Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Magistrale in Matematica

ALGORITMI E RICORSIVITÀ: MACCHINE DI TURING E ALGORITMI DI MARKOV

Tesi di Laurea in Principi della Matematica

Relatore:
Chiar.mo Prof.
Piero Plazzi

Presentata da:
Federica Frisoni

I Sessione
Anno Accademico 2013-2014

A*lla mia famiglia*
e a Simone

Introduzione

Intorno alla metà degli anni Trenta del secolo scorso, studiosi di varie parti del mondo si occuparono di definire in maniera rigorosa i concetti di algoritmo e computabilità. Le ragioni che spinsero a questi studi sono dovute ad alcune importanti questioni sollevate in quegli anni nella ricerca sui fondamenti della matematica. Un matematico che diede un notevole contributo in tal senso fu David Hilbert, che formulò 23 problemi, tra cui quello che in tedesco si chiama *Entscheidunsgproblem*, traducibile come Problema della decidibilità: nell'ambito della logica del primo ordine ci si chiede se esiste un procedimento atto a riconoscere gli enunciati (dimostrabili) veri distinguendoli da quelli che non lo sono. Altro quesito, sempre di Hilbert, è quello noto come Decimo problema, il quale poneva il seguente interrogativo: esiste un procedimento effettivo per decidere se un qualunque polinomio a coefficienti interi in un numero finito di variabili ha soluzioni intere? La ricerca di un eventuale procedimento da esibire per poter rispondere in modo affermativo o la dimostrazione che un tale procedimento non esiste per rispondere in maniera negativa, esigevano una definizione rigorosa di procedimento effettivo e cioè di algoritmo, mediante cui è poi possibile stabilire anche il significato di funzione computabile. Quest'ultimo concetto è stato precisato in varie maniere e degno di nota è il fatto che tutte descrivono la medesima classe di funzioni. Ciò significa che ci troviamo di fronte a una famiglia "oggettiva" di funzioni. Una definizione soddisfacente è, però, stata fornita solo per funzioni aritmetiche $f : \mathbb{N}^k \to \mathbb{N}$ e non per funzioni di variabile reale.
L'obiettivo di questa tesi è presentare alcune delle più importanti definizioni

di tale classe di funzioni: dopo un primo approfondimento sul concetto intuitivo di algoritmo, ne viene presentata una prima descrizione "aritmetica" tramite le funzioni ricorsive (capitolo 1). Un secondo approccio, descritto nel capitolo 2, è quello dato in termini di macchine di Turing: esso è storicamente il più importante, in quanto, pur trattandosi di un modello ideale di calcolo, il suo funzionamento, in particolare per quanto riguarda quella che Turing definì macchina universale, prefigura quello dei moderni calcolatori elettronici. In questo stesso capitolo verrà esposta una dimostrazione dell' equivalenza tra i concetti di ricorsività e computabilità tramite le macchine Turing. Nel terzo capitolo vengono, invece, introdotti gli algoritmi Markov, basati su sistemi di riscrittura di stringhe, mediante opportune regole di sostituzione. Anche in questo caso viene fornita una dimostrazione di equivalenza con le funzioni ricorsive. Completa la tesi un breve cenno ad un'ulteriore idea equivalente di algoritmo, basata sul concetto di funzione, il λ-K-calcolo, introdotto da A. Church.

Indice

Capitolo 1

Il concetto intuitivo di algoritmo e la ricorsività

1.1 Il concetto di algoritmo

Il concetto di algoritmo, ossia di procedura generale, è fondamentale in molti campi della matematica. Di esso daremo inizialmente una definizione intuitiva e successivamente analizzeremo alcune fra le descrizioni rigorose che ne sono state date e che fanno parte di quella branca della matematica denominata *teoria della computabilità*.
Il termine "algoritmo" deriva dalla latinizzazione del nome del matematico persiano Muhammad ibn Musa al-Khwarizmi vissuto nel IX secolo, considerato uno dei primi autori ad aver fatto riferimento a questo concetto. Al-Khwarizmi è famoso per aver introdotto i cosiddetti numeri arabi: la sua opera "(Libro) di al-Khwarizmi sui numeri indiani" (825 d.C. circa) fu tradotta in latino come "Algorismi de numero Indorum" ed esercitò una profonda influenza nei secoli successivi: fu con questo libro, infatti, che in Europa si iniziò ad usare il sistema di notazione decimale posizionale. Le procedure che permettevano di effettuare calcoli in notazione decimale divennero, così, note come"Algorismi" o "Algoritmi" ed in séguito, lo stesso termine fu applicato in generale alle procedure di calcolo necessarie per ottenere un determinato

risultato.

Così, oggi, con la parola algoritmo, designamo una procedura generale, ossia una successione di istruzioni che indicano un procedimento la cui esecuzione è specificata nel dettaglio, si esplica in un numero finito di passi e mediante cui è possibile ottenere la risposta per ogni problema pertinente. In svariati campi della matematica è facile trovare esempi di algoritmi, basti pensare alla procedura per eseguire la divisione euclidea, oppure quella per calcolare lo sviluppo decimale approssimato della radice quadrata di un dato numero naturale, o ancora, quella per il calcolo del massimo comun divisore tra due numeri. Con l' avvento dell' informatica e lo sviluppo dei calcolatori elettronici, poi, il termine algoritmo ha conosciuto un'ulteriore e feconda diffusione, in quanto è stato possibile affidare l' esecuzione di tali procedure anche alle macchine, programmate in opportuni linguaggi, riuscendo a ridurre notevolmente i tempi di computazione. In questo ambito un algoritmo si configura come un metodo di risoluzione di un problema, in cui si presuppone l' esistenza di:

- un committente, che pone il problema di cui si desidera conoscere la soluzione;
- un esecutore;
- un insieme di dati che rappresentano la descrizione della particolare istanza del problema da risolvere;
- una sequenza di istruzioni, determinate senza ambiguità, che l' esecutore deve eseguire;
- un insieme di dati che rappresentano il risultato.

È importante sottolineare che l' esecuzione deve avvenire in un numero finito (non importa quanto elevato) di passi, in funzione della particolare istanza del problema. Nella teoria matematica degli algoritmi che presenteremo, i dati devono appartenere ad insiemi finiti di simboli, la cui riunione costituisce l'

alfabeto associato all' algoritmo in questione. Su un piano teorico, è possibile restringersi a considerare un alfabeto composto da un solo segno, per esempio |, cosicché le parole di questo alfabeto sono del tipo: $|, ||, |||, \ldots$ e si possono far corrispondere ai numeri naturali 0,1,2, ...
Al concetto di algoritmo sono, inoltre, strettamente connessi i concetti di computabilità, decidibilità, enumerabilità, che esporremo brevemente.

Computabilità. Chiamiamo funzioni *effettivamente computabili* quelle funzioni il cui valore è calcolabile tramite un procedimento sistematico e ben determinato, in un numero finito di passi, per qualunque dato argomento. Vale a dire quelle funzioni per cui esiste un algoritmo che ne fornisce il valore per ogni dato argomento. Ci concentreremo prevalentemente su funzioni i cui argomenti e i cui valori sono numeri naturali e dunque, parole nell' alfabeto $\{0, 1, 2, \ldots\}$. Più in generale diremo che il dominio delle funzioni computabili è costituito da tutte le parole P su un dato alfabeto finito A. Alcuni esempi abbastanza elementari di funzioni computabili sono la somma, il prodotto, l' elevamento a potenza.

Enumerabilità. Sia f una funzione computabile avente per dominio i numeri naturali. Allora l' immagine di f, che indichiamo con $M = f(\mathbb{N})$, è determinata dalla sequenza $f(0), f(1), f(2), \ldots$ in cui vengono elencati, eventualmente con ripetizioni, gli elementi di M. In tal caso si dice che f enumera gli elementi di M nella sequenza $f(0), f(1), f(2), \ldots$ e inoltre, un insieme Ω è *enumerabile* se è vuoto o se esiste una funzione computabile la cui immagine coincide con Ω. Analogamente, si definiscono anche le relazioni (o predicati) enumerabili: sia R una relazione binaria tra le parole su un alfabeto A. R è enumerabile se è la relazione vuota o se esistono due funzioni unarie computabili f e g tali che la sequenza di coppie ordinate $< f(0), g(0) >, < f(1), g(1) >, < f(2), g(2) >, \ldots$ elenca, eventualmente con ripetizioni, l' insieme di tutte le coppie ordinate che stanno nella relazione R.

