.

- E_1 è l'insieme di tutte le stringhe nell'alfabeto di input $A = \{s_1, \ldots, s_M\}$ *accettate* da T, "orlate" dal carattere h e con il simbolo q_1 posizionato appena prima dell'inizio della stringa stessa. Quindi è la *fotografia iniziale* di tutte le sequenze finite di caratteri in A che, mediante una serie finita di sostituzioni, possono essere ridotte a q utilizzando le relazioni di $\Xi(T)$.

- $\tilde{\Omega}$ è l'insieme di tutte le possibili parole positive nell'alfabeto di nastro di T: $A \cup \{q, h, s_0, q_0, q_1, \ldots, q_N\}$. Il quale contiene, oltre a tutte le quintuple che corrispondo a ID della macchina di Turing nominata, infinite altre stringhe senza significato come ad esempio: $hq_1q_2hhhhhs_0$ o $hhhhh\ldots$.

- $\tilde{E}$ è l'insieme di tutte le parole in $\tilde{\Omega}$ che sono uguali a q in $\Xi(T)$. Qui non ci si deve confondere: infatti di primo acchito si è portati a pensare che E_1 e $\tilde{E}$ siano in realtà scritture diverse per lo stesso insieme. In realtà non è così e il seguente esempio dovrebbe chiarire il concetto.

  Esempio 3.20. Se $hq_1\omega h = q$ in $\Xi(T)$ allora si possono definire le parole: $\omega_1, \ldots, \omega_t$ sull'alfabeto $\{h, s_0, s_1, \ldots, s_M, q_0, \ldots, q_n\}$ tali che è possibile scrivere la seguente catena di operazioni elementari:

$$hq_1\omega h \equiv \omega_1 \to \omega_2 \to \cdots \to \omega_t \equiv hq_0h \to q$$

Ogni parola $\omega_1, \ldots, \omega_t$ è banalmente uguale a q in Ξ, perciò tutte appartengono a $\tilde{E}$; tuttavia solo ω_1 appartiene a E_1 perché, come già detto tale insieme collezione tutte le ID *iniziali* che porteranno ad un ID accettante.

È chiaro che E_1 è un sottoinsieme ricorsivo di Ω_1 se e solo se E è un sottoinsieme ricorsivo di Ω. Con queste notazioni, il lemma 3.18 permette di scrivere:

$$E_1 = \tilde{E} \cap \Omega_1 \tag{3.9}$$

Ora si assuma che T sia la TM T^* (sempre con stato accettante e terminale q_0) del teorema 3.5, questo significa che E e dunque pure E_1 è RE-non-ricorsivo.
Ora si ponga l'attenzione sulla parte destra dell'equazione (3.9), se entrambi i sottoinsiemi fossero ricorsivi allora anche la loro intersezione lo sarebbe[13], contraddicendo quanto appena detto.
L'insieme Ω_1 è ricorsivo, è facile descrivere una macchina di Turing T_{Ω_1} che si arresti sempre, il cui linguaggio è composto da tutte e sole le stringhe del tipo $hq_1\omega h$ con $\omega \in A$. Sia la TM nominata così definita:

$T_{\Omega_1} = (Q = \{\varrho_0, \ldots, \varrho_4\}, \Sigma = A \cup h \cup q_1, \Sigma \cup B, \delta, \varrho_0, B, F = \{\varrho_4\})$
dove δ è data dalla seguente tabella:

	Simboli			
Stato	h	q_1	$\omega_i \ \forall \omega_i \in A$	B
ϱ_0	(ϱ_1, h, R)	-	-	—
ϱ_1	-	(ϱ_2, q_1, R)	-	-
ϱ_2	(ϱ_3, h, R)	-	(ϱ_2, ω_i, R)	-
ϱ_3	-	-	-	(ϱ_4, B, R)
ϱ_4	-	-	-	-

Tabella 3.2: La funzione di transizione δ di T_{Ω_1}.

Il funzionamento di T_{Ω_1} è semplice, innanzitutto non modifica mai il nastro, come si può notale dalla tabella la testina lascia il carattere letto intonso. Essa scorre l'input da sinistra a destra, necessariamente i primi due simboli devono essere h e q_1 dopodiché, entrata nello stato ϱ_2, scorre tutta la stringa ω. Appena legge il carattere h significa che la stringa è terminata e quindi il carattere successivo deve essere un blank.

Dato che Ω_1 è ricorsivo allora non lo è $\tilde{E} = \{\tilde{\omega} \in \tilde{\Omega} : \tilde{\omega} = q \text{ in } \Xi(T)\}$.
Quindi, $\Psi = \Xi(T^*)$ ha world problem non risolvibile.

(*ii*) Si definisca

$$\tilde{S} = \{ \text{ parole h-speciali } h\alpha h \text{ tali che: } h\alpha h = q \text{ in } \Xi(T^*)\}$$

[13]Si veda il teorema 2.8 del capitolo 3.

Si supponga che $\tilde{S}$ sia un sottoinsieme ricorsivo di $\tilde{\Omega}$, allora $\tilde{S} \cap \Omega_1$ sarebbe un sottoinsieme ricorsivo di Ω_1. Tuttavia $\tilde{S} \cap \Omega_1 = E_1$ e si giunge ad una contraddizione. □

Per una scrittura più chiara e semplice si scrivono in maniera differente i generatori e le relazioni del semigruppo di Markov-Post $\Xi(T^*)$ usando il seguente corollario.

Corollario 3.21. *Le seguenti due affermazioni sono vere:*

(i) Esiste un semigruppo finitamente presentato

$$\Xi = \langle q, q_0, \ldots, q_N, s_0, \ldots, s_M \mid F_i q_{i_1} G_i = H_i q_{i_2} K_i, i \in I \rangle$$

con word problem non risolvibile, dove F_i, G_i, H_i, K_i sono s-parole positive (anche vuote) e $q_{i_1}, q_{i_2} \in \{q, q_0, \ldots, q_N\}$.

(ii) Non esiste alcun algoritmo (o processo di decisione), avente come ingressi un carattere q_{i_j} arbitrario e due s-parole positive X e Y, che riesca a determinare se $X q_{i_j} Y = q$ in Ξ oppure no.

*Dimostrazione (*i*).* Il generatore h del semigruppo $\Xi(T^*)$ in questa presentazione non compare. Ad ogni modo questo problema è facilmente superabile in quando si può "reindirizzare" tale carattere verso una delle s-lettere, ad esempio $h = s_M$. Una volta preso atto di questa "rimappatura" e riscrivendo le relazioni di conseguenza, si ottiene proprio la scrittura presentata nel primo punto del corollario che si sta dimostrando. □

*Dimostrazione (*ii*).* Si ripercorre passo passo il punto (*ii*) della dimostrazione del teorema 3.19 mutatis mutandis.

□

3.2.7 Teorema di Novikov-Boone-Britton

Il word problem definito sui gruppi è stato considerato per la prima volta da M. Dehn (1910) e da A. Thue (1914). La soluzione fu scoperta da P.S. Novikov (1955) e, indipendentemente, da W.W. Boone (1954-1957) e da J.L. Britton (1958).
Nel 1959, Boone esibì un gruppo finitamente presentato relativamente conciso e semplice (rispetto e quelli scoperti fino ad allora in quest'ambito) e dimostrò che esso vanta word problem non risolvibile.
In contrasto con la dimostrazione "combinatoria" di Novikov e Boone, quella di Britton si basa sulle proprietà delle estensioni HNN (che gli permisero di scoprire il lemma che oggi porta il suo nome).
Nel 1963, Britton fornì una dimostrazione circa la non risolvibilità del word problem per

gruppi molto semplice, se confrontata con quella di Boone; si presenterà la sua dimostrazione in questa sottosezione, con alcune modifiche migliorative apportate da Boone, D.J. Collins e C.F. Miller.
Tutto ciò che serve per comprendere tale dimostrazione è già stato descritto in questo capitolo, in particolare il corollario 3.21 che assicura l'esistenza di un semigruppo Ξ con word problem non risolvibile. La dimostrazione che verrà presentata cercherà di legare l'uguaglianza di parole in un gruppo con l'uguaglianza di parole in un semigruppo. In questo passaggio è fondamentale osservare attentamente gli "esponenti" dei caratteri: infatti a diferenza dei gruppi dove qualsiasi parola arbitraria ha significato, nei semigruppo possono esistere solo parole positive[14].

> Prima di arrivare al teorema vero e proprio si richiamano alcuni concetti introdotti nelle pagine precedenti e se ne presentano di nuovi.
> Se $X \equiv s_{\beta_1}^{e_1} \dots s_{\beta_m}^{e_m}$ è una s-parola (non necessariamente positiva), allora $X^\sharp \equiv s_{\beta_1}^{-e_1} \dots s_{\beta_m}^{-e_m}$. Si noti che se X e Y sono s-parole, allora $(X^\sharp)^\sharp \equiv X$ e $(XY)^\sharp \equiv X^\sharp Y^\sharp$.
> Si ricorda inoltre che per ogni macchina di Turing T esiste un semigruppo "associato" $\Xi = \Xi(T)$ presentato nel modo seguente:
>
> $$\Xi = \langle q, q_0, \dots, q_N, s_0, \dots, s_M \mid F_i q_{i_1} G_i = H_i q_{i_2} K_i,\ i \in I \rangle$$
>
> dove F_i, G_i, H_i, K_i sono s-parole positive (anche vuote), mentre q_{i_1}, q_{i_2} appartengono all'insieme: $\{q, q_0, \dots, q_N\}$.

Per ogni macchina di Turing T, si definisce ora un gruppo $\mathfrak{B} = \mathfrak{B}(T)$ successivamente verrà dimostrato che su di esso il word problem non è risolvibile nel caso in cui T sia proprio la TM T^* del teorema di Markov-Post.
Il gruppo $\mathfrak{B}(T)$ vanta la seguente presentazione:

generatori: $q, q_0, \dots, q_N, s_0, \dots, s_M, r_i,\ i \in I, x, t, k$;

relazioni: $\forall i \in I$ e $\forall \beta = 0, \dots, M$:

$$xs_\beta = s_\beta x^2$$
$$r_i s_\beta = s_\beta x r_i x$$
$$r_i^{-1} F_i^\sharp q_{i_1} G_i r_i = H_i^\sharp q_{i_2} K_i$$
$$t r_i = r_i t$$
$$tx = xt$$
$$k r_i = r_i k$$
$$kx = xk$$
$$k(q^{-1}tq) = (q^{-1}tq)k$$

[14] Si ricorda che in un semigruppo, per nessun elemento, è garantita l'esistenza del proprio inverso.

Se X e Y sono s-parole alla si definisce

$$(Xq_jY)^* \equiv X^\sharp q_j Y$$

dove $q_j \in \{q, q_0, \ldots, q_N\}$.

Definizione 3.22. *Una parola Υ è* SPECIALE *se $\Upsilon \equiv X^\sharp q_j Y$ dove X e Y sono s-parole positive e $q_j \in \{q, q_0, \ldots, q_N\}$.*

Se Υ è speciale, allora $\Upsilon \equiv X^\sharp q_j Y$ dove X e Y sono s-parole positive e quindi $\Upsilon^* \equiv (X^\sharp q_j Y)^* \equiv X q_j Y$ è una parola positiva; allora si può dire che Υ^* rappresenta un elemento del semigruppo Ξ.

Lemma 3.23 (Boone)**.** *Sia T una macchina di Turing con stato accettante (e terminale) q_0, avente semigruppo associato $\Xi = \Xi(T)$ (scritto usando la stessa notazione del corollario 3.21).*
Se Υ è una parola speciale, allora:

$$k(\Upsilon^{-1} t \Upsilon) = (\Upsilon^{-1} t \Upsilon)k \quad in \quad \mathfrak{B} = \mathfrak{B}(T) \iff \Upsilon^* = q \quad in \quad \Xi(T)$$

Teorema 3.24 (Novikov-Boone-Britton)**.** *Esiste un gruppo $\mathfrak{B}$ finitamente presentato con word problem non risolvibile.*

Dimostrazione. Sia T la macchina di Turing T^* del teorema di Markov-Post. Se ci fosse un algoritmo in grado di determinare, per una parola speciale qualsiasi Υ, se

$$k\Upsilon^{-1} t \Upsilon k^{-1} \Upsilon^{-1} t^{-1} \Upsilon = 1 \text{ in } \mathfrak{B}(T)$$

allora tale processo di decisione sarebbe in grado di determinare se $\Upsilon^* = q$ in $\Xi(T^*)$. Tuttavia il corollario 3.21(*ii*) afferma che non non esiste alcun algoritmo del genere per $\Xi(T^*)$. □

Rabin nel 1958 e successivamente Baumslag, Boone e B. H. Neumann sempre nello stesso anno si basarono sulla scoperta di Novikov per dimostrare che, praticamente tutti i problemi che riguardano i gruppi finitamente presentati, sono generalmente non risolvibili [5]; ad esempio:

- determinare se il gruppo è banale;
- determinare se il gruppo è finito;
- determinare se il gruppo è libero;
- ...

Capitolo 4

Protocollo Shpilrain-Zapata

Molti problemi "difficili" provenienti dalla teoria combinatoria dei (semi)gruppi sono utilizzati in crittografia: essi possono trovare impiego nella fase iniziale, quindi nello scambio di chiavi tra le entità che vogliono iniziare una comunicazione protetta ("*key establishment protocol*"), o addirittura costituire il nucleo di un crittosistema completo a chiave pubblica. Molti di questi problemi sono *problemi di ricerca*, dunque della seguente natura: "data una particolare proprietà $\mathcal{P}$ trovare almeno un oggetto che la soddisfi" (molto spesso viene garantita in partenza l'esistenza di almeno un elemento che possa vantare $\mathcal{P}$). Finora nessun protocollo crittografico basato su un problema di ricerca in un (semi)gruppo non commutativo è stato preso seriamente in considerazione come alternativa ai protocolli moderni già affermati (ad esempio RSA) basati su (semi)gruppi commutativi.
Nessuno di essi è stato giudicato infatti sufficientemente sicuro per essere impiegato; questo è un vero peccato, in quanto molti di essi si sono dimostrati più efficienti di RSA [2].

I due matematici Shpilrain e Zapata suggeriscono invece di staccarsi dai problemi di ricerca, ed usare i *problemi decisionali* come fondamenta per realizzare lo scambio di chiavi, oppure l'intero crittosistema. Si ricorda che i problemi decisionali sono della seguente natura: "data una proprietà $\mathcal{P}$ e un oggetto $\mathcal{O}$, determinare se l'oggetto $\mathcal{O}$ in questione soddisfa la proprietà $\mathcal{P}$ oppure no".

Usando il "*word problem*" i due matematici nominati poc'anzi hanno ideato un sistema di cifratura avente le seguenti caratteristiche: (1) Bob trasmette ad Alice una sequenza cifrata di bit che quest'ultima potrà decifrare correttamente con probabilità "molto vicina" a 1; (2) l'intruso, Eva, alla quale è garantita una capacità computazionale grande a piacere (ma fissata), non potrà decifrare la sequenza di bit inviata da Bob usando un attacco a forza bruta. In altre parole la velocità di calcolo che può vantare Eva, per quanto alta, non darà alcuna informazione circa la sequenza di bit intercettata.

