Linguaggi
Grammatiche e Automi

Vincenzo Manca

Dipartimento di Informatica

Università di Verona

Indice

Capitolo 1

Sistemi Monoidali

La teoria dei linguaggi formali, sviluppatasi a partire dai primi anni del Novecento, ha dimostrato una centralità sorprendente nelle teorie matematiche della computazione. Inoltre, la scoperta della natura discreta dell'informazione in tanti meccanismi biologici fondamentali (DNA, sintesi proteica, metabolismi biochimici) ha reso tale teoria un campo in grande sviluppo in cui potere ricercare modelli adeguati per un gran numero di fenomeni.

Un **monoide** è determinato da un insieme M su cui è definita un'operazione binaria associativa e che contiene un elemento neutro e, indifferente rispetto a tale operazione $\cdot$ ($e \cdot x = x \cdot e = x \quad \forall x \in M$). Dato un **alfabeto** finito A le stringhe su tale alfabeto costituiscono l'insieme A^* detto anche **universo linguistico** su A. Tale insieme è naturalmente dotato di una struttura di monoide, detto **monoide libero** su A, in cui l'operazione binaria è la concatenazione tra stringhe e l'elemento neutro è la stringa nulla λ. Un **linguaggio** sull'alfabeto A è un sottoinsieme di A^*.

Tra i linguaggi, in quanto insiemi, si possono applicare le usuali operazioni insiemistiche di **unione** (spesso denotata con $+$), **intersezione** e **complemento**. Un **morfismo** tra due linguaggi è una funzione f tale che $f(xy) = f(x)f(y)$, dato un linguaggio L, $f(L)$ è il linguaggio le cui stringhe sono immagini di stringhe di L secondo f. Una **codifica** di un linguaggio in un altro è una funzione iniettiva dal primo al secondo. Ricordiamo che ogni linguaggio può essere codificato in un linguaggio sull'alfabeto binario $\{0, 1\}$ e in un linguaggio sull'alfabeto unario $\{1\}$ (ovvero in un insieme di naturali).

La nozione di linguaggio sta alla matematica discreta come quella di insieme sta a tutta la matematica. Utilizzando un aforisma potremmo dire che se in matematica tutto è insieme, in matematica discreta tutto è linguaggio.

In senso lato, una struttura discreta è qualcosa in ultima analisi rappresentabile con un linguaggio (o con più linguaggi) e una struttura discreta finita qualcosa che si può rappresentare con una stringa (o con un linguaggio finito).

Questo giustifica l'interesse teorico per i linguaggi e le classi di linguag-

gi, i metodi per definirli e le loro caratteristiche algebriche e computazionali. La **teoria dei linguaggi formali**, sviluppatasi a partire dai primi anni del '900, ha dimostrato una centalità sorprendente nelle teorie matematiche della computazione e quindi in tutta l'informatica teorica. Inoltre, la scoperta della natura discreta dell'informazione in tanti meccanismi biologici fondamentali (DNA, sintesi proteica, metabolismi biochimici) ha reso la teoria dei linguaggi formali un campo in grande sviluppo in cui potere ricercare modelli adeguati per la modellizzazione di un gran numero di fenomeni.

La nozione stessa di **problema**, formulato entro qualche rappresentazione dei suoi dati, equivale a quella di linguaggio. Data una formula proposizionale, quando possiamo dire che essa è soddisfacibile (ovvero rispondere affermativamente al problema di verificarne la soddisfacibilità)? Quando, previa sua rappresentazione, possiamo dire che essa appartiene al linguaggio SAT. In effetti, la teoria matematica dei problemi e dei loro gradi di risolubilità può considerarsi una specializzazione della teoria della calcolabilità e dei linguaggi formali, che va sotto il nome di teoria della **complessità computazionale**. Il problema (o meglio metaproblema) più famoso in tale teoria consiste nello stabilire se le due classi di linguaggi: P (in cui l'appartenenza di una stringa è **decidibile deterministicamente** in tempo **polinomiale**, rispetto alla lunghezza della stringa) ed NP (in cui l'appartenenza di una stringa è decidibile **non deterministicamente** in tempo polinomiale, rispetto alla lunghezza della stringa) sono uguali o diverse. Si dimostra che il linguaggio SAT ha complessità massima nella classe NP, nel senso che tutti gli altri problemi di NP possono essere ricondotti a SAT in tempo polinomiale. In tale questione si intersecano in modo inestricabile logica, teoria dei linguaggi e teoria della complessità.

Esempi di linguaggi sono i seguenti.

- Tutte le espressioni corrette secondo una sintassi (per esempio i programmi sintatticamente corretti di un certo linguaggio di programmazione).
- Una funzione sui naturali o sulle stringhe di un alfabeto, vista come insieme di coppie (argomento, risultato).
- La rappresentazione di un numero in una qualche base (linguaggio infinito se il numero è irrazionale).
- L'insieme di stringhe che codificano (secondo qualche criterio) una classe di strutture finite (grafi, alberi, tabelle, formule).
- Le sequenze genetiche del genoma di un organismo biologico.
- L'insieme di stati che può assumere un sistema dinamico simbolico.

Soffermiamoci su questo ultimo punto che presenta sviluppi ed interpretazioni di grande interesse.

Un **sistema dinamico simbolico** è un sistema che cambia nel tempo e i cui stati sono descrivibili con stringhe.

Il problema di stabilire se il sistema arriva ad un certo stato corrisponde a quello di poter decidere se una data stringa appartiene ad un certo linguaggio. Analogamente gli stati raggiungibili a partire da certi stati iniziali costituiscono un linguaggio. Gli stati in cui il sistema rimane stabile nel tempo (stati finali) sono un altro linguaggio, e gli stati che dopo un tempo finito portano a certi stati finali sono un altro linguaggio ancora.

I sistemi dinamici naturali continui sono oggetto di studio della fisica (sistema solare, sistemi idrodinamici, sistemi metereologici); tuttavia anche per essi le simulazioni discrete forniscono modelli approssimati, ma estremamente utili, allorquando gli strumenti classici (equazioni differenziali) diventano di complessità proibiliva.

Esempi di sistemi dinamici discreti sono: un sistema di calcolo, o una parte di esso; una rete digitale di comunicazione, in cui ad ogni istante certi pacchetti informativi si trovano sui nodi di un grafo di comunicazione; un archivio o una base di dati con evoluzione temporale; una deduzione (secondo qualche calcolo logico); l'insieme delle possibili configurazioni di una scacchiera durante un gioco (a scacchi, a dama, ...); una miscela biochimica che si trasforma secondo reazioni tra le sue molecole e secondo parametri dipendenti da condizioni esterne (acidi nucleici, proteine, e bio-molecole in genere, sono rappresentabili per molti versi come stringhe di opportuni alfabeti).

In tutti i tipi di sistemi dinamici (naturali o artificiali, continui o discreti, chiusi o interattivi, deterministici o no) rimane centrale il problema della descrizione matematica con forme descrittive quanto più rigorose, generali e sintetiche.

Un primo passo consiste nel passare da descrizioni informali a descrizioni matematicamente rigorose che garantiscono una maggiore affidabilità. In passi successivi si tende a descrizioni più complete, o più semplici, o più generali.

Le famose leggi di Keplero sono una descrizione chiara dei moti planetari (orbite ellittiche, costanza della velocità areolare, tempi di percorrenza). La descrizione Newtoniana in termini della forza gravitazionale e del secondo principio della dinamica è un'altra descrizione dello stesso sistema, con il vantaggio di ricondurlo a principi generali validi per un'ampia classe di sistemi meccanici.

In termini del tutto generali, la descrizione di un sistema dinamico consiste sempre nel passaggio da un sistema rappresentativo ad un altro in modo che vi sia un guadagno (in termini di accuratezza, semplicità, generalità).

Il passaggio da una rappresentazione ad un'altra è la base della problematica della teoria dei linguaggi formali.

Infatti, il problema centrale di una teoria matematica dei linguaggi consiste nei metodi formali (finitistici) di rappresentazione dei linguaggi e nei rapporti

tra i vari sistemi rappresentativi. Nel seguito presenteremo alcuni metodi ormai standard.

Mei metodi formali che studieremo posiamo individuare un'altra importante analogia con lo studio dei sistemi dinamici studiati dalla fisica. Un aspetto notevole nel passaggio dalla descrizione Kepleriana dei moti planetari a quella Newtoniana consiste nell'utilizzo di caratteristiche del moto locali piuttosto che globali. Il fatto che le orbite dei moti siano ellittiche (con il sole in uno dei fuochi) è un dato della descrizione Kepleriana, mentre in quella Newtoniana è piuttosto una conseguenza della legge di attrazione gravitazionale e della seconda legge della dinamica ($f = ma$). Tali leggi determinano equazioni differenziali che stabiliscono in ultima analisi il tipo di rapporto che deve intercorrere tra il differenziale secondo dello spostamento infinitesimo istantaneo e l'infinitesimo temporale in cui questo avviene.

Quello che avviene nel passare da una descrizione globale di un linguaggio (la forma delle sue stringhe o di tutto il linguaggio) ad una descrizione tramite grammatica o automa è qualcosa di molto simile. Infatti una grammatica o un automa stabiliscono solo le trasformazioni locali che possono essere effettuate a partire da certe stringe iniziali per generare o riconoscere una stringa del linguaggio.

Capitolo 2

Stringhe e Linguaggi

Definizione 1. *Un* alfabeto *è un insieme A, finito e non vuoto, di oggetti detti* simboli*.*

Definizione 2. *Una* stringa*, detta anche* parola*, è una sequenza finita di simboli di un alfabeto A. L'insieme delle stringhe su A, che indichiamo con* $A^\star$*, si chiama* universo linguistico *di A.*

Notazioni. Con $\mathbb{N}$ si indica l'insieme dei numeri interi non negativi, e con λ la parola vuota (priva di simboli).

Definiamo le seguenti operazioni su stringhe:

- concatenazione c (- -)

$$\begin{array}{rccc} c: & A^\star \times A^\star & \to & A^\star \\ & (\alpha, \beta) & \mapsto & \alpha\beta \end{array}$$

- lunghezza l ($\|$)

$$\begin{array}{rccc} l: & A^\star & \to & \mathbb{N} \\ & \alpha & \mapsto & |\alpha| \end{array}$$

  dove $|\alpha|$ è il numero dei simboli della parola α.

Su un alfabeto $A = \{a_1, a_2, \cdots, a_n\}$ si consideri la seguente struttura algebrica

$$R = (A^\star, \mathbb{N}, --, \|, \lambda, a_1, a_2, \cdots, a_n)$$

determinata dagli insiemi $A^\star$ e $\mathbb{N}$, che chiamiamo *domini* dell'algebra, dalle funzioni totali concatenazione e lunghezza (definite su un prodotto cartesiano i cui fattori sono domini, e a valori in uno dei domini), e dalle funzioni nullarie (con zero argomenti) coincidenti con gli elementi dell'alfabeto e con la parola vuota. R è una **struttura relazionale**.

Così come tutta la matematica si basa sugli insiemi, tutta la matematica discreta si basa sui linguaggi.

Definizione 3. *Un* linguaggio *(su A) è un elemento di $\mathcal{P}(A^\star)$.*

Notazione. Indichiamo un generico linguaggio con la lettera L, variamente decorata ($L', L_1, L_2, \cdots$), e riserviamo la lettera $\mathcal{L}$, variamente decorata, per indicare una classe di linguaggi.

Oltre alle usuali operazioni insiemistiche di **unione** ($\cup$), **intersezione** ($\cap$) e **differenza** ($\setminus$), sui linguaggi abbiamo la **concatenazione** ($\cdot$) e l'**iterazione** ($*$), rispettivamente:

- $L_1 \cdot L_2 = \{\alpha\beta \,/\, \alpha \in L_1, \beta \in L_2\} \qquad \forall L_1, L_2 \in \mathcal{P}(A^\star)$.
- $L^* = \bigcup_{n\in\mathbb{N}} L^n \qquad \forall L \in \mathcal{P}(A^\star)$

  dove L^n è definito per induzione $\begin{cases} L^0 \stackrel{def}{=} \{\lambda\} \\ L^n = L^{n-1} \cdot L \quad n \geq 1. \end{cases}$

  Il simbolo $*$ si chiama anche *stella di Kleene*.

Come si è detto, $A^\star$ è un monoide rispetto alla concatenazione (ovviamente associativa), avente λ come elemento neutro. Anche $\mathcal{P}(A^\star)$ è un monoide rispetto alla concatenazione, e ha come elemento neutro L^0.

Sui linguaggi si possono definire funzioni, e in particolare morfismi.

Definizione 4. *Siano A e B due alfabeti. Chiamiamo* morfismo *un'applicazione h tale che*

$$\begin{array}{rccll} h: & A^\star & \to & B^\star & \\ & \alpha\beta & \mapsto & h(\alpha)h(\beta) & \forall \alpha, \beta \in A^\star \\ & \lambda & \mapsto & \lambda & \end{array}$$

dove $h(\alpha)h(\beta)$ è la concatenazione di $h(\alpha)$ e $h(\beta)$.
Di conseguenza, chiamiamo morfismo inverso *l'applicazione h^{-1} su $B^\star$ tale che*

$$\begin{array}{rccl} h^{-1}: & B^\star & \to & \mathcal{P}(A^\star) \\ & \beta & \mapsto & \{\alpha \in A^\star \,/\, h(\alpha) = \beta\}. \end{array}$$

Sono di particolare interesse i morfismi *non erasing*, ovvero quelli tali che $h^{-1}(\lambda) = \{\lambda\}$.

Ci sono quattro metodi per definire linguaggi: due di tipo matematico (insiemistico) e due di tipo algoritmico (tramite processi di calcolo), rispettivamente:

1. i **metodi logici** descrivono un linguaggio tramite le sue proprietà, cioè definendo una proprietà P tale che $L = \{\alpha \in A^\star \,/\, P(\alpha)\}$.

   Esempio 5 (linguaggio bisomatico)**.** *Sia* $\mathbf{L_b} = \{\mathbf{a^n b^n} \,/\, \mathbf{n} \in \mathbb{N}\}$ *su* $A = \{a, b\}$ *il linguaggio che esprime l'***accrescimento bipolare***, possiamo descriverlo come l'insieme delle stringhe* α *costituite da due sottostringhe* γ *e* β*, aventi uguale lunghezza e tali che i simboli di* γ *siano tutti* a *e quelli di* β *tutti* b*.*

$$L_b = \quad \{\alpha \in A^\star \,/\, \alpha = \gamma\beta \wedge |\gamma| = |\beta| \wedge \mathcal{A}(\gamma) \wedge \mathcal{B}(\beta)\} \quad \textit{dove}$$

$$\begin{aligned} \mathcal{A}(\gamma) &\equiv \ \forall\, x,\, \varepsilon,\, \xi \quad (\gamma = \varepsilon x \xi \Rightarrow x = a) \\ \mathcal{B}(\beta) &\equiv \ \forall\, x,\, \varepsilon,\, \xi \quad (\beta = \varepsilon x \xi \Rightarrow x = b) \end{aligned}$$

   Abbiamo dunque descritto il linguaggio bisomatico L_b *tramite una proprietà (espressa in termini di concatenazione e lunghezza di stringhe) definita sulla struttura relazionale* $(A^\star, --\,, \|, \lambda, a, b)$.

2. i **metodi algebrici** costruiscono un linguaggio tramite operazioni quali $\cup\,,\ \setminus\,,\cdot\,,\ *\,, f$ (morfismo), partendo da certi linguaggi iniziali.

3. i **metodi generativi**, ovvero **grammatiche**, consistono in algoritmi che producono tutte e sole le parole del linguaggio.

4. i **metodi riconoscitivi**, ovvero **automi**, consistono in algoritmi che per ogni $\alpha \in A^\star$ dicono se α appartiene o meno ad un linguaggio dato.

Osservazione 6. *Da un algoritmo riconoscitivo per un linguaggio* L *se ne ottiene uno generativo (l'algoritmo che prende tutte le parole di* $A^\star$ *e genera tutte e sole quelle che riconosce appartenenti ad* L*), il viceversa non è vero (come vedremo più avanti).*

Capitolo 3

Gerarchia di Chomsky

Notazioni. $A \subset B$ indica che A è *strettamente* incluso in B, e $A \subset_f B$ indica che A è un sottinsieme *finito* di B.