Decidibilità relativa e assoluta. Siano M_1, M_2 insiemi di parole su un alfabeto A, dove $M_1 \subseteq M_2$. Diremo che M_1 è *decidibile relativamente*

ad M_2 se esiste un algoritmo finito grazie a cui è possibile determinare per ogni parola P di M_2, se $P \in M_1$ o $P \notin M_1$. Un tale algoritmo viene detto procedura di decisione.
Un insieme M_1 di parole su un alfabeto A è detto *decidibile* se M_1 è decidibile relativamente all' insieme di tutte le parole su A. Analogamente si definiscono le relazioni decidibili. Sia M un insieme di parole su un alfabeto A. La sua funzione caratteristica sia definita per ogni $P \in$A come segue:

$$\chi_M(P) = \begin{cases} 0 & \text{se} \quad P \in M \\ 1 & \text{se} \quad P \notin M \end{cases}$$

M è decidibile se e solo se la sua funzione caratteristica è computabile.

Riassumiamo i diversi tipi di algoritmo e le loro relazioni con le nozioni appena esposte:

algoritmi di calcolo per funzioni: è questo il caso della computabilità di funzioni

algoritmi di decisione per relazioni: corrispondono alla computabilità di funzioni caratteristiche

algoritmi di generazione di valori: corrispondono alla enumerazione dei valori di una funzione.

Iniziamo fornendo una prima definizione rigorosa, in termini "aritmetici", dell' idea inuitiva di funzione effettivamente computabile: quella di funzione *ricorsiva*. Considereremo nella nostra trattazione sulle funzioni ricorsive funzioni della forma $f : \mathbb{N}^k \to \mathbb{N}$. Esse sono definite per ogni k-upla di numeri naturali, sono a valori in $\mathbb{N}$ e vengono dette funzioni aritmetiche. Dapprima daremo la definizione di funzioni iniziali, a partire dalle quali applicando dei dati schemi si ottengono le funzioni ricorsive primitive, e infine, da queste arriveremo a definire le funzioni ricorsive. Mostreremo infine qualche esempio. Ci serviremo delle lettere i, j, k, m, n, r per indicare gli indici, delle lettere x, y, z per indicare delle variabili in $\mathbb{N}$, della scrittura ${}^n x$ per indicare delle n-uple $(x_1, \ldots, x_n)$.

1.2 Definizioni

Definizione 1.1. Chiamiamo *funzioni iniziali*:

- La funzione successore $S : \mathbb{N} \to \mathbb{N}$, $S(x) = x' = x + 1$
- La funzione proiezione sulla j-esima componente $U_j^k : \mathbb{N}^k \to \mathbb{N}$, $U_j^k(x_1, \ldots, x_k) = x_j$
- La funzione costante zero $Z : \mathbb{N} \to \mathbb{N}$, $Z(x) = 0$.

Definiamo due schemi mediante cui, a partire dalle funzioni iniziali, è possibile ottenere nuove funzioni.

Definizione 1.2. Sostituzione. Siano $n, r \geq 1$, $h_1, \ldots, h_r$ funzioni $n-$arie, g funzione $r-$aria. Si dice che f è definita da g per sostituzione di $h_1, \ldots, h_r$ se

$$f({}^n x) = g(h_1({}^n x), \ldots, h_r({}^n x)).$$

Ricorsione semplice. Sia $m \geq 0$, g una funzione m-aria, h una funzione $(m+2)$-aria. Si dice che f è definita per ricorsione da g e da h se

$$\begin{cases} f({}^m x, 0) = g({}^m x) \\ f({}^m x, y') = h({}^m x, y, f({}^m x, y)). \end{cases}$$

g ed h vengono dette rispettivamente *base* e *passo* della ricorsione; invece y ed ${}^m x$ vengono dette rispettivamente *variabile di ricorsione* e *parametri*. Per $m = 0$ i parametri sono assenti e la funzione g è una costante.

Definizione 1.3. Una funzione f è *ricorsiva primitiva* (abbr. r.p.) se è una funzione iniziale, oppure se è ottenibile a partire dalle funzioni iniziali, applicando un numero finito di volte gli schemi di sostituzione e ricorsione semplice.

Le funzioni iniziali sono inuitivamente computabili e applicandovi i due suddetti schemi, si ottengono ancora funzioni computabili, quindi abbiamo

che tutte le funzioni ricorsive primitive lo sono.
Esistono anche altri schemi costruttivi, derivati da quelli appena presentati, mediante cui è possibile ottenere funzioni r.p. da altre funzioni r.p.:

- **Identificazione di variabili:** sia $1 \leq h, k \leq n$, per ogni $x_1, \ldots, x_n$ valga
$$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_{k-1}, x_h, x_{k+1}, \ldots, x_n)$$
Se g è r.p. ed f si ottiene da g identificando le variabili di posto h e $k > h$, allora f è r.p.

- **Permutazione di variabili:** sia π una permutazione di $\{1, \ldots, n\}$ e per ogni $x_1, \ldots, x_n$ valga $f(x_1, \ldots, x_n) = g(x_{\pi(1)}, \ldots, x_{\pi(n)})$.
Se g è r.p. ed f si ottiene da g tramite una permutazione delle variabili, allora anche f lo è.

- **Sostituzione parziale di variabili:** si può sostituire una singola variabile x_k con $h({}^n x)$ in g ottenendo:
$$f({}^n x) = g(x_1, \ldots, \underbrace{h({}^n x)}_{k}, \ldots, x_n).$$
Se g è r.p., lo è anche f.

- **Inserimento di variabili fittizie:** sia g funzione n-aria, in essa si può inserire una variabile fittizia x_{n+1}:
$$f(x_1, \ldots, x_{n+1}) = g(x_1, \ldots, x_n).$$
Allora se g è r.p., lo è anche f. Iterando questo schema si può inserire in g un qualunque numero di variabili fittizie; così, aggiungendo variabili fittizie in $Z(x)$, si ha che è r.p. la costante $Z({}^n x) = 0$ per ogni n.

Casi particolari dello schema di ricorsione sono invece:

- **Ricorsione parzialmente senza parametri:**
$$\begin{cases} f({}^m x, 0) = g({}^m x) \\ f({}^m x, y') = h(y, f({}^m x, y)). \end{cases}$$

Se g ed h sono r.p., lo è anche f. Facendo comparire in h parametri fittizi $x_1, \ldots, x_m$, il presente schema discende come caso particolare dello schema di ricorsione semplice.

- **Ricorsione uniforme:**

$$\begin{cases} f({}^m x, 0) = g({}^m x) \\ f({}^m x, y') = h({}^m x, f({}^m x, y)). \end{cases}$$

Se g ed h sono r.p., lo è anche f. Facendo comparire in h come fittizia, la variabile di ricorsione y, il presente schema discende come caso particolare dello schema di ricorsione semplice.

Tutti questi casi si possono combinare tra loro ottenendo altre regole derivate, grazie a cui, a partire da funzioni r.p. si ottengono comunque funzioni r.p..

1.3 Esempi di funzioni ricorsive primitive

Tutte le funzioni di uso comune in aritmetica sono ricorsive primitive; vediamone ora alcuni esempi.

- Somma $x + y = \begin{cases} x + 0 = x \\ x + y' = (x + y)' \end{cases}$ cioè $\begin{cases} f(x, 0) = U_1^1(x) \\ f(x, y') = S(f(x, y)) \end{cases}$

- Prodotto $x \cdot y = \begin{cases} x \cdot 0 = 0 \\ x \cdot y' = xy + x \end{cases}$ cioè se g è la funzione addizione,

$$\begin{cases} f(x, 0) = Z(x) \\ f(x, y') = g(f(x, y), x) \end{cases}$$

- Elevamento a potenza $x^y = esp(x, y) = \begin{cases} esp(x, 0) = 1 \\ esp(x, y') = esp(x, y) \cdot x \end{cases}$

  Si pone per convenzione $0^0 = 1$.

- Fattoriale $x! = \begin{cases} 0! = 1 \\ y'! = y! \cdot y' \end{cases}$

- Predecessore $Pred(x) = \begin{cases} Pred(0) = 0 \\ Pred(y') = y \end{cases}$

- Differenza modificata $x \dot{-} y = \begin{cases} x \dot{-} 0 = x \\ x \dot{-} y' = Pred(x \dot{-} y) \end{cases}$

- Differenza assoluta(distanza) d: $d(x,y) = |x - y| = (x \dot{-} y) + (y \dot{-} x)$

- Segno $sgn(x) = \begin{cases} sgn(0) = 0 \\ sgn(y') = 1. \end{cases}$
  È, questo, un primo esempio di funzione caratteristica: una funzione che assume solo i valori 0 e 1. sgn è la funzione caratteristica del singoletto $\{0\}$.