4.1 Using decision problem in public key cryptography

4.1.1 Introduzione

Ricercando alternative più efficienti o sicure ai protocolli crittografici esistenti (come RSA), molti ricercatori hanno ideato sistemi di cifratura completi o semplicemente key establishment protocol basati su search problem difficili riguardanti la teoria combinatoria dei (semi)gruppi. Nominandone solo alcuni che sono stati utilizzati: il problema di ricerca del coniugato, il problema dell'isomorfismo (versione "di ricerca"), il problema della scomposizione (versione "di ricerca") nonché il problema dell'appartenenza.
Come già detto, essi sono tutti della seguente natura: data una proprietà $\mathcal{P}$ e la garanzia che esistano oggetti con la proprietà $\mathcal{P}$, trovare almeno un elemento, appartenete ad un insieme $\mathcal{S}$, che la possieda.
Si intuisce facilmente che la sicurezza del protocollo crittografico associato a ciascuno dei problemi nominati si basa sulla limitatezza della capacità di calcolo dell'avversario (tipicamente subesponenziale). Infatti, l'insieme $\mathcal{S}$ molto spesso risulta essere ricorsivamente enumerabile; in questo modo è possibile scrivere un programma che come output dia tutti e soli gli elementi di $\mathcal{S}$, finché non si scopre l'oggetto con proprietà $\mathcal{P}$ (si assume che la verifica di tale caratteristica si possa effettuare efficientemente). La tecnica appena descritta è solitamente chiamata "attacco a forza bruta" o, in inglese, "brute force attack".

Shpilrain e Zapata suggeriscono di usare i *problemi decisionali*, provenienti dalla teoria combinatoria dei gruppi come nucleo per nuovi sistemi di cifratura moderni o per un public key establishment protocol innovativi. Secondo loro infatti, proprio nella natura di tali problemi è celata la proprietà che permetterebbe di vincere l'ultima sfida lanciata dalla crittografia moderna: quella di creare un crittosistema sicuro contro un attacco a forza bruta condotto da un avversario con capacità di calcolo essenzialmente illimitata.
Più precisamente il modello computazionale proposto è il seguente: il sistema di cifratura viene spiegato in ogni suo dettaglio all'avversario[1], il quale successivamente sceglie la capacità di calcolo che vorrebbe disporre per attaccarlo. Una volta decisa, lo sfidante non la potrà più cambiare, cioè non gli è consentito accelerare la computazione oltre il limite che lui stesso ha scelto.

In particolare il protocollo proposto dai due matematici Shpilrain e Zapata utilizza il *word problem* (trattato ampiamente nel capitolo 3), esso ha la seguente formulazione: "data una presentazione ricorsiva di un gruppo G e un elemento $g \in G$ dire se $g = e$ in G".
Per semplificare la scrittura nel corso di questo capitolo si indicherà con il simbolo $\bar{R}_\Gamma$ l'insieme (infinito) di tutti i defining relator della presentazione di gruppo Γ, compresi i loro inversi e i loro coniugati (nell'alfabeto dei generatori di Γ). Qualora fosse chiara la presentazione alla quale si fa riferimento si scriverà semplicemente $\bar{R}$.

[1] Assecondando il principio di Kerckhoffs [4].

Come già detto per i problemi decisionali presentati nella sezione 3.1 del capitolo 3 anche il word problem, data la sua natura, è composto da due parti: la "Yes-part" e la "No-part". Se la presentazione del gruppo è ricorsiva, come è richiesto nella formulazione del problema, allora la "Yes-part" è RE poiché si possono enumerare tutti i prodotti tra elementi di $\bar{R}$
Tuttavia il numero di fattori, in un prodotto del genere, necessari per rappresentare una parola di lunghezza n pari a e in G, può essere molto grande se comparato appunto con n. In particolare ci sono gruppi G, con word problem risolvibile, e parole w di lunghezza n pari a e in G tali che il numero di fattori in ogni loro scomposizione in elementi di $\bar{R}$ può essere esponenziale in n.
Si pensi infatti alla dimostrazione del teorema 3.10 del capitolo 3: per dimostrare che una parola $w \equiv \alpha_1\alpha_2 \dots \alpha_n$ lunga n è pari a e_G la si dovrebbe esprimere attraverso una giustapposizione di elementi ς_i in questo modo: $w = \tilde{w} \equiv \varsigma_1\varsigma_2 \dots \varsigma_k$ per un qualche k finito. Ciascun ς_i è una parola in nell'alfabeto dei generatori del gruppo G del seguente tipo: $\omega_j\rho_k\omega_j^{-1}$, dove ρ_k è una parola in $\{r_1, \dots, r_m\}$ (i relatori di G) mentre i simboli della stringa ω_j sono scelti a partire dall'alfabeto dei generatori del gruppo G. Non c'è alcun legame tra la lunghezza n della parola w e il numero di fattori k della parola $\tilde{w}$; inoltre la lunghezza della parola $\tilde{w}$, ovvero $|\tilde{w}|$, può essere molto maggiore di n (si ricorda infatti che k è solo il numero di fattori, non la lunghezza della parola ovvero il numero di caratteri che la compongono).
Inoltre in molti gruppi la "No-part" non è ricorsiva, quindi un eventuale attacco brute force, come quello descritto nell'introduzione 4.1.1, è irrealizzabile contro questa "parte" del problema.
In base a quanto detto si descriverà un protocollo crittografico avente le seguenti caratteristiche:

1. Bob trasmette ad Alice una sequenza cifrata di bit che quest'ultima potrà decifrare correttamente con probabilità "molto vicina" a 1.

2. Eva, l'intruso, alla quale è garantita una capacità computazionale grande a piacere (ma fissata), non potrà decifrare la sequenza di bit inviata da Bob usando un attacco a forza bruta. In altre parole la velocità di calcolo che può vantare Eva, per quanto alta, non darà alcuna informazione circa la sequenza di bit intercettata.

4.1.2 Il protocollo

In questa sottosezione viene fornito uno schema semplificato del protocollo, i dettagli verrano approfonditi nella parti successive.
Protocollo:

1. Si considera di *dominio pubblico* un insieme di presentazioni di gruppi aventi word problem risolvibile (efficientemente).

2. Alice sceglie casualmente una particolare presentazione Γ dall'insieme nominato. Ad essa applica una serie di trasformazioni "*isomorphism preservation*" ottenendo così un'altra presentazione Γ' che, data la natura delle trasformazioni utilizzate, identifica comunque lo stesso gruppo presentato da Γ. Successivamente Alice elimina da Γ' alcuni relatori, ottenendo una terza presentazione "accorciata" $\hat{\Gamma}$. Quest'ultima viene pubblicata mentre Γ e Γ' devono essere mantenute segrete. In base a quanto detto le seguenti due proposizioni sono vere: (i) $\Gamma \cong \Gamma' \ncong \hat{\Gamma}$; (ii) $w = e$ in $\hat{\Gamma} \Rightarrow w = e$ in Γ.[2]

3. Bob, per cifrare una sequenza di bit sensibile che vuole mandare ad Alice, procede in questo modo: trasmette una parola pari a e in $\hat{\Gamma}$ (e quindi anche in Γ') al posto di ogni "1" presente nella sequenza binaria da spedire, e una parola che non rappresenta l'elemento neutro al posto di ciascun "0". Naturalmente si deve generare una stringa differente per ogni bit della sequenza da codificare.

4. Alice, quando riceve una parola $\hat{w}$ deve capire se essa rappresenta l'elemento neutro in $\hat{\Gamma}$ oppure no. Per fare ciò in primo luogo mappa, attraverso l'isomorfismo noto solamente a lei che lega le due presentazioni, $\hat{w}$ (vista come una parola in $\hat{\Gamma}$) nella parola corrispondete in Γ: w, dopodiché risolve il word problem: "$w = e$ in Γ ?".

Molti punti del protocollo appaiono immediatamente non banali infatti, come già detto, verranno ampiamente descritti nelle sezioni di questo capitolo che seguiranno.
A priori costruire un elemento diverso da e in Γ' sembra essere la parte più difficile di questo crittosistema in quanto Bob (colui che dovrà eseguire questo calcolo) non conosce la suddetta presentazione. Come verrà spiegato nella parte 4.1.5 di questo capitolo, in realtà Bob dovrà solamente generare una *parola casuale sufficientemente lunga* e si vedrà che, con certezza quasi assoluta, essa non sarà pari a e in Γ'.

Si ricorda ancora una volta che, il principale vantaggio di questo protocollo rispetto a quelli già esistenti, consiste nel rendere impossibile ad un eventuale avversario un attacco a forza bruta. Un approccio di questo tipo è infatti il più ovvio e immediato (anche se molto spesso computazionalmente oneroso) per tentare di violare un protocollo di cifratura.

[2]Informalmente, si dirà che una parola è uguale a "e" in Γ (o, in generale, in una presentazione qualsiasi), se tale parola è pari all'elemento neutro del gruppo G identificato dalla presentazione in questione (si veda la sezione 1.3 del capitolo 1)

Durante la trasmissione (fase (3) del protocollo poc'anzi descritto) Eva potrebbe essere in grado di riconoscere tutte le parole pari a e della sequenza codificata generando (off-line) una lista[3] formata da prodotti di elementi di $\bar{R}_{\hat{\Gamma}}$ per poi ricercare in essa una corrispondenza tra gli elementi presenti e le parole intercettate durante la comunicazione tra Bob e Alice. Se una parola w trasmessa da Bob compare nella lista allora Eva deduce che $w = e$ in $\hat{\Gamma}$ quindi era un "1" della sequenza binaria originale, se non compare allora $w \neq e$ in $\hat{\Gamma}$ dunque era uno "0". Tuttavia questo modo di procedere conduce ad un altro problema: come può Eva concludere che w non si trova nella lista se tale lista è infinita? Naturalmente l'avversario in questione potrebbe accontentarsi di generare un lista finita "sufficientemente lunga" e affermare che se la parola non compare in essa, allora *probabilmente* $w \neq e$.
Questo è un punto cruciale: anche Bob, quando deve produrre una parola $w \neq e$ in $\hat{\Gamma}$ la compone casualmente: come è stato detto precedentemente una parola casuale *probabilmente* non sarà pari a e in $\hat{\Gamma}$.
Segue ora un riassunto della situazione, dal punto di vista di Alice e dal punto di vista di Eva. Come verrà spiegato, il "*probabilmente*" di Eva e quello di Bob, sono di natura molto differente.

Dati disponibili:	**Alice**: presentazioni: Γ, $\Gamma' \cong \Gamma$ e $\hat{\Gamma}$.
	Bob: presentazione $\hat{\Gamma}$.
	Eva: presentazione $\hat{\Gamma}$.

Tabella 4.1: Dati disponibili ad Alice, Bob e all'attaccante Eva.

[3]Come già detto nell'introduzione l'insieme di parole pari a e in $\hat{\Gamma}$ è RE poiché si possono enumerare tutti i possibili prodotti aventi come fattori elementi di $\bar{R}_{\hat{\Gamma}}$ (come suggerisce il teorema 3.10 del capitolo 3). Tuttavia il numero di fattori, in un prodotto del genere, necessari per rappresentare una parola di lunghezza n pari a e in G, può essere molto grande se comparato appunto con n. In particolare ci sono gruppi G, con word problem risolvibile, e parole w di lunghezza n pari a e in G tali che il numero di fattori in ogni loro scomposizione in elementi di $\bar{R}_{\hat{\Gamma}}$ può essere esponenziale in n.

$w = e$ in $\hat{\Gamma}$	**Alice**: per comporre una parola pari a e in $\hat{\Gamma}$ Bob usa come "mattoni" solamente gli elementi di $\bar{R}_{\hat{\Gamma}}$ (opportunamente mescolati come verrà descritto più avanti) quindi *sicuramente* la sequenza prodotta rappresenta l'elemento neutro. Alice risolve il word problem in Γ determinando che $w = e$ in tale presentazione.
	Eva: affinché possa riconoscerla correttamente, la probabilità $P_2(f(N))$ che una parola w lunga N pari a e generata da Bob abbia un "certificato" di lunghezza $\leq f(N)$ il quale garantisca che $w = e$ in $\hat{\Gamma}$, dovrebbe convergere a 1 per $N \to \infty$.
$w \neq e$ in $\hat{\Gamma}$	**Alice**: Per per rappresentare gli "0" Bob compone semplicemente una parola casuale lunga N (nell'alfabeto dei generatori di Γ); Alice poi risolve *correttamente* il word problem su di essa. Ora ci si chiede: può essere che, una parola composta casualmente, rappresenti l'elemento neutro in $\hat{\Gamma}$? Affinché la trasmissione possa procedere correttamente correttamente la probabilità $P_1(N)$ che una parola casuale w di lunghezza N non rappresenti l'elemento neutro deve convergere a 1 (velocemente) per $N \to \infty$.
	Eva: se una parola non viene trovata nella lista allora la interpreta come "0". Ecco che la funzione $P_2(f(N))$ gioca un ruolo determinante.

Tabella 4.2: Differenza tra $P_1(N)$ e $P_2(f(N))$.

Si nota subito che le funzioni P_1 e P_2 sono di natura molto differente. La prima è stata studiata affondo e in particolare è noto che in ogni gruppo infinito G, $P_1(N)$ converge a 1 per $N \to \infty$; si veda la sezione 4.1.5 per maggiori dettagli.
Diversamente la funzione $P_2(f(N))$ è molto più complessa e attualmente, oggetto di vivace ricerca. Risultati parziali sembrano suggerire che $P_2(f(N))$ possa non convergere[4] affatto a 1 per $N \to \infty$ [2]. Ricapitolando la "fiducia" che Alice ripone su P_1 è giustificata, molto meno quella che Eva ripone su P_2.

Concludendo questa sottosezione si fornisce una breve panoramica circa le parti di questo capitolo che seguiranno. La parte che riguarda l'efficienza e i parametri suggeriti per la realizzazione del protocollo è la 4.1.6. Il passo (2), quindi l'algoritmo che Alice dovrà eseguire per generare la propria chiave pubblica, è discusso in 4.1.4. Il punto (3), l'operazione di cifratura svolta da Bob, è descritta in 4.1.5.

[4]La funzione P_2 è comunque strettamente legata al particolare algoritmo usato da Bob per produrre le parole pari a 1.

Per quanto riguarda qualche informazione di carattere tecnico, è stato verificato (in [2]) che il tempo necessario per la codifica di un bit è quadratico nella lunghezza della parola utilizzata per rappresentarlo; in più altri test (sempre in [2]) hanno messo in evidenza che questo protocollo ha un coefficiente di espansione pari a 150: ogni bit della sequenza originale è sostituito con una parola che mediamente è lunga 150 bit. Inoltre il tempo che Alice impiegherà per decifrare ogni parola, ovvero per capire se essa rappresenta un "1" o uno "0" è limitato da $C \cdot |w|$ dove $|w|$ è la lunghezza della parola e C è una costante strettamente legata al particolare isomorfismo che relaziona Γ a Γ' (e viceversa). È proprio questo collegamento tra le due presentazioni, noto ad Alice ma sconosciuto a Eva, a porre la seconda in una posizione di netto svantaggio rispetto al legittimo destinatario del messaggio (si fa notare che Bob non deve conoscere l'isomorfismo utilizzato da Alice per poter codificare il messaggio). Infine la sicurezza semantica verrà brevemente presa in considerazione alla fine della sottosezione 4.1.5.