Definizione 7. *Si chiama* grammatica di Chomsky *una quaterna* $\mathbf{G} = (\mathbf{A}, \mathbf{T}, \mathbf{S}, \mathbf{P})$ *di cui*
A *è l'alfabeto*
$T \subset A$ *è un insieme di simboli detti* terminali
$S \in A \backslash T$ *è un simbolo chiamato* start del linguaggio
$P \subset_f A^\star \times A^\star$ *è l'insieme delle* produzioni.
Notazione. $(\alpha, \beta) \in P$ *si scrive* $\alpha \rightarrow_G \beta$.
A partire dalla grammatica G si definisce una relazione di riscrittura ad un passo $\Rightarrow_{\mathbf{G}}$

$$w \Rightarrow_G w\prime \quad \Leftrightarrow \quad \exists\, \alpha,\, \beta,\, \gamma,\, \delta \in A^\star \quad (\alpha \rightarrow_G \beta) \wedge (w = \gamma\alpha\delta) \wedge (w\prime = \gamma\beta\delta).$$

A partire dalla relazione $\Rightarrow_G$ *si definisce una* riscrittura a più passi $\Rightarrow^\star_{\mathbf{G}}$

$$w \Rightarrow^\star_G w\prime \Leftrightarrow \exists w_1, \cdots, w_n \quad (w_1 = w) \wedge (w_n = w\prime) \wedge (\forall i \in \{1, \cdots, n-1\}\ w_i \Rightarrow_G w_{i+1}).$$

Definizione 8. *Sia G una grammatica di Chomsky. L'insieme*

$$\mathbf{L(G)} = \{\alpha \in \mathbf{T}^\star \,/\, \mathbf{S} \Rightarrow^\star_{\mathbf{G}} \alpha\}$$

si chiama linguaggio generato da G.

Quindi, per generare un linguaggio da una grammatica G = (A, T, S, P), si parte dal simbolo *iniziale* S e si fa una riscrittura a più passi, ovvero un rimpiazzamento iterato di sottostringhe secondo le produzioni P.

Per come sono definite le produzioni di una grammatica, di volta in volta possono esserci più regole applicabili, per esempio si potrebbe avere la stringa $w = \gamma\alpha\delta$ e tra le produzioni date $\alpha \rightarrow_G \beta$, $\alpha \rightarrow_G \vartheta$, $\gamma \rightarrow_G \eta$, $\delta \rightarrow_G \lambda$. Inoltre, una stessa regola può essere applicabile in diversi modi se una sottostringa

rimpiazzabile appare più volte in una stringa, per esempio sulla stringa $\gamma\alpha\eta\alpha\vartheta$ l'applicazione della regola $\alpha \to_G \beta$ può produrre $\gamma\beta\eta\alpha\vartheta$ oppure $\gamma\alpha\eta\beta\vartheta$.

In questi casi si procede in modo *non deterministico*, ovvero scegliendo arbitrariamente tra le regole applicabili e tra i vari modi di applicarle.

Ci si ferma quando non ci sono più regole da poter applicare. La parola ottenuta con a parte del linguaggio se e solo se è un elemento di $T^\star$. Una stringa che non può essere trasformata in un elemento di $T^\star$ o che non proviene da S si dice **parassita**.

Esempio 9 (linguaggio trisomatico)**.** *Ci proponiamo di descrivere*

$$\mathbf{L_t} \stackrel{\mathbf{def}}{=} \{\mathbf{a^n b^n c^n} \,/\, \mathbf{n} \in \mathbb{N}\}$$

*che esprime l'***accrescimento trisomatico***.*

Possiamo farlo con il seguente **sistema di Post**, di cui una grammatica di Chomsky è una versione semplificata.
Si prendano due *assiomi*, che per ipotesi sono parole del linguaggio, e una *regola* (che indichiamo con la notazione $\frac{\cdots}{\cdots\cdots}$):

$$\begin{cases} \textit{Assiomi}: & \lambda, abc \\ \textit{regola}: & \frac{X\,a\,b\,Y}{a\,X\,a\,b\,b\,Y\,c} \quad X, Y \text{ parole qualsiasi.} \end{cases}$$

La regola consiste nel fattorizzare una stringa secondo il "pattern" $XabY$ per opportuni valori delle variabili X e Y su $\{a, b, c\}^\star$, una volta rinvenuto tale pattern in una data stringa si aggiunge una a all'inizio, una c alla fine e una b prima della stringa associata a Y secondo il pattern.
Si dimostra per induzione che partendo dall'assioma abc e iterando l'applicazione della regola si ottengono tutte e sole parole del tipo $a^n b^n c^n$, con $n \geq 2$ che dipende dal numero (arbitrario) di volte che si è applicata la regola.
Un esempio di derivazione è

$$\begin{array}{lcll} (ass) & abc & X = \lambda & Y = c \\ & \hline aabbcc & X = a & Y = bcc \\ & \hline aaabbbccc & X = aa & Y = bbccc \\ & \hline \cdots & \cdots & \cdots \end{array}$$

Questo modo di procedere è diverso da quello della relazione di riscrittura della Definizione 7, perché è basato su una regola globale, infatti tutte le volte che si applica la regola si "guarda" tutta la stringa.

Le grammatiche invece basano la loro forza computazionale sull'adozione di un **principio di località**.

Notazione. Con le lettere minuscole indichiamo gli elementi di T e con le maiuscole quelli di $A \setminus T$.

Nel caso del linguaggio trisomatico, se G_t è la grammatica individuata dalle seguenti produzioni

$$P_{G_t} \begin{cases} S & \to & abc \\ S & \to & aSBc \\ cB & \to & Bc \\ bB & \to & bb \end{cases}$$

sull'alfabeto $A = \{a, b, c, B\}$, abbiamo che

Proposizione 10. $L(G_t) = \{a^n b^n c^n \ / \ n \in \mathbb{N}\}$.

Vediamo un esempio pratico di come si ottengono le stringhe del linguaggio trisomatico tramite le produzioni di cui sopra.

Partiamo da S. Possiamo applicare la prima o la seconda regola, scegliendo la prima otteniamo subito la parola $abc \in L_t$, la seconda invece ci dà $aSBc$ e ci porta nuovamente nella condizione in cui le regole applicabili sono le prime due. Scegliamo di applicare, per esempio, un altro paio di volte la seconda regola e otteniamo $aaaSBcBcBc$. Su questa stringa le produzioni applicabili sono le prime tre, e come prima si sceglie arbitrariamente il percorso da seguire. Per esempio si può applicare una volta la prima regola e sei volte la terza per ottenere $aaaabBBBcccc$, che terminalizza nella stringa $a^4b^4c^4 \in L_t$ applicando tre volte la quarta regola.

La struttura trisomatica ha un grande interesse biologico perché lo sviluppo di un embrione (nel senso di accrescimento del numero di cellule) nelle sue tre parti (mesoderma, endoderma, ectoderma) segue una legge del tipo $\mathbf{a^{f(n)}b^{g(n)}c^{h(n)}}$, dove $f(n)$, $g(n)$, $h(n)$ indicano lo stato di sviluppo delle tre parti.

Si dimostra che tutti gli algoritmi generativi di un tale tipo di stringhe devono introdurre meccanismi di comunicazione/sincronizzazione/cooperazione/correlazione tra agenti che operano localmente con un raggio d'azione limitato.

Esempio 11 (linguaggio dei duplicati)**.** *Il linguaggio*

$$L_d \stackrel{def}{=} \{\alpha\alpha \ / \ \alpha \in \{a, b\}^\star\}$$

in natura si collega alla riproduzione, attraverso il processo di autoduplicazione del DNA.

Consideriamo le seguenti produzioni

$$P_{G_d} \begin{cases} S & \to & X_a Y_a S \\ S & \to & X_b Y_b S \\ S & \to & X_a{}' Y_a{}' \\ S & \to & X_b{}' Y_b{}' \\ & & \\ Y_i X_j & \to & X_j Y_i \\ Y_i X_j{}' & \to & X_j{}' Y_i \\ & & \\ X_i X_j{}' & \to & X_i{}' j \\ Y_i Y_j{}' & \to & Y_i{}' j \\ X_a{}' & \to & a \\ Y_a{}' & \to & a \\ X_b{}' & \to & b \\ Y_b{}' & \to & b \end{cases}$$

associate all'alfabeto $A = \{S, X_a, X_b, Y_a, Y_b, X_a{}', Y_a{}', X_b{}', Y_b{}', a, b\}$, i terminali $T = \{a, b\}$ (quindi $i, j \in T$), e il simbolo iniziale S.

Proposizione 12. *La grammatica G_d genera il linguaggio dei duplicati.*

Dimostrazione. Applicando le regole in un certo ordine si terminalizza: le prime quattro creano sequenze del tipo $X_a Y_a X_b Y_b X_b{}' Y_b{}'$, la coppia successiva porta in cima alla stringa tutti i simboli con X, e le ultime sei portano a "maturazione" la parola, creando stringhe del tipo $\alpha\alpha$ con $\alpha \in \{a, b\}^\star$. Si osserva facilmente che tale terminalizzazione introduce implicitamente una strategia, in quanto applicando le regole secondo un ordine diverso si ottengono parole parassite (non appartenenti a $T^\star$), che quindi non fanno parte del nostro linguaggio. □

Queste stringhe parassite sono gli individui scartati dalla selezione naturale. La natura preferisce sprecare energia e materiale pur di mantenere meccanismi *anarchic*i ed evitare meccanismi di controllo centralizzati che stabiliscono l'ordine di applicazione della regole. Si dimostra che la distinzione tra T e A è fondamentale (per esempio una **grammatica pura**, dove $A = T$, non può generare $a^n b^n c^n$).
Notazione. Indichiamo l'insieme dei non terminali con $N = A \setminus T$.

Distinguiamo regole ($(\alpha, \beta) \in P$) di tipo

- 0: $\alpha \in A^\star N A^\star$ (generale)
- 1: $(0) \wedge (|\alpha| \leq |\beta|)$ (contestuale e monotona)
- 2: $(1) \wedge (|\alpha| = 1)$ (libera da contesto)

- 3: $(2) \wedge (\beta \in T \cup TN)$ (regolare)

L'ulteriore condizione $X \to \lambda$, con X che non occorre in alcuna β tale che $\alpha \to_G \beta$, equivale ad aggiungere la parola λ al linguaggio generato dalla grammatica di tipo i.

Le condizioni della regola 0 impediscono a stringhe terminalizzate di continuare la derivazione, l'introduzione di simboli non terminali e la scelta di accettare solo parole di $T^\star$ (**filtraggio**) definisce implicitamente una *strategia*. In generale infatti, il modello generativo di Chomsky è molto dispendioso: si deriva a caso applicando ogni produzione possibile, tanto ci pensa la strategia globale a scartare il lavoro inutile (stringhe che non terminalizzano) e a prendere quelle buone.

Osservazione 13. *Ogni regola di tipo i è necessariamente di tipo $i-1$ (con i = 1,2,3).*

Definizione 14. *Una grammatica si dice* di tipo i *(con* i = 0,1,2,3*) se tutte le sue produzioni sono di tipo* i. *$\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ sono le classi di linguaggi generati rispettivamente da grammatiche di tipo* 0, 1, 2, 3 *e sono dette* classi di Chomsky*.*

Si dimostra che:

$$\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3$$

un linguaggio L è di pipo i se $\exists G$ grammatica di tipo $i$tale che $L = L(G)$ e $L \neq L(G') \quad \forall G'$ di tipo i + 1.

Il seguente teorema scaturisce mettendo insieme vari risultati dimostrati nel seguito.

Teorema 15 (Chomsky)**.** $\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3$.

La seguente definizione è del tutto informale. Vedremo nel seguito come essa può essere completamente formalizzata.

Definizione 16. *Sia f una funzione da A^* in A^* (A alfabeto finito). Diciamo che f è calcolabile se esiste un sistema fisico che assunto uno stato che codifica un ingresso di f si evolve nel tempo raggiungendo uno stato finale solo se f è definita sull'ingresso considerato, ed in tal caso lo stato finale del sistema codifica il risultato che f associa a tale ingresso.*

Definizione 17. *Un linguaggio L su un alfabeto A si dice* ricorsivamente enumerabile *o* semidecidibile *o anche* effettivamente generabile *se L è il codominio di una funzione calcolabile. Indichiamo con RE la classe dei linguaggi ricorsivamente enumerabili.*

Definizione 18. *Sia A un alfabeto e $\forall n \in \mathbb{N}$ ¡n¿ una codifica di n in tale alfabeto (per esempio $<n>= a \cdots a$ n volte un $a \in A$). Sia $f : \mathbb{N} \to A^\star$ una funzione (totale) di enumerazione di stringhe, tale che $f(n) = \alpha$ se vi è una sequenza finita di stringhe $\alpha_1, \alpha_2, \cdots, \alpha_m$ tale che $\alpha_1 =<n>, \alpha_m = \alpha$, e α_{i+1}, per $i = 1, \cdots, m-1$, è ottenuta da α_i rimpiazzando una sottostringa γ con una sottostringa δ ove $|\gamma| \leq 2$ e $|\delta| \leq 2$. Diciamo in tal caso che l'insieme delle immagini $f(\mathbb{N})$ è* generabile per bi-rimpiazzamenti.

Come dimostreremo in seguito (vedi *Teorema Centrale di Rappresentazione*), risulta

$$\mathcal{L}_\mathbf{0} = \mathbf{RE}.$$

In genere si dice *computazionalmente universale* un qualsiasi formalismo che fornisca un metodo di costruzione per ogni linguaggio in RE.

Definizione 19. *Chiamiamo* **REC** *la classe dei linguaggi L la cui funzione caratteristica f_L tale che $f_L(x) = 1 \Leftrightarrow x \in L$ e $f_L(x) = 0 \Leftrightarrow x \notin L$ è calcolabile. Tali linguaggi si dicono* ricorsivi.

Vediamo una significativa caratterizzazione di REC.

Notazione. Indichiamo il complementare di un linguaggio L con $\overline{L} = A^\star \setminus L$.

Proposizione 20. $L \in REC \Leftrightarrow L, \overline{L} \in RE$.

Dimostrazione.

$\Rightarrow$) Per ogni parola di $A^\star$ sappiamo dire in tempo finito se appartiene ad L oppure a $\overline{L}$. Un algoritmo effettivo che genera L è quello che, prima genera tutte le stringhe di $A^\star$ secondo un certo criterio (per esempio seguendo l'ordine lessicografico), poi applica a ciascuna l'algoritmo di riconoscimento di L e nel caso la parola venga riconosciuta la produce in uscita. L'algoritmo effettivo che genera $\overline{L}$ è analogo, ma produce in uscita le parole accettate dall'algoritmo di riconoscimento di $\overline{L}$. Quindi $L, \overline{L} \in RE$.

$\Leftarrow$) Sia $\alpha \in A^\star$ una stringa di cui vogliamo testare l'appartenenza a L, e siano f_1 e f_2 le funzioni (Definizione 18) tali che $f_1(\mathbb{N}) = L$ e $f_2(\mathbb{N}) = \overline{L}$. Trattandosi di linguaggi complementari, esiste un naturale n per cui o $f_1(n)$ oppure $f_2(n)$ coincide con α, nel primo caso $\alpha \in L$ e nel secondo $\alpha \notin L$. Il calcolo di $f_1(n)$ e $f_2(n)$ per $n = 1, 2, 3, \cdots$ rappresenta un algoritmo che, presa una stringa, in un numero finito di passi sa dire se tale stringa appartiene o meno a L. Quindi $L \in REC$. □

Osservazione 21. $REC \subseteq RE$.

Osservazione 22. *Un modo semplice per vedere che le grammatiche sono numerabili è quello di osservare che sono individuate dalle loro produzioni, le quali, viste come concatenazione di un numero finito di regole, sono stringhe su un certo alfabeto, e quindi appartengono al suo universo linguistico, che è numerabile.*

Vediamo dunque che

Proposizione 23. $\mathcal{L}_1 \subset REC \subset RE$.

Dimostrazione.

REC $\subset$ **RE**) In base all'osservazione che $REC \subseteq RE$. Esibiamo ora un linguaggio non ricorsivo $L_k \in RE$: si numerino le grammatiche G_i e le parole α_i di $A^\star$, $\forall (i,j) \in \mathbb{N} \times \mathbb{N}$ si vada a controllare se G_i nei primi j passi ha trovato α_i, in caso positivo si metta α_i in L_k altrimenti no. Questo è un metodo generativo per costruire L_k, quindi $L_k \in RE$, ma non esiste un algoritmo capace di stabilire in tempo finito se una stringa appartiene a L_k o no. Infatti, si dimostra per assurdo che $\overline{L_k} = \{\alpha_i \,/\, \alpha_i \notin L(G_i)\} \notin RE$:
$\overline{L_k} \in RE \Rightarrow \exists\, d$ tale che $\overline{L_k} = L(G_d)$, ma $\alpha_d \in \overline{L_k} \Leftrightarrow \alpha_d \notin L(G_d) \Leftrightarrow \alpha_d \notin \overline{L_k}$.
L_k è ricorsivamente enumerabile ma per la Proposizione 20 non è ricorsivo, in quanto $\overline{L_k} \notin RE$, quindi $L_k \in RE \setminus REC$.