- Controsegno $\overline{sgn(x)} = \begin{cases} \overline{sgn}(0) = 1 \\ \overline{sgn}(y') = 0. \end{cases}$
  $\overline{sgn}$ è la funzione caratteristica di $\mathbb{N}^+$, cioè del complementare di $\{0\}$. È r.p. per ricorsione semplice e inoltre si può osservare che $\overline{sgn}(x) = 1 \dot{-} sgn(x)$.

- $\varepsilon(x,y) = sgn(|x - y|)$. Allora $\varepsilon(x,y) = 0$ se $x = y$, $\varepsilon(x,y) = 1$ altrimenti. È la funzione caratteristica dell' identità su $\mathbb{N}$.

- $\delta(x,y) = \overline{sgn(|x - y|)}$. Allora $\delta(x,y) = 0$ se $x \neq y$, $\delta(x,y) = 1$ altrimenti. È la funzione caratteristica della non identità su $\mathbb{N}$.

- Quoziente e resto della divisione di y per x : $qu(x,y)$ e $re(x,y)$
  $qu(x,y) = \begin{cases} qu(x,0) = 0 \\ qu(x,y') = qu(x,y) + \delta(x, S(re(x,y))). \end{cases}$
  $re(x,y) = \begin{cases} re(x,0) = 0 \\ re(x,y') = S(re(x,y)) \cdot \varepsilon(x, S(re(x,y))). \end{cases}$
  Da notare che $re(0,y) = y$ e $qu(0,y) = 0$.

- Numero di divisori di x $D(x)$:
$$D(x) = \sum_{z \leq x} \delta(0, re(z,x))$$

dove si somma 1 ogni volta che z divide x e $D(0) = D(1) = 1$.

- Massimi e minimi:

$$max(x, y) = x\dot{-}(x\dot{-}y)$$

$$min(x, y) = y + (x\dot{-}y).$$

- Sommatorie e produttorie finite: se la funzione f è r.p., allora anche le funzioni g_1, g_2 e h_1, h_2 lo sono:

$$g_1({}^n x, z) = \sum_{y<z} f({}^n x, y) = \begin{cases} 0 & se \quad z = 0 \\ f({}^n x, 0) + \ldots + f({}^n x, z-1) & se \quad z > 0 \end{cases}$$

g_2 si definisce in termini di g_1:

$$\sum_{y\leq z} f({}^n x, y) = \sum_{y<z+1} f({}^n x, y) = g_2({}^n x, z).$$

$$h_1({}^n x, z) = \prod_{y<z} f({}^n x, y) = \begin{cases} 1 & se \quad z = 0 \\ f({}^n x, 0) \cdot \ldots \cdot f({}^n x, z-1) & se \quad z > 0 \end{cases}$$

h_2 si definisce in termini di h_1:

$$\prod_{y\leq z} f({}^n x, y) = \prod_{y<z+1} f({}^n x, y) = h_2({}^n x, z).$$

Infatti possiamo esprimere g_1 tramite lo schema di ricorsione semplice:
$\begin{cases} g_1({}^n x, 0) = 0 \\ g_1({}^n x, z+1) = g_1({}^n x, z) + f({}^n x, z). \end{cases}$

Possiamo inoltre ottenere g_2 a partire da g_1:

$$g_2({}^n x, z) = g_1({}^n x, z+1).$$

Analogamente, si dimostra che anche i prodotti sono r.p..

Alcune di queste funzioni sono solo ausiliarie, altre, invece, verranno utilizzate in séguito per dimostrare importanti risultati. Altre funzioni r.p., inoltre, si ottengono dalle relazioni r.p.:

Definizione 1.4. Sia $R \subseteq \mathbb{N}^k$ una relazione k-aria. Diremo che è ricorsiva primitiva (r.p.) se la sua funzione caratteristica $\chi_R(x)$ in $\mathbb{N}^k$ è r.p.. Per k=1 invece che di relazioni, si preferisce parlare di *proprietà* o *insiemi* r.p..

Osservazioni:

1. Combinazioni booleane (cioè mediante connettivi) di relazioni r.p. sono r.p. quindi
2. $\cup, \cap$ finite, complementi, differenze di sottoinsiemi r.p. di $\mathbb{N}^k$ sono r.p.
3. Ogni sottoinsieme finito di $\mathbb{N}^k$ è r.p..
4. Le usuali relazioni d' ordine in $\mathbb{N}$ sono r.p.: ad esempio, $<$ ha per funzione caratteristica $\chi_<(x,y) = \delta(0, y \dot{-} x)$, che sappiamo essere r.p.
5. L' insieme dei numeri primi, Pr, è r.p.. La sua funzione caratteristica è infatti $\chi_{Pr}(x) = max(\varepsilon(D(x), 2), \chi_>(x, 1))$ che è evidentemente r.p..

Vedremo che è r.p. anche la successione dei numeri primi.

Definizione 1.5. Sia $R \subseteq \mathbb{N}^{m+1}$ una relazione $(m+1)$-aria. Definiamo quantificazioni limitate rispettivamente *esistenziale* e *universale* di R su y le relazioni $(m+1)$-arie S e T, tali che

- $({}^m x, z) \in S \Leftrightarrow$ esiste un $y \leq z$ per cui vale $R({}^m x, y)$; $S = S({}^m x, z)$ si indica con $(\exists y_{\leq z}) R({}^m x, y)$;
- $({}^m x, z) \in T \Leftrightarrow$ per ogni $y \leq z$ vale $R({}^m x, y)$; $T = T({}^m x, z)$ si indica con $(\forall y_{\leq z}) R({}^m x, y)$.

Se R è r.p. lo sono anche S e T.

Definizione 1.6. Sia $R \subseteq \mathbb{N}^{m+1}$ una relazione $(m+1)$-aria, allora la funzione f, anch' essa $(m+1)$-aria, si dice definita a partire da R mediante l' applicazione del *μ-operatore (o operatore di minimo) nel caso limitato* se:

$$f({}^m x, z) = \begin{cases} min\{y \in \mathbb{N}; y \leq z \wedge R({}^m x, y)\} & \text{se l' insieme non è vuoto} \\ z' & \text{altrimenti} \end{cases}$$

Una tale f è r.p..
In genere, per indicare una tale funzione $f = f({}^m x, z)$ si scrive $(\mu y_{\leq z}) R({}^m x, y)$.

Esempio 1.1. La successione crescente $(p_m)_{m\in\mathbb{N}}$ dei numeri primi è r.p.. Essa si può vedere come funzione unaria. Dalla dimostrazione di Euclide sull' infinità dei numeri primi segue che $p_{m+1} \leq (p_m! + 1)$. Con questa disuguaglianza si può definire il μ-operatore limitato

$$p_{m+1} = (\mu y_{\leq (p_m!+1)})(Pr(y) \wedge y > p_m)$$

Inoltre $p_0 = 2$.

Esempio 1.2. Altra funzione r.p. che in séguito utilizzeremo è $exp(i, x)$, definita per mezzo del μ-operatore nel caso limitato. Essa associa ad ogni x l' esponente (i-esimo) di $p_i, i \geq 0$, nella scomposizione di x in numeri primi. Nella definizione che vedremo si sfruttano anche la successione dei numeri primi ed il teorema fondamentale dell' aritmetica sull' unicità della fattorizzazione dei numeri naturali $x \geq 2$:

$$exp(i, x) = (\mu y_{\leq x})(p_i^y | x \wedge p_i^{y+1} \nmid x).$$

Convenzionalmente $exp(i, 0) = exp(i, 1) = 0$.

1.4 Funzioni ricorsive

Ora, per ampliare la classe delle funzioni ricorsive primitive, presentiamo un terzo schema, in aggiunta a quelli di sostituzione e ricorsione semplice.

Definizione 1.7. μ-operatore e schema di minimo. Sia $f = f({}^n x, z)$ una funzione $(n+1)$-aria ($n \geq 1$). Essa si dice *regolare* in z se l' equazione $f({}^n x, z) = 0$ ha almeno una soluzione in z per ogni ${}^n x \in \mathbb{N}^n$; allora la funzione g (n-aria) si dice definita mediante il μ-operatore se

$$g({}^n x) = min\{z \in \mathbb{N}; f({}^n x, z) = 0\} = \mu z(f({}^n x, z) = 0)$$

Esso è detto schema di minimo e sottointenderemo sempre che verrà applicato nel caso regolare.