4.1.3 Insieme di presentazioni e algoritmo di Dehn

Ci sono molte classi di gruppi finitamente presentati con word problem risolvibile: gruppi con un solo relatore, gruppi iperbolici, gruppi nilpotenti, gruppi metabeliani.... In ogni caso Alice deve essere in grado di generare o scegliere da un insieme già popolato, una presentazione di un gruppo con word problem risolvibile in maniera efficiente. Tale restrizione limita le tipologie di tali entità matematiche che possono essere utilizzate in questo contesto. Shpilrain e Zapata suggeriscono di usare gruppi appartenenti alla classe "small cancellation". I relatori dei gruppi appartenenti a tale categoria soddisfano una "condizione metrica" molto semplice e verificabile efficientemente. Si prosegue ora seguendo l'esposizione di [6] e richiamando alcuni concetti utili.
Sia F(X) il gruppo libero su base $X = \{x_i \text{ con } i \in I\}$ dove I è un insieme numerabile. Sia $\epsilon_k \in \{\pm 1\}$ dove $1 \leq k \leq n$. Una parola $w(x_1 \dots x_n) = x_{i_1}^{\epsilon_1} x_{i_2}^{\epsilon_2} \dots x_{i_n}^{\epsilon_n}$ in $F(X)$ con tutti gli x_i non necessariamente distinti è una parola RIDOTTA se $x_{i_k}^{\epsilon_k} \neq x_{i_{k+1}}^{-\epsilon_{k+1}}$; inoltre una parola $w(x_1 \dots x_n)$ si dice RIDOTTA CICLICAMENTE se è una parola ridotta con $x_{i_1}^{\epsilon_1} \neq x_{i_n}^{-\epsilon_n}$.

Definizione 4.1. *Un insieme R di parole in $F(X)$ ridotte ciclicamente si dice* SIMMETRIZZATO *se $\forall r \in R$:*

(i) tutte le sue permutazioni cicliche, ridotte ciclicamente, appartengono a R.

(ii) tutte le permutazioni cicliche di r^{-1}, ridotte ciclicamente, appartengono a R.

Ogni insieme finito R, nel caso non lo fosse, può essere simmetrizzato efficientemente.

Definizione 4.2. *Sia G un gruppo con presentazione $\langle X; R\rangle$ dove R è simmetrizzato; una parola non vuota $u \in F(X)$ è chiamata* PEZZO RELATIVO[5] A *R se esistono due relatori*

[5]Dato che si lavorerà con un simmetrizzato alla volta d'ora in avanti verrà omessa la parte "relativo a R" e si dirà semplicemente "pezzo".

distinti $r_1, r_2 \in R$ *tali che* $r_1 \equiv uv_1$ *e* $r_2 \equiv uv_2$ *per qualche* $v_1, v_2 \in F(X)$ *senza alcuna cancellazione tra* u *e* v_1 *o tra* u *e* v_2.

L'ipotesi "small cancellation" assicura che questi *pezzi* occupino un prefisso relativamente piccolo di ciascun elemento di R:

(i) Il gruppo $G = \langle X; R\rangle$ appartiene alla classe $C(P)$ se nessun elemento di R è prodotto di meno di P *pezzi*.

(ii) Il gruppo $G = \langle X; R\rangle$ appartiene alla classe $C'(\lambda)$ se $\forall r \in R$ tale che $r \equiv uv$ e u è un pezzo, si ha che $|u| < \lambda r$.

Sia dato un gruppo $G = \langle x_1, \ldots, x_n; R\rangle$ con R simmetrizzato, appartenente alla classe $C'(\frac{1}{6})$; allora il seguente teorema è vero:

Teorema 4.3 (Dehn-Greendlinger)**.** *Sia* $w(x_1 \ldots x_n)$ *una parola ridotta ciclicamente. Se* $w = e$ *in* G *allora una qualche permutazione ciclica di* w *contiene una sottoparola* $\mathbf{u}$ *tale che* $r = \mathbf{u}\beta$ *è un relatore appartenete a* R *(*β *potrebbe essere la parole vuota* ϵ*). Inoltre si ha che* $|\mathbf{u}| > \frac{1}{2}|r|$.

Questo teorema fu scoperto prima da Dehn nel 1912, successivamente Greendlinger fornì una dimostrazione combinatoria dello stesso (a differenza di quella a carattere "geometrico" di Dehn) nel 1960 [5]. Esso suggerisce una procedura, nota come "algoritmo di Dehn" per risolvere il word problem in gruppi di questo tipo:

Algoritmo 4.1 Algoritmo di Dehn per risolvere il word problem in un gruppo $C'(\frac{1}{6})$

WP-Dehn's algorithm($G = \langle x_1, x_2, \ldots x_n \mid R\rangle, w$)

```
1  while (w ≠ ε) do
2      trovare un relatore r ∈ R con r ≡ uv e |u| > |v| tale che w ≡ bud^a
3      if (tale r non esiste) then
4          return "w ≠ e in G"
5      else
6          una volta trovato tale r riscrivere w nel modo seguente: w ≡ bv^{-1}d^b
7      return "w = e in G"
```

[a] Si fa notare che un r di questo tipo soddisfa $|r| < 2|w|$. L'insieme S di tutte le parole in $x_1, \ldots, x_n$ che hanno lunghezza strettamente minore di $2|w|$ è finito. Poiché R è ricorsivo si può effettivamente calcolare la lista di tutti gli elementi $R' = S \cap R$ e cercare all'interno di essa il relatore r desiderato.

[b] Dato che $uv = e$ allora $u = v^{-1}$ (questo spiega come mai è lecito effettuare tale sostituzione). Dato che $|u| > |v|$ segue banalmente che la nuova formulazione di w è più corta rispetto alla precedente.

Come già anticipato, uno dei principali risultati della "small cancellation theory" garantisce che l'algoritmo di Dehn è valido per ogni gruppo $C'(\frac{1}{6})$ [6].
È stato dimostrato inoltre che in generale, un gruppo finitamente generato è un small cancellation group [7]; quindi per selezionare un gruppo di questo tipo Alice (dopo aver stabilito l'insieme dei generatori) genera semplicemente alcune parole casuali (sull'alfabeto dei generatori) e controlla se il loro simmetrizzato soddisfa la condizione $C'(\frac{1}{6})$. Se non è così ripete la procedura. Concludendo questa sezione, si fornisce qui di seguito, la procedura formale che Alice dovrà seguire per ottenere la presentazione Γ nominata nel protocollo descritto in 4.1.2.

1. Alice fissa un numero intero k, $10 \leq k \leq 20$ di generatori per la sua presentazione Γ: $x_1, x_2, \ldots, x_k$.

2. Alice genera m parole casuali appartenenti all'alfabeto $\{x_1, \ldots, x_k\}$. Il numero m di relatori da produrre dovrebbe essere $10 \leq m \leq 30$, con lunghezza $12 \leq l_i \leq 20$ (dove $l_i = |r_i|$ con $i = 1, \ldots, m$). In questo modo Alice ottiene una presentazione Γ ancora "grezza".

3. Alice applica una serie di trasformazioni di Tietze (che verranno descritte nella prossima sezione 4.1.4) per ottenere a partire da Γ una nuova presentazione Γ'. Successivamente, essa elimina da quest'ultima circa la metà dei relatori ottenendo così la presentazione "ridotta" $\hat{\Gamma}$.

4. Una volta calcolata la presentazione $\hat{\Gamma}$, Alice aggiunge ai suoi relatori la relazione[6]:

$$x'_i = \prod_{j=1}^{M} [x'_i, w_j] \tag{4.1}$$

   dove x'_i è una parola casuale sull'alfabeto formato dai generatori di $\hat{\Gamma}$ mentre le w_j sono anch'esse parole casuali sul medesimo alfabeto aventi lunghezza 1 o 2, inoltre $M = 10$. Questa relazione viene aggiunta per rendere vano un eventuale "quotient attack" (vedere sezione 4.2.1). Si ricordi ora quanto detto nel punto (2) del protocollo presentato in 4.1.2: "[...] (ii) $w = e$ in $\hat{\Gamma} \Rightarrow w = e$ in Γ [...]". Tale proposizione non è più vera in quanto nella presentazione $\hat{\Gamma}$ è stata aggiunta una relazione (la 4.1) che in Γ non compare affatto. Per risolvere tale questione Alice calcola la preimmagine di 4.1 attraverso l'isomorfismo esistente tra Γ e $\hat{\Gamma}$ dopodiché aggiunge l'equazione trovata tra i relatori di Γ[7]. In questo modo Γ vanta $m + 1$ defining relator.

5. Alice controlla se la presentazione finita Γ soddisfa la condizione $C'(\frac{1}{6})$ (quasi sicuramente sarà così [7]); in caso contrario ripete nuovamente la procedura dal punto (1).

[6] La notazione per il *commutatore* usata in questa tesi è : $[a, b] = a^{-1}b^{-1}ab$.

[7] Nel caso non si voglia usare una presentazione ibrida (quindi con relatori e relazioni mescolati insieme) si può facilmente convertire una relazione in un relatore: ad esempio un'eventuale relazione $r = ab^{-1}cd$ può essere convertita in questo relatore: $rd^{-1}c^{-1}ba^{-1}$.

4.1.4 Trasformazioni di Tietze: isomorfismi elementari

In questa sezione viene spiegato come Alice potrà effettivamente eseguire il passo (2) del protocollo descritto nell'introduzione. Per prima cosa verranno presentate le trasformazioni di Tietze: qualsiasi isomorfismo tra due gruppi finitamente presentati si può ottenere tramite una sequenza finita di tali trasformazioni.
Un'altra cosa veramente importante è che ogni trasformazione di Tietze è facilmente invertibile: Alice quindi, può sempre calcolare l'isomorfismo inverso che conduce da Γ' a Γ.
Un gruppo G può avere molte presentazioni, infatti dato un suo insieme di elementi generatori ci sono molte collezioni possibili di defining relator che possono essere utilizzate per identificare il medesimo gruppo G; si continua seguendo la trattazione di [5].
Nel 1908, H. Tietze dimostrò che, data una presentazione:

$$G \;=\; \langle x_1, x_2 \dots \mid r_1, r_2 \dots \rangle \tag{4.2}$$

di un gruppo G ogni altra possibile presentazione dello stesso gruppo si può ottenere applicando a 4.2 un certo numero (finito) delle seguenti trasformazioni:

(T1) *Introduzione di un nuovo generatore*: se K è una parola nell'alfabeto $\{x_1, x_2 \dots\}$ allora in 4.2 si aggiunga un simbolo y tra i generatori e, sempre in 4.2, tra i relatori si introduca yK^{-1} (oppure $K^{-1}y$ o $y^{-1}K$ o Ky^{-1} o la relazione $y = K$).

(T2) *Cancellazione di un generatore*: se un relatore in 4.2 è della forma yK (o compare la relazione $y = K^{-1}$), dove y è un generatore e K è una parola in cui, tra le lettere che la compongono, non compare il simbolo y allora si può togliere quest'ultimo(a) dall'insieme dei generatori e cancellare yK tra i relatori. Dopodiché sostituire y con K^{-1} nei defining relator rimanenti.

(T3*) *Cancellazione di un relatore*: se un relatore elencato tra $\{r_1, r_2, \dots\}$ può essere derivato[8] dagli altri defining relator presenti allora può essere cancellato da tale insieme.

(T4*) *Introduzione di un nuovo relatore*: se una parola W è derivabile da $\{r_1, r_2, \dots\}$ allora la si può aggiungere alla lista dei defining relator in 4.2.

Esempio 4.4. Si mostra che il gruppo

$$\langle a, b, c \mid (ab)^2 ab^2 \rangle$$

è un gruppo libero con due generatori. Per far ciò si introducono due nuovi generatori usando (T1), x e y; tra i relatori vengono aggiunti $xb^{-1}a^{-1}$ (corrispondente a $x = ab$) e $yb^{-2}a^{-1}$ (dunque $y = ab^2$). Si ottiene dunque:

$$\langle a, b, c, x, y \mid (ab)^2 ab^2, xb^{-1}a^{-1}, yb^{-2}a^{-1} \rangle$$

[8] Per la definizione di "derivabile" si veda la sezione 1.3.2 del capitolo 1.

Applicando (T4*) si aggiunge il relatore x^2y alla presentazione. Esso può essere banalmente derivato dagli altri relatori: $x^2y = x^2[(ab)^2ab^2]^{-1}y = (ab)^2[(ab)^2ab^2]^{-1}ab^2 = e$. La presentazione ottenuta è la seguente:

$$\langle a, b, c, x, y \mid (ab)^2ab^2, x^2y, xb^{-1}a^{-1}, yb^{-2}a^{-1}\rangle$$

Ora anche il relatore $(ab)^2ab^2$ può essere derivato dagli altri, infatti $[x^2]^{-1}(ab)^2ab^2[y]^{-1} = e$, quindi utilizzando (T3*) può essere eliminato dall'insieme dei defining relator:

$$\langle a, b, c, x, y \mid x^2y, xb^{-1}a^{-1}, yb^{-2}a^{-1}\rangle$$

Si aggiungono altri due relatori mediante (T4*): $by^{-1}x$ e $ax^{-1}yx^{-1}$. Si ottiene allora:

$$\langle a, b, c, x, y \mid x^2y, xb^{-1}a^{-1}, yb^{-2}a^{-1}, by^{-1}x, ax^{-1}yx^{-1}\rangle$$

Ora, applicando (T2) si possono cancellare i due generatori a e b:

$$\langle c, x, y \mid x^2y, xy^{-1}xx^{-1}yx^{-1}, y(y^{-1}x)^2x^{-1}yx^{-1}\rangle$$

Il relatore $xy^{-1}xx^{-1}yx^{-1}$ è un relatore banale, e si può tranquillamente eliminare; mentre $y(y^{-1}x)^2x^{-1}yx^{-1}$ può essere derivato dagli altri relatori, infatti $y(y^{-1}x)^2x^{-1}yx^{-1} = y^{-1}xyx^{-1} = y^{-1}(x^{-1}x)xyx^{-1} = y^{-1}x^{-1}[x^2y]x^{-1} = y^{-1}x^{-2} = y^{-1}x^{-2}x^2y = e$. Applicando (T3*) si ottiene:

$$\langle c, x, y \mid x^2y\rangle$$

La parola yx^2 è derivabile dal relatore x^2y infatti $y(x^2y)^{-1}x^2 = yy^{-1}x^{-2}x^2 = e$, quindi la si può aggiungere tra i relatori, usando (T4*)

$$\langle c, x, y \mid x^2y, yx^2\rangle$$

Ora il relatore yx^2 permette di usare (T2) eliminando quindi il generatore y, ottenendo così la presentazione "classica" di un gruppo libero con due generatori:

$$\langle c, x \mid \emptyset\rangle$$

In realtà Shpilrain e Zapata, nel loro articolo [2], presentano le trasformazioni di Tietze (T3*) e (T4*) in maniera leggermente diversa (ma del tutto equivalente):

(T3) *Applicazioni di un automorfismo*: si applica un automorfismo del gruppo libero generato da $\{x_1, x_2, \dots\}$ a tutto l'insieme dei relatori $\{r_1, r_2, \dots\}$. La definizione è abbastanza vaga tuttavia, per quanto concerne il protocollo in questione, si useranno solo automorfismi del tipo: $x_i \to x_i{x_j}^{\pm 1}$ o $x_i \to {x_j}^{\pm 1}x_i$ per un certo i, mentre $x_k \to x_k\ \forall k \neq i$. Per maggior chiarezza, vedere l'esempio 4.6.