$\mathcal{L}_\mathbf{1} \subseteq$ **REC**) Vedremo che la condizione $|\alpha| \leq |\beta|$ su $\alpha \to \beta$ fa scendere nei decidibili, ovvero che, presa una stringa α e un linguaggio $L(G) \in \mathcal{L}_1$, si riesce a stabilire se $\alpha \in L$ o se $\alpha \notin L$. Consideriamo tutte le possibili sequenze appartenenti ad $A^\star$ di lunghezza al più $|\alpha|$, S, $\alpha_1, \alpha_2, \cdots, \alpha_n$, sia queste che le produzioni di G sono in numero finito. Quindi in tempo finito si riesce a verificare se α è derivabile tramite G o no, e quindi se $\alpha \in L(G)$ o no.

Grazie al Teorema Centrale di Rappresentazione, tutto questo è sufficiente per poter dire che

- $\mathcal{L}_1 \subset \mathcal{L}_0$

e quindi dimostrare una delle inclusioni del teorema 15, ma si ha un risultato ancora più forte:

$\mathcal{L}_\mathbf{1} \subset$ **REC**) Si dimostra con un ragionamento diagonale (alla Russell).
Si numerino le grammatiche G_i di $\mathcal{L}_1$ e le stringhe x_i dell'universo linguistico, poi si consideri $L_R \stackrel{def}{=} \{x_i \,/\, x_i \notin L(G_i)\}$. Siccome gli $L(G_i)$ sono decidibili questo linguaggio è ben definito ed è REC, ma non è generato da nessuna grammatica di tipo 1, quindi $L_R \in REC \setminus \mathcal{L}_1$. □

Osservazione 24. *A parte il linguaggio L_R definito nella dimostrazione precedente, non si conoscono altri linguaggi ricorsivi che non siano $\mathcal{L}_1$.*

Capitolo 4

Regolarità, Contestualità, Universalità

Passiamo alla caratterizzazione di $\mathcal{L}_3$.
Tutti i sistemi di riconoscimento si rifanno a meccanismi introdotti da Kleene che prendono il nome di **Automi a Stati Finiti (ASF)**. Intuitivamente un tale automa è da intendersi come una "scatola" M che può assumere un numero finito di stati Q, e messa in uno stato $q_0 \in Q$ (detto iniziale) prende in input una parola e la "mangia" lettera per lettera secondo delle regole di transizione: $Q \times A \rightarrow_t Q$ del tipo $(q, x) \mapsto_t q'$, arrivando alla fine in uno stato in cui tutta la parola di input è stata scandita. Se dopo che la parola è stata scandita M si trova in uno stato finale di $F \subseteq Q$, si dice che la parola viene *accettata* da M.

Definizione 25. *Un* automa *è una quintupla M = (A, Q, q_0, F, t).*

Definizione 26. *Il linguaggio generato da un automa M è*

$$L(M) = \{\alpha \ / \ q_0\alpha \Rightarrow^\star q \in F\}.$$

La differenza tra una grammatica e un automa sta rispettivamente nel generare/riconoscere una parola α:
nel primo caso $S \Rightarrow^\star \alpha$
nel secondo $q_0\alpha \Rightarrow^\star q \in F$.

Osservazione 27. *Un automa si può vedere come una "grammatica invertita".*

Esempio 28. *Il linguaggio* $\mathbf{L_g} = \{\mathbf{a^n b^m} \ / \ \mathbf{n, m} \in \mathbb{N}\}$ *con* $T = \{a, b\}$ *si può generare con una grammatica che ha le seguenti produzioni*

$$\begin{cases} S & \rightarrow & a \\ S & \rightarrow & b \\ S & \rightarrow & aS \\ S & \rightarrow & bS_1 \\ S_1 & \rightarrow & bS_1 \\ S_1 & \rightarrow & b \end{cases}$$

oppure con un automa che ha come stati $Q = \{q_1, q_2, q_3\}$, stati finali $F = \{q_1, q_2\}$, come stato iniziale q_1, come alfabeto $A = \{a, b\}$ e come transizioni:

$$\begin{cases} q_1 a & \to & q_1 \\ q_1 b & \to & q_2 \\ q_2 b & \to & q_2 \\ q_2 a & \to & q_3. \end{cases}$$

Infatti, con queste transizioni vengono riconosciute tutte e sole le parole del linguaggio.

Osservazione 29. *L'automa legge le stringhe da sinistra a destra.*

Per esempio la stringa a^3b^2a non viene accettata, perché nella lettura vengono necessariamente applicate, nell'ordine, le seguenti transizioni

$$\begin{array}{lcl} q_1 a & \to & q_1 \\ q_1 a & \to & q_1 \\ q_1 a & \to & q_1 \\ q_1 b & \to & q_2 \\ q_2 b & \to & q_2 \\ q_2 a & \to & q_3 \end{array}$$

e dopo aver letto la parola l'automa si trova nello stato non finale q_3, quindi non riconosce la stringa come facente parte del linguaggio. È evidente che tutte le volte che l'automa legge una a dopo una b finisce nello stato q_3.

Invece, una stringa del tipo a^nb^m viene sempre riconosciuta, perché l'automa applica le seguenti transizioni e finisce in q_2 che è un elemento di F:

$$\underbrace{q_1 a \to q_1\ ,\ \cdots\ ,\ q_1 a \to q_1}_{n \text{ volte}} \quad q_1 b \to q_2 \quad \underbrace{q_2 b \to q_2\ ,\ \cdots\ ,\ q_2 b \to q_2}_{m-1 \text{ volte}}.$$

Se $m = 0$ vengono effettuate solo le n transizioni che hanno nelle premesse il simbolo a.

Teorema 30 (Kleene)**.** $\{L(M) \,/\, M \in ASF\} = \mathcal{L}_3$.

Cenno di Dimostrazione. Data una grammatica di tipo 3, avente quindi produzioni $\alpha \to_G \beta$ con $\alpha = X \in N$ e $\beta \in T \cup TN$, si ha un $M \in ASF$ che genera lo stesso linguaggio, e viceversa:

$$\begin{array}{lcl} S & \to & aS_1 \\ S_1 & \to & bS_2 \\ & \vdots & \\ S_j & \to & b \end{array} \quad \Longleftrightarrow \quad \begin{array}{lcl} q_0 a & \to & q_1 \\ q_1 b & \to & q_2 \\ & \vdots & \\ q_j b & \to & q_f \end{array}$$

□

Definizione 31. *Chiamiamo* $REG = Clos(FIN, \cdot, *, +)$ *la classe dei* linguaggi regolari. *FIN rappresenta la classe dei linguaggi finiti, e Clos ne indica la chiusura algebrica rispetto alle tre operazioni indicate.*

Osservazione 32. *Nella definizione precedente FIN può essere sostituito dalla classe dei linguaggi costituiti dai singoletti dell'alfabeto, infatti ogni linguaggio finito è dato da unioni e concatenazioni di simboli.*

Teorema 33 (Kleene)**.** $\mathcal{L}_3 = REG$.

Dimostrazione. Andiamo a dimostrare, equivalentemente, l'uguaglianza $\{L(M) \ / \ M \in ASF\} = Clos(FIN, \cdot, *, +)$, ovvero che ogni $L(M)$, con $M \in ASF$, è ottenibile con unioni, concatenazioni e stella di Kleene di linguaggi finiti (su un alfabeto A). Il viceversa è ovvio.
In generale se $Q = \{q_1, \cdots, q_n\}$ sono gli stati di M, di cui q_1 è lo stato iniziale e $F = \{q_m, \cdots, q_n\}$ gli stati finali, consideriamo il linguaggio

$$L_{i,j}{}^n \stackrel{def}{=} \{\alpha \in A^\star \ / \ q_i\alpha \to q_{i_1}\beta \to \cdots \to q_j \in F, \ i, i_1, \cdots, j \in \{1, \cdots, n\}\}$$

fatto dalle parole α che permettono di passare da q_i a q_j attraverso stati intermedi che appartengono sempre a Q. Si osserva che $L_{i,j}{}^0 \subseteq A \ \forall i, j \in \mathbb{N}$ e che risulta

$$L_{i,j}{}^{k+1} = L_{i,k+1}{}^k \cdot (L_{k+1,k+1}{}^k)^* \cdot L_{k+1,j}{}^k + L_{i,j}{}^k \quad \forall i, j \in \mathbb{N}.$$

Infatti, per andare da q_i a q_j (avendo $k+1$ stati) si può passare o meno da q_{k+1} (il k+1-esimo stato), e i due tipi di percorso sono compresi rispettivamente in $L_{i,k+1}{}^k \cdot (L_{k+1,k+1}{}^k)^* \cdot L_{k+1,j}{}^k$ e in $L_{i,j}{}^k$. Pertanto il linguaggio generato da M è $L(M) = \{L_{1,m}{}^n \cup L_{1,m+1}{}^n \cup \cdots \cup L_{1,n}{}^n\}$, ottenuto evidentemente come unioni, concatenazioni e stella di Kleene di linguaggi finiti. □

Proposizione 34. $L \in \mathcal{L}_3 \Leftrightarrow$ *L è ottenibile da un opportuno linguaggio finito con un meccanismo di duplicazione cancellazione e filtraggio.*

L'equivalenza $\mathcal{L}_3 = REG$ si basa sul non determinismo degli ASF, ma gli automi deterministici, in cui ad ogni passo vi è un'unica transizione applicabile, generano la stessa classe di linguaggi, come risulta dal seguente teorema.

Teorema 35 (Kleene)**.** $ASF = DASF$.

Dimostrazione.
$\supseteq$) Gli automi a stati finiti deterministici sono, per definizione, un caso particolare di quelli non deterministici.
$\subseteq$) Dato un automa a stati finiti $M = (A, Q, q_0, F, \to)$ non deterministico, se ne può costruire uno equivalente deterministico:

$$M' = (A, \mathcal{P}(Q), \{q_0\}, F', \to')$$

dove $F' \stackrel{def}{=} \{X \subseteq Q \,/\, X \cap F \neq \emptyset\}$ e $M' : Xa \to' \bigcup_{q \in X}\{q' \,/\, qa \to q'\}$ dove $X \subseteq Q$ e $a \in A$. Praticamente si rende M deterministico passando attraverso il parallelismo. Gli stati finali di M' sono insiemi che contengono almeno uno stato finale di F, quindi per ogni percorso computazionale di M' che riconosce una certa stringa ve n'è almeno uno di M che riconosce la stessa stringa, e viceversa. Pertanto $L(M) = L(M')$. □

Un automa $M \in ASF$ presa una $\alpha \in A^\star$ alla fine della lettura si trova in uno stato in cui si conclude la computazione. Questo induce una partizione finita di $A^\star$ (è una relazione d'equivalenza ad indici finiti) dove

$$[q_i] = \{\alpha \,/\, q_0\alpha \Rightarrow^\star q_i\}.$$

In generale, dato $L \subseteq A^\star$, definiamo la seguente relazione (binaria) d'equivalenza:

$$\alpha \sim_L \beta \Leftrightarrow (\alpha\gamma \in L \Leftrightarrow \beta\gamma \in L \quad \forall\gamma \in A^\star)$$

che risulta invariante per concatenazioni destre, ossia

$$\alpha \sim_L \beta \Rightarrow \alpha\gamma \sim_L \beta\gamma \quad \forall\gamma \in A^\star.$$

Teorema 36 (Myhill-Nerode)**.** *Sia L un linguaggio. Ogni ASF determina una relazione d'equivalenza a indici finiti invariante per concatenazioni destre e viceversa, ovvero*

$$\{[\alpha]_{\sim_L} \,/\, \alpha \in A^\star\} \in \mathbb{N} \;\Leftrightarrow\; L = L(M),\ M \in ASF.$$

Corollario 37. *L'automa M_L del teorema è il minimo (ovvero quello col minimo numero di stati) tra quelli che accettano il linguaggio L.*

4.1 Linguaggi Contestuali

Definizione 38. *Una grammatica (su un alfabeto A) si dice* contestuale *quando le sue produzioni sono del tipo*

$$\alpha X\beta \to \alpha\gamma\beta \qquad X \in N,\ \alpha, \beta, \gamma \in A^\star, \gamma \neq \lambda.$$

La classe dei linguaggi generati da grammatiche contestuali si chiama **CS** *(*context sensitive*).*

Teorema 39. $CS = \mathcal{L}_1$.

Dimostrazione.

$\subseteq$) Una grammatica contestuale è monotona per definizione.

$\supseteq$) Le produzioni di una grammatica monotona sono del tipo

$$X_1X_2\cdots X_n \to Y_1Y_2\cdots Y_m \qquad m \geq n$$

e si possono facilmente tradurre con regole di una grammatica contestuale ove ogni simbolo X diventa X':

$$\begin{aligned}
\underbrace{\lambda}_{\alpha} X_1' \underbrace{X_2' \cdots X_n'}_{\beta} &\to \underbrace{\lambda}_{\alpha} Y_1' \underbrace{X_2' \cdots X_n'}_{\beta} \\
\underbrace{Y_1'}_{\alpha} X_2' \underbrace{X_3' \cdots X_n'}_{\beta} &\to \underbrace{Y_1'}_{\alpha} Y_2' \underbrace{X_3' \cdots X_n'}_{\beta} \\
&\vdots \\
\underbrace{Y_1' Y_2' \cdots Y_{n-1}'}_{\alpha} X_n' \underbrace{\lambda}_{\beta} &\to \underbrace{Y_1' Y_2' \cdots Y_{n-1}'}_{\alpha} Y_n' \cdots Y_m' \underbrace{\lambda}_{\beta}
\end{aligned}$$

Il punto cruciale in questo passaggio è che $m \geq n$. □

4.2 Linguaggi Liberi

Definizione 40. *Una grammatica (su un alfabeto A) si dice* acontestuale*, o più usualmente,* libera *quando le sue produzioni sono del tipo*

$$X \to \gamma \qquad X \in N,\ \gamma \in A^\star,\ \gamma \neq \lambda.$$

La classe dei linguaggi generati da grammatiche libere si chiama **CF** (context free).

Per definizione risulta

$$\mathbf{CF} = \boldsymbol{\mathcal{L}_2}.$$

Identificando il *tempo* con il numero di passi di un algoritmo in funzione della lunghezza della stringa da riconoscere, si dimostra che il tempo di riconoscimento di una stringa di $L \in \mathcal{L}_2$ è di ordine cubico rispetto alla lunghezza della stringa.

Proposizione 41. $CF \subseteq CS$.

Dimostrazione. Una grammatica libera è una grammatica contestuale particolare, ovvero le sue produzioni sono del tipo

$$\alpha X \beta \to \alpha \gamma \beta \qquad X \in N,\ \alpha, \beta, \gamma \in A^\star, \gamma \neq \lambda$$

come quelle di una grammatica contestuale, ma soddisfano l'ulteriore condizione $\alpha, \beta = \lambda$. □

Vediamo la caratterizzazione di Bar Hillel-Chomsky dei linguaggi CF:

Lemma 42 (Pumping)**.** *Sia $L \in CF$ infinito. $\exists\, p, q \in \mathbb{N}$:*

$$\forall \alpha \in L \quad con \quad |\alpha| \geq p \quad \exists\, u,\ v,\ w,\ x,\ y\ :$$

$$\alpha = uvwxy, \qquad |vwx| \leq q \quad vx \neq \lambda, \qquad \forall i \in \mathbb{N} \quad uv^i wx^i y \in L$$

v e x si dicono "ancelle" della parola w in α.

Cenno di Dimostrazione. L è infinito e le regole di una grammatica libera sono del tipo $X \to \gamma$ con $X \in N,\ \gamma \in A^\star, \gamma \neq \lambda$. Siccome N è finito, per il principio dei cassetti deve esistere un elemento $Y \in N$ *autogenerante*, cioè tale che partendo da Y e applicando le produzioni iterativamente si ottiene una parola contenente Y. Si può quindi ripetere questo stesso iter un numero arbitrario i di volte prima di far terminalizzare la Y con w.
La differenza espressiva tra $\mathcal{L}_2$ e $\mathcal{L}_1$ (o equivalentemente tra CF e CS) è che l'uno può esprimere al massimo simmetrie binarie, l'altro anche ternarie.

Corollario 43. $L_t \in \mathcal{L}_1 \setminus \mathcal{L}_2$.

Dimostrazione. La simmetria ternaria del linguaggio trisomatico è discriminante: essendo generato da una grammatica di tipo 1 (Proposizione 10) appartiene a $\mathcal{L}_1$, ma non può appartenere a $\mathcal{L}_2$ che ha simmetria binaria. □

Pertanto

Corollario 44. $CF \subset CS$.

E questo dimostra che $\mathcal{L}_2 \subset \mathcal{L}_1$, ovvero un'altra delle inclusioni del teorema 15.