Definizione 1.8. Una funzione si dice *μ-ricorsiva* o semplicemente *ricorsiva* se è una funzione iniziale o se è ottenibile a partire dalle funzioni iniziali, applicando un numero finito di volte gli schemi di sostituzione, ricorsione semplice e di minimo.

Osservazione. Le funzioni e le relazioni r.p. sono ricorsive. Non vale il viceversa, infatti, è possbile esibire un esempio di funzione ricorsiva ma non r.p., la cosiddetta funzione di Ackermann [Hermes 1975, Cap.14, §3].

Analogamente, se consideriamo relazioni $R \subseteq \mathbb{N}^{n+1}$, regolari in z, cioè tali che per ogni ${}^n x$ esiste almeno uno z per cui vale $R({}^n x, z)$, possiamo definire lo schema di minimo per relazioni:

Definizione 1.9. Sia R una relazione ($n+1$-aria) regolare in z, allora a partire da questa si può definire una funzione g (n-aria) mediante il μ- operatore se

$$g({}^n x) = \mu z(R({}^n x, z)) = min\{z \in \mathbb{N}; R({}^n x, z)\}.$$

La definizione che abbiamo appena dato di funzioni ricorsive riveste particolare importanza poichè fornisce una definizione matematica adeguata delle idee più informali di computabilità ed algoritmo, individuando una classe naturale di funzioni, che il logico americano Alonzo Church (Washington, 14 giugno 1903 − Hudson, 11 agosto 1995) identifica, tramite la cosiddetta *Tesi di Church*, con quella delle funzioni effettivamente computabili. Essa, in quanto tesi, è un' affermazione di tipo filosofico e non è dimostrabile. Tuttavia, altre classi di funzioni come quelle Turing-computabili e quelle computabili nel senso di Markov, ne costituiscono un rafforzamento, infatti nei capitoli successivi dimostreremo come tali concetti descrivono anch'essi la medesima classe di funzioni.

Capitolo 2

Macchine di Turing

2.1 Cenni biografici su Alan Turing [Hodges]

Figura 2.1: Alan Turing

È nel 1936 che Alan Turing, alla età di soli 24 anni, lascia il suo più noto contributo elaborando il concetto della cosiddetta Macchina di Turing, descritta in un articolo dal titolo *"On computable numbers with an application to Entscheidung problem"*[Turing 1937] (sui numeri calcolabili con una applicazione al problema della decisione), articolo che verrà completato nel 1937

all' interno di un biennio di studi presso la Università di Princeton (Stati Uniti d' America). Tale articolo, come il titolo stesso lascia intendere, trae ispirazione dal problema della decidibilità, formulato da Hilbert.

Per motivare l' originalità dell' approccio "concreto" di Turing alla computabilità, possiamo considerare alcuni passi della sua biografia. Nato il 23 giugno 1912 a Londra da Julius ed Hethel Turing, secondo di due fratelli, Alan viene affidato all'età di circa due anni alla famiglia del Colonnello Inglese Ward: il padre Julius, infatti, è un alto funzionario dell' *Indian Civil Service* ed è portato a prestare servizio a Madras, in India. Sin dalla tenera età Alan Turing, tuttavia, mal gradisce il rigore proprio di quella famiglia e delle prime scuole che deve frequentare, maturando piuttosto un interesse rivolto alle scienze sperimentali, prima fra tutte la chimica. Tale interesse matura in séguito alla lettura di un libro che gli viene donato durante la sua infanzia: "Natural wonder every child should know" di Brewster, di cui più tardi dirà che fu il libro che gli "aprì gli occhi alla scienza": il suo interesse per gli esseri viventi in quanto complesse macchine funzionanti divenne notevole, unito alla passione per l' invenzione di nuove macchine artificiali. Anche nel 1922, quando viene ammesso alla Hazelhurst Public School, si distingue dagli altri compagni per la sua indole da vero sperimentatore ed affianca a letture di fondamenti della matematica (quali gli scritti di Russel) testi ben più affini alle scienze naturali, come "La natura del mondo fisico", di A. S. Eddington. In séguito l' interesse di Turing si sposta verso la matematica e nel 1931 egli riesce a vincere una cospicua borsa di studio presso il King's College (a Cambrdge). Egli si ritrova a conoscere Pure Mathematics, libro di G. H. Hardy su cui si sono formate generazioni di studenti inglesi. In questo testo è insita la predilezione per la matematica pura a dispetto di quella applicata, considerata meno nobile agli occhi di Hardy e non solo. Di fronte a questa diatriba, Alan Turing mantiene un interesse per entrambi i campi, senza mai dimenticare quindi del tutto la sua affinità alle materie più legate alla natura. I grandi sconvolgimenti politici europei degli anni '30 non suscitano in Turing − a differenza di molti suoi compagni di studi − grandi

reazioni, se non una ferma condanna nei confronti dei regimi filonazisti, che comunque non sfugge all' attenzione dei servizi dell' Intelligence inglese e che negli anni successivi lo coinvolgerà in prima persona.

Intanto, nel 1935, la maturazione intellettuale di Turing raggiunge un punto cruciale quando, nel contesto di un corso di logica tenuto a Cambridge da M. H. A. Newman, viene a conoscenza degli scritti di Gödel riguardanti il problema della Decidibilità, ossia, data una proposizione matematica, il problema dell'esistenza di un metodo definito applicabile ad essa per stabilire se questa sia dimostrabile o meno. Ed è proprio grazie agli stimoli offerti da questo problema che Alan Turing, come si diceva, approda nel 1936-1937 al concetto della omonima macchina, fornendo in sostanza una definizione di funzioni effettivamente computabili.

Se fino al 1939 l' impegno politico di Turing era stato piuttosto marginale, quasi inesistente, con lo scoppio del secondo conflitto mondiale egli viene coinvolto in questioni della massima importanza. L' Intelligence britannica vuole infatti la sua partecipazione nell' impresa di decifrare il codice utilizzato dalle macchine *Enigma*, impiegate dai Tedeschi per le comunicazioni belliche: si tratta in sostanza di individuare il sistema con cui i messaggi vengono codificati prima di essere trasmessi, in modo da poter spiare le comunicazioni tra gli alti comandi e le truppe. Turing non è l'unico a lavorare a questa impresa militare: assieme a lui una vera e propria squadra di matematici e logici, con sede a Bletchley Park, opera nel tentativo di forzare le macchine Enigma; egli è, però, il primo ad avere successo, dimostrando ancora una volta la sua genialità. Tali impegni bellici perdurano per il Matematico inglese fino al 1944, culminando anche nella costruzione di dispositivi attivi in grado di inficiare le comunicazioni nemiche. Parallelamente agli impegni legati alla guerra, Turing si cimenta con la progettazione di una macchina computazionale elettronica e, nel 1945, grazie all' ingresso nel National Physical Laboratory (Teddington), ottiene i mezzi per la costruzione dell' *Automatic Calculating Engine (ACE)*. Rispetto alla macchina costruita un anno prima da John von Neumann *(Electronic Discrete Variable Automatic Computer,*

EDVAC), il calcolatore sviluppato da Turing ha il vantaggio di avere un programma con possibilità di modifiche. Oltre ad ACE, Turing lavora, nel 1949, alla realizzazione di un altro calcolatore, il MADAM *(Manchester Automatic Digital Machine*), con l' importante ruolo di direttore del Computing Laboratory (Manchester). La crescente relazione con l' universo dei calcolatori spinge Turing ad alcune importanti riflessioni circa la capacità delle macchine di pensare: attività che si concretizza nel 1950 con un articolo dal titolo "Computing machinery and Intelligence, pubblicato per la rivista Mind, in cui egli propone un test (famoso oggi come *test di Turing*), un criterio per fornire una risposta al quesito: una macchina è sufficientemente intelligente se è in grado di fornire risposte per noi non distinguibili rispetto a quelle fornite da un essere umano. I primi anni '50 vedono, poi, Turing impegnato in studi di biomatematica e meccanica quantistica. La vita di Alan Turing si conclude il 7 giugno 1954 (all' età di soli 42 anni): egli viene ritrovato morto, probabilmente suicida, per avvelenamento. La ragione è quasi sicuramente da ricercarsi in alcune vicende personali che lo hanno coinvolto negli ultimi anni: una storia d' amore con un ragazzo conosciuto nel 1951, che una volta conclusasi porta quest' ultimo a ricattare Turing. Il Matematico decide così di denunciare il fatto firmando in realtà la propria condanna di fronte all' opinione pubblica: l' accusa di omosessualità è troppo dura per il rigido clima inglese e Turing viene addirittura arrestato. Grazie ai suoi studi ed ai preziosi contributi nel campo della teoria della computabilità e dei primi calcolatori, oggi Turing è ricordato come uno dei principali padri dell' informatica e dell' intelligenza artificiale.