(T4) *Cambiare i defining relator*: rimpiazzare l'insieme $r_1, r_2, \dots$ dei defining relator con un altro insieme $r'_1, r'_2, \dots$ avente la stessa chiusura normale[9]. Ciò significa che ciascun $r'_1, r'_2, \dots$ appartiene al sottogruppo normale (del gruppo libero generato da $\{x_1, x_2, \dots\}$) generato da $r_1, r_2, \dots$ e viceversa.

In questa tesi si intenderanno le trasformazioni di Tietze come descritte in (T1), (T2), (T3) e (T4).

Esiste anche una versione "ricorsiva" di (T4), utile nel caso si voglia implementare questa particolare trasformazione in pratica; infatti non è molto chiaro come un calcolatore possa sostituire un insieme di parole con un altro, avente la stessa chiusura normale:

(T4′) Si può aggiungere all'insieme dei defining relator R un elemento scelto tra questi: $r_i^{-1}, r_i r_j, r_i r_j^{-1}, r_j r_i, r_j r_i^{-1}, x_k^{-1} r_i x_k, x_k r_i x_k^{-1}$ dove $r_i \neq r_j$ sono entrambi relatori in R e x_k è un simbolo generatore qualsiasi.

Si suggerisce di applicare molte trasformazioni di tipo (T4′) nella parte (2) del protocollo per "mescolare" la presentazione originale Γ. In particolare dopo alcune trasformazioni di quel tipo l'insieme R dei relatori non dovrebbe più soddisfare la proprietà $C'(\frac{1}{6})$. Un altro espediente che Alice potrebbe usare per confondere il suo avversario, consiste nel calcolare il prodotto libero tra il suo gruppo e il gruppo banale presentato in una forma diversa da quella standard. Per fare ciò Alice può aggiungere alla sua presentazione dei nuovi generatori $z_1, \dots, z_q$ e dei nuovi relatori $s_1(z_1, \dots, z_q), \dots, s_2(z_1, \dots, z_q)$ tali che la presentazione $\langle z_1, \dots, z_q \mid s_1, \dots, s_t \rangle$ definisca il gruppo banale.
Dopodiché è consigliabile applicare alcune trasformazioni di tipo (T3) e (T4′) per amalgamare i nuovi relatori con i precedenti. Molte presentazioni non-standard del gruppo banale sono elencate in [8], se ne riportano alcune a titolo di esempio:

1. $\langle x, y | x^2 = y^3, xyx = yxy \rangle$
2. $\langle x, y | x^{-1}yx = y^2, x = yx^2yx^{-2} \rangle$
3. $\langle x, y | x^{-1}y^2x = y^3, x^2 = yxy^{-1} \rangle$
4. $\langle x, y | x^{-1}y^2x = y^3, x^2 = yxy \rangle$

Come si può notare, nelle presentazioni del gruppo banale elencate qui sopra, il numero di relatori è pari al numero di generatori; questa forma particolare di presentazioni è detta *bilanciata* ("balanced presentation").
Vale inoltre il seguente teorema [8]:

[9]Si ricorda che la CHIUSURA NORMALE (in inglese "normal closure" o anche "conjugate closure") di un sottoinsime S di un gruppo G è il sottogruppo, sempre di G, generato da $S^G = \{g^{-1}sg$ dove $g \in G$ e $s \in S\}$ e si indica con $\langle S^G \rangle$ o con $\langle S \rangle^G$. Esso è sempre un sottogruppo normale di G.

Teorema 4.5. *Qualunque presentazione del tipo:*

$$\langle x, y, z \mid x = z \cdot [y^{-1}, x^{-1}, z], y = x \cdot [y^1, x^{-1}, z^{-1}] \cdot [z^1, x], w \rangle$$

dove w è una parola in $\{x, y, z\}$ la cui somma degli esponenti in x, y e z è pari a ± 1 presenta il gruppo banale.[10]

Alice dopo aver mescolato la presentazione Γ utilizzando i metodi precedentemente descritti, dovrebbe spezzare i relatori in modo che essi siano composti al massimo da quattro simboli (intuitivamente questo permette una diffusione[11] più semplice ed efficiente della presentazione). Segmentare i relatori non è un'operazione difficile, basta infatti applicare una serie di trasformazioni di tipo (T1); dopodiché "stagionare" la presentazione ottenuta mediante alcune trasformazioni (T3), sempre per ottenere una diffusione migliore.
Sia Γ una presentazione $\langle x_1, \ldots, x_k \mid r_1, \ldots, r_m \rangle$; si espone qui di seguito una procedura che prevede l'introduzione di nuovi generatori al fine di diminuire la lunghezza dei defining relator presenti. Sia $r_1 \equiv x_i x_j u$ con $1 \leq i, j \leq k$ mentre u è una parola (anche vuota) in $\{x_1, \ldots, x_k\}$. Si introduce ora un nuovo generatore x_{k+1} e un nuovo relatore $r_{m+1} \equiv x_{k+1}^{-1} x_i x_j$ (esattamente come dice (T1)).
La presentazione $\langle x_1, \ldots, x_k, x_{k+1} \mid r_1, \ldots, r_m, r_{m+1} \rangle$ è ovviamente isomorfa a Γ. Si prosegue introducendo un nuovo relatore $r_1' \equiv x_{k+1} u$, a questo punto si nota facilmente che r_1 può essere derivato dagli altri, infatti: $r_1 \equiv x_i x_j u = x_i x_j r_{m+1}^{-1} u = x_{k+1} u \equiv r_1'$ e quindi si può eliminare dall'insieme dei defining relator. Si ottiene dunque la presentazione $\langle x_1, \ldots, x_k, x_{k+1} \mid r_1', \ldots, r_m, r_{m+1} \rangle$ anch'essa isomorfa a Γ ma la lunghezza di uno dei suoi defining relator, nella fattispecie r_1, è diminuita di 1.
Ripetendo questa procedura Alice può abbassare a piacere la lunghezza dei relatori (naturalmente questo comporta una crescita del numero di generatori). L'obiettivo finale è quello di averli tutti (o quantomeno la maggior parte di essi) espressi mediante parole di lunghezza 3 o 4 (non più corti). I relatori più lunghi eventualmente possono anche essere scartati nella fase in cui da Γ' se ne eliminano circa a metà, al fine di ottenere la presentazione ridotta $\hat{\Gamma}$.
Si conclude questa sottosezione con un semplice esempio che mostra come le trasformazioni di Tietze possono essere utilizzate per segmentare i relatori al fine di poterli rappresentare mediante parole "non troppo lunghe". In particolare nell'esempio di partirà con una presentazione avente due relatori di lunghezza 5 e tre generatori e si raggiungerà, alla fine, una presentazione isomorfa con quattro relatori di lunghezza 3 o 4 e cinque generatori. Si ricorda che il simbolo $\cong$ significa "isomorfo a".

Esempio 4.6.

$$\langle x_1, x_2, x_3 \mid x_1^2 x_2^3,\ x_1 x_2^2 x_1^{-1} x_3 \rangle \cong$$

[10] La notazione di commutatore può essere ulteriormente estesa ricorsivamente in questo modo: $[X_1 \ldots X_{n+1}] = [[X_1 \ldots X_n] X_{n+1}]$ per sottoinsiemi $X_1 \ldots X_{n+1}$ di un gruppo G.

[11] L'operazione di mascherare un elemento prima della sua trasmissione, in ambito crittografico spesso si indica con *diffusione*. Tale concetto viene approfondito nella sezione 4.3.2 di questo capitolo.

$$\cong \langle x_1, x_2, x_3, x_4 \mid x_4 = x_1^2,\ x_4 x_2^3,\ x_1 x_2^2 x_1^{-1} x_3 \rangle \cong$$

$$\cong \langle x_1, x_2, x_3, x_4, x_5 \mid x_5 = x_1 x_2^2,\ x_4 = x_1^2,\ x_4 x_2^3,\ x_5 x_1^{-1} x_3 \rangle$$

Applicando la trasformazione di tipo (T3): $x_5 \to x_1 x_5,\ x_i \to x_i,\ i \neq 5$ si ottiene la presentazione isomorfa:

$$\langle x_1, x_2, x_3, x_4, x_5 \mid x_5 = x_2^2,\ x_4 = x_1^2,\ x_4 x_2^3,\ x_1 x_5 x_1^{-1} x_3 \rangle \cong$$

$$\cong \langle x_1, x_2, x_3, x_4, x_5 \mid x_2^2 x_5^{-1},\ x_1^2 x_4^{-1},\ x_4 x_2^3,\ x_1 x_5 x_1^{-1} x_3 \rangle$$

4.1.5 Generare elementi casuali in gruppi finitamente presentati

In questa sottosezione sono descritte le operazioni che Bob dovrà eseguire durante il passo (3) del protocollo:

- per spedire un elemento pari a e in $\hat{\Gamma}$, egli dovrebbe riuscire a costruire una parola, usando come alfabeto gli elementi di $\bar{R}_{\hat{\Gamma}}$ il più *casualmente* possibile ;
- per spedire un elemento diverso da e in $\hat{\Gamma}$, Bob deve semplicemente comporre una parola *casuale* sufficientemente lunga, usando come alfabeto i generatori di $\hat{\Gamma}$. Come è già stato anticipato, con ottime probabilità, una stringa di questo tipo non sarà uguale all'elemento neutro (tale questione viene approfondita alla fine di questa sottosezione);

In una **parola uguale a e in $\hat{\Gamma}$** trasmessa da Bob, i relatori che la compongono, non devono essere facilmente individuabili (e nemmeno parti abbastanza grandi di essi). L'unica proprietà di $\hat{\Gamma}$ che si può sfruttare per celare la struttura di una stringa che rappresenta l'elemento neutro è che la maggior parte dei defining relator sono di lunghezza 3 o 4.
Ecco la procedura nel dettaglio, con $\hat{\Gamma} = \langle x'_1, x'_2, \dots\ x'_k \mid \hat{r_1}, \hat{r_2}, \dots \rangle$.

1. Creare una parola della forma: $u = s_1 \dots s_p$ dove ciascun s_i è un elemento scelto casualmente dall'insieme dei defining relator $\{\hat{r_1}, \hat{r_2}, \dots\}$ unito all'insieme dei loro inversi e dei loro coniugati in questa forma: $\alpha s_i \alpha^{-1}$ dove α è una parola sull'alfabeto dei generatori $\{x'_1, x'_2, \dots\}$ di lunghezza 1 o 2. La quantità p di lettere che compongono u deve essere sufficientemente grande: almeno dieci volte il numero di defining relator in $\hat{\Gamma}$.
2. Inserire circa $\frac{2p}{k}$ espressioni della forma $x'_j (x'_j)^{-1}$ o $(x'_j)^{-1} x'_j$ in posizioni casuali all'interno della parola u (qui k indica il numero di generatori x'_i di $\hat{\Gamma}$), per valori aleatori di j nell'intervallo $[1, k]$.

3. Andando da sinistra a destra lungo u, ogniqualvolta si incontri una coppia di lettere consecutive che fa parte di un defining relator, sostituirla con l'inverso della parte rimanente del medesimo relatore. Ad esempio, se ci fosse un relatore $r'_i = x'_1x'_2x'_3x'_4$ nel caso si incontri una sottoparola $x'_1x'_2$ in u allora dovrebbe essere sostituita da $(x'_4)^{-1}(x'_3)^{-1}$ (ovviamente $x'_1x'_2 = (x'_4)^{-1}(x'_3)^{-1}$); nel caso si trovasse la coppia $x'_2x'_3$ sempre in u allora la si dovrebbe cambiare in $(x'_1)^{-1}(x'_4)^{-1}$. Se ci fosse più di una possibilità per il rimpiazzo di una sottoparola, se ne sceglie semplicemente una a caso.

4. Cancellare eventuali sequenze del tipo $x'_j(x'_j)^{-1}$ o $(x'_j)^{-1}x'_j$ da u.

I passi (2)-(4) andrebbero ripetuti circa p volte.

Si indica ora, sempre con u, la parola ottenuta dopo aver eseguito la procedura sopra descritta. Bob successivamente compone la parola $w = [x'_i, u]$ e applica nuovamente le fasi (2)-(4) approssimativamente $\frac{|w|}{2}$ volte, dove $|w|$ è la lunghezza di w.
Questa fase finale serve a rendere la parola w in questione (che è uguale a e in $\hat{\Gamma}$ e quindi anche in Γ') indistinguibile da un'altra parola, diversa dall'elemento neutro: anch'essa infatti sarà della stessa forma: $w = [x'_i, u]$ (come verrà descritto tra poco). Qui x'_i è lo stesso generatore usato per costruire la relazione $x'_i = \prod_{j=1}^{M}[x'_i, w_j]$ inserita da Alice nel passo (3) del protocollo.
Come verrà spiegato nella sottosezione 4.2.1, esprimere anche le parole $w \neq e$ in questa forma è necessario sia per rendere inefficace un eventuale "quotient attack" sia per una questione di sicurezza semantica.

Quando Bob vuole trasmettere una **parola diversa da *e* in $\hat{\Gamma}$** per prima cosa deve scegliere una parola casuale u appartenente al sottogruppo dei commutatori[12]. Scegliere una stringa di questo tipo è facile: per prima cosa si genera una sequenza casuale di generatori v, dopodiché si modificano gli esponenti delle lettere che la compongono in modo che la somma degli esponenti, per ciascun simbolo, sia pari a zero [2]. La lunghezza di u dovrebbe essere circa la stessa della parola $u = e$ in $\hat{\Gamma}$ costruita da Bob nel capoverso precedente. Bob ora calcola $w = [x'_i, u]$, dove x'_i è lo stesso generatore usato per costruire la relazione $x'_i = \prod_{j=1}^{M}[x'_i, w_j]$ inserita da Alice nel passo (3) del protocollo. Infine per "nascondere" u, Bob applica le fasi (2)-(4) approssimativamente $\frac{|w|}{2}$ volte, dove $|w|$ è la lunghezza di w.

> Concludendo questa sezione si spiega come mai, una parola casuale sufficientemente lunga, quasi sicuramente non sarà pari a 1 in Γ (tale presentazione si intende costruita come nel passo (3) del protocollo); inoltre si parlerà anche di sicurezza semantica.

[12]Sia G un gruppo. Il sottogruppo generato da tutti i commutatori di G è detto SOTTOGRUPPO DEI COMMUTATORI.