Proposizione 45. $L \in CF \not\Rightarrow \overline{L} \in CF$.

Dimostrazione. Si osservi che CF è chiusa rispetto all'unione (unendo le produzioni di due grammatiche libere ottengo una grammatica libera che genera l'unione dei linguaggi corrispondenti alle grammatiche di partenza). Ma CF non è chiusa per intersezione: $a^n b^n c^m, a^p b^k c^k \in CF$ mentre $a^n b^n c^n \notin CF$.
Se fosse chiusa per complementazione, dati $L_1, L_2 \in CF$, secondo le leggi di De Morgan avremmo $\overline{L_1} \cup \overline{L_2} \in CF$ e $\overline{(\overline{L_1} \cup \overline{L_2})} = L_1 \cap L_2 \in CF$, che è assurdo. □

Corollario 46. $L_b \in \mathcal{L}_2 \setminus \mathcal{L}_3$.

Dimostrazione. Il linguaggio bisomatico è generato dalla grammatica $G_b = (\{a, b, S\}, \{a, b\}, S, P_b)$ di tipo 2, dove

$$P_b \begin{cases} S & \to & ab \\ S & \to & aSb \end{cases}$$

quindi $L_b \in \mathcal{L}_2$.
Se per assurdo $L_b \in \mathcal{L}_3$, allora $L_b = L(M)$ per qualche $M \in ASF$, e per il principio dei cassetti (le classi di equivalenza sono finite e i naturali non lo sono) deve esistere $k \in \mathbb{N} \setminus \{0\}$ tale che $[a^n] \simeq [a^{n+k}]$. Trattandosi di una relazione invariante per concatenazioni destre, M accetta $a^n b^n$ così come $a^{n+k} b^n$, e questo è assurdo perché, per $k > 0$, $a^{n+k} b^n \notin L_b$. □

Pertanto

Corollario 47. $\mathcal{L}_3 \subset \mathcal{L}_2$.

Questo corollario verifica l'ultima delle inclusioni della gerarchia di Chomsky, la cui completa dimostrazione si conclude con la dimostrazione del Teorema Centrale di Rappresentazione che sarà data in seguito. □

La scelta di termini come "alfabeto", "parola", "universo linguistico", "grammatica", "linguaggio", è motivata dal fatto che la gerarchia di Chomsky, delle grammatiche e dei linguaggi formali, è nata dallo studio della sintassi dei linguaggi naturali.

Secondo una terminologia alternativa a quella introdotta dal teorema di Chomsky sulle classi di linguaggi, possiamo esprimere la gerarchia di Chomsky come di seguito.

$$FIN \subset REG \subset CF \subset CS \subset REC \subset RE$$

La classificazione di Chomsky è di fondamentale importanza nella teoria dei linguaggi formali. Inoltre, si possono trovare classi intermedie tra REG e CF, tra CF e CS, tra CS e RE, e oltre RE entro tutta la classe dei linguaggi, e all'interno di tale gerarchia vi sono sottogerarchie (alcune infinite) estremamente complesse e articolate.

Ci si rende conto quindi come entro il numerabile vi sia una ricchezza di gradi di complessità generativa analoga a quella scoperta da Cantor per gli insiemi infiniti. La complessità generativa corrisponde a caratteristiche computazionali molto importanti. Vari linguaggi di grande interesse computazionale si trovano in classi intermedie tra CS e CF.

4.3 Contestualità e Universalità

Andiamo a vedere come il passaggio dalla contestualità all'universalità sia sorprendentemente semplice.

Definizione 48. *Una* proiezione *h è un particolare morfismo che cancella i simboli che non fanno parte di un certo alfabeto. Sia Π l'insieme delle proiezioni e FL una classe di linguaggi, definiamo $\Pi(FL)$ come la classe dei linguaggi $\{h(L) \,/\, h \in \Pi$ e $L \in FL\}$.*

Proposizione 49. $RE = \Pi(CS)$.

Dimostrazione. Sia $L \in \mathcal{L}_0$ e G la grammatica che lo genera.
Ogni regola di G non monotona è del tipo $\alpha x \beta \to \alpha\beta$ con $x \in N$, e $x \neq \lambda$. Aggiungendo il simbolo # a T e sostituendo questo tipo di regola con $\alpha x \beta \to \alpha \# \beta$ si ottiene una grammatica G' monotona, che genera un $L' \in CS$ avente come parole quelle di L intercalate da un certo numero di #.
Quindi, partendo da $L \subseteq A^\star$ e considerando il seguente morfismo h

$$\begin{cases} h(x) = x & \forall x \in A \\ h(\#) = \lambda \end{cases}$$

si deduce che esiste $L' \in CS$ tale che $L = h(L')$. □

Pertanto le parole di un qualsiasi linguaggio ricorsivamente enumerabile si ottengono da quelle di un linguaggio di CS per rimozione di un numero qualsivoglia di gratelle, ovvero attraverso un tipo di morfismo molto particolare.
Ma abbiamo un risultato ancora più forte.

Teorema 50 (Savich)**.** *Per ogni linguaggio $L \in RE$, esiste un linguaggio $L' \in CS$ tale che $L = \{\alpha \,/\, \exists n \in \mathbb{N} \; t.c. \; \alpha \#^n \in L'\}$ dove $\# \notin A$.*

Dimostrazione. Si suppone di avere G:0 che genera L RE. Trasformiamo G nel modo seguente.

1. Si apicizzano le regole di G :, quindi si trasformano le regole di tipo 0 secondo la trasformazione (aggiungendo $perp, \$, \#$ come non terminali): $\alpha' X' \beta' \to \alpha'\beta' \Longrightarrow \alpha' X \beta' \to \alpha' \perp \beta'$
2. $S \to \$S'$
3. $\perp X' \to X' \perp$
4. $\$X' \to X\$$
5. $\$ \perp \to \# \perp$
6. $\# \perp \to \#\#$

La grammatica così ottenuta è ovviamente di tipo 1. □

Definizione 51. *Dati i linguaggi L, L', si definisce il seguente* linguaggio dei prefissi $Pref(L, L') = \{\alpha \,/\, \alpha\beta \in L, \beta \in L'\}$.

Quindi il teorema di Savitch 50 dice che $RE = Pref(CS, REG)$, in quanto risulta che per ogni linguaggio $L \in RE$ esiste un linguaggio $L' \in CS$ tale che $L = Pref(L', \#^\star)$.

Osservazione 52. *La cancellazione caratterizza la non ricorsività.*

4.4 Macchine di Turing

Turing nel suo saggio "On Computable real numbers, with application to the Entscheidungsproblem (Proc. of the London Math. Soc., 2-42, 1936)" tenta di carpire l'essenza del calcolo osservando un uomo che computa. L'uomo legge, cancella, scrive simboli. Di fronte ad ogni simbolo decide cosa fare guidato da una serie *finita* di regole. In ogni momento ha una quantità di carta finita, ma potenzialmente non vi sono limiti alla quantità di carta disponibile.

Come spiegheremo, l'agente di calcolo suggerito da Turing in quel saggio è in linea di principio la macchina da calcolo più potente che possa essere definita.

Intuitivamente la macchina ideata da Turing è una "scatola" che può assumere un numero finito di stati fisici Q (tra i quali c'è lo stato iniziale q_0), con un nastro di lunghezza finita ma illimitata. Su ogni casella del nastro può esserci un solo simbolo di un alfabeto finito A (a cui appartiene anche un simbolo speciale B. La macchina è dotata di un organo di lettura che 'legge' una casella per volta. Essa si comporta secondo un **programma**, ovvero una sequenza finita di istruzioni, che sono esprimibili come elementi di $Q \times A \times A \times Q \times \{0, 1\}$, dove $\{0, 1\}$ (o un qualsiasi insieme di cardinalità due) indica l'insieme di movimenti dell'organo di lettura (a destra o a sinistra).

Per convenzione all'inizio della computazione il nastro contiene una stringa detta **input**, e la testina del lettore è posizionata sul primo simbolo diverso da B alla sinistra del'input (le caselle alla destra e alla sinistra dell'input sono contengono il simbolo B).

Per vedere come funziona una tale macchina si carichi un input e si vada nello stato iniziale q_0. Verranno eseguite le istruzioni del programma con il seguente significato:

$$(q_i, a_j, a_k, q_l, Y)$$

vuol dire che trovandosi a leggere il simbolo a_j nello stato q_i la macchina deve scrivere a_k al posto di a_j e passare nello stato q_l spostando l'organo di lettura secondo il movimento Y. Quando non vi sono più istruzioni applicabili la macchina si ferma. L'**output** della macchina quando si è fermata si trova nella parte di nastro compresa tra i simboli diversi da B posti all'estrema sinistra e e destra del nastro.

Una macchina M si può vedere sia come **accettore**, quando prende una stringa in input e dopo un tempo finito raggiunge uno stato finale, che come **generatore**, quando si considerano tutte le stringhe che alla fine delle computazioni si trovano sul nastro, al variare di tutte le possibili stringhe di ingresso.

M può esser vista anche come automa **input/output** che calcola una funzione $f : A^\star \to A^\star$, ovvero prende in input $\alpha \in A^\star$ e, dopo un tempo finito, scrive sul nastro la parola $f(\alpha)$.

Osservazione 53. *In generale il comportamento della MdT è non deterministico, infatti a parità di premesse (le prime due componenti dell'istruzione) si possono avere più azioni (rappresentate dalle ultime tre componenti dell'istruzione). Come tra le produzioni di una grammatica, in generale si hanno diverse istruzioni aventi le stesse premesse senza che ci sia alcun criterio di scelta.*

Osservazione 54. *La novità della macchina di Turing, rispetto ad altri automi, è che oltre a leggere essa può anche scrivere, ma la sua vera forza sta nella capacità di andare avanti e indietro, quindi nel poter elaborare l'informazione in modo bidirezionale.*

Ogni $M \in MdT$ può essere "ridotta" ad un'equivalente $M' \in MdT$ che lavora su un nastro semidefinito (ovvero che si può allungare solo in una direzione).
Infatti, se per esempio M' ha un nastro che si può allungare solo a destra, aggiungendo al programma una routine di copiatura (che va a cercare l'ultimo termine a destra $\neq B$, lo sposta di una casella a destra, e fa lo stesso per tutti i simboli precedenti) viene simulato l'allungamento a sinistra, e quindi con M' si possono fare tutte le operazioni possibili per M.

Può anche essere concepita una MdT con k nastri, avente A^k come alfabeto, Q^k come insieme di stati, un lettore con k "occhi", e k-uple di movimenti a destra e a sinistra nelle istruzioni del programma. Si dimostra la seguente proposizione.

Proposizione 55. *La potenza generativa di una MdT con k nastri è uguale a quella di una MdT con 1 nastro.*

Per definizione una MdT deterministica è un caso particolare di MdT non deterministica, eppure si dimostra che lavorare con MdT deterministiche non è affatto una limitazione.

Proposizione 56. $DMdT = MdT$.

Cenno di Dimostrazione. Per riportarsi ad una macchina deterministica basta scaricare il non determinismo delle regole sull'input. Per esempio, sia n il numero massimo di istruzioni aventi le stesse premesse, allora ad ogni input α si aggiunge un prefisso numerico avente lunghezza qualsiasi che varia in modo random su $\{1, 2, \cdots, n\}^\star$. In questo modo si introduce dall'esterno una strategia di scelta, togliendo il non determinismo della macchina.
Per esempio: se $q_i, a \rightarrow_1 \cdots, \quad q_i, a \rightarrow_2 q_j, b, dx, \quad q_i, a \rightarrow_3 \cdots$, la macchina che legge il seguente input $[23133 \cdots a \cdots]$ quando si trova nello stato q_i, sceglie la regola 2 secondo il prefisso, cancella il 2 dal prefisso e applica $\rightarrow_2$, passando quindi a leggere $[3133 \cdots b \cdots]$ nello stato q_j e con il lettore spostato

di una casella a destra.
Se il prefisso dovesse "esaurirsi" prima che la macchina abbia completato la computazione il processo si arresta e la macchina non produce niente, ciò non altera la sua potenza generativa, che viene valutata la variare di tutti i possibili ingressi, e il suo comportamento rimane deterministico. □

Una *macchina a registri* è un automa dotato di
i) un programma con le istruzioni numerate
ii) un contatore che in ogni istante contiene l'indirizzo dell'istruzione corrente
iii) una memoria avente un numero finito di locazioni numerate R_i, dette *registri* e contenenti numeri.

Le istruzioni del programma hanno i seguenti formati:

$$op_j\ k$$
$$op_{i,j}\ k$$
$$test_{i,j}\ h, k$$

dove con *op* viene indicata un'operazione aritmetica.
Nel primo caso l'istruzione dice di eseguire l'operazione *op* sul contenuto del registro j (per esempio incrementare o decrementare di 1 il contenuto di R_j) e di passare poi all'istruzione numerata con k. Nel secondo caso l'istruzione dice di eseguire l'operazione *op* tra R_i e R_j (per esempio copiare il contenuto di R_i in R_j, o incrementare R_i del contenuto di R_j) e di passare all'istruzione numerata con k. Nel terzo caso l'istruzione dice di confrontare il contenuto di R_i con quello di R_j per vedere se vale una certa relazione (generalmente l'uguaglianza), se vale tale relazione si passa all'istruzione numerata con h, altrimenti si passa all'istruzione numerata con k.
L'idea di base è avere locazioni e istruzioni numerate che gestiscono un processo di alterazione dei contenuti delle locazioni.

Si dimostra che una macchina di Turing si riconduce ad una macchina a registri.

Proposizione 57. *$MdT = RAM$ (Random Access Machine).*

Dimostrazione. Basta prendere le caselle della MdT (che chiamiamo $R_1, R_2, \cdots, R_k$) "elastiche" (ognuna in grado di contenere un naturale), un modo per incrementare di 1 il contenuto di una casella/registro INC_j e uno per decrementare DEC_j ($j \in \{1, 2, \cdots, k\}$), e infine far diventare il programma una lista di operazioni indirizzate (eliminando così gli spostamenti a destra e a sinistra) più un contatore che contiene l'indirizzo dell'istruzione corrente. □

Teorema 58 (Shannon '56)**.** *Per ogni $M \in MdT$ esiste $M' \in MdT$ equivalente avente solo due stati. Per ogni $M \in MdT$ esiste $M' \in MdT$ equivalente avente solo due simboli.*

Notazione. Indichiamo con $|A|$ la cardinalità di un insieme A.

Definizione 59. *Chiamiamo* dimensione simboli-stati *di una macchina di Turing universale M il numero $|A| \times |Q|$, dove A e Q sono l'alfabeto e l'insieme degli stati di M.*

La minimizzazione della dimensione delle macchine di Turing universale è di grande interesse, perché significa indagare su strutture "piccole" in grado di "calcolare tutto il calcolabile". Attualmente la dimensione più piccola che è stata trovata è 28.

Definizione 60. *Una funzione $f : A^\star \to A^\star$ si dice* Turing calcolabile *se esiste una $M_f \in MdT$ che prendendo una stringa $\alpha \in A^\star$ come input mi dà $f(\alpha)$ come output.*

Per esempio, sia $A = \{0, 1\}$.
$f : \mathbb{N}^k \to \mathbb{N}$ è Turing calcolabile se esiste $M_f \in MdT$ tale che

$$M_f : 1^{n_1+1}\, 0\, 1^{n_2+1}\, 0\, \cdots\, 1^{n_k+1}\, 0 \mapsto 1^{m+1} \Leftrightarrow f(n_1, n_2, \cdots, n_k) = m.$$

Ovvero se esiste una macchina di Turing che con la codifica

$$\begin{cases} 0 & \mapsto & 1 \\ 1 & \mapsto & 11 \\ \vdots & & \\ n & \mapsto & 1^{n+1} \end{cases}$$

calcola la funzione f.