2.2 Macchine di Turing

Abbiamo già asserito come un algoritmo o una procedura generale, debba essere costituita da un numero finito di passi ben determinati, senza ambiguità, che non lasci, cioè, nulla all' arbitrio. È, allora, possibile immaginare, come fece Turing, che tali passi possano essere eseguiti da una macchina,

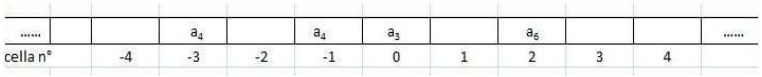
che rappresenta in questo caso, un modello di calcolo. Un modello di calcolo è un'astrazione di un esecutore reale, in cui si omettono dettagli irrilevanti allo studio di un algoritmo per risolvere un problema. Esistono modelli più generali di altri (un dato modello è più generale di un altro se tutti i problemi risolubili con quest' ultimo sono risolubili anche con il primo). In questo senso, la macchina di Turing costituisce un modello di calcolo assolutamente generale, dunque, molto potente. Per fissare le idee, seguendo [Hermes 1975], assumiamo che il compito dell' esecutore sia quello di calcolare il valore di una funzione per un dato argomento seguendo le istruzioni che gli vengono fornite. Per eseguire i calcoli che a partire dai dati di input porteranno ad ottenere il risultato, la macchina di Turing prevede l'uso di un nastro unidimensionale suddiviso in celle, estendibile indefinitamente. Quasi tutte le celle sono vuote, tranne un numero finito, in cui può essere impresso un simbolo proprio tra $\{a_1, \ldots, a_n\}$. Il simbolo vuoto, invece, viene detto anche improprio e denotato con a_0 o $*$. In ciascuna cella può venire impresso al più un simbolo. I simboli impressi sulle celle del nastro formano quella che definiremo *espressione del nastro*. In quanto al calcolo, esso si svolge eseguendo istruzioni finite e possiamo immaginarlo suddiviso in tanti singoli passi elementari. Ad ogni passo è possibile prendere in esame e cambiare il simbolo di una sola cella, o scrivendone uno, che indicheremo con a_k, in una cella precedentemente vuota, o sostituendone uno ad un altro già impresso. Gli altri possibili passi di calcolo prevedono lo spostamento a destra di una cella (d), a sinistra di una cella (s) o di fermarsi (f). Per stabilire quale dei passi deve essere eseguito in un dato momento, ci si basa sulla *configurazione di macchina*. Essa è descritta dall' istruzione del calcolo, dal simbolo impresso nella cella in esame in quel momento, e dalla cella in esame stessa, che viene detta *cella osservata*. Più formalmente, consideriamo la seguente

Definizione 2.1. Sia A un alfabeto costituito dai simboli $\{a_1, \ldots, a_N\}, N \geq 1$, a_0 il simbolo vuoto e d, s, f altri simboli non in A. Definiamo macchina di Turing $\mathcal{M}$ sull' alfabeto A identificandola con una matrice di $M(N+1)$ righe e 4 colonne $(M \geq 1)$ della forma

c_1	a_0	b_1	$c_1^{'}$
$\dots$	$\dots$	$\dots$	$\dots$
c_1	a_N	b_{N+1}	$c_{N+1}^{'}$
c_2	a_0	b_{N+2}	$c_{N+2}^{'}$
$\dots$	$\dots$	$\dots$	$\dots$
c_2	a_N	b_{2N+2}	$c_{2N+2}^{'}$
$\dots$	$\dots$	$\dots$	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$
c_M	a_0	$b_{(M-1)N+M}$	$c_{(M-1)N+M}^{'}$
$\dots$	$\dots$	$\dots$	$\dots$
c_M	a_N	b_{MN+M}	$c_{MN+M}^{'}$

dove $c_1, \dots, c_M$ sono numeri naturali distinti ≥ 0, $c_j^{'} \in \{c_1, \dots, c_M\}$, per $j = 1, \dots, MN + M$ e $b_j \in \{a_0, \dots, a_N, d, s, f\}$. Tale matrice viene anche detta *tavola della macchina* $\mathcal{M}$. I c_j sono detti *stati*. Essi sono una descrizione delle configurazioni interne della macchina durante il calcolo. In particolare c_1 è lo stato iniziale e verrà talvolta indicato anche con $c_{\mathcal{M}}$. La presenza di uno stato terminale c_j viene invece determinata da una riga che inizia con $c_j a_k f$. La mancanza di arbitrio nello svolgimento del calcolo viene tradotta nel fatto che per ogni coppia c_j, a_i esiste esattamente una riga che inizia con tali simboli. Infine, chiamiamo l' alfabeto A, l' alfabeto di $\mathcal{M}$.

Nastro per il calcolo e configurazioni. Le celle del nastro siano numerate con numeri interi x, come rappresentato in figura:



Indicheremo con $B(x) = a_j$ per $j = 0, 1, \dots, N$ il contenuto della cella x. Le celle vuote per convenzione si ammette che abbiamo impresso il simbolo a_0 e si assume che i simboli propri siano impressi solo in un numero finito di celle.

Definizione 2.2. Una *configurazione* K di una macchina di Turing $\mathcal{M}$ è una tripla ordinata $< A, B, C >$, dove A è il numero di una cella, B un' espressione del nastro e C uno stato. Una configurazione $< A, B, C >$ viene detta *iniziale* se $C = c_1$ ed ogni configurazione corrisponde in modo univoco alla riga della tavola di $\mathcal{M}$ che inizia con $C\ B(A)$, che chiamiamo *riga della configurazione* K. Una configurazione viene detta *terminale* se la riga di tale configurazione inizia con $C\ B(A)f$.

Ora, se $K =< A, B, C >$ è una configurazione non terminale a cui corrisponde quindi la riga $CB(A)bc'$, con $b \neq f$, possiamo considerare la funzione $F(K) =< A', B', C' >$ che associa a K la *configurazione consecutiva*:

$$A' = \begin{cases} A & se \ \ b \neq d \wedge b \neq s \\ A+1 & se \ \ b = d \\ A-1 & se \ \ b = s \end{cases}$$

$$B'(x) = \begin{cases} B(x) & se \ \ x \neq A \\ B(x) & se \ \ x = A \wedge (b = d \vee b = s) \\ b & se \ \ x = A \wedge (b \neq d \vee b \neq s) \end{cases}$$

$$C' = c'$$

Il significato di F è il seguente [Hermes 1975]: ad ogni passo del calcolo si passa da una certa configurazione a quella successiva. Più precisamente con l'n-esimo passo del calcolo si passa dalla $(n-1)$-esima configurazione alla n-esima. Se l' n-esima configurazione è terminale, diremo che la macchina cessa di operare dopo l' n-esimo passo.
Fissati una macchina $\mathcal{M}$, un numero A e una funzione B, possiamo determinare una configurazione iniziale $K_0 =< A, B, c_{\mathcal{M}} >$ e diremo che la macchina $\mathcal{M}$ viene applicata all' espressione del nastro B sulla cella A. Ora, se K_0 non è terminale, esiste una configurazione consecutiva $K_1 = F(K_0)$ e diremo che $\mathcal{M}$ cambia nel primo passo da K_0 a K_1. Analogamente, se K_1 non è terminale, esiste una configurazione $K_2 = F(K_1)$ e così via. Se nessuna delle configurazioni $K_0, K_1, K_2, \ldots$ è terminale, esisteranno infinite

$K_n, \forall n$ e diremo che la macchina non cessa mai di operare. Al contrario, se è possibile individuare un indice i tale che $K_i =< A_i, B_i, C_i >$ è una configurazione terminale, avremo che non esisterà una configurazione consecutiva K_{i+1} e diremo che $\mathcal{M}$ cessa di operare (cioè si ferma) dopo l'i-esimo passo, sull' espressione del nastro B_i e sulla cella A_i. In particolare il simbolo a_j contenuto nella cella A_i viene determinato da $B_i(A_i)$. Diremo allora che $\mathcal{M}$ cessa di operare su a_j. Se quest' ultimo $a_j \neq a_0$, allora esiste una parte di nastro che comprende la cella A_i ed eventualmente anche altre celle ad essa contigue, impressa con simboli propri, che formano una certa parola W. In questo caso, diremo che $\mathcal{M}$ cessa di operare sulla parola W.
Sia K_n configurazione non terminale, se la riga corrispondente è $C_nB(A_n)bc$ con $b = s$ oppure $b = d$, allora diremo che nell' $(n+1)$-esimo passo la macchina si muove rispettivamente a sinistra o destra. Se invece b è uguale ad un certo a_j, che in A_n viene impresso il simbolo a_j.