Come ogni altro gruppo, anche il gruppo G presentato da $\Gamma = \langle x_1, x_2 \dots \mid R \rangle$ è pari al gruppo quoziente $G = F/N$ (si veda la sottosezione 1.3.2 del capitolo 1) dove N è la chiusura normale[13] dell'insieme dei definig relator R, e F è il gruppo libero generato da $\{x_1, x_2 \dots\}$. Detto ciò, per stimare la probabilità che una parola casuale in $x_1, x_2 \dots$ non appartenga a N (e quindi non sia uguale a e in G) si dovrebbe studiare la DENSITÀ ASINTOTICA del complemento di N nel gruppo libero F [9]. Risulta più semplice tuttavia lavorare con la densità asintotica di R stesso, la sua definizione formale è:

$$\rho_F(N) = \lim_{n\to\infty} \left\lceil \frac{|\{u \in N : \ |u| \leq n\}|}{|\{u \in F : \ |u| \leq n\}|} \right\rceil$$

Si deduce che la che la densità asintotica dipende, in generale, dalla base di F $\{x_1, x_2 \dots\}$; inoltre è degno di nota il seguente risultato ottenuto da Woess [10]:

Proposizione 4.7. *Se il gruppo $G = F/R$ è infinito allora $\rho_F(N) = 0$*

Poiché il gruppo presentato da Γ (vedere sezione 4.1.3) è infinito la proposizione 4.7 garantisce la correttezza della procedura che Bob attua per produrre una parola diversa da e.
Come si può notare nella definizione di $\rho_F(N)$ la variabile n tende a infinito. Naturalmente le parole trasmesse da Bob devono essere di lunghezza ragionevole (dai 100 ai 200 simboli). Risulta molto importante dunque conoscere quanto rapidamente converge a zero, il valore della frazione che definisce la densità asintotica.
Sempre Woss dimostrò che nei *gruppi non amenabili* (i gruppi di tipo "small cancellation" sono una particolare categoria di gruppi non amenabili) la "velocità" di tale convergenza è esponenziale. La questione in realtà è molto complicata ed esula dalla trattazione di questa tesi, per approfondimenti si veda [10].
Si noti dunque che i gruppi small cancellation sono doppiamente adatti per questo protocollo:

1. hanno word problem risolvibile, mediante l'algoritmo di Dehn;
2. la probabilità che una parola casuale di lunghezza n in $\{x_1, x_2 \dots\}$ non sia uguale a e in G converge esponenzialmente a 1 per $n \to \infty$;

Per concludere si prende in considerazione la sicurezza semantica delle parole inviate da Bob durante la trasmissione protetta. La trattazione è volutamente informale in quanto introdurre una misura di probabilità in un gruppo infinito è tutt'altro che banale (si veda [11]). I principi fondamentali della sicurezza semantica sono rispettati da questo protocollo. Ad esempio quando Bob deve costruire una parola $u \neq e$ semplicemente assembla una *sequenza casuale* di caratteri; in essa un attaccante non potrà trovare alcuna struttura utile per una decodifica illecita. Si potrebbe obiettare facendo notare il fatto che in realtà, al posto di uno "0", non viene inviata proprio una sequenza casuale di lettere bensì la parola $w = [x'_i, u]$ che non sembra affatto aleatoria. A questo punto, bisogna ricordare che:

[13] Si ricorda che la CHIUSURA NORMALE (in inglese "normal closure" o anche "conjugate closure") di un sottoinsime S di un gruppo G è il sottogruppo, sempre di G, generato da $S^G = \{g^{-1}sg$ dove $g \in G$ e $s \in S\}$ e si indica con $\langle S^G \rangle$ o con $\langle S \rangle^G$. Esso è sempre un sottogruppo normale di G.

- l'effettiva parola trasmessa da Bob *rappresenta* l'elemento $[x'_i, u]$ tuttavia $w \not\equiv [x'_i, u]$. Infatti Bob, come descritto all'inizio di questa sottosezione, applica su di essa numerose operazioni per "distruggere" la sua struttura;

- date le specifiche del protocollo, la cosa veramente importante è che una parola uguale a e in $\hat{\Gamma}$ sia indistinguibile da una parola diversa da e. Ecco perché gli elementi uguali a e in $\hat{\Gamma}$ inviati da Bob, sono anch'essi della forma $[x'_i, u]$;

Detto questo la questione della sicurezza semantica, per quanto riguarda questo protocollo, si riduce alla seguente domanda: "una parola u che rappresenta l'elemento neutro in $\hat{\Gamma}$ è indistinguibile da una parola casuale della stessa lunghezza?".
Pur non essendo disponibile una dimostrazione formale, esperimenti svolti al computer condotti da Shpilrain e Zapata (in [2]) hanno mostrato che, quando la maggior parte dei relatori hanno lunghezza al massimo 4, la risposta alla domanda quasi sicuramente è "si". Infatti l'analisi delle frequenze condotta sulla parola u in questione, per quanto riguarda i singoli caratteri e tutte le sottoparole di lunghezza 2 e 3, non porta a nessun risultato particolare (se una parola casuale fosse sottoposta alla medesima analisi si otterrebbero gli stessi risultati).

4.1.6 Parametri suggeriti

In questa sottosezione, si presenta una raccolta di tutti i parametri suggeriti in questo capitolo riguardanti il protocollo di Shpilrain e Zapata.

1. Il numero di generatori x_i nella presentazione privata di Alice Γ è k, un numero random appartenente all'intervallo $10 \leq k \leq 20$.

2. I relatori $r_1, \ldots, r_m$ della presentazione privata di Alice Γ sono parole casuali a partire dall'alfabeto $\{x_1 \ldots, x_k\}$. Qui m è un numero random appartenente all'intervallo $10 \leq m \leq 30$, e la lunghezza l_i di r_i parte da un minimo di 12 per arrivare ad un massimo di 20 caratteri. Inoltre, a questi relatori ne va aggiunto un altro, di natura particolare; per i dettegli vedere la sottosezione 4.1.3.

3. L'isomorfismo che relaziona le due presentazioni Γ e Γ', noto *solamente* ad Alice, è costituito da un prodotto, in ordine casuale di s_1 trasformazioni di tipo (T1) e s_2 trasformazioni di tipo (T3) (vedere la sottosezione 4.1.4). Ogni relatore introdotto mediante (T1) è semplicemente una parola casuale la cui lunghezza è un numero random intero appartenente all'intervallo $[12, 20]$. I paramenti s_1 e s_2 non sono specificati, tuttavia la loro somma dovrebbe essere come minimo 50; più precisamente, man mano che $s_1 + s_2$ si avvicina a tale quantità, si dovrebbero utilizzare sempre più (T1) al fine di abbattere la lunghezza dei relatori; circa il 30% di essi, alla fine di questa procedura, dovrebbe avere lunghezza al massimo 4. Dopo aver fatto ciò Alice scarta il 70% dei relatori in maniera tale che almeno il 50% di quelli sopravvissuti abbia lunghezza al massimo 4.

4. Bob cifra la sua sequenza binaria sensibile, spedendo al posto di ogni bit, una parola prodotta come descritto nella sezione 4.1.5. In questo paragrafo si preciserà la lunghezza di tali stringhe. Si ricorda il fatto che Bob comincia a costruire una parola $w = e$ in $\hat{\Gamma}$ mediante un prodotto di p parole scelte casualmente tra l'insieme dei defining relator, compresi i loro inversi e i loro coniugati (quest'ultimi ottenuti considerando solo parole, sull'alfabeto dei generatori, di una o due lettere). Il parametro p è un numero intero random scelto all'interno dell'intervallo $[5, 12]$, in questo modo una parola w sarà lunga circa 150. Le seguenti operazioni di diffusione aumentano di poco la lunghezza della parola, non intaccando l'efficienza del protocollo [2].
Infine, si ricorda che Bob crea una parola $w \neq e$ in $\hat{\Gamma}$ della forma $[x'_i, u]$, dove u è una parola casuale appartenente al sottogruppo dei commutatori del gruppo libero generato da $x'_1, x'_2, \ldots$ (i generatori della presentazione pubblica $\hat{\Gamma}$). La lunghezza l di u è un numero random intero scelto nell'intervallo $[65, 85]$. In questo modo la lunghezza di $w = [x'_i, u]$, pari a $2l + 2$, risulta essere circa 150, proprio come nel caso $w = e$ considerato nel capoverso precedente.

4.2 Possibili attacchi

4.2.1 Quotient attack al protocollo Shpilrain-Zapata

In questa sottosezione viene presentato un possibile attacco al protocollo di Shpilrain-Zapata, dopodiché si analizzerà la sua efficacia.
In quest'ultima parte di questo capitolo si utilizzerà un terminologia riguardante la teoria dei gruppi non presentata nel capitolo 1; è necessaria quindi una (breve) introduzione.
Un gruppo G si dice ABELIANO (o commutativo) se $[a, b] = e_G \ \forall a, b \in G$, dove $[a, b] = a^{-1}b^{-1}ab$ (si veda alche la definizione 1.3 del capitolo 1). Tale concetto può essere generalizzato in molti modi diversi.

Definizione 4.8. *Un gruppo G è detto* METABELIANO *se $[[x, y], [z, t]] = e_G \ \forall x, y, z, t \in G$ esplicitando la scrittura: $(w^{-1}x^{-1}wx)(y^{-1}z^{-1}yz) = (y^{-1}z^{-1}yz)(w^{-1}x^{-1}wx) \ \forall w, x, y, z \in G$. Equivalentemente un gruppo G è metabeliano se il suo sottogruppo dei commutatori è abeliano. Tutti i gruppi abeliani sono banalmente metabeliani.*

Definizione 4.9. *La* SERIE CENTRALE DISCENDENTE *di un gruppo G è definita come la catena di sottogruppi normali:*

$$G = G_1 \trianglerighteq G_2 \trianglerighteq \cdots \trianglerighteq G_n \trianglerighteq \cdots$$

dove $G_{n+1} = [G_n, G]$ dunque il sottogruppo generato da tutti i commutatori $[x, y]$ tali che $x \in G_n$ e $y \in G$.

Si noti che $G_2 = [G, G]$ è il sottogruppo dei commutatori di G, a volte chiamato anche *sottogruppo derivato* di G.

Definizione 4.10. *Si consideri la seguente definizione ricorsiva:*

$$G^{(0)} \stackrel{\triangle}{=} G$$
$$G^{(n)} \stackrel{\triangle}{=} [G^{(n-1)}, G^{(n-1)}]$$

Dalla definizione si ricava che $G^{(1)} = [G, G]$ *dunque rappresenta il sottogruppo dei commutatori (o sottogruppo derivato);* $G^{(2)} = [[G, G], [G, G]]$ *è detto secondo sottogruppo derivato;* $G^{(3)}$ *terzo sottogruppo derivato e così via.*
La seguente catena:

$$\cdots \triangleleft G^{(2)} \triangleleft G^{(1)} \triangleleft G^{(0)} = G$$

è detta SERIE DERIVATA *del gruppo* G*.*

Definizione 4.11. *Un gruppo* G *è detto* NILPOTENTE DI CLASSE $c \geq 1$ *se la sua serie centrale discendente conduce al gruppo banale in* c *passi.*

Eva, utilizzando un quotient attack che verrà descritto a breve, potrebbe cercare di identificare nel flusso di parole che Bob spedisce ad Alice, quelle che rappresentano gli "0" della sequenza binaria originale. Per far ciò Eva aggiunge alla presentazione pubblica $\hat{\Gamma}$ molti altri relatori (a sua scelta) cercando di ottenere la presentazione di un gruppo H avente word problem risolvibile, o più precisamente, la presentazione di un gruppo H per il quale Eva possa facilmente *riconoscere* la risolvibilità del word problem. La procedura appena descritta di chiama *quotient test* (vedere [9]).

Usando una metafora, il quotient test potrebbe essere spiegato così: Eva cerca di costruire un setaccio utile per filtrare gli elementi che Bob invia ad Alice. Il setaccio fa passare gli elementi uguali a e mentre trattiene quelli diversi dall'elemento neutro. L'attaccante parte con un setaccio avente maglie strettissime (inutilizzabile) e, passo dopo passo, allarga sempre di più la trama dello strumento.

Si sta delineando quindi una situazione di questo tipo: appena l'intruso riconosce un quoziente H con word problem risolvibile ha finalmente concluso la costruzione del setaccio. Può ora testare le parole intercettate su questo gruppo:

- se una parola w risultasse diversa da e allora sicuramente Bob, con essa, ha cifrato uno "0" (si ricordi che per ottenere H Eva ha solamente aggiunto relatori);
- nel caso $w = e$ in H si hanno due possibilità:
  (i) w è pari a e anche in $\hat{\Gamma}$ quindi, a maggior ragione, anche in H.
  (ii) $w = e$ in H è un falso positivo; a causa dell'allargamento della maglia filtrante, attraversano il setaccio anche elementi diversi da e.

L'obiettivo di Eva dunque, è quello di raggiungere, mediante l'inserimento di relatori, dei quozienti riconoscibili; ad esempio abeliani o in generale nilpotenti. Si capisce allora che Eva non può aggiungere relatori casuali, ma solamente quelli che vantano una determinata struttura in modo da indirizzare l'evoluzione della presentazione verso un quoziente utile per la decifratura.

Esempio 4.12. Per ottenere un quoziente abeliano Eva dovrebbe aggiungere i relatori $[x'_i, x'_j]$ per ogni coppia di generatori x'_i, x'_j in $\hat{\Gamma}$.

Di fatto un quotient attack a questo protocollo può essere implementato servendosi solamente di un quoziente nilpotente o di un quoziente metabeliano di $\hat{\Gamma}$ [2].
Proprio per far fallire questo genere di attacco, Alice aggiunge il relatore $x'_i = \prod_{j=1}^{M}[x'_i, w_j]$ a $\hat{\Gamma}$ (vedere la sezione 4.1.3) e per lo stesso motivo, Bob costruisce una parola della forma $[x'_i, u]$ quando vuole trasmettere una parola diversa da 1 in $\hat{\Gamma}$ (vedere la sezione 4.1.5). Un quotient attack di tipo metabeliano su un elemento della forma $[x'_i, u] \neq e$ non produrrà alcun risultato utile: dato che $x'_i = \prod_{j=1}^{M}[x'_i, w_j]$, l'elemento x'_i appartiene al sottogruppo dei commutatori (o sottogruppo derivato) quindi $[x'_i, u]$ appartiene al secondo gruppo derivato.
In questo modo (vedere la definizione di gruppo metabeliano 4.8) in un quoziente metabeliano un elemento di questo genere sarebbe sempre uguale a e (si ricordi che l'elemento $[x'_i, u]$ prodotto da Bob invece, non è uguale a e in $\hat{\Gamma}$).
Inoltre un elemento della forma $[x'_i, u]$ appartiene ad ogni termine della serie centrale discendente, infatti:

$$[x'_i, u] = [\prod_{j=1}^{M}[x'_i, w_j], u] = [\prod_{j=1}^{M}[\prod_{j=1}^{M}[x'_i, w_j], w_j], u] = \dots$$

Questa serie di uguaglianze fa fallire anche un quotient attack nilpotente.