Esempio 61. *Una MdT con le seguenti istruzioni, dove R e L indicano rispettivamente gli spostamenti a destra e sinistra e q_4 è uno stato finale, calcola deterministicamente la somma tra due numeri secondo la codifica appena vista.*

1. $(q_0, 1, 1, q_0, R)$
2. $(q_0, 0, 1, q_1, R)$
3. $(q_1, 1, 1, q_1, R)$
4. $(q_1, 0, 0, q_2, L)$
5. $(q_2, 1, 0, q_3, L)$
6. $(q_3, 1, 0, q_4, L)$

Supponiamo infatti di voler calcolare $f(n, m) = n + m$.
Carichiamo il nastro con la sequenza $[\ \underbrace{1 \cdots 1}_{n+1\ volte}\ 0\ \underbrace{1 \cdots 1}_{m+1\ volte}\ ]$ secondo la codifica.
La macchina esegue nell'ordine

- *$n+1$ volte la prima istruzione, scorrendo così la prima sequenza di 1 dell'input*
- *1 volta la seconda istruzione, con la quale scrive 1 al posto dello 0 che nell'input separa le due sequenze di 1, modificando così la stringa iniziale in una lista di $n+m+3$ simboli uguali a 1, e passa dallo stato q_0 allo stato q_1*
- *$m+1$ volte la terza istruzione, scorrendo così l'ultima sequenza di 1 dell'input*
- *1 volta la quarta istruzione, con la quale passa nello stato q_2 e posiziona il lettore sull'ultimo degli 1 che ci sono sul nastro*
- *1 volta la quinta istruzione, con la quale cancella l'ultimo degli 1 (sostituendolo con lo 0) e passa nello stato q_3*
- *1 volta la sesta istruzione, con la quale cancella quello che ora è diventato l'ultimo degli 1 (sostituendolo con lo 0) e passa nello stato q_4.*

Non essendoci altre istruzioni applicabili e trovandosi la macchina nello stato finale q_4 la computazione termina, fornendo come output la sequenza $\underbrace{1\cdots1}_{n+m+1\ volte}$ *, che secondo la codifica rappresenta il numero $n+m$.*

Osservazione 62. *Il calcolabile si riconduce chiaramente ai naturali, nella pratica infatti si può avere a che fare solo con troncamenti di sequenze infinite, quindi con razionali che riportiamo ai naturali.*

Notazione. Scriviamo $M(\alpha) = \beta$ quando M con input α produce β come output.

Definizione 63. *Data $M \in MdT$,* L(M) *può essere definito o come* linguaggio di accettazione

$$Input(M) = \{\alpha \in A^\star \,/\, \exists \beta \in A^\star \quad M(\alpha) = \beta\}$$

o come linguaggio di generazione

$$Output(M) = \{\alpha \in A^\star \,/\, \exists \beta \in A^\star \quad M(\beta) = \alpha\}.$$

Risulta infatti che

Teorema 64. $\{Input(M) \,/\, M \in MdT\} = \{Output(M) \,/\, M \in MdT\}$.

Dimostrazione.

$\supseteq$) Consideriamo il nastro di M' suddiviso in due parti (la potenza generativa della macchina rimane equivalente), nella prima parte scriviamo una stringa α, e nella seconda una stringa generata non deterministicamente $\beta \in A^\star$. Si fa lavorare M sulla seconda parte del nastro, se $M(\beta) = \alpha$, allora α viene accettata, ovvero M' finisce in uno stato di accettazione, altrimenti no. È chiaro che M' riconosce tutte le stringhe che genera M, quindi $\alpha \in Output(M) \Rightarrow \alpha \in Input(M')$.

$\subseteq$) M' abbia nuovamente il nastro suddiviso in due parti. Presa una stringa $\alpha \in Input(M)$, ne scriviamo una copia su ciascuna parte del nastro, e su una delle due facciamo lavorare M. Quando M riconosce α producendo una certa stringa β M' si ferma, ignora la parte del nastro contenente β e genera la parola α che trova scritta sull'altra parte. Quindi $\alpha \in Output(M')$. □

Osservazione 65. *Se $L \neq \emptyset$ e $L = Output(M)$, allora vi è una M' tale che $L = Output(M')$ ed M' è una macchina totale, ovvero che si ferma per qualsiasi ingresso.*
Infatti M' funziona prendendo un ingresso lungo n e "decodificandolo" nella coppia (i, j) di $\mathbb{N} \times \mathbb{N}$ (calcolando la funzione enumerativa del quadrato di Cantor), quindi eseguendo j passi di M sulla stringa α_i (in un qualche ordinamento effettivo di stringhe, per esempio quello lessicografico), se in j passi M si ferma su α_i (ovvero la riconosce in non più di j passi) M' la produce in uscita, altrimenti M' fornisce un certo α_0 di L.

Abbiamo una versione per funzioni della famosa

- **Tesi della universalità computazionale (Church '36).**
  f è calcolabile sse (previa codifica) è Turing calcolabile.

e una versione per linguaggi

- **Tesi della universalità computazionale (Church '36).** *Un linguaggio è effettivamente generabile sse si riesce a generarlo con una macchina di Turing.*

la quale ci permette di definire RE usando una terminologia alternativa a quella della Definizione 18.

Definizione 66. $RE = \{Output(M) \,/\, M \in MdT\}$.

Quindi ogni $M \in MdT$ si può vedere come un particolare automa che genera linguaggi.

Teorema 67 (Centrale di Rappresentazione)**.** $RE = \mathcal{L}_0$.

Dimostrazione.

$\subseteq$) Secondo la precedente definizione di linguaggi ricorsivamente enumerabili, sia $L(M) \in RE$ con $M \in MdT$. Prendiamo la grammatica G_M avente le produzioni scritte di seguito, che suddividiamo in Regole di Ingresso, di Programma, di Uscita e di Allungamento, e vediamo come tale grammatica generi lo stesso linguaggio della macchina di Turing M, di cui riesce a simularne le operazioni.

$$\textit{Regole di Ingresso} = \begin{cases} S_0 & \to & \$S_1 \\ S_1 & \to & S\sharp \\ S & \to & Sx \quad x \in A \\ S & \to & q_0 \end{cases}$$

dove S_0 è il simbolo iniziale e q_0 un simbolo che rappresenta lo stato iniziale. Ci sono due simboli che rappresentano l'inizio e la fine del nastro $\$ \cdots \sharp$.
La terza produzione simula la trascrizione di simboli dell'alfabeto sul nastro, e produrre la stringa $\$q_0\alpha\sharp$ con queste istruzioni corrisponde a caricare la macchina con un ingresso α.

$$\textit{Regole di Programma} = \begin{cases} q_i x & \to & y q_j \\ q_i x & \to & \bar{q}_j y \\ z\bar{q}_j & \to & q_j z \end{cases}$$

dove $\bar{q}_j$ sono simboli che non appartengono al programma P della macchina.
La prima di queste regole si ha $\forall(q_i, x, y, q_j, dx) \in P$ (istruzioni con spostamento a destra) e le altre due $\forall(q_i, x, y, q_j, sn) \in P$ (istruzioni con spostamento a sinistra).

$$\textit{Regole di Uscita} = \begin{cases} q_f x & \to & x q_f \\ q_f \sharp & \to & q_\sharp \, \sharp' \\ B q_\sharp & \to & q_\sharp \, \sharp' \\ x q_\sharp & \to & q_{f'} x \quad x \neq B \end{cases}$$

dove q_f è uno stato finale e $q_{f'}$ non sta nel programma.
Queste istruzioni spostano q_f fino alla fine del nastro e poi simulano la sostituzione di B con $\sharp'$ da destra a sinistra, ci saranno delle analoghe regole di uscita (con $\$'$ invece di $\sharp'$) su $q_{f'}$ che va all'inizio del nastro.
Si aggiungono poi le altre

$$\textit{Regole di Uscita} = \begin{cases} \sharp' & \to & \lambda \\ \$' & \to & \lambda \end{cases}$$

e infine le regole che ci permettono di poter allungare il nastro, che servono a non far fermare processi che richiedono spostamenti all'estrema destra o all'estrema sinistra

$$\textit{Regole di Allungamento} = \begin{cases} \sharp & \rightarrow & B\sharp \\ \$ & \rightarrow & \$B. \end{cases}$$

Dalla rappresentazione appena data si ottiene che la MdT è simulabile con un **sistema di rimpiazzamento binario** (ovvero tutte le stringhe che figurano nelle regole non hanno più di due simboli).
Abbiamo dunque $L(G_M) = \{\alpha \,/\, S_0 \Rightarrow^\star \alpha\}$ e per costruzione risulta $L(G_M) = \mathit{Output}(M)$. Questo vuol dire che per ogni $M \in MdT$ esiste una grammatica G_M tale che $L(M) = L(G_M)$, e si tratta di una grammatica di tipo 0.

$\supseteq$) Data una grammatica G di tipo 0, possiamo costruire una $M \in MdT$ tale che $L(G) = L(G_M)$.
La macchina inizia con S sul nastro. Ogni volta che sul nastro trova α, e $\alpha \rightarrow_G \beta$, avvia una procedura che riscrive α in β. Eseguita una riscrittura si va in uno stato particolare col quale ci si muove *non deterministicamente* in un punto a caso del nastro, e da lì, spostandosi verso destra, si cerca nuovamente la prima produzione applicabile. □

Corollario 68 (Forma Normale di Kuroda, '64)**.** *Ogni grammatica è equivalente ad una con regole di cancellazione (del tipo $X \rightarrow \lambda$) e regole del tipo $X \rightarrow Y$, $XY \rightarrow UV$, $X \rightarrow YZ$.*

Dimostrazione. Segue direttamente dal teorema precedente. Ogni grammatica di tipo 0 è equivalente ad una macchina di Turing che è equivalente ad un sistema di rimpiazzamento binario del tipo espresso nell'enunciato. □

Definizione 69. *Con* **2RE** *si indica la classe dei linguaggi ricorsivamente enumerabili generati da un sistema di rimpiazzamento binario (o grammatica di Kuroda).*

Dal Teorema Centrale di Rappresentazione segue direttamente (per costruzione) che

Teorema 70. $\mathcal{L}_0 = 2RE$.

Così come ogni grammatica di tipo 0 può essere rappresentata in forma di Kuroda, una grammatica libera si può riportare nella **Forma di Chomsky**:

$$\begin{cases} X & \rightarrow & YZ \\ X & \rightarrow & a \end{cases}$$

Altri esempi di Forme Normali sono

- Forma di Penttonen

(abbiamo contestualità laterale perché possiamo utilizzare solo 2 simboli, e in più uno dei due non viene trasformato)

$$\begin{cases} AB & \to & AC \\ & oppure & \\ AB & \to & CB \end{cases}$$

- Forma di Geffert

  (forma S-lineare, perché S compare una sola volta)

$$\begin{cases} S & \to & uSv \\ ABC & \to & \lambda \end{cases}$$

con $S, A, B, C \in N$ e $u, v \in \{A, B, C\}^\star \cup T$ (ovvero in u e v non c'è S).

Definizione 71. *Un* LBA (Linear Bounded Automa) *è una Macchina di Turing che lavora in uno spazio lineare rispetto alla stringa α d'ingresso (se accetta) o d'uscita (se genera).*

Un LBA lavora in esattamente $k|\alpha|$ caselle per una certa costante k, che può essere presa uguale a 1 (per quanto visto nella Proposizione 55).

Definizione 72. $NLINSPACE \stackrel{def}{=} \{L \,/\, L = L(M),\ M \in LBA\}$ *è la classe dei linguaggi generabili in modo non deterministico in uno spazio lineare.*

Notazione. P indica *polinomiale* ed NP indica *polinomiale non deterministico.*

Vediamo una interessante caratterizzazione di CS.

Teorema 73 (Kuroda)**.** $CS = NLINSPACE$.

Dimostrazione.

$\supseteq$) Si tratta di dimostrare che se $M \in LBA$ allora $L(M) \in CS$, e questo discende dal teorema centrale di rappresentazione. Si consideri infatti la grammatica G_M vista nella dimostrazione del Teorema 67, e si tolgano le regole di uscita (che contengono tutte le regole non monotone di G_M) e quelle di allungamento che di conseguenza sarebbero inutili. Togliere queste regole significa considerare un $M \in LBA$ perchè, non essendoci più la possibilità di allungare, M lavora in esattamente $|\alpha|$ caselle.
Nel generare la parola α M ha bisogno di lavorare su due copie di α (per quanto visto nella dimostrazione del Teorema 64), ma questo non altera la nostra dimostrazione perché lavorare su due nastri è equivalente a lavorare su uno (vedi prop. 55). La grammatica corrispondente a L(M) è ora di tipo CS.

$\subseteq$) Sia $L \in CS$, costruiamo in questi termini una macchina che lo accetti: si metta la generica stringa α sul nastro e si applichino le regole della grammatica che genera L lette al contrario (premessa al posto di conseguenza e viceversa), se si ottiene S (simbolo iniziale) α viene accettata come parola facente parte di L, altrimenti no. Siccome la grammatica che genera L è monotona, quello che abbiamo costruito è un LBA che riconosce L. □

Lemma 74. *Se $M \in LBA$ esiste $M' \in LBA$ tale che $L(M') = \overline{L(M)}$.*

Cenno di dimostrazione. La prova del lemma si basa sul fatto che è possibile rappresentare in uno spazio lineare il numero di configurazioni assumibili da un LBA a partire da un input fissato.
Combinando il lemma 74 con il teorema di Kuroda 73 si ottiene subito la chiusura per complementazione di CS.

Corollario 75 (Immermann '87)**.** $L \in CS \Leftrightarrow \overline{L} \in CS$.

Capitolo 5

Chiusura e Decidibilità

La teoria dei linguaggi formali è caratterizzata da quattro tipi di teoremi:

1. Universalità / Gerarchia

   (stabilire la posizione di una classe di linguaggi nella gerarchia di Chomsky - vedi Definizione 14).

2. Equivalenza / Normalizzazione

   (per esempio. REG = L(ASF), o anche MdT=DMdT).

3. Chiusura / Non Chiusura

   (verificare se determinate operazioni, per esempio $\cup, \setminus, \cap, h, \cdots$, applicate a linguaggi di una classe, producono sempre linguaggi della stessa classe oppure no).

4. Decidibilità / Indecidibilità

   (stabilire se certe proprietà possono essere sempre stabilite con metodi algoritmici oppure no).

   Nei capitoli precedenti si sono presentati vari risultati relativi ai primi due tipi di teoremi, ora mostreremo alcuni importanti risultati relativi agli altri due.

Osservazione 76. *I problemi si possono vedere come insiemi di coppie (dati, soluzione corrispondente), quindi, previa un'opportuna codifica, possono essere studiati equivalentemente come linguaggi. In tal modo un problema si riduce alla decisione di appartenenza di una stringa ad un linguaggio.*

Vediamo un importante teorema di Chiusura.

Teorema 77 (Ginzburg)**.** *Sia $\mathcal{C}$ una classe di Chomsky, allora*

$$L \in \mathcal{C} \Rightarrow L \cap R \in \mathcal{C} \quad \forall R \in REG.$$

Dimostrazione. Il teorema si dimostra per casi. Nel seguito, diremo "apicizzare una grammatica G" per intendere una trasformazione che produce una nuova grammatica, ove per ogni suo simbolo A di G si aggiunge un nuovo simbolo A' (non terminale) e di conseguenza ogni regola viene tradotta in una regola con i corrispondenti simboli apicizzati, aggiunta alle regole della grammatica.

Se $L \in RE$ l'enunciato è ovvio.

Se $L \in REG = \mathcal{L}_3$, basta dimostrare che l'intersezione di due linguaggi regolari è regolare. Infatti, REG è chiusa per unione (unendo le produzioni di due grammatiche regolari si ottiene una grammatica regolare che genera l'unione dei linguaggi generati dalle grammatiche di partenza). REG è chiusa per complementazione: sia $L = L(M)$ con $M = (A, Q, q_0, F, t)$ automa a stati finiti, allora $\overline{L} = L(M')$ con $M' = (A, Q, q_0, Q \backslash F, t)$, che è sempre un automa a stati finiti. Quindi REG è chiuso per intersezione (in quanto $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$).

Consideriamo ora i casi CF e CS. Sia $R \in REG$, sappiamo che $\exists M = (A, Q, q_0, F, t) \in ASF$ tale che $R = L(M)$. 'Indicheremo con $\equiv$ la traduzione di una regola di G in una nuova regola della grammatica del linguaggio intersezione che si vuole costruire.

- $\underline{L \in CF\,,\, R \in REG \Longrightarrow L \cap R \in CF}$

  Si Trasformi G, utilizzando M, in G' avente gli stessi terminali di G e come non terminali i simboli apicizzati e indicizzati.

  - $S \to S_{q_0,q_f}\ \forall q_f \in F$
  - $X \to YZ \equiv X_{q_1,q_2} \to Y_{q_1,q_3} Z_{q_3,q_2}\ \forall q_1, q_2, q_3 \in M$
  - $X \to Y \equiv X_{q_1,q_2} \to Y_{q_1,q_2}\ \forall q_1, q_2 \in M$
  - $a_{q,q'} \to a \equiv qa \to q'\ \forall a \in T\,,\ \forall q, q' \in M$

  Questa grammatica, partendo da $S_{q_0 q_f}$, arriva a generare stringhe del tipo $a_{q_0 q_1} b_{q_1 q_2} \cdots c_{q_k q_f}$, dove q_0 è uno stato iniziale e q_f uno stato finale. Un tale tipo di stringa terminalizza solo se l'automa riconosce la parola secondo la sequenza computazionale $aq_0 \to q_1, bq_1 \to \cdots \to q_k, cq_k \to q_f$.

  Consideriamo finalmente il caso CS.