Definizione 2.3. Date due macchine di Turing $\mathcal{M}_1$ ed $\mathcal{M}_2$, esse si dicono equivalenti se esiste una funzione biettiva φ tra gli stati di $\mathcal{M}_1$ e di $\mathcal{M}_2$, tale che

- ogni riga $cabc'$ di $\mathcal{M}_1$ viene portata in $\varphi(c)ab\varphi(c')$ di $\mathcal{M}_2$

- $\varphi(c_{\mathcal{M}_1}) = c_{\mathcal{M}_2}$.

Osservazione. Due macchine di Turing equivalenti, $\mathcal{M}_1$ ed $\mathcal{M}_2$, se applicate alla stessa espressione del nastro B sulla stessa cella A, producono le stesse sequenze $< A_n, B_n(x) >$. Infatti se $\mathcal{M}_1$ produce le configurazioni $< A_n, B_n(x), C_n >$, dato che vi è una corrispondenza biunivoca φ degli stati delle due macchine, si ha anche una corrispondenza tra le rispettive configurazioni: $< A_n, B_n(x), C_n > \leftrightarrow < A_n, B_n(x), \varphi(C_n) >$

Definizione 2.4. [Computabilità nel senso di Turing] Sia $A_0 = \{a_1, \ldots, a_N\}$ un alfabeto, diremo che una funzione unaria f è *computabile nel senso di Turing* se esiste una macchina di Turing $\mathcal{M}$ su un alfabeto A, con $A_0 \subset A$, tale che per qualsiasi parola W su A_0 che viene impressa sul nastro (che

per il resto è vuoto), se $\mathcal{M}$ viene posta su una qualsiasi cella, essa cessa di operare dopo un numero finito di passi dopo una parola che rappresenta il valore $f(W)$ della funzione.
Sia $f(W_1, \ldots, W_k)$ una funzione k-aria, con $W_1, \ldots, W_k$ parole su A_0. Diciamo che f è computabile nel senso di Turing se esiste una macchina di Turing $\mathcal{M}$ su un alfabeto A, con $A_0 \subset A$ tale che, applicata su qualunque cella del nastro, in cui sono stampati soltanto gli argomenti $W_1, \ldots, W_k$ separati dal simbolo vuoto a_0, $(W_1 a_0 W_2 a_0 \ldots a_0 W_k)$, cessa di operare dopo un numero finito di passi dopo la parola $f(W_1, \ldots, W_k)$.
Per completezza definiamo anche funzioni di zero argomenti, che sono sempre Turing-computabili: esse assumono un solo valore che può essere una qualsiasi parola W. Una funzione 0-aria è computabile nel senso di Turing se esiste una macchina di Turing che applicata ad un nastro vuoto, cessa di operare in un numero finito di passi dopo il valore della funzione W.

Riproponiamo anche le definizioni di enumerabilità e decidibilità, questa volta riferite alle macchine di Turing.

Definizione 2.5. Un insieme M è detto *enumerabile nel senso di Turing* se è vuoto o se coincide con l'immagine di una funzione Turing-computabile.

Definizione 2.6. Un insieme M di parole su un alfabeto A_0 è detto *decidibile nel senso di Turing* se esiste una macchina di Turing $\mathcal{M}$ su un alfabeto A ($A_0 \subset A$) e due simboli distinti $a_i, a_j \in A$, propri o impropri, tali che, applicando $\mathcal{M}$ su una qualunque cella del nastro, avente impressa una parola W su A_0 o eventualmente vuoto, $\mathcal{M}$ cessa di operare dopo un numero finito di passi su a_i se $W \in M$, oppure su a_j se $W \notin M$.
Siano M_1, M_2 due insiemi di parole su A_0 con $M_1 \subset M_2$. Diciamo che M_1 è *decidibile nel senso di Turing relativamente a* M_2 se esiste una macchina di Turing $\mathcal{M}$ su un alfabeto A ($A_0 \subset A$) e due simboli $a_i, a_j \in A$, propri o impropri, tali che, applicando $\mathcal{M}$ su una qualunque cella del nastro, contenente una parola $W \in M_2$ o eventualmente vuoto, cessa di operare dopo un numero finito di passi su a_i se $W \in M_1$ oppure su a_j se $W \notin M_1$. Una

proprietà di parole è Turing-decidibile se l' insieme delle parole che godono di tale proprietà è Turing-decidibile.

Le definizioni 2.4, 2.5 e 2.6 verranno applicate soprattutto al caso aritmetico: rappresentiamo d' ora in poi i numeri naturali che compariranno come arogmenti di funzioni Turing-computabili, con delle sequenze di sbarre |. Più precisamente lo zero con una sbarra, l' 1 con ||, e in generale n con $n+1$ sbarre. Identifichiamo poi la sbarra | con a_1, quindi $A_0 = \{|\}$.

2.3 Macchine elementari

Date le prime definzioni e stabilite le dovute convenzioni, introduciamo tre primi semplici tipi di macchine grazie a cui se ne possono ricavare di più complesse.

Macchina di destra d: supponiamo di avere un alfabeto $\{a_0, a_1, \ldots, a_N\}$, allora d è determinata dalla tavola:

$$\begin{array}{c} 0\ a_0\ d\ 1 \\ \ldots\ldots\ldots \\ 0\ a_N\ d\ 1 \\ 1\ a_0\ f\ 1 \\ \ldots\ldots\ldots \\ 1\ a_N\ f\ 1 \end{array}$$

Essa può essere applicata su una qualsiasi cella del nastro, contenente una qualsiasi espressione (che non verrà alterata) ed effettua un singolo passo, cessando di operare sulla cella contigua a destra rispetto alla cella in esame.

Macchina di sinistra s: viene applicata ad una espressione del nastro qualsiasi (senza alterarla) su una cella qualsiasi e si muove di una sola cella verso sinistra. La sua tavola è data da

$$\begin{array}{c} 0\ a_0\ s\ 1 \\ \ldots\ldots\ldots \\ 0\ a_N\ s\ 1 \end{array}$$

$$1\ a_0\ f\ 1$$
$$\ldots\ldots\ldots$$
$$1\ a_N\ f\ 1$$

Macchina $a_j (1 \leq j \leq N)$**:** Fissato j, questa macchina viene applicata ad una espressione del nastro qualsiasi su una cella qualsiasi e, se la cella in esame contiene già il simbolo a_j, cessa di operare dopo 0 passi sulla stessa cella lasciando l'espressione del nastro inalterata, mentre se la cella in esame è segnata con un simbolo diverso da a_j, questa macchina cessa di operare dopo un singolo passo, sulla stessa cella, imprimendovi la lettera a_j. La sua tavola è

$$0\ a_0\ a_j\ 0$$
$$0\ a_1\ a_j\ 0$$
$$\ldots\ldots\ldots$$
$$0\ a_j\ f\ 0$$
$$\ldots\ldots\ldots$$
$$0\ a_N\ a_j\ 0$$

Mostriamo un esempio significativo di macchina di Turing che richiameremo successivamente nella dimostrazione dell' equivalenza tra Turing-computabiltà e ricorsività. Sia $\{|\}$ il nostro alfabeto, che risulta sufficiente per esprimere i numeri naturali (0=|, 1=||, 2=|||, ecc). La seguente tavola di macchina, applicata ad un nastro (altrimenti vuoto) su cui è impressa una sequenza di sbarre che rappresentano il numero naturale dato in input, prende come cella in esame quella vuota (contrassegnata con $*$), che si trova immediatamente dopo l' ultima sbarra. Essa aggiunge una sbarra alla sequenza e cessa di operare a destra di questa. Essa permette di calcolare il successore di ogni numero naturale:

$$0 * |\ 0$$
$$0\ |\ d\ 1$$
$$1 * f\ 1$$
$$1\ |\ f\ 1$$

Posta sulla prima cella vuota a destra dell' input, scrive | e rimane nello stato iniziale 0 (prima riga). Trovandosi nello stato 0 e rimanendo nella stessa cella, sovrascrive |, si sposta a destra di una cella e passa nello stato 1 (seconda riga). Trovandosi nello stato 1 su una cella vuota, si ferma, conservando lo stato 1 (terza riga). La quarta riga, ai fini dei passi di calcolo, è superflua, ma viene comunque inserita nella tavola di macchina perché ci siano tutte le possibili combinazioni stato/cella.
Tuttavia, mediante questa macchina non è possibile affermare che il successore è una funzione Turing-computabile, in quanto può essere applicata solo su una particolare cella. Ritorneremo su questo particolare più avanti, nella prossima sezione.