4.3 Teoria combinatoria dei gruppi e crittografia a chiave pubblica

Dopo le prime due sezioni del capitolo 4 si sente la necessità di formalizzare il concetto di sistema di cifratura a chiave pubblica basato su un "problema difficile" della teoria combinatoria dei gruppi [12].
Come già detto, negli ultimi anni c'è stato (e c'è ancora), un gran fermento a riguardo: oltre al protocollo di Shpilrain e Zapata descritto nella sezione 4.1.2, un altro protocollo degno di nota è quello proposto da Anshel-Anshel-Goldfeld e da Ko-Lee in [13] e in [14] rispettivamente. Il forte sviluppo in questo ramo e la continua nascita di protocolli di questo tipo, ha comportato una riscrittura della definizione di crittosistema.

4.3.1 Crittosistemi algebrici a chiave pubblica

La componente principale di un crittosistema a chiave pubblica è una *funzione unidirezionale*; essa costituisce il "nucleo della sicurezza" attorno al quale si sviluppa e viene implementato l'intero sistema di cifratura.

Definizione 4.13. *Siano S e T due insiemi. Una* FUNZIONE UNIDIREZIONALE *è una funzione $f : S \to T$ calcolabile efficientemente tale che, data un'immagine $y = f(x)$ è computazionalmente oneroso determinare la preimmagine $x \in S$.*

Si assume che S e T siano strutture algebriche associative aventi una singola operazione binaria (semigruppi, gruppi,...) e un unico elemento neutro e. Questi oggetti matematici, in ambito crittografico, sono detti PIATTAFORME. Sia la coppia $\langle X \mid R \rangle$ una presentazione del semigruppo S, dove $X = \{x_1, x_2, \dots\}$ è l'insieme dei generatori si S e $R = \{r_1 = r'_1, r_2 = r'_2 \dots\}$ è l'insieme delle defining relation.

Definizione 4.14. *L'insieme di tutte le funzioni $S \to S$ chiuso sotto l'operazione di composizione è semigruppo ed è chiamato* SEMIGRUPPO DELLE TRASFORMAZIONI TOTALI *(full transformation semigroup) di S e si indica con $\mathcal{T}_S$.*

Definizione 4.15. *Una funzione $t \in \mathcal{T}_S$ è* BEN DEFINITA *in S se $\forall\ w, w' \in S$ tali che $w = w'$ si ha che $t(w) = t(w')$.*

L'insieme delle funzioni ben definite contenuto in $\mathcal{T}_S$ può essere utilizzato per ottenere una diffusione in S cioè per riscrivere in maniera diversa un elemento appartenente alla piattaforma S utilizzando le defining relations (il concetto di diffusione viene approfondito in 4.3.2).

Proposizione 4.16. *Sia $T \subseteq \mathcal{T}_S$ il sottoinsieme delle funzioni ben definite in S. La funzione $f : S \times T \to S$ descritta da $f(w, t) \to t(w)$ soddisfa il requisito fondamentale di "funzione unidirezionale" se calcolare w da $t(w) = f(w, t)$ è computazionalmente oneroso.*

Definizione 4.17. *Un* CRITTOSISTEMA ALGEBRICO A CHIAVE PUBBLICA *è una tupla $(S, T, f, \mathcal{H}, h)$ che soddisfa le seguenti caratteristiche:*

- *S e T sono strutture algebriche (ad esempio semigruppi);*
- *$f : S \times T \to S$ è una funzione unidirezionale ben definita in S: dato un elemento $t \in T$ (da mantenere segreto) e un qualsiasi elemento $w \in S$, (che può anche essere reso pubblico) è computazionalmente oneroso determinare t a partire da $f(w, t)$ e per ogni $w' \in S$ tale che $w' = w$ si ha che $f(w, t) = f(w', t)$;*
- *$\mathcal{H}$ è un insieme ausiliario di strutture algebriche utili per definire protocolli specifici (scambio di chiavi, codifica, decodifica, eccetera);*
- *$h : X \times Y \to X$ è una funzione ausiliaria (definita per protocolli specifici), dove X e Y appartengono all'insieme $\{S, T, H \in \mathcal{H}\}$;*

Definizione 4.18. *Si selezionino due piattaforme S e T per costruire un crittosistema algebrico a chiave pubblica $(S, T, f, \mathcal{H}, h)$ (vedere la definizione 4.17), dove l'insieme ausiliario $\mathcal{H}$ è $\{A, B \subseteq T$ tale che $\forall\ \alpha \in A$ e $\forall\ \beta \in B$ si ha che $\alpha\beta = \beta\alpha\}$. Il protocollo per*

lo SCAMBIO DI CHIAVI *tra due entità, Alice e Boob, è definito come segue:*

Protocollo

1. *Si considera di dominio pubblico il semigruppo S, una parola $w \in S$ e un insieme di generatori per ciascuno dei semgruppi contenuti in $\mathcal{H}$.*
2. *Alice sceglie una parola $\alpha \in A$ (da mantenere segreta) la quale soddisfi $f(w, \alpha) \neq e$ e trasmette $f(w, \alpha) = w\alpha$ a Bob.*
3. *Bob sceglie una parola $\beta \in B$ (da mantenere segreta) la quale soddisfi $f(w, \beta) \neq e$ e trasmette $f(w, \beta) = w\beta$ ad Alice.*
4. *Alice calcola $f(w\beta, \alpha) = w\beta\alpha$ mentre Bob calcola $f(w\alpha, \beta) = w\alpha\beta$. Entrambe le entità ottengono $w\alpha\beta = w\beta\alpha$ come chiave segreta condivisa.*

Esempio 4.19. Il protocollo Diffie-Hellman per lo scambio di chiavi [4] diventa un'istanza del protocollo definito in 4.18. Nella fattispecie il gruppo moltiplicativo formato da tutti i numeri interi modulo un primo viene scelto come piattaforma.

4.3.2 Diffusione di un elemento

L'operazione di mascherare un elemento prima della sua trasmissione, in ambito crittografico spesso si indica con *diffusione*. Si pensi, ad esempio, a tutte le trasformazione di Tietze che Bob applica ad una parola $w = e$ o ad una parola $w \neq e$ prima di spedirla ad Alice nel protocollo di Shpilrain-Zapata descritto in sezione 4.1.2.
Il perché si debba camuffare un elemento prima della trasmissione è abbastanza ovvio: prendendo sempre come riferimento il crittosistema presentato all'inizio di questo capitolo, se si inviasse una parola pari a e, semplicemente come prodotto di relatori $r_1 r_2 \ldots$ l'avversario, con una semplice ispezione della parola si accorgerebbe che essa rappresenta l'elemento neutro.
Si nota subito che in un crittosistema basato sulla teoria dei numeri la diffusione è "automaticaticamente" fornita dal sistema di notazione decimale usato per rappresentare i numeri coinvolti nella varie fasi dei protocollo. Ad esempio, nel prodotto $7 \cdot 3 = 21$ i fattori 3 e 7 non posso essere riconosciuti con una semplice ispezione.
Nei (semi)gruppi astratti ciò non avviene, il risultato di una "moltiplicazione" è indicato semplicemente concatenando i fattori che lo compongono: $a \cdot b = ab$; risulta evidente che uno sforzo aggiuntivo per "mescolare" un prodotto è assolutamente necessario.
Il meccanismo di mascheramento è di fondamentale importanza per qualsiasi protocollo di cifratura a chiave pubblica basato sulla computazione simbolica.
Naturalmente bisogna garantire che la funzione che si occupa di "travestire" l'elemento che si vuole spedire sia unidirezionale. In un lavoro recente di Myasnikov e Ushakov [15] è emerso che, parlando informalmente, in un gruppo "generico" lo sforzo richiesto per camuffare un elemento e lo sforzo necessario per riportarlo nella sua forma originale sono molto simili. Questo è ovviamente inaccettabile per qualsiasi per qualsiasi applicazione

crittografica.
Al momento attuale la diffusione di protocolli crittografici basati su computazione simboli è ostacolato principalmente proprio dalla mancanza di un metodo generale e accettato dalla comunità che consenta un mascheramento efficace degli elementi.
Come già detto nella sottosezione 4.1.4 di questo capitolo per una diffusione efficace del protocollo Shpilrain-Zapata i due matematici suggeriscono di spezzare i relatori in modo tale che, nella presentazione pubblica $\hat{\Gamma}$ la maggior parte di essi abbia lunghezza 3 o 4. La seguente obiezione tuttavia, è lecita:

> Un'avversario, servendosi sempre delle defining relation, potrebbe riconvertire la presentazione con relatori "brevi" in quella originale.

A questo punto però si deve ricordare che Alice non solo riduce la lunghezza dei relatori, ma dalla presentazione privata Γ ne scarta circa il 70%, ciò rende impossibile ripristinare la presentazione originale a partire da quella pubblica $\hat{\Gamma}$.

4.4 Implementazione del protocollo di Shpilrain e Zapata

L'intero protocollo suggerito dai due matematici Shpilrain e Zapata è stato realizzato in linguaggio C (ambiente di sviluppo Xcode).
Le variabili globali del programma sono:

- `int num_gen`

  rappresenta il numero di generatori, impostato a 20;

- `int num_rel`

  la variabile indica il numero di relatori, di default vale 20;

- `int k_cod`

  indica il numero di coppie del tipo $x_i x_i^{-1}$ che vengono aggiunte (in ogni iterazione) nella parola che Bob spedisce ad Alice per realizzare la diffusione della stessa. Di default vengo aggiunte, ogni volta, 50 coppie del tipo indicato;

- `int n_i`

  rappresenta il numero di cicli di diffusione, sono previsti 2000 iterazioni per raggiungere un mascheramento adeguato;

Le costanti:

- `LMIN_REL`

  Indica la lunghezza minima dei relatori in Γ impostato a 15;

- `LMAX_REL`

  Rappresenta la lunghezza massima dei relatori in Γ, di default vale 20;

- `PERC_REL_SCARTATI`

  Indica la percentuale dei relatori scartati nel passaggio tra la presentazione Γ' e $\hat{\Gamma}$ è impostata a 20;

Le principali variabili del programma sono:

- `generatori[100]`

  È un array contenente tutti i simboli generatori;

- `lunghezze[100]`

  La cella $i-$esima di questo vettore memorizza la lunghezza del realore $i-$esimo;

- `*relatori[100]`

  Il puntatore memorizzato in $i-$esima posizione è collegato alla prima cella di memoria ove è mantenuto il relatore $i-$esimo;

Sono presenti 13 funzioni:

- `void stampa_gen(int *generatori)`

  La funzione stampa la lista dei generatori;

- `void stampa_rel(int *lunghezze, int *relatori[], int nr)`

  Questa procedura elenca, a schermo, la lista dei relatori;

- `void iniz_gen(int *generatori)`

  All'inizio questa funzione popola l'insieme dei generatori;

- `void iniz_rel(int *lunghezze, int *relatori[])`

  Questa procedura si occupa di comporre i relatori a partire dai simboli generatori;

- `void ordina_rel(int *lunghezze, int *relatori[])`

  Questa funzione ordina i relatori per lunghezza;

- `void accorcia_rel(int *lunghezze, int *relatori[], int *generatori,...)`

  Questa procedura è fondamentale nel passaggio da Γ a Γ': si occupa infatti di accorciare la lunghezza dei relatori;

- `void elimina_rel(int *lunghezze, int *relatori[], int *generatori,...)`

  Questa funzione scarta la percentuale indicata di relatori dalla presentazione corrente;

- `int cprimo(int *lunghezze, int *relatori[], int *generatori)`

  Questa procedura effettua il test $C'(\frac{1}{6})$ sull'insieme dei relatori;

- `int riduci (int *parola, int lparola)`

  Questa funzione riduce, se possibile, la parola passata come parametro;

- `void patternmatching(int *pattern, int lpattern, int *testo,...)`

  Una semplice procedura di pattern matching;

- `int codifica_1 (int *lunghezze, int *relatori[],...)`

  Questa funzione si occupa di generare una parola pari a 1;

- `int decodifica (int *lunghezze, int *relatori[], int *generatori,...)`

  Simulazione di decodifica di Alice (si hanno quindi a disposizione le presentazioni Γ, Γ' e $\hat{\Gamma}$);

- `int decodifica_2 (int *lunghezze, int *relatori[], int *generatori,...)`

  Simulazione di decodifica non autorizzata (si è a conoscenza della sola presentazione $\hat{\Gamma}$);

La realizzazione del protocollo è risultata piuttosto complicata in quanto non risulta affatto intuitivo come gestire e manipolare delle presentazioni di gruppi attraverso un programma.
Per facilitare la comprensione degli output a video si sono utilizzati i numeri interi come caratteri, racchiusi tra parentesi tonde.
La realizzazione del protocollo segue pedissequamente le indicazioni di Shpilrain e Zapata (descritte nelle prime sezioni di questo capitolo); si distaccano solo in due punti:

1. il numero di cicli per la diffusione è molto maggiore: 2000 (quello consigliato è 200)
2. la percentuale di relazioni scartate è del 20% (a fronte di quella indicata: 50%)

Test pratici dimostrano che, con questi parametri, un utente non autorizzato riesce, in media, a decifrare correttamente un bit su dieci.
Viene riportato, qui di seguito, un'istanza di comunicazione tra due entità: Alice (destinatario) e Bob (mittente). Come si potrà notare, il fattore di espansione è piuttosto elevato: per codificare un bit si deve comporre una parola avente 693 caratteri e l'operazione di diffusione raddoppia la lunghezza della parola originale.
Per codificare 10 bit, si impiegano[14] (compresa l'operazione di generazione della chiave): 34,33 secondi in media; quindi circa 3 secondi e mezzo per bit.

Generatori della presentazione Γ:

Generatore 1 : 1	Generatore 11 : 11
Generatore 2 : 2	Generatore 12 : 12
Generatore 3 : 3	Generatore 13 : 13
Generatore 4 : 4	Generatore 14 : 14
Generatore 5 : 5	Generatore 15 : 15
Generatore 6 : 6	Generatore 16 : 16
Generatore 7 : 7	Generatore 17 : 17
Generatore 8 : 8	Generatore 18 : 18
Generatore 9 : 9	Generatore 19 : 19
Generatore 10 : 10	Generatore 20 : 20

Tabella 4.3: Insieme dei generatori di Γ.

[14]Processore: Intel Core 2 Duo, 2 GHz; Memoria: 4 GB.