- $\underline{L \in CS\,,\, R \in REG \Longrightarrow L \cap R \in CS}$

  Si estenda G nella grammatica G', utilizzando M, dapprima apicizzando G, quindi aggiungendo le seguenti regole, in cui i simboli indicizzati e apicizzati sono non terminali, mentre i terminali coincidono con quelli di G.

  - $X'_{q_0}\alpha \to Y'_{q_0}\beta \quad \forall \ X'\alpha \to Y'\beta \in G'$

- $a'_{q_i} b' \to a b'_{q_j} \quad \forall \; q_i a \to q_j \in M$
- $b'_{q_i} \to b \quad \forall \; q_i b \to q_f \in M$

Questa grammatica, partendo da S_{q_0}, genera stringhe praticamente uguali a quelle che avrebbe generato G, con la sola differenza di avere degli apici e l'indice q_0 sempre come indice del simbolo alla estrema sinistra. Nel caso che si pervenga in tal modo ad una stringa costituita interamente da terminali apicizzati, l'indice q_0 si sposta secondo le ultime due regole di sopra terminalizzando se e solo se la stringa corrispondeva ad una stringa accettata dall'automa M. □

Definizione 78. *Si dicono* AFL (Abstract Family Language) *le classi di linguaggi chiuse rispetto a* $\cdot, +, *, h, h^{-1}, \cap R$*, dove* h *e* h^{-1} *indicano rispettivamente un morfismo e il suo inverso, e* $R \in REG$*.*

Teorema 79. *Le classi di Chomsky sono AFL.*

Corollario 80. *Le classi AFL sono chiuse per gsm (Definizione 97) non erasing (Definizione 4).*

Il seguente teorema esprime una sorta di compattezza, dice che l'informazione necessaria a caratterizzare un morfismo è un'informazione finita.

Teorema 81. *Dati due morfismi* h_1, h_2*, e un linguaggio* $L \in RE$ *esiste* $F \subset L$ finito *tale che*

$$h_1 = h_2 \text{ su } F \Leftrightarrow h_1 = h_2 \text{ su } L.$$

Questo risulta particolarmente interessante alla luce di quanto segue.

Teorema 82. *Sia* $L \in RE$*. Esistono* h_1, h_2 *morfismi, un linguaggio* $R \in REG$ *e una proiezione* h *tale che*

$$L = h(Eq(h_1, h_2) \cap R)$$

dove $Eq(h_1, h_2) \stackrel{def}{=} \{\alpha \;/\; h_1(\alpha) = h_2(\alpha)\}$.

Dimostrazione.

$\subseteq$) Sia $L \in RE$, $G = (A, T, S, P)$ una grammatica tale che $L = L(G)$, e α_k una generica parola di L, quindi costituita da simboli terminali e ottenibile tramite una certa derivazione $S \Rightarrow_G \alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \cdots \Rightarrow_G \alpha_k$.
Definiamo un alfabeto $A^{\perp} = A \cup \{B, \#, P_1, \cdots, P_n, F\} \cup T'$ dove T' è l'insieme di tutti i simboli di T apicizzati, P_i è un simbolo speciale che si ha per ogni regola $\alpha_i \to_G \beta_i$ di P, B,# e F sono dei simboli aggiunti.
Su tale alfabeto definiamo i morfismi h_1 e h_2 come segue

	B	$\#$	P_i	X	a'	a	F
h_1	$S\#$	$\#$	β_i	X	a	λ	λ
h_2	λ	$\#$	α_i	X	a	a	$\#$

dove $a \in T$ e $X \in N$ (ricordiamo che con N si indica $A \setminus T$, par.**??**).

Notazione. Con A^+ si indica $A^\star \setminus \lambda$.

Sia infine $R \stackrel{def}{=} B \cdot [(N \cup T')^* \cdot P \cdot (N \cup T')^*]^+ \cdot T^\star \cdot F$ un linguaggio regolare, e h il morfismo che cancella tutti i simboli di $A^\perp \setminus T$.
Risulta $\alpha_k \in h(Eq(h_1, h_2) \cap R)$, e l'idea di base di una tale costruzione è quella di simulare la derivazione di una parola α_k come la "corsa" di due morfismi che si uguagliano solo sull'ultimo passo della derivazione, ovvero su α_k.
Vediamone un esempio concreto.
L sia generato da una grammatica con le seguenti produzioni:

$$\begin{array}{lllll} p_1: & S & \rightarrow & ACCC \\ p_2: & CC & \rightarrow & CD \\ p_3: & AC & \rightarrow & a \\ p_4: & DC & \rightarrow & ACC \\ p_5: & AC & \rightarrow & \lambda \\ p_6: & C & \rightarrow & b \end{array}$$

e sia per esempio ab la parola di L di cui vogliamo dimostrare l'appartenenza a $h(Eq(h_1, h_2) \cap R$. Essa è prodotto della derivazione $S \rightarrow_{p_1} ACCC \rightarrow_{p_2} ACDC \rightarrow_{p_3} aDC \rightarrow_{p_4} aACC \rightarrow_{p_5} aC \rightarrow_{p_6} ab$.

Consideriamo quindi la seguente stringa

$$\gamma_{ab} = BP_1\#AP_2C\#P_3DC\#a'P_4\#a'P_5C\#a'P_6\#abF$$

ottenuta dalla derivazione di ab apicizzando tutti i simboli terminali, sostituendo ad ogni sottostringa che è premessa della regola p_i che verrà applicata il simbolo P_i corrispondente, e alla $\rightarrow$ il simbolo #, e poi mettendo B all'inizio della stringa e F alla fine. Una tale stringa appartiene evidentemente ad R.
Osserviamo che $\gamma_{ab} \in Eq(h_1, h_2)$ (in quanto $h_1(\gamma) = h_2(\gamma) = S\#ACCC\#ACDC\#aDC\#aACCC\#aC\#ab\#$) e che $h(\gamma_{ab})$ è proprio ab.
In generale dunque, presa $\alpha_k \in T^\star$ questa appartiene a $h(Eq(h_1, h_2) \cap R)$ per come sono stati definiti h, h_1, h_2 e R, infatti si costruisce facilmente col criterio visto sopra una $\gamma_k \in Eq(h_1, h_2) \cap R$ e tale che $h(\gamma_k) = \alpha_k$.

$\supseteq$) Una volta costruiti i morfismi h, h_1, h_2 (h individua un insieme T) e definito il linguaggio regolare R come sopra, si tratta di verificare che $\alpha \in h(Eq(h_1, h_2) \cap R) Rightarrow h(\alpha) \in L$. Per dimostrare tale inclusione si verifica che $h_1(\alpha) = h_2(\alpha)$ se α rappresenta la derivazione di una stringa di L. Per come sono definiti i due morfismi vi è uguaglianza solo alla fine della stringa che rappresenta tale derivazione, applicando poi a questa il morfismo h si ottiene la parte finale di tale derivazione che coincide proprio con la stringa generata. □

Ovvero i concetti di morfismo e regolarità sono sufficientemente potenti da esprimere tutti i linguaggi ricorsivamente enumerabili. Ma questo teorema può essere rinforzato, e in questo caso è il concetto di morfismo a diventare fondamentale:

Teorema 83. *Dato $L \in RE$ esistono due morfismi h_1, h_2 ed una proiezione h tali che $L = h(Eq(h_1, h_2))$.*

Vediamo un'applicazione di questi teoremi di indecidibilità.

PCP (Post's Correspondence Problem)

Supponiamo di avere quante copie vogliamo di n distinte tessere di domino così fatte

$$\boxed{\begin{array}{c}\alpha_1\\ \hline \beta_1\end{array}} \qquad \boxed{\begin{array}{c}\alpha_2\\ \hline \beta_2\end{array}} \qquad \cdots \qquad \boxed{\begin{array}{c}\alpha_n\\ \hline \beta_n\end{array}}$$

dove α_i, β_i con $i = 1, 2, \cdots, n$ sono parole su un certo alfabeto.
Ci si chiede se esiste una sequenza (di lunghezza k qualsiasi) di questi mattoncini, $i_1, i_2, \cdots, i_k \in \{1, 2, \cdots, n\}$, tale che $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_k}$.

Equivalentemente, possiamo prendere come alfabeto $\{1, 2, \cdots, n\}$ e formulare il PCP come la ricerca di una parola γ su tale alfabeto tale che $h_1(\gamma) = h_2(\gamma)$, dove

$$\begin{cases} h_1(1) = \alpha_1 & h_1(2) = \alpha_2 & \cdots & h_1(n) = \alpha_n \\ h_2(1) = \beta_1 & h_2(2) = \beta_2 & \cdots & h_2(n) = \beta_n. \end{cases}$$

Ovvero, risolvere un problema di Post significa prendere due morfismi e andare a vedere se $Eq(h_1, h_2) \neq \emptyset$.

Corollario 84. *In generale non esiste un algoritmo che risolve il problema di Post.*

Dimostrazione. Si dimostra per assurdo.
Sia $L \in RE$, per il teorema sopra $\exists h \in \Pi$ tale che $L = h(Eq(h_1, h_2))$ per una qualche coppia di morfismi h_1, h_2, che individua a sua volta un PCP.
Saper risolvere il problema di Post vuol dire saper dire se $Eq(h_1, h_2) = \emptyset$ o no, e quindi saper dire se $L = \emptyset$ o no, ma come abbiamo visto (nel corollario del teorema di Savich) questo è un problema indecidibile. □

Osservazione 85. *Dati due morfismi h_1 e h_2, $Eq(h_1, h_2) \in REC$.*
Infatti, presa una stringa α sappiamo sempre dire se $\alpha \in Eq(h_1, h_2)$ o no, basta vedere se $h_1(\alpha) = h_2(\alpha)$ o no.
Quindi, per un problema di Post, se potessimo sapere che esiste una soluzione potremmo anche trovarla in tempo finito, basterebbe numerare le parole e verificarne via via l'appartenenza a $Eq(h_1, h_2)$.

Corollario 86. *Non può esistere un algoritmo che in tempo finito ci dica se un linguaggio di CS è vuoto o no: è infatti un problema indecidibile.*

Dimostrazione. Sia $L \in RE$. Per il teorema di Savitch $\alpha \in L \Leftrightarrow L' \cap \alpha\{\#\}^\star \neq \emptyset$, dove $L' \cap \alpha\{\#\}^\star \in CS$ per il Teorema 77 di Ginzburg (in quanto intersezione di un linguaggio regolare con uno appartenente a CS). Siccome l'appartenenza di una stringa ad un generico linguaggio ricorsivamente enumerabile è un problema indecidibile, non esiste un algoritmo che ci permette di dire in tempo finito se un linguaggio CS è diverso dall'insieme vuoto. Si osservi che, a maggior ragione, non sappiamo dirlo per un generico linguaggio $L \in RE$. □

Capitolo 6

Rimpiazzamento, Trasduzione, Regolazione

Nel '68 Lindenmayer studiava l'accrescimento delle alghe rosse, e pensò di descriverlo con una grammatica. Si accorse però che le grammatiche di Chomsky non erano adatte a tale scopo, infatti, con la riscrittura di sottostringhe si riesce a descrivere una mutazione (trasformazione locale) ma non l'accrescimento contemporaneo di *tutte* le cellule dell'alga.
E così è nata l'idea dei **sistemi di riscrittura parallela** $(\mathbf{A}, \mu, \alpha)$ dove $A = \{a_1, a_2, \cdots, a_n\}$ è l'$alfabeto$, $\alpha = parola\ iniziale$, $\mu = morfismo$ su A:

- $\mu : A \to A^\star$ (quando c'è determinismo, **DOL-System**)
- $\mu : A \to \mathcal{P}(A^\star)$ (quando c'è non determinismo, **OL-System**)

$\mu = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ \downarrow & \downarrow & & \downarrow \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{pmatrix}$ con β_i stringa o insieme di stringhe rispett.

C'è un forte assunto: ogni cellula lavora a prescindere dal contesto, senza interazione.
Una **derivazione** è dunque l'applicazione iterata del morfismo a tutti i simboli della parola, si parte da α.

Definizione 87. *Dato un sistema $S = (A, \mu, \alpha)$, chiamiamo L(S) il linguaggio sull'alfabeto A, ottenuto partendo dalla parola α e applicando μ ripetutamente, fin quando possibile.*

Introducendo la distinzione tra terminali T e non, (A, μ, T, α), si hanno gli **EOL-Systems** (E="extended"), e una versione di questi sistemi sono gli **ETOL-Systems**, che ad ogni passo di derivazione hanno più morfismi tra cui scegliere in modo non deterministico (T="table"), se la scelta è deterministica si finisce negli **EDTOL-Systems**. Senza l'utilizzo dei terminali troviamo anche le classi **TOL-System** e **DTOL-System**.

Esempio 88. *Per generare il linguaggio trisomatico considero il sistema EOL* $(\{a, b, c, F\}, \mu, \{a, b, c\}, abc)$ *col morfismo* μ

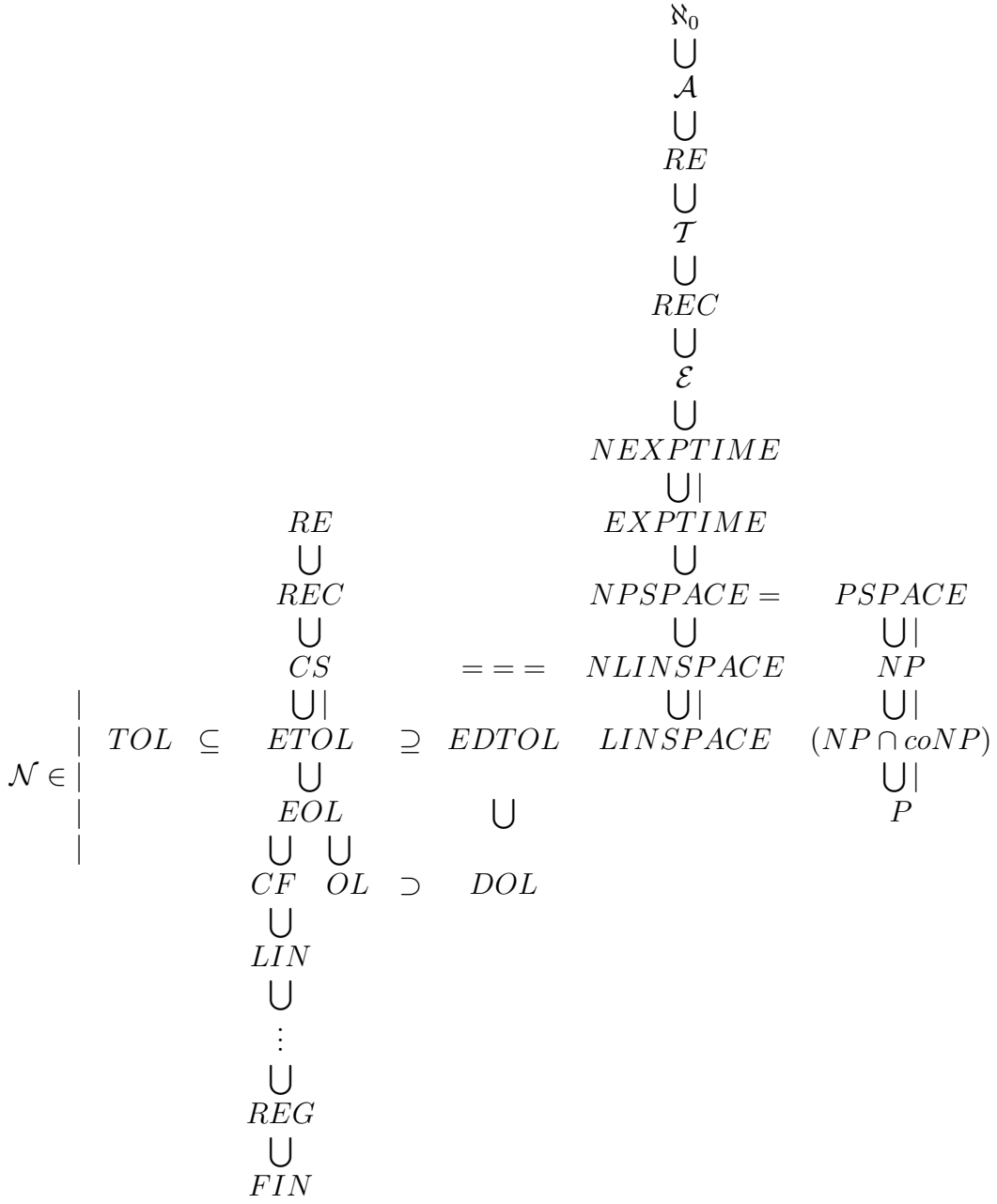
$$\begin{pmatrix} a & b & c & a & b & c & F \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \{aa, a\} & \{bb, b\} & \{cc, c\} & F & F & F & F \end{pmatrix}$$

È un unico blocco ma posso applicare le regole in modo arbitrario purché ogni lettera della parola subisca una sostituzione (ci sono anche regole che mandano in se stessi). Per terminalizzare, le regole di μ forzano un modo di procedere che evita di introdurre F (che invece evidenzia la mancanza di sincronizzazione).