Combinazioni di macchine e diagrammi. Come si usa fare per progettare e descrivere i programmi, introduciamo i diagrammi di flusso per costruire una macchina complessa $\mathcal{M}$ a partire da altre $\mathcal{M}_1, \ldots, \mathcal{M}_r$. Siano date delle macchine di Turing $\mathcal{M}_1, \ldots, \mathcal{M}_r$ sull' alfabeto A=$\{a_1, \ldots, a_N\}$, allora è possibile formare un diagramma D tale che:

- In D compare almeno una macchina tra $\mathcal{M}_1, \ldots, \mathcal{M}_r$. Ogni macchina può comparire solo un numero finito di volte;

- Vi è un punto iniziale del diagramma, che coincide con una occorrenza di una tra le macchine $\mathcal{M}_1, \ldots, \mathcal{M}_r$. Esso è individuato da una freccia circolare $\circlearrowleft$ attorno al simbolo della macchina;

- Le macchine che compaiono nel diagramma sono collegate da frecce numerate, che indicano la macchina a cui passare successivamente;

- Da ogni simbolo può partire al massimo una freccia con uno stesso numero $j = 0, 1, \ldots, N$, ma possono partire più frecce, se con numeri diversi;

La tavola della macchina $\mathcal{M}$, costituita dalla combinazione di tutte le macchine che occorrono del diagramma viene determinata, a meno di equivalenze, nel modo seguente:

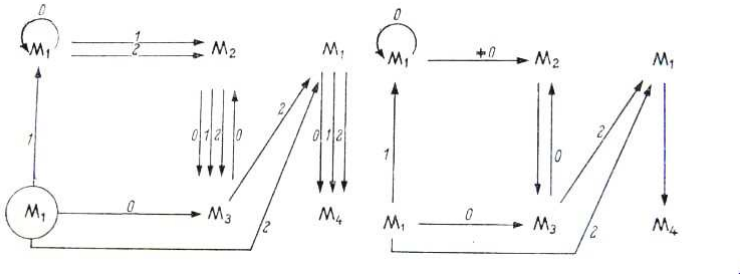


Figura 2.2: Esempi di diagrammi

- Si producono le tavole di ogni macchina $\mathcal{M}_i$ che compare nel diagramma;

- due diverse tavole non possono contenere uno stesso stato;

- si scrive un'unica tavola costituita dalle singole tavole. La tavola della macchina iniziale va posta per prima, mentre le altre possono essere messe in un ordine arbitrario;

- se il simbolo di una macchina $\mathcal{M}_i$ che occorre in un determinato punto del diagramma è collegato per mezzo di una freccia $\xrightarrow{k}$ al simbolo di un'altra macchina $\mathcal{M}_j$, allora le righe del tipo ca_kfc' che compaiono nella tavola di $\mathcal{M}_i$ vanno cambiate in $ca_ka_kc_{\mathcal{M}_j}$, dove $c_{\mathcal{M}_j}$ è lo stato iniziale della tavola di $\mathcal{M}_j$.

Dopodichè non è difficile seguire il modo di operare di D: se si applica $\mathcal{M}$ su una cella A del nastro la cui espressione è data da $B(x)$, inizierà ad eseguire i passi della macchina $\mathcal{M}'$ denotata dal simbolo iniziale. Giunta alla configurazione terminale di $\mathcal{M}'$, contraddistinta da una riga del tipo ca_kfc', se $\mathcal{M}'$ non è collegata mediante nessuna freccia a nessun'altra macchina, la riga terminale rimane tale e il calcolo è finito. In caso contrario, se vi è una freccia di collegamento $\xrightarrow{k}$ tra $\mathcal{M}'$ ed un'altra macchina $\mathcal{M}''$, allora si trasforma l'ultima riga di $\mathcal{M}'$ in $ca_ka_kc_{\mathcal{M}''}$, che ora viene ad essere la riga iniziale di $\mathcal{M}''$. Si prosegue, quindi, con l' applicazione della macchina

$\mathcal{M}''$ e così via. Dunque, intuitivamente, $\mathcal{M}$ esegue i passi delle macchine $\mathcal{M}', \mathcal{M}'', \ldots$ nell' ordine determinato dal diagramma.

2.4 Macchine particolari

Basandoci sulle macchine elementari appena introdotte, mostriamo altre macchine di cui ci serviremo successivamente per provare l' equivalenza tra ricorsività e Turing-computabilità.

Grande macchina di destra D (di sinistra S): essa non altera l'espressione del nastro, bensì si muove dalla cella in cui viene applicata a quella immediatamente a destra (sinistra). Se questa è vuota, la macchina cessa di operare, altrimenti se è segnata, continua a spostarsi a destra (sinistra) fino a quando trova una cella vuota.

Macchina di ricerca a destra δ e a sinistra σ: dalla cella su cui viene applicata si sposta a destra (sinistra) di una cella. Se questa è segnata, cessa di operare, altrimenti se è vuota continua a spostarsi a destra (sinistra) finchè non trova una cella segnata, su cui cessa di operare.

Macchina di ricerca R: applicata su un nastro con almeno una cella segnata, ricerca in modo sistematico, una volta a destra una volta a sinistra, una cella segnata e appena la trova, cessa di operare, senza modificare l' espressione del nastro. Per tenere in memoria i limiti (sia a destra che a sinistra) della ricerca, viene usata la lettera a_1 esclusivamente come indicatore, che viene poi spostato di volta in volta verso l' esterno, fino a raggiungere la cella segnata ricercata.

Macchina di conclusione a destra $\mathcal{D}$ e a sinistra $\mathcal{S}$: a partire dalla cella su cui viene applicata, si muove verso destra (sinistra), finché non si trova sulla seconda di due celle vuote consecutive (senza contare la cella in esame iniziale). Arrivata a questo punto, torna indietro di una cella verso sinistra (destra), senza alterare l' espressione del nastro.

Macchina di traslazione a sinistra T: sposta una parola, lettera per lettera, di una cella verso sinistra.

Macchina di traslazione τ: se W_1 e W_2 sono due parole separate e delimitate da uno spazio $*W_1 * W_2*$, questa macchina, applicata sulla cella dopo W_2, si sposta a sinistra, fino a portarsi sull'ultima lettera di W_1, dopodiché cancella una cella alla volta e fa scorrere indietro W_2, fino a farla coincidere con l' originario inizio di W_1 e cessa di operare sulla cella a destra di W_2.
Macchina di cancellazione C: cancella i calcoli intermedi che vengono prodotti, sotto forma di proposizione X, alla cui sinistra sono due celle vuote e alla cui destra seguono una cella vuota e poi il risultato W (si ricordi che il risultato è l'ultima parola non vuota del nastro) poi, sposta verso sinistra il risultato W, finché l' inizio della parola traslata W non si trova sulla prima delle due celle vuote.
La macchina copiatrice K: viene applicata sulla cella successiva alla parola W che vogliamo copiare, alla cui destra tutte le celle del nastro sono vuote. Cessa di operare dopo un numero finito di passi; le celle che erano segnate in origine rimangono tali e a destra della parola iniziale W stampa una copia di W, separata da essa tramite una cella vuota. K cessa di operare dopo l' ultima cella segnata dalla parola copiata.
Macchine n-copiatrici K_n: vengono usate spesso per calcolare funzioni a più argomenti e quando la parola da copiare non è posta all' estrema destra, quando cioè la copiatura dev' essere operata al di là di alcune parole stampate in mezzo. K_n opera in modo analogo a K, saltando le parole intermedie che non devono essere copiate e copiando la n-esima verso sinistra; pertanto si ha che $K_1 = K$.