Relatori della presentazione Γ (ordinati per lunghezza):

Relatore 1, lughezza 15 :(3)(19)(-15)(12)(-11)(-17)(14)(11)(8)(7)(3)(1)(-2)(1)(-6)
Relatore 2, lughezza 15 :(10)(4)(12)(-3)(-13)(-13)(-13)(2)(6)(-15)(12)(17)(14)(-2)(16)
Relatore 3, lughezza 15 :(14)(13)(-17)(-11)(-10)(4)(11)(-19)(-12)(10)(-18)(-17)(15)(-1)(12)
Relatore 4, lughezza 15 :(7)(-15)(-17)(-17)(-13)(-9)(7)(14)(-9)(4)(-20)(-8)(9)(-15)(11)
Relatore 5, lughezza 15 :(-3)(-7)(8)(10)(-15)(-20)(12)(10)(-13)(15)(-20)(-13)(-13)(1)(10)
Relatore 6, lughezza 16 :(-4)(-3)(-3)(11)(9)(-8)(17)(11)(3)(3)(17)(-11)(5)(-10)(1)(3)
Relatore 7, lughezza 16 :(3)(-20)(6)(13)(-3)(17)(-13)(2)(3)(-13)(-19)(-5)(17)(3)(-20)(12)
Relatore 8, lughezza 16 :(-15)(-17)(-19)(-8)(-10)(11)(8)(3)(9)(-15)(-15)(-12)(-11)(-4)(-14)(-12)
Relatore 9, lughezza 17 :(-2)(4)(-11)(-11)(-6)(-17)(13)(-11)(-18)(9)(10)(12)(7)(7)(16)(-5)(-9)
Relatore 10, lughezza 17 :(-15)(-4)(15)(6)(13)(2)(-7)(-13)(8)(9)(11)(-13)(7)(-10)(12)(16)(-9)
Relatore 11, lughezza 17 :(2)(-7)(-14)(-18)(-1)(15)(-1)(14)(3)(16)(6)(11)(-1)(9)(11)(-14)(19)
Relatore 12, lughezza 17 :(-19)(-17)(-9)(-19)(5)(-16)(-13)(-17)(18)(-7)(16)(-14)(7)(-2)(9)(13)(19)
Relatore 13, lughezza 17 :(-8)(-5)(-10)(-10)(-16)(-4)(-6)(20)(14)(-20)(3)(-7)(-1)(-20)(16)(-7)(20)
Relatore 14, lughezza 18 :(16)(19)(-4)(17)(2)(-5)(-18)(17)(-19)(6)(4)(-18)(20)(17)(-13)(-15)(-13)(3)
Relatore 15, lughezza 18 :(-14)(-14)(-18)(12)(-20)(-11)(-11)(7)(6)(-3)(14)(17)(18)(-17)(-4)(19)(-16)(-5)
Relatore 16, lughezza 18 :(6)(3)(10)(-13)(-7)(-15)(-6)(12)(15)(11)(-12)(16)(18)(18)(-13)(-16)(-16)(2)
Relatore 17, lughezza 19 :(14)(-11)(-5)(12)(12)(-3)(9)(-8)(7)(19)(15)(7)(-20)(1)(-16)(-18)(-20)(14)(7)
Relatore 18, lughezza 19 :(-9)(-2)(20)(-14)(-14)(17)(-10)(-4)(10)(-4)(-13)(11)(-2)(-8)(12)(12)(-18)(1)(11)
Relatore 19, lughezza 19 :(13)(-15)(-15)(13)(-1)(8)(10)(17)(11)(-15)(4)(-1)(-1)(-8)(15)(2)(4)(10)(-2)
Relatore 20, lughezza 20 :(14)(8)(-15)(-11)(-18)(-1)(12)(20)(12)(-15)(-6)(9)(3)(13)(9)(-20)(-11)(20)(-13)(17)

Relatori della presentazione Γ':

Relatore 1, lughezza 8 :(21)(22)(23)(24)(25)(26)(27)(-6)
Relatore 2, lughezza 7 :(28)(29)(30)(31)(32)(33)(34)
Relatore 3, lughezza 8 :(35)(36)(37)(38)(39)(40)(41)(12)
Relatore 4, lughezza 8 :(42)(43)(44)(45)(46)(47)(48)(11)
Relatore 5, lughezza 7 :(49)(50)(51)(52)(53)(54)(55)
Relatore 6, lughezza 8 :(56)(57)(58)(59)(60)(61)(62)(3)
Relatore 7, lughezza 7 :(63)(64)(65)(66)(67)(68)(69)
Relatore 8, lughezza 8 :(70)(71)(72)(73)(74)(75)(76)(-12)
Relatore 9, lughezza 9 :(77)(78)(79)(80)(81)(82)(83)(84)(-9)
Relatore 10, lughezza 7 :(85)(86)(87)(88)(89)(90)(91)
Relatore 11, lughezza 15 :(87)(-14)(-18)(-1)(41)(14)(3)(16)(6)(11)(-1)(9)(11)(-14)(19)
Relatore 12, lughezza 14 :(-19)(-17)(-9)(-19)(-84)(-13)(-17)(18)(-7)(16)(-14)(-87)(-44)(19)
Relatore 13, lughezza 17 :(-8)(-5)(-10)(-10)(-16)(-4)(-6)(20)(14)(-20)(3)(-7)(-1)(-20)(16)(-7)(20)
Relatore 14, lughezza 18 :(16)(19)(-4)(17)(2)(-5)(-18)(17)(-19)(6)(4)(-18)(20)(17)(-13)(-15)(-13)(3)
Relatore 15, lughezza 16 :(-14)(-14)(-18)(12)(-20)(78)(7)(6)(-3)(14)(-40)(-17)(-4)(19)(-16)(-5)
Relatore 16, lughezza 17 :(6)(3)(10)(-13)(-7)(-15)(-6)(12)(15)(11)(-12)(16)(18)(18)(-13)(-16)(-34)
Relatore 17, lughezza 17 :(14)(-11)(-5)(12)(29)(58)(7)(19)(15)(7)(-20)(1)(-16)(-18)(-20)(14)(7)
Relatore 18, lughezza 18 :(-9)(-2)(20)(-14)(-14)(17)(-10)(-37)(-4)(-13)(11)(-2)(-8)(12)(12)(-18)(1)(11)
Relatore 19, lughezza 16 :(13)(-15)(-53)(-1)(50)(-23)(-15)(4)(-1)(-1)(-8)(15)(2)(4)(10)(-2)
Relatore 20, lughezza 20 :(14)(8)(-15)(-11)(-18)(-1)(12)(20)(12)(-15)(-6)(9)(3)(13)(9)(-20)(-11)(20)(-13)(17)
Relatore 21, lughezza 3 :(-21)(3)(19)
Relatore 22, lughezza 3 :(-22)(-15)(12)
Relatore 23, lughezza 3 :(-23)(-11)(-17)
Relatore 24, lughezza 3 :(-24)(14)(11)
Relatore 25, lughezza 3 :(-25)(8)(7)
Relatore 26, lughezza 3 :(-26)(3)(1)
Relatore 27, lughezza 3 :(-27)(-2)(1)
Relatore 28, lughezza 3 :(-28)(10)(4)
Relatore 29, lughezza 3 :(-29)(12)(-3)
Relatore 30, lughezza 3 :(-30)(-13)(-13)
Relatore 31, lughezza 3 :(-31)(-13)(2)
Relatore 32, lughezza 3 :(-32)(6)(22)
Relatore 33, lughezza 3 :(-33)(17)(14)
Relatore 34, lughezza 3 :(-34)(-2)(16)
Relatore 35, lughezza 3 :(-35)(14)(13)

Relatore 36, lughezza 3 :(-36)(-17)(-11)
Relatore 37, lughezza 3 :(-37)(-10)(4)
Relatore 38, lughezza 3 :(-38)(11)(-19)
Relatore 39, lughezza 3 :(-39)(-12)(10)
Relatore 40, lughezza 3 :(-40)(-18)(-17)
Relatore 41, lughezza 3 :(-41)(15)(-1)
Relatore 42, lughezza 3 :(-42)(7)(-15)
Relatore 43, lughezza 3 :(-43)(-17)(-17)
Relatore 44, lughezza 3 :(-44)(-13)(-9)
Relatore 45, lughezza 3 :(-45)(7)(14)
Relatore 46, lughezza 3 :(-46)(-9)(4)
Relatore 47, lughezza 3 :(-47)(-20)(-8)
Relatore 48, lughezza 3 :(-48)(9)(-15)
Relatore 49, lughezza 3 :(-49)(-3)(-7)
Relatore 50, lughezza 3 :(-50)(8)(10)
Relatore 51, lughezza 3 :(-51)(-15)(-20)
Relatore 52, lughezza 3 :(-52)(12)(10)
Relatore 53, lughezza 3 :(-53)(-13)(15)
Relatore 54, lughezza 3 :(-54)(-20)(30)
Relatore 55, lughezza 3 :(-55)(1)(10)
Relatore 56, lughezza 3 :(-56)(-4)(-3)
Relatore 57, lughezza 3 :(-57)(-3)(11)
Relatore 58, lughezza 3 :(-58)(9)(-8)
Relatore 59, lughezza 3 :(-59)(-23)(3)
Relatore 60, lughezza 3 :(-60)(3)(17)
Relatore 61, lughezza 3 :(-61)(-11)(5)
Relatore 62, lughezza 3 :(-62)(-10)(1)
Relatore 63, lughezza 3 :(-63)(3)(-20)
Relatore 64, lughezza 3 :(-64)(6)(13)
Relatore 65, lughezza 3 :(-65)(-3)(17)
Relatore 66, lughezza 3 :(-66)(31)(3)
Relatore 67, lughezza 3 :(-67)(-13)(-19)
Relatore 68, lughezza 3 :(-68)(-5)(17)
Relatore 69, lughezza 3 :(-69)(63)(12)
Relatore 70, lughezza 3 :(-70)(-15)(-17)

Relatore 71, lughezza 3 :(-71)(-19)(-8)
Relatore 72, lughezza 3 :(-72)(-10)(11)
Relatore 73, lughezza 3 :(-73)(8)(3)
Relatore 74, lughezza 3 :(-74)(48)(-15)
Relatore 75, lughezza 3 :(-75)(-12)(-11)
Relatore 76, lughezza 3 :(-76)(-4)(-14)
Relatore 77, lughezza 3 :(-77)(-2)(4)
Relatore 78, lughezza 3 :(-78)(-11)(-11)
Relatore 79, lughezza 3 :(-79)(-6)(-17)
Relatore 80, lughezza 3 :(-80)(13)(-11)
Relatore 81, lughezza 3 :(-81)(-18)(9)
Relatore 82, lughezza 3 :(-82)(10)(12)
Relatore 83, lughezza 3 :(-83)(7)(7)
Relatore 84, lughezza 3 :(-84)(16)(-5)
Relatore 85, lughezza 3 :(-85)(-15)(-4)
Relatore 86, lughezza 3 :(-86)(15)(64)
Relatore 87, lughezza 3 :(-87)(2)(-7)
Relatore 88, lughezza 3 :(-88)(-13)(8)
Relatore 89, lughezza 3 :(-89)(9)(-80)
Relatore 90, lughezza 3 :(-90)(7)(-39)
Relatore 91, lughezza 3 :(-91)(16)(-9)

Relatori della presentazione $\hat{\Gamma}$:

Relatore 1, lughezza 8 :(21)(22)(23)(24)(25)(26)(27)(-6)
Relatore 2, lughezza 7 :(28)(29)(30)(31)(32)(33)(34)
Relatore 3, lughezza 8 :(35)(36)(37)(38)(39)(40)(41)(12)
Relatore 4, lughezza 8 :(42)(43)(44)(45)(46)(47)(48)(11)
Relatore 5, lughezza 7 :(49)(50)(51)(52)(53)(54)(55)
Relatore 6, lughezza 8 :(56)(57)(58)(59)(60)(61)(62)(3)
Relatore 7, lughezza 7 :(63)(64)(65)(66)(67)(68)(69)
Relatore 8, lughezza 8 :(70)(71)(72)(73)(74)(75)(76)(-12)
Relatore 9, lughezza 9 :(77)(78)(79)(80)(81)(82)(83)(84)(-9)
Relatore 10, lughezza 7 :(85)(86)(87)(88)(89)(90)(91)
Relatore 11, lughezza 15 :(87)(-14)(-18)(-1)(41)(14)(3)(16)(6)(11)(-1)(9)(11)(-14)(19)
Relatore 12, lughezza 14 :(-19)(-17)(-9)(-19)(-84)(-13)(-17)(18)(-7)(16)(-14)(-87)(-44)(19)
Relatore 13, lughezza 17 :(-8)(-5)(-10)(-10)(-16)(-4)(-6)(20)(14)(-20)(3)(-7)(-1)(-20)(16)(-7)(20)
Relatore 14, lughezza 18 :(16)(19)(-4)(17)(2)(-5)(-18)(17)(-19)(6)(4)(-18)(20)(17)(-13)(-15)(-13)(3)
Relatore 15, lughezza 16 :(-14)(-14)(-18)(12)(-20)(78)(7)(6)(-3)(14)(-40)(-17)(-4)(19)(-16)(-5)
Relatore 16, lughezza 17 :(6)(3)(10)(-13)(-7)(-15)(-6)(12)(15)(11)(-12)(16)(18)(18)(-13)(-16)(-34)
Relatore 17, lughezza 17 :(14)(-11)(-5)(12)(29)(58)(7)(19)(15)(7)(-20)(1)(-16)(-18)(-20)(14)(7)
Relatore 18, lughezza 18 :(-9)(-2)(20)(-14)(-14)(17)(-10)(-37)(-4)(-13)(11)(-2)(-8)(12)(12)(-18)(1)(11)
Relatore 19, lughezza 16 :(13)(-15)(-53)(-1)(50)(-23)(-15)(4)(-1)(-1)(-8)(15)(2)(4)(10)(-2)
Relatore 20, lughezza 20 :(14)(8)(-15)(-11)(-18)(-1)(12)(20)(12)(-15)(-6)(9)(3)(13)(9)(-20)(-11)(20)(-13)(17)
Relatore 21, lughezza 3 :(-21)(3)(19)
Relatore 22, lughezza 3 :(-26)(3)(1)
Relatore 23, lughezza 3 :(-30)(-13)(-13)
Relatore 24, lughezza 3 :(-31)(-13)(2)
Relatore 25, lughezza 3 :(-32)(6)(22)
Relatore 26, lughezza 3 :(-33)(17)(14)
Relatore 27, lughezza 3 :(-34)(-2)(16)
Relatore 28, lughezza 3 :(-35)(14)(13)
Relatore 29, lughezza 3 :(-37)(-10)(4)
Relatore 30, lughezza 3 :(-38)(11)(-19)
Relatore 31, lughezza 3 :(-39)(-12)(10)
Relatore 32, lughezza 3 :(-40)(-18)(-17)
Relatore 33, lughezza 3 :(-42)(7)(-15)
Relatore 34, lughezza 3 :(-43)(-17)(-17)
Relatore 35, lughezza 3 :(-45)(7)(14)

Relatore 36, lughezza 3 :(-46)(-9)(4)
Relatore 37, lughezza 3 :(-48)(9)(-15)
Relatore 38, lughezza 3 :(-50)(8)(10)
Relatore 39, lughezza 3 :(-51)(-15)(-20)
Relatore 40, lughezza 3 :(-53)(-13)(15)
Relatore 41, lughezza 3 :(-55)(1)(10)
Relatore 42, lughezza 3 :(-58)(9)(-8)
Relatore 43, lughezza 3 :(-59)(-23)(3)
Relatore 44, lughezza 3 :(-60)(3)(17)
Relatore 45, lughezza 3 :(-62)(-10)(1)
Relatore 46, lughezza 3 :(-66)(31)(3)
Relatore 47, lughezza 3 :(-67)(-13)(-19)
Relatore 48, lughezza 3 :(-70)(-15)(-17)
Relatore 49, lughezza 3 :(-71)(-19)(-8)
Relatore 50, lughezza 3 :(-72)(-10)(11)
Relatore 51, lughezza 3 :(-73)(8)(3)
Relatore 52, lughezza 3 :(-74)(48)(-15)
Relatore 53, lughezza 3 :(-76)(-4)(-14)
Relatore 54, lughezza 3 :(-77)(-2)(4)
Relatore 55, lughezza 3 :(-78)(-11)(-11)
Relatore 56, lughezza 3 :(-81)(-18)(9)
Relatore 57, lughezza 3 :(-82)(10)(12)