Esempio 89. *Un sistema EDTOL che genera il linguaggio trisomatico.*

$$\mu_1 = \begin{pmatrix} A & B & C & a & b & c \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ aA & bB & cC & a & b & c \end{pmatrix} \quad \mu_2 = \begin{pmatrix} A & B & C & a & b & c \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ a & b & c & a & b & c \end{pmatrix}.$$

Le seguenti sono relazioni di inclusione tra alcune classi di linguaggi.



dove $\mathcal{A}$ = gerarchia aritmetica
$\mathcal{T}$ = gerarchia di Post
$\mathcal{E}$ = gerarchia elementare

La classe $\mathcal{N}$ include i linguaggi di programmazione, i linguaggi naturali, i linguaggi morfogenetici (linguaggi che descrivono accrescimenti di forme natuarali, come alghe, arbusti, etc.), e *SAT*.

Una classe di sistemi molto simili agli OL-Systems è stata definita nel '65 ed è quella delle Matrix Grammars, che introducono una *regolazione*, ovvero una strategia di applicazione delle regole. Una **matrix grammar** G è definita da

$$\mathbf{G} = (\mathbf{A}, \mathbf{T}, \mathbf{M}, \alpha)$$

con A alfabeto, T terminali, M matrici e α assioma iniziale.
Le matrici sono delle sequenze di regole:

$$\begin{array}{llll} M_1: & \alpha_1 \to \beta_1 & \alpha_2 \to \beta_2 & \alpha_3 \to \cdots \\ M_2: & \gamma_1 \to \delta_1 & \gamma_2 \to \delta_2 & \gamma_3 \to \cdots \\ M_3: & \varepsilon_1 \to \eta_1 & \varepsilon_2 \to \eta_2 & \varepsilon_3 \to \cdots \\ \cdots & \cdots & \cdots & \cdots \end{array}$$

Si sceglie arbitrariamente la M_j da applicare, e una volta scelta si devono applicare tutte le regole nell'ordine, se una regola in M_j non è applicabile il processo "abortisce" e si va a scegliere un'altra M_i.

Esempio 90. *Se prendiamo* $A = \{A, B, C, A', B', C', a, b, c, F\}$, $T = \{a, b, c\}$ *e* $\alpha = ABC$, *il seguente sistema di matrici*

$$\begin{array}{llll} M_1: & S \to ABC & & \\ M_2: & A \to aA & B \to bB & C \to cC \\ M_3: & A \to a & B \to b & C \to c \end{array}$$

genera il linguaggio trisomatico.

Sia **MAT** la classe dei linguaggi generati da matrici, abbiamo

Teorema 91. $MAT^{\lambda}_{AC} = RE$.

λ indica il fatto che sono ammesse regole di cancellazione, e AC = *appearing checking* significa che alcune produzioni possono avere un $*$ che indica la condizione "se la regola non è applicabile, invece di abortire si passi alla successiva", senza questo accorgimento non si genera tutto RE.

I seguenti sono altri formalismi grammaticali con meccanismi di regolazione.
Le **grammatiche programmate o condizionate**, che aggiungono condizioni all'applicazione delle regole (per esempio la presenza di un certo γ nella stringa), o le **scattered rules** applicano sostituzioni in varie parti della parola, e diventano **Indian** se la sostituzione parallela avviene per uno stesso simbolo. Un caso particolare di "scattered" sono le **Russian**, che possono richiedere la

sostituzione parallela di uno stesso simbolo in tutte le sue occorrenze o l'usuale sostituzione sequenziale di Chomsky.
Tutti questi metodi di riscrittura mirano a superare la sequenzializzazione del rimpiazzamento di Chomsky.

I **trasduttori** sono automi in cui la nozione di stato si combina con la nozione di rimpiazzamento di un simbolo con una stringa.

Definizione 92. *Un* GSM *è un automa che, scorrendo la parola, in corrispondenza ad ogni simbolo esegue due azioni: cambia stato e sostituisce il simbolo con una stringa, non può tornare indietro e alla fine ha sostituito una stringa con un'altra.*

Definizione 93. *I* trasduttori iterati IFT *estendono i GSM: si parte da una stringa assioma, si applicano regole GSM, quando si arriva ad uno stato finale la stringa fa parte del linguaggio. Alla fine di una trasduzione, indipendentemente dall'essere giunti in uno stato finale o meno, la parola ottenuta può essere data in ingresso all'automa per un'ulteriore trasduzione (che può produrre una stringa o no).*

Osservazione 94. *I meccanismi di trasduzione sono completi dal punto di vista computazionale e sono molto importanti in natura.*

Teorema 95 (Manca, Martin-Vide, Păun)**.** $RE = IFT_4$.

Teorema 96. $CF = IFT_2$.

Definizione 97. $f : L \to L$ *si dice* gsm *se è calcolata tramite GSM.*

Definizione 98. *Lo* Shuffle *di* $x, y \in A^\star$ *è*

$$x \sqcup\!\sqcup y = \{\alpha_1\beta_1\alpha_2\beta_2 \cdots \alpha_k\beta_k \;/\; \alpha_1\alpha_2 \cdots \alpha_k = x, \beta_1\beta_2 \cdots \beta_k = y\}.$$

Definizione 99. *Si chiama* Twin Shuffle (0,1) *il linguaggio*

$$\mathbf{TS_{01}} = \{\alpha \sqcup\!\sqcup \overline{\alpha} \;/\; \alpha \in \{0,1\}^\star\}$$

dove $\overline{\alpha} \in \{\overline{0}, \overline{1}\}^\star$ *è ottenuta da* α *barrando i simboli.*

Vediamo ora una interessante caratterizzazione di RE tramite gsm e Shuffle.

Teorema 100 (Engerlfriet-Rozemberg '80)**.** *Per ogni linguaggio L di RE esiste un gsm* g *tale che* $L = g(TS_{01})$*. In generale abbiamo*

$$RE = \bigcup_{g \in gsm} g(TS_{01}).$$

Si dimostra facilmente che si può stabilire una biunivocità calcolabile fra tutte le possibili sequenze di DNA e le parole di Twin Shuffle (0,1):

Proposizione 101. $DNA \leftrightarrow TS_{01}$.

Ovvero il DNA risulta computazionalmente completo, modulo funzioni gsm.

Capitolo 7

Sistemi Monoidali Derivativi e Deduttivi

Tutti i formalismi di riscrittura che abbiamo visto si basano su meccanismi di rimpiazzamento. I sistemi derivazionali su stringhe forniscono un quadro concettuale più ampio in cui rientrano sia i sistemi simbolici basati sul rimpiazzamento che altri sistemi basati su meccanismi combinatori diversi.
Tali sistemi hanno la seguente struttura

$$(\mathbf{A}, \mathbf{R}, \vdash, \perp)$$

dove A è un alfabeto, R regole combinatorie (relazioni che mi dicono come "mescolare" stringhe per ottenerne altre), $\vdash$ regole che stabiliscono come costruire una derivazione con regole combinatorie, $\perp$ un criterio per stabilire quando una derivazione termina producendo un risultato.
Questo schema generale comprende tre filoni principali.

1. **Sistemi di Post**.

   (a) **RS** (Replacing Systems).
   Comprendono le grammatiche di Chomsky, gli automi, le macchine di Turing e i trasduttori (vedi paragrafi precedenti).

   (b) **CG** (Contextual Grammars) introdotte da Marcus negli anni '60.
   (Vedi il bi-insert nel seguito della Definizione 102)

   (c) **OL**.
   Sistemi di riscrittura parallela introdotti da Lindenmayer nel '68 (vedi par. 6).

   (d) **RRS** (Regulated Replacing Systems).
   Sistemi di riscrittura parallela con regolazione, Matrix, Ordered, e Conditional Grammars (vedi il par. 6).

2. Sistemi a Ricombinazione (**H-Systems**).

   Nell'87 Tom Head modella l'operazione di ricombinazione del DNA con gli **HS** (H-Systems), introducendo lo "splicing" come nuovo meccanismo combinatorio (vedi il seguito della Definizione 102).

   Da queste nuove idee si sviluppa la MC (Molecular Computing) e tutti i modelli che hanno trovato ampio spazio nella modellizzazione dei sistemi biologici, Evolutionary Computing, Sticker Systems, Insert-Deletion, etc.

3. **GS** (Grammar Systems).
   Grammatiche che lavorano in cooperazione. La novità di questi sistemi consiste nella ricerca di un controllo della sincronizzazione, per esempio nel regolare l'accrescimento di parti lontane.[**?**]

4. **PS** (Sistemi a Membrana).
   Sistemi basati su membrane che contengono sia regole localizzate che regole di propagazione. Assumono grande importanza la temporalità, la località e la molteplicità (tempo, spazio, materia).

Definizione 102. *Siano A, V un alfabeto e un insieme di variabili.*
Uno schema combinatorio *è definito dalla seguente formula*

$$r(p_1, p_2, \cdots, p_k) : \frac{t_1 t_2 \cdots t_m}{s_1 s_2 \cdots s_n}$$

con $t_i, s_i \in (A \cup V)^\star$ e $p_i \in V$.

Le p_i sono alcune delle variabili presenti in $t_1 t_2 \cdots t_m$ e $s_1 s_2 \cdots s_n$ e si dicono *parametri*, le altre variabili $X_1, \cdots, X_h$ si dicono *proprie*.

Definizione 103. *Si dice* regola combinatoria *un'* istanza *di uno schema combinatorio $r(p_1, p_2, \cdots, p_k)$, che si ottiene istanziando i parametri dello schema con stringhe di $A^\star$ e si indica con $r(w_1, w_2, \cdots, w_k)$ se l'assegnamento alle variabili è $p_1 := w_1, \cdots, p_k := w_k$. Una regola combinatoria $r(w_1, w_2, \cdots, w_k) = \frac{u_1 \cdots u_m}{v_1 \cdots v_n}$ determina una relazione $\rightarrow_{r(w_1, w_2, \cdots, w_k)}$ di n+m argomenti tale che*

$$(\alpha_1, \cdots, \alpha_m, \beta_1, \cdots, \beta_n) \in \rightarrow_{r(w_1, w_2, \cdots, w_k)}$$

se esiste un assegnamento delle variabili proprie in corrispondenza del quale $u_1 \cdots u_m, v_1 \cdots v_n$ forniscono come valori $\alpha_1 \cdots \alpha_m, \beta_1 \cdots \beta_n$ rispettivamente.

Per esempio, se

$$r(x, y) = \frac{xZy \quad yZVx}{xxZyVx}$$

allora:

$$r(a, b) = \frac{aZb \quad bZVa}{aaZbVa}$$

e:

$$(acb, bcda, aacbda) \in \rightarrow_{r(a,b)}$$

Alla base delle grammatiche e degli automi vi è lo schema

- **replace**

$$r(u, w) : \frac{xuy}{xwy}$$

  dove u, w sono i parametri e x e y le variabili proprie.
  Sotto forma di regola grammaticale si scrive $u \rightarrow w$.

Nei due schemi che seguono abbiamo il meccanismo combinatorio delle grammatiche contestuali.

- **insert**

$$r(u, w, v) : \frac{xuvy}{xuwvy}$$

  dove u,w,v sono i parametri e x e y le variabili esterne.

- **bi-insert** (introdotto da Marcus)

$$r(u, w, v) : \frac{xwy}{xuwvy}$$

Infine lo

- **splicing (singolo)**

$$r(u, v, u', v') : \frac{xuvy \quad zu'v'w}{xuv'w}$$

  che taglia e ricombina, proprio come fanno gli enzimi con il DNA. Nello **splicing generale** a parità di premesse avremmo ottenuto le due stringhe $xuv'w$ e $zu'vy$.

  La principale differenza tra lo splicing e il rimpiazzamento è che il primo ammette due premesse e il secondo una sola.

Definizione 104. *Ogni regola con una premessa e un risultato si chiama* rewriting.

Il replace è un caso particolare di rewriting.

Nel generico sistema derivazionale su stringhe, sia R un sistema di regole combinatorie, $\vdash$ la chiusura transitiva di $\rightarrow_r$ e $\perp$ il criterio che considera parole facenti parte del linguaggio solo quelle costituite da simboli terminali. Questo è un **sistema di Post (extended)** che chiamiamo **unregulated**.
Negli anni '60 si cercò di introdurre meccanismi di regolazione per regole combinatorie (per esempio con le matrici).

Definizione 105. *Un sistema di Post è* ranged *quando ad ogni variabile X è associato un sottinsieme proprio dell'alfabeto $B_X \subset A$, e $X \in {B_X}^\star$.*

Lemma 106. *$Post = PostNonRanged$.*

Il seguente è un risultato solo teorico (non praticabile), ma garantisce che si possa fare tutto con le regole rewriting.

Lemma 107. *Per ogni sistema di Post esiste un sistema Rewriting equivalente.*

È evidente, grazie alla Turing-completezza delle grammatiche, che

Lemma 108. *Per ogni Rewriting esiste un Replacing equivalente.*

Teorema 109 (Post Normal Form)**.** *Ogni sistema di Post può essere equivalentemente formulato con regole della forma $\frac{\alpha X}{X\beta}$.*

Osservazione 110. *Questo tipo di classificazione dei sistemi derivativi li colloca in base alle loro caratteristiche intensionali (come sono fatti) piuttosto che estensionali (i linguaggi che generano).*

7.1 Sistemi Monoidali Deduttivi

Nel seguito si assumono le nozioni basilari di Logica Matematica.

Definizione 111. *Dato un modello $\mathcal{M}$=(Dominio, operazioni, relazioni), $A \subseteq D$ è* rappresentabile *se esiste ϕ tale che $a \in A \Leftrightarrow \mathcal{M} \models \phi(a)$.*

Esempio 112. *Nel modello aritmetico $AR = (\mathbb{N}, +, \cdot, 0, 1)$ consideriamo il sottinsieme di $\mathbb{N}$ dei numeri primi P. Esso si rappresenta con:*

$$n \in P \Leftrightarrow AR \models \neg n = 0 \wedge \neg n = 1 \wedge \forall x, y\ (n = x \cdot y \rightarrow x = n \vee y = n).$$

Definizione 113. *Una* teoria Φ *è un insieme di formule deduttivamente chiuso* ($\Phi \models \phi \Rightarrow \phi \in \Phi$).

In virtù della completezza della logica predicativa si ha che

$$\Phi \models \phi \Leftrightarrow \Phi \vdash \phi$$

ovvero ϕ è conseguenza logica della teoria Φ sse esiste un calcolo logico che partendo dagli assiomi di Φ porta a ϕ.

Definizione 114. *I* termini chiusi *T di una teoria Φ sono termini che non hanno variabili. Un insieme $A \subseteq T$ è* rappresentabile *in Φ tramite ϕ se $a \in A \Leftrightarrow \Phi \models \phi(a)$.*

Definizione 115. *Una* segnatura monoidale Σ su un alfabeto A *include un'operazione binaria - - , un simbolo λ (stringa vuota), e tutte le costanti $a, b, c, \cdots$, come simboli dell'alfabeto.*

Definizione 116. *Una* teoria monoidale*, su una segnatura monoidale Σ, è una Σ-teoria che include gli assiomi di monoide.*

Esempio 117. *Il linguaggio trisomatico è generato con una teoria monoidale sull'alfabeto $A = \{a, b, c\}$.*
Gli assiomi sono

$$\Phi \begin{cases} \phi(\lambda) \\ \phi(abc) \\ \phi(xaby) \rightarrow \phi(axabbyc) \end{cases}$$

Risulta $\alpha \in L_t \Leftrightarrow \Phi \vdash \phi(\alpha)$, e il lavoro generativo viene "dato in appalto" al calcolo logico.

Un altro modo per generare questo linguaggio con un calcolo logico, cioè come la chiusura deduttiva degli assiomi, si ha per esempio considerando

$$\begin{cases} R(\lambda, \lambda, \lambda) & \\ R(x, y, z) & \rightarrow \quad (ax, by, cz) \\ R(x, y, z) & \rightarrow \quad L(xyz) \end{cases}$$

Tutte le formule di tipo $L(\alpha)$ che si ottengono con queste regole hanno la forma $a^n b^n c^n$.