Abbiamo già presentato alcuni esempi di macchine di Turing applicabili solo su particolari celle del nastro, che non permettono, dunque, di affermare che le rispettive funzioni sono Turing-computabili. Per ovviare a questo problema introduciamo un teorema [Hermes 1975, Cap.2, §9.1], in cui prendendo in considerazione delle macchine aggiuntive, si prova la Turing-computabilità di tali funzioni.

Teorema 2.4.1. *Sia h una funzione n-aria ($n \geq 1$) definita per tutte le parole non vuote su un alfabeto A=$\{a_1, \ldots, a_N\}$ e a valori nello stesso alfabeto A.*

Sia $\mathcal{N}$ una macchina su A tale che se la si applica dopo l'ultimo simbolo di una qualsiasi n-upla di parole $< W_1, \ldots, W_n >$, essa cessa di operare su una cella che chiameremo c e non altera l'espressione originale del nastro. Sia $\mathcal{M}'$ una macchina tale che se stampiamo sul nastro (altrimenti vuoto) una n-upla $W_1, \ldots, W_n$ di argomenti e se l' applichiamo sulla cella c, allora $\mathcal{M}'$ cessa di operare dopo un numero finito di passi dopo il valore della funzione $h(W_1, \ldots, W_n)$. Allora h è Turing- computabile e la macchina $\mathcal{M}$ che la calcola è data da $\mathcal{M} = R\,\mathcal{D}\mathcal{N}\mathcal{M}'$.

Infatti R ricerca una cella segnata dagli argomenti, poi $\mathcal{D}$ si posiziona sulla cella a destra dell'ultima cella contenente gli argomenti, ed $\mathcal{N}$ sulla cella c, su cui infine è applicata $\mathcal{M}'$ per calcolare $h(W_1, \ldots, W_n)$.

2.5 Equivalenza tra ricorsività e Turing-computabilità

Le funzioni a cui faremo riferimento nel corso delle dimostrazioni di questa sezione, sono quelle definite per tutte le n-uple di numeri naturali, a valori in $\mathbb{N}$.

2.5.1 Turing-computabilità standard

Per effettuare la dimostrazione più agevolmete introduciamo la nozione di *Turing-computabilità standard*, che presenta alcune peculiarità rispetto a quella vista precedentemente. Indicheremo i numeri naturali con sequenze di sbarre, a differenza della Turing-computabilità, dove ci riferivamo a funzioni definite su parole arbitrarie non vuote. Utilizzeremo quindi l' alfabeto A={|} e non più un alfabeto generico $a_1, \ldots, a_N$. Inoltre, la cella presa in esame all' inizio del calcolo ora è definita in modo univoco, quindi, non potremo più applicare una macchina su una cella qualunque. Infine, non assumeremo più che il nastro per il calcolo sia inizialmente vuoto, tranne che per l' argomento dato in input: accetteremo, infatti, anche che alla sinistra degli argomenti dati in input ci siano altre celle, separate da questi da un certo spazio, arbitrariamente segnate, mentre a destra dell' input il nastro dovrà essere vuoto.

Con la locuzione "metà nastro" ci riferiremo alla parte di nastro determinata da una data cella x (x compresa) e da tutte quelle alla sua destra. Con la locuzione "lista di argomenti" di una funzione n-aria, invece, designeremo quella parte del nastro contenente i simboli e le parole $*W_1 * W_2 * \ldots * W_n$. Dunque, la prima cella della lista degli argomenti deve essere vuota e l'ultima contiene l'ultimo simbolo di W_n. Per funzioni di zero argomenti la lista di argomenti non contiene alcuna cella.

Definizione 2.7. Sia f una funzione n-aria. Diremo che f è Turing-computabile in modo standard se esiste una macchina di Turing $\mathcal{M}$ su un alfabeto $\{|\}$, tale che se scriviamo una n-upla di argomenti sul nastro per il calcolo, e se la metà nastro H la cui prima cella è quella immediatamente a destra della lista degli argomenti, risulta vuota, allora la macchina, *applicata sulla prima cella di* H, cessa di operare dopo un numero finito di passi, verificando le seguenti condizioni:

1. gli argomenti dati in input rimangono nella posizione iniziale (non vengono spostati)
2. il valore della funzione ha inizio sulla seconda cella di H, in modo che vi sia una cella vuota che separa gli argomenti dal risultato
3. $\mathcal{M}$ si trova sulla cella immediatamente dopo l'ultima sbarra della sequenza che rappresenta il valore della funzione
4. la metà nastro H è vuota, tranne che per il valore della funzione
5. nel corso del calcolo vengono prese in esame solo le celle della lista di argomenti e le celle di H.

Per funzioni di zero argomenti la definizione diventa: f è Turing-computabile in modo standard se esiste una macchina di Turing $\mathcal{M}$ su un alfabeto $\{|\}$, tale che se applichiamo $\mathcal{M}$ su una cella qualsiasi A e se la metà nastro H la cui prima cella è A risulta vuota, allora $\mathcal{M}$ cessa di operare dopo un numero finito di passi, verificando le seguenti condizioni:

1. il valore della funzione ha inizio sulla seconda cella di H
2. $\mathcal{M}$ si trova sulla cella immediatamente dopo l'ultima sbarra della sequenza che rappresenta il valore della funzione
3. la metà nastro H è vuota, tranne che per il valore della funzione
4. nel corso del calcolo vengono prese in esame solo le celle di H.

Osservazione: una presentazione alternativa. Se si adopera questo alfabeto, diviene ancora più importante delimitare gli argomenti in input, l'input complessivo, i vari passaggi, la fine del risutato, eccetera: questo può essere fatto mediante gli stati interni di $\mathcal{M}$, sequenze di $*$ e così via. Un altro analogo procedimento significativo è quello proposto da R. Penrose in [Penrose, 1992], che sarà esposto di sèguito.
Roger Penrose (8 agosto 1931), fisico e matematico britannico in due libri, quello citato (e anche in [Penrose 1996]), in cui espone argomentazioni contro qualsiasi modello puramente computazionale della comprensione umana, sostenendo che esistono differenze intrinseche ed ineliminabili fra l' intelligenza artificiale e quella dell' uomo, propone una sua descrizione divulgativa della macchina di Turing. Come abbiamo già lasciato intendere, essa non è un oggetto fisico, ma costituisce un concetto matematico astratto, estraneo alle consuete idee matematiche del tempo in cui fu introdotta. Ha un numero finito di stati interni ed agisce su un nastro unidimensionale, infinitamente estensibile, adatto a contenere input, spazi di calcolo ed output potenzialmente illimitati. Esso è composto da una sequenza di celle e ognuna di esse può contenere un segno oppure no. I segni impressi sul nastro devono comunque essere in numero finito. Nella descrizione di Penrose, equivalente a quella già esposta, le celle segnate vanno indicate col simbolo 1, quelle vuote con lo 0. Vi è, poi, un dispositivo di lettura/scrittura che esamina, una per volta, le celle del nastro, che all' inizio del calcolo deve essere completamente bianco, tranne che per i segni che definiscono i dati di input (espressi tramite una stringa finita di 1 eventualmente intervallati da 0) su cui la macchina deve eseguire le operazioni. I dati di input vanno posti alla destra della testina

di lettura. A seconda dello stato interno della macchina e dalla natura del segno in esame, viene determinato in modo univoco:

- se il segno nella cella deve venire alterato o no
- il nuovo stato della macchina
- se il dispostivo si deve poi muovere lungo il nastro di un passo verso destra (indicato con D), verso sinistra (indicato con S) o se deve fermarsi (indicato con STOP).

Dunque, dati lo stato iniziale e l' input, la macchina deve operare in modo completamente deterministico, adempiendo ai compiti descritti nei tre punti sopra. Se e quando, infine, dopo una serie finita di passi di calcolo, la macchina si arresta, il risultato si deve trovare alla sinistra del dispostivo di lettura/scrittura.
Vediamo, ora, nei dettagli, il modo di operare di una macchina di Turing come proposto da Penrose. Esso viene specificato completamente da una sequenza di istruzioni. Innanzitutto, gli stati interni sono numerati coi numeri naturali 0,1,2,3,.... Un' istruzione tipo può essere la seguente: 0$\mathtt{0}\rightarrow\mathtt{11}$R, cioè dallo stato interno iniziale 0, se sulla cella si legge il simbolo `0`, allora passare nello stato 1, scrivere in tale cella il simbolo `1` e muoversi lungo il nastro di una cella verso destra. L' insieme di tutte le istruzioni per calcolare una