Codifica: parola uguale a e in $\hat{\Gamma}$ prodotta da Bob prima della diffusione (lunghezza 349):

(63)(64)(65)(66)(67)(68)(69)(66)(34)(-10)(-1)(55)(-34)(-66)(-51)(-15)(-20)(-32)(6)(22)(56)(57)(58)(59)(60)(61)(62)(3)
(-50)(8)(10)(9)(17)(-15)(13)(53)(-17)(-9)(33)(38)(19)(13)(67)(-38)(-33)(50)(57)(-38)(11)(-19)(-57)(-50)(90)(74)(-50)(8)
(10)(-74)(-90)(-42)(7)(-15)(74)(26)(70)(71)(72)(73)(74)(75)(76)(-12)(-26)(-74)(-58)(9)(-8)(35)(59)(13)(-15)(-53)(-1)(50)
(-23)(-15)(4)(-1)(-1)(-8)(15)(2)(4)(10)(-2)(-59)(-35)(35)(47)(-10)(-1)(55)(-47)(-35)(25)(72)(49)(50)(51)(52)(53)(54)(55)
(-72)(-25)(-77)(-2)(4)(-9)(18)(81)(28)(60)(-14)(-7)(45)(-60)(-28)(-34)(-2)(16)(50)(30)(-17)(-3)(60)(-30)(-50)(-1)(-3)(26)
(30)(7)(-3)(-8)(73)(-7)(-30)(-3)(-8)(73)(63)(64)(65)(66)(67)(68)(69)(32)(42)(-62)(-10)(1)(-42)(-32)(56)(57)(58)(59)(60)
(61)(62)(3)(1)(75)(-11)(-48)(-47)(-46)(-45)(-44)(-43)(-42)(-75)(-1)(42)(53)(-62)(-10)(1)(-53)(-42)(19)(13)(67)(-10)(-8)
(50)(-14)(-17)(33)(23)(62)(77)(78)(79)(80)(81)(82)(83)(84)(-9)(-62)(-23)(8)(19)(56)(57)(58)(59)(60)(61)(62)(3)(-19)(-8)
(44)(76)(-69)(-68)(-67)(-66)(-65)(-64)(-63)(-76)(-44)(30)(91)(28)(29)(30)(31)(32)(33)(34)(-91)(-30)(-14)(-14)(-18)(12)
(-20)(78)(7)(6)(-3)(14)(-40)(-17)(-4)(19)(-16)(-5)(86)(31)(-70)(-15)(-17)(-31)(-86)(13)(13)(30)(-77)(-2)(4)(28)(55)(-91)
(-90)(-89)(-88)(-87)(-86)(-85)(-55)(-28)(14)(83)(-3)(-8)(73)(-83)(-14)(11)(11)(78)(-34)(-2)(16)(56)(67)(13)(13)(30)(-67)
(-56)(78)(84)(-37)(-10)(4)(-84)(-78)(63)(69)(8)(-9)(58)(-69)(-63)(15)(43)(-3)(-8)(73)(-43)(-15)(-17)(-3)(60)(34)(16)(13)
(-18)(-18)(-16)(12)(-11)(-15)(-12)(6)(15)(7)(13)(-10)(-3)(-6)

Codifica: parola uguale a e in $\hat{\Gamma}$ prodotta da Bob dopo l'operazione di diffusione (lunghezza 693):

(67)(-3)(21)(-14)(35)(63)(64)(65)(46)(-4)(74)(15)(15)(66)(67)(68)(69)(66)(34)(62)(-1)(-1)(55)(-34)(-3)(34)(-16)(13)(38)
(-3)(21)(71)(8)(-38)(56)(57)(58)(59)(46)(76)(14)(18)(81)(-3)(19)(71)(73)(60)(61)(38)(19)(38)(-3)(21)(78)(62)(77)(-4)(13)
(66)(58)(50)(72)(78)(38)(19)(17)(-7)(42)(-14)(35)(67)(-3)(21)(-14)(76)(35)(13)(30)(2)(77)(35)(53)(43)(17)(46)(76)(-7)
(45)(33)(38)(19)(2)(77)(76)(35)(67)(-38)(4)(76)(15)(70)(74)(26)(70)(71)(72)(73)(73)(-3)(19)(71)(55)(62)(-1)(-1)(74)(75)
(76)(39)(37)(-4)(-1)(-3)(46)(76)(-7)(45)(58)(73)(-3)(13)(53)(13)(53)(46)(-4)(35)(59)(12)(39)(37)(76)(35)(70)(38)(19)(-11)
(-3)(60)(70)(-3)(60)(-14)(35)(-1)(50)(59)(-3)(-7)(42)(58)(73)(-3)(46)(-3)(-31)(66)(-1)(-1)(46)(76)(14)(58)(13)(53)(13)(66)
(-3)(13)(31)(77)(-1)(55)(77)(76)(35)(67)(19)(-59)(47)(62)(-1)(-1)(55)(-47)(-35)(25)(72)(49)(50)(51)(8)(-9)(58)(-3)(-8)(73)
(23)(59)(-3)(52)(53)(54)(55)(78)(38)(19)(82)(39)(62)(-1)(-25)(46)(76)(18)(40)(15)(-9)(74)(13)(53)(33)(-9)(58)(50)(72)
(10)(72)(78)(18)(81)(28)(60)(-14)(-7)(45)(43)(33)(-14)(13)(53)(46)(-4)(74)(15)(-3)(-28)(50)(30)(43)(33)(-14)(-3)(73)(-3)
(-9)(58)(35)(-13)(-14)(60)(-30)(62)(-1)(-8)(-1)(-3)(26)(30)(45)(-14)(-3)(-3)(21)(71)(73)(-7)(-30)(-3)(-8)(73)(63)(-1)(55)
(72)(-11)(64)(65)(66)(67)(68)(69)(56)(57)(58)(59)(60)(61)(62)(13)(53)(-7)(42)(26)(75)(78)(82)(-12)(72)(15)(46)(-4)(-47)
(-4)(58)(8)(-14)(-7)(-44)(-3)(60)(33)(-14)(78)(11)(82)(-12)(72)(13)(53)(-7)(-75)(-1)(-3)(21)(-14)(35)(67)(62)(-1)(-3)(21)
(71)(50)(58)(8)(46)(76)(17)(43)(33)(23)(62)(77)(78)(79)(80)(81)(82)(83)(32)(-22)(-6)(84)(46)(76)(35)(30)(13)(-1)(-8)(50)
(12)(39)(72)(38)(19)(78)(59)(-3)(66)(-3)(-7)(45)(74)(15)(-14)(35)(53)(46)(76)(34)(-16)(13)(73)(-16)(13)(31)(34)(-3)(19)
(56)(57)(58)(50)(62)(-1)(46)(76)(14)(58)(59)(60)(61)(62)(21)(-11)(38)(71)(44)(76)(-69)(-68)(-3)(21)(13)(-3)(-31)(-65)
(-64)(-63)(43)(-31)(66)(-3)(33)(-14)(33)(58)(8)(46)(-44)(30)(2)(34)(-16)(91)(67)(35)(67)(19)(-14)(-3)(-7)(45)(9)(46)(76)
(21)(13)(28)(29)(30)(66)(-3)(32)(33)(34)(-91)(-14)(35)(-14)(35)(-14)(4)(76)(81)(46)(76)(-17)(33)(-10)(82)(15)(51)(78)
(45)(59)(59)(-3)(23)(70)(17)(15)(-3)(23)(4)(76)(32)(-22)(-3)(35)(-14)(35)(30)(-40)(15)(70)(-4)(-3)(21)(-16)(-5)(13)(-14)
(35)(30)(13)(66)(-3)(77)(76)(-17)(33)(28)(71)(50)(72)(11)(78)(19)(55)(-91)(-90)(-89)(-88)(-87)(-86)(-85)(37)(76)(35)(67)
(19)(-1)(55)(-10)(-1)(-28)(-7)(45)(83)(-3)(74)(2)(77)(76)(35)(53)(15)(46)(-4)(-3)(21)(71)(73)(-3)(23)(59)(-83)(-14)(11)
(12)(39)(72)(78)(56)(67)(13)(-14)(35)(30)(-67)(-56)(63)(69)(50)(62)(-1)(46)(-4)(58)(-69)(-63)(9)(46)(76)(35)(53)(43)
(-3)(19)(71)(73)(17)(17)(70)(-3)(-1)(55)(72)(-11)(60)(34)(2)(34)(13)(81)(46)(76)(-7)(45)(46)(76)(-7)(45)(58)(73)(-3)
(40)(17)(-16)(12)(78)(38)(-3)(21)(46)(15)(46)(-4)(74)(13)(53)(76)(35)(67)(19)(74)(13)(53)(39)(72)(-1)(55)(72)(78)(32)
(-22)(-14)(35)(53)(42)(15)(-14)(33)(4)(76)(43)(-3)(60)(35)(62)(-1)(-3)(-8)(50)(37)(76)(-7)(45)(-6)

Codifica: parola uguale a e in $\hat{\Gamma}$ ricevuta da Alice già mappata in Γ (lunghezza 314):

(3)(-20)(6)(13)(-3)(17)(-13)(2)(3)(-13)(-19)(-5)(17)(3)(-20)(12)(-4)(-3)(-3)(11)(9)(-8)(17)(11)(3)(3)(17)(-11)(5)(-10)(1)(3)
(9)(-15)(-15)(3)(1)(-15)(-17)(-19)(-8)(-10)(11)(8)(3)(9)(-15)(-15)(-12)(-11)(-4)(-14)(-12)(-1)(-3)(15)(15)(-9)(14)(13)(17)
(11)(3)(13)(-15)(-15)(13)(-1)(8)(10)(17)(11)(-15)(4)(-1)(-1)(-8)(15)(2)(4)(10)(-2)(-3)(-11)(-17)(-13)(-14)(8)(7)(-10)(11)
(-3)(-7)(8)(10)(-15)(-20)(12)(10)(-13)(15)(-20)(-13)(-13)(1)(10)(-11)(10)(-7)(-8)(3)(-20)(6)(13)(-3)(17)(-13)(2)(3)(-13)
(-19)(-5)(17)(3)(-20)(12)(-4)(-3)(-3)(11)(9)(-8)(17)(11)(3)(3)(17)(-11)(5)(-10)(1)(3)(1)(-12)(-11)(-11)(15)(-9)(8)(20)(-4)
(9)(-14)(-7)(9)(13)(17)(17)(15)(-7)(11)(12)(-1)(-11)(-17)(-10)(1)(-2)(4)(-11)(-11)(-6)(-17)(13)(-11)(-18)(9)(10)(12)(7)(7)
(16)(-5)(-9)(-1)(10)(17)(11)(8)(19)(-4)(-3)(-3)(11)(9)(-8)(17)(11)(3)(3)(17)(-11)(5)(-10)(1)(3)(-19)(-8)(-13)(-9)(-4)(-14)
(-12)(20)(-3)(-17)(5)(19)(13)(-3)(-2)(13)(-17)(3)(-13)(-6)(20)(-3)(14)(4)(9)(-13)(16)(-9)(10)(4)(12)(-3)(-13)(-13)(-13)
(2)(6)(-15)(12)(17)(14)(-2)(16)(9)(-16)(13)(13)(-14)(-14)(-18)(12)(-20)(-11)(-11)(7)(6)(-3)(14)(17)(18)(-17)(-4)(19)(-16)
(-5)(10)(4)(1)(10)(9)(-16)(-12)(10)(-7)(13)(-11)(-9)(-8)(13)(7)(-2)(-13)(-6)(-15)(4)(15)(-10)(-1)(-4)(-10)(-2)(16)(16)(13)
(-18)(-18)(-16)(12)(-11)(-15)(-12)(6)(15)(7)(13)(-10)(-3)(-6)

Bibliografia

[1] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 3rd ed. Upper Saddle River, NJ: Pearson Addison-Wesley, 2007.

[2] V. Shpilrain and G. Zapata, "Using decision problems in public key cryptography," *Groups-Complexity-Cryptology*, vol. 1, no. 1, pp. 33–39, 2009.

[3] N. Lauritzen, *Concrete Abstract Algebra*. Cambridge University Press, 2003.

[4] A. Languasco and A. Zaccagnini, *Introduziona alla Crittografia*. Ulrico Hoepli, 2004.

[5] W. Magnus, A. Karras, and D. Solitar, *Combinatorial Group Theory*, 2nd ed. New York: Dover, 1976.

[6] R. C. Lyndon and P. E. Schupp, *Combinatorial Group Theory*. Springer-Verlag, 1977.

[7] G. N. Arzhantseva and A. Y. Ol'shanskii, *The class of groups all of whose subgroups with lesser number of generators are free is generic*. Math. Notes, 1996, vol. 59.

[8] A. G. Myasnikov, A. G. Myasnikov, and V. Shpilrain, "On the andrews-curtis equivalence," *Contemp. Math., Amer. Math. Soc*, vol. 296, pp. 183–198, 2002.

[9] I. Kapovich, A. Myasnikov, P. Schupp, and V. Shpilrain, "Generic-case complexity, decision problems in group theory and random walks," *J. Algebra*, vol. 264, pp. 665–694, 2003.

[10] W. Woess, "Cogrowth of groups and simple random walks," *Arch. Math.*, vol. 41, pp. 363–370, 1983.

[11] A. V. Borovik, R. V. Borovik, A. G. Myasnikov, and V. Shpilrain, "Measuring sets in infinite groups," 2002.

[12] V. Shpilrain and G. Zapata, "Combinatorial group theory and public key cryptography," *Applicable Algebra in Engineering, Communication and Computing*, vol. 17, no. 3–4, pp. 291–302, 2006.

[13] I. Anshel, M. Anshel, and D. Goldfeld, "An algebraic method for public-key cryptography," *Mathematical Research Letters*, vol. 6, no. 3/4, pp. 287–291, 1999.

[14] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. sung Kang, and C. Park, "New Public-Key Cryptosystem using Braid Groups," in *Advances in Cryptology – CRYPTO 2000*, ser. Lecture Notes in Computer Science, M. Bellare, Ed., vol. 1880. Springer, 2000, pp. 166–183.

[15] M. Alexei and U. Alexander, "Random van kampen diagrams."

Ringraziamenti

Ringrazio il mio relatore, il prof. Alberto Tonolo, che con pazienza e passione ha saputo aiutarmi e condurmi durante la stesura di questa tesi.
Ringrazio la mia famiglia: il mio papà, la mia mamma e mio fratello Davide, che mi hanno sempre sostenuto durante tutto il periodo universitario e non mi hanno mai fatto mancare nulla.
Un grazie sentito ai miei nonni.
Grazie anche a tutti i miei amici d'infanzia, delle superiori: Fresk, Ross e Guidi e ai miei amici dell'Università Simone, Alessio, Stefano, Giovanni e Nicolò.

Infine ringrazio Francesca, per non avermi mai lasciato solo.