Vediamo ora una teoria monoidale, in versione equazionale, che genera il linguaggio trisomatico

$$\begin{cases} S(x) = A(x)B(x)C(x) \\ A(x+1) = aA(x) \\ B(x+1) = bB(x) \\ C(x+1) = cC(x) \\ A(0) = B(0) = C(0) = \lambda \end{cases}$$

dove A,B,C,0 sono simboli funzionali, e +1 è un simbolo funzionale unario associativo.
Se $S(n)$ sta per $S(0 + 1 + 1 + \cdots + 1)$ con n volte 1 si ha che:

$$S(n) = \alpha \in T^\star \Leftrightarrow \alpha = a^n b^n c^n.$$

Definizione 118. *Un* sistema monoidale *deduttivo$M = (A, \Sigma, \Phi)$ è caratterizzato da un alfabeto A, una segnatura monoidale Σ, e una teoria monoidale Φ su Σ. Un simbolo relazionale k-ario R di Φ, rappresenta in M una relazione R_M k-aria su $A^\star$:*

$$R_M(\alpha_1, \cdots, \alpha_k) \Leftrightarrow \Phi \models R(\alpha_1, \cdots, \alpha_k).$$

La traduzione monoidale di un sistema simbolico ha due gruppi di assiomi:

1. assiomi relativi al tipo di sistema
2. assiomi specifici del sistema

I sistemi monoidali sono Universali [**?**].
Infatti, se ML è la classe dei linguaggi che sono rappresentabili tramite sistemi monoidali (con predicati unari, ovvero nelle condizioni della Definizione 118 si ha $R_M(\alpha) \Leftrightarrow \Phi \models R(\alpha)$) risulta che ML=RE.

Collegando il Teorema Centrale con le teorie monoidali si riesce a dare una rappresentazione logica della Macchina di Turing. Infatti, ora daremo una teoria logica che risulterà equivalente alla MdT, e da questo segue l'universalità dei sitemi monoidali.

Notazione. Maiuscole = predicati, minuscole = variabili.
Si considerino i seguenti assiomi.

$$\begin{cases} Q_0(q) \to T(q) & (\textit{lo stato iniziale sul nastro}) \\ S_0(x) \to S(x) & (B \textit{ è un simbolo}) \\ C(x) \to S(x) & (\neg\, B \textit{ è un simbolo}) \\ Q_0(q) \wedge T(qx) \wedge S(y) \to T(qxy) & (\textit{carica il nastro}) \\ R(qx, q'y) \wedge T(wqxz) \to T(wyq'z) & (\textit{movimento a destra}) \\ L(qx, q'y) \wedge T(wvqxz) \wedge S(v) \to T(wq'vyz) & (\textit{movimento a sinistra}) \\ S_0(x) \wedge Q(q) \wedge T(qz) \to T(xqz) & (\textit{allungam. nastro}) \\ S_0(x) \wedge Q(q) \wedge T(zq) \to T(zqx) & (\textit{allungam. nastro}) \\ T(xqy) \wedge F(q) \to D(xy) & (\textit{fine della computazione}) \\ S_0(x) \wedge D(xy) \to D(y) & (\textit{pulisce a sinistra}) \\ S_0(y) \wedge D(xy) \to D(x) & (\textit{pulisce a destra}) \\ D(\lambda) \to O(\lambda) & (\textit{se legge } \lambda \textit{ l'output è nullo}) \\ D(x) \wedge C(x) \to O(x) & (\textit{se legge solo } x \textit{ l'output è } x) \\ D(xyz) \wedge C(x) \wedge C(z) \to O(xyz) & (\textit{produce l'output}) \end{cases}$$

Questi assiomi (teoria TT), che danno una descrizione logica della MdT, sono sufficienti a simularne le computazioni.

Intuitivamente, $T(xqy)$ esprime che sul nastro di input della macchina si trova la parola xy, la macchina è nello stato q, legge il primo simbolo della parola y, che occupa la parte destra del nastro, mentre alla sinistra della casella di lettura si trova la parola x.
La formula $S(x)$ indica che x è un simbolo, $S_0(x)$ che x è il simbolo speciale B e $C(x)$ che x è un simbolo diverso da B, $Q(q)$ che q è uno stato e $Q_0(q)$ che q è lo stato iniziale.

La formula $R(qx, q'y)$ esprime l'istruzione (q, x, q', y, dx), mentre $L(qx, q'y)$ esprime (q, x, q', y, sn).
La formula $F(q)$ indica che q è uno stato finale; il predicato D serve a cancellare, nel nastro finale, i simboli bianchi estremi; e la formula $O(x)$ dice che x è l'uscita ottenuta quando la macchina si ferma.

Abbiamo visto gli assiomi generali, quelli della famiglia delle macchine di Turing, per completare la descrizione monoidale del sistema mancano gli assiomi specifici. Consideriamo dunque una particolare macchina di Turing $M = (A, B, Q, q_0, P_M, Q_f)$ e a TT aggiungiamo gli assiomi specifici di M.

$$A(M) \begin{cases} S_0(B) & \\ Q_0(q_0) & \\ C(a) & \forall a \in A \; a \neq B \\ Q(q) & \forall q \in Q \\ F(q) & \forall q \in Q_f \\ R(qa, q'b) & \forall (q, a, q', b, dx) \in P_M \\ L(qa, a'b) & \forall (q, a, q', b, dx) \in P_M \end{cases}$$

□

Il seguente teorema, che collega il concetto di computazione a quello di calcolo logico, è conseguenza immediata della costruzione della teoria TT appena vista e fornisce una formulazione ulteriore dell'universalità monoidale.

Teorema 119. *Data $M \in MdT$, si ha $A(M) \cup TT \vdash O(\alpha) \Leftrightarrow \alpha \in L(M)$.*

Capitolo 8

NP Completezza e Soddisfacibilità

I linguaggi hanno un semi-ordine dato dalla **riducibilità polinomiale**:

Definizione 120. *Diciamo che $L_1 \leq L_2$ sse esiste f, esprimibile mediante una macchina di Turing che lavora in tempo polinomiale rispetto alla lunghezza della stringa $|\alpha|$, tale che $\alpha \in L_1 \Rightarrow f(\alpha) \in L_2$.*

$L_1 \leq L_2$ vuol dire che è possibile tradurre il problema L_1 in L_2 con un costo polinomiale. Si tratta di un quasi ordinamento (è una relazione riflessiva e transitiva ma non antisimmetrica) ed è interessante trovare linguaggi/problemi massimali rispetto a questo q.o.

Definizione 121. *Chiamiamo* **P** *la classe dei linguaggi tali che l'appartenenza di una stringa sia risolubile con MdT* **deterministiche** *in tempo polinomiale.*

Definizione 122. *Chiamiamo* **NP** *la classe dei linguaggi tali che l'appartenenza di una stringa sia risolubile con MdT* **non deterministiche** *in tempo polinomiale.*

NP può essere caratterizzato (vedi dimostrazione della proposizione 56) come la classe dei linguaggi L per cui esiste una funzione Turing calcolabile in tempo polinomiale $f : A^\star \times A^\star \to \{0,1\}$ tale che $\alpha \in L \Leftrightarrow \exists\gamma : f(\alpha,\gamma) = 1$. In tal modo, per stabilire se $\alpha \in L$ occorre generare non deterministicamente una $\gamma \in A^\star$, detto *certificato*, e verificare in tempo polinomiale se $f(\alpha,\gamma) = 1$. Tali linguaggi si dicono **verificabili in tempo polinomiale**.

Osservazione 123. $P \subseteq NP$.
La validità dell'uguaglianza $P \stackrel{?}{=} NP$ è un problema aperto.

Definizione 124. *Un linguaggio massimale in NP, rispetto alla riducibilità polinomiale "$\leq$" (vedi Definizione 120), si dice* NP-Completo.

In NP troviamo un gran numero di problemi combinatori di importanza fondamentale.

Esempio 125. Il problema del sequenziamento genomico *consiste nell'andare a vedere qual è la sequenza del genoma umano, che è una doppia stringa (con le coppie di basi complementari) lunga* $n \approx 3.2 \cdot 10^9$ *sull'alfabeto* $B = \{A, T, C, G\}$.
Il problema viene così formulato

$$\begin{cases} |X| = n \\ L \subseteq sub(X) \end{cases}$$

dove la stringa X è l'incognita e L, che rappresenta i nostri dati (più grande è L e più dati ho), è un linguaggio noto (i suoi elementi si ottengono da tante copie di X frammentate variamente in pezzetti di lunghezza simile) che si ottiene in laboratorio con tecniche che garantiscono la struttura "random" delle sue parole. sub(X) è l'insieme formato da tutte le possibili sottostringhe di X.
La formulazione equazionale del problema è la seguente.

$$\begin{cases} \{X\} = B^n \cap \{X\} \\ sub(X) \cap L = L \end{cases}$$

Avendo L e la lunghezza della stringa devo poter risalire ad X.

Definizione 126. *Si chiama* **co$\mathcal{L}$** *la classe dei linguaggi L tali che* $\overline{L} \in \mathcal{L}$.

Si è constatato che $NP \cap coNP$ contiene problemi "difficili".

Una formula proposizionale è un'espressione che collega variabili booleane ($x_1, x_2 \cdots x_n \in \{0, 1\}$) tramite le operazioni binarie di disgiunzione ($x_1 \vee x_2 = 0$ sse $x_1, x_2 = 0$) e congiunzione ($x_1 \wedge x_2 = 1$ sse $x_1, x_2 = 1$), una tale formula si esprime facilmente come sistema di equazioni booleane e si chiama *istanza per SAT*.
Data un'istanza ϕ per SAT, se ϕ è vera per qualche assegnamento di valori di verità delle sue variabili allora si dice *soddisfacibile*, ovvero è una formula di SAT.

Definizione 127. *SAT è il linguaggio delle formule proposizionali soddisfacibili.*

Quindi $\alpha \in SAT$ sse α è vera per qualche valore delle sue variabili proposizionali.
Si dicono *letterali* le variabili proposizionali x del sistema o le loro negate ($\neg x$, aventi valore di verità opposto a quello della variabile corrispondente x). Consideriamo dunque ogni formula di SAT come congiunzione di *clausole* (che sono disgiunzioni di letterali).

SAT è NP. Infatti si dimostra facilmente che SAT è verificabile in tempo polinomiale (vedi seguito della Definizione 122), anzi lineare: generato come cerificato un assegnamento delle variabili in modo non deterministico, si verifica se esso soddisfa la formula in tempo lineare.

Definizione 128. 3-SAT *sono le formule proposizionali soddisfacibili aventi non più di 3 letterali per ogni clausola. Se tutte le clausole hanno esattamente tre letterali si parla di* 3-SAT esatto*.*

Proposizione 129. *3-SAT* $\cong$ *3-SAT esatto.*

Dimostrazione. Ogni clausola di *3-SAT* che dovesse avere meno di tre letterali si riconduce equivalentemente alla congiunzione di due o quattro clausole con tre letterali:

$$C = L_1 L_2 \text{ è } soddisfacibile \Leftrightarrow \text{ è } soddisfacibile \begin{cases} C_1 = L_1 L_2 v \\ C_2 = L_1 L_2 \neg v \end{cases}$$

$$C = L_1 \text{ è } soddisfacibile \Leftrightarrow \text{ è } soddisfacibile \begin{cases} C_1 = L_1 v u \\ C_2 = L_1 \neg v u \\ C_3 = L_1 v \neg u \\ C_4 = L_1 \neg v \neg u \end{cases}$$

dove L_1 e L_2 sono letterali e u e v sono delle variabili aggiunte. □

Lemma 130. *SAT* $\cong$ *3-SAT.*

Dimostrazione. È ovvio che ogni formula di 3-SAT è una particolare formula di SAT, passiamo quindi a dimostrare l'equivalenza nell'altro verso.
Sia $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ una formula di SAT (quindi C_i è soddisfacibile $\forall i = 1, 2, \cdots, n$), ognuna di queste clausole è del tipo

$$C = L_1 \cdots L_j L_{j+1} \cdots L_k$$

per qualche k, dove ogni L_i è un letterale.
Consideriamo una nuova variabile v, e spezziamo la clausola C nelle due clausole

$$\begin{cases} C_1 = L_1 \cdots L_j v \\ C_2 = L_{j+1} \cdots L_k \neg v \end{cases}$$

abbiamo che

$$C \text{ è } soddisfacibile \Leftrightarrow C_1 \text{ e } C_2 \text{ sono } soddisfacibili$$

quindi l'istanza di partenza rimane equivalente sostituendo C_1 e C_2 a C. Applicando a tutte le clausole questo procedimento iterato, ad ogni passo aumenta di 1 il numero delle variabili e delle clausole, ma diminuisce il numero di letterali presenti in ogni clausola del sistema.
Una generica clausola con tre letterali, non viene ulteriormente ridotta dal nostro procedimento: $C = L_1L_2L_3$ diventa per esempio

$$\begin{cases} C_1 = L_1L_2v \\ C_2 = L_3\neg v \end{cases}$$

ma C_1 continua a contenere tre letterali. □

Teorema 131. *3-SAT* $\in$ *ETOL.*

Dimostrazione. Un ETOL non deterministico che genera tutte e sole le formule di *3-SAT* ha 3 sostituzioni (definite come riscritture non deterministiche), μ_1, μ_2 e μ_3.

$$\mu_1 = \begin{pmatrix} S & S \\ \downarrow & \downarrow \\ (x \vee y \vee z)S & (x \vee y \vee z) \end{pmatrix}$$

dove $x, y, z \in \{T, \neg T, F, \neg F\}$ in modo che T oppure $\neg F$ occorrano almeno una volta. Con μ_1 si genera un numero arbitrario di clausole vere (aventi per costruzione non più di tre letterali).

$$\mu_2 = \begin{pmatrix} S & T & T & F & v \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ R & 1T & 1 & 1F & v \end{pmatrix} \quad \mu_3 = \begin{pmatrix} S & T & F & F & v \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ R & 1T & 1 & 1F & v \end{pmatrix}$$

dove $1, R$ sono i simboli terminali e $v \in \{(\ ,\ )\}$. Una variabile proposizionale si identifica con una sequenza di 1.
Applicando μ_2 o μ_3 si sostituisce in modo sincronizzato 1 a T o F, e l'applicazione iterata del morfismo incrementa di 1 la stringa.
Si ottiene dunque la congiunzione di tante clausole (vere per costruzione) rappresentate da parentesi (quando si usa $T \to 1$ o $F \to 1$ alla variabile relativa viene associato valore di verità vero o falso rispettivamente).
Abbiamo costruito con un sistema ETOL tutte le formule proposizionali soddisfacibili (con un arbitrario numero di clausole e variabili), quindi tale sistema genera *3-SAT*. □

Lemma 132. *L'insieme delle tautologie LT* $\notin$ *CF.*

Dimostrazione. Su un alfabeto V si consideri il linguaggio regolare

$$R = \{x \vee y \to z \ / \ x, y, z \in V^\star\}$$

e il linguaggio delle tautologie LT (quello delle formule proposizionali vere per qualsiasi assegnamento delle variabili).
Per il cor.80 risulta $R \cap LT = \{x \vee x \rightarrow x \ / \ x \in V^\star\} \notin CF$. Infatti, $f : x \vee x \rightarrow x \mapsto a^n b^n c^n$ è una gsm che associa tale linguaggio a $L_t \notin CF$ (se $R \cap LT$ appartenesse a CF, che è chiusa per gsm, anche L_t apparterrebbe a CF, e questo è assurdo). □

Teorema 133. *$SAT \notin CF$.*

Osservazione 134. *SAT è considerato rispetto ad una certa codifica ma si passa da una codifica all'altra in tempo polinomiale, a volte addirittura lineare.*

Infine vediamo un risultato degli anni '80, che giustifica la grande attenzione che ruota attorno al linguaggio SAT.

Teorema 135. *SAT è NP-Completo.*

Cenno di Dimostrazione. Si riesce ad esprimere qualsiasi istanza di un problema NP tramite la soddisfacibilità di una formula proposizionale (Definizione 124). Ovvero, preso $L \in NP$ esiste un polinomio f tale che $\alpha \in L \Leftrightarrow \phi_{\alpha,L} \in SAT$ con $|\phi_{\alpha,L}| = f(|\alpha|)$.

Bibliografia

1. Calude C., Păun Gh., Computing with cells and atoms, Taylor & Francis, London, 2001.

2. J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata, Languages, and Computation, Addison Wesley, 2001 (Trad. It. Pearson Education Italia, 2003).

3. Manca V., Logica Matematica, Bollati Boringhieri, Torino, 2001.

4. Păun Gh., Rozenberg G., Salomaa A. (Eds.), DNA Computing: New Computing Paradigms, Springer-Verlag, Berlin - Heidelberg, 1998.

5. Rozenberg G., Salomaa A. (Eds.), Handkook of Formal Languages, 3 Vols. Springer-Verlag, Berlin - Heidelberg, 1997.

6. Salomaa A., Formal Languages, Academic Press, New York, 1973.

7. Salomaa A., Jewels of Formal Language Theory, Computer Science Press, Rockville, 1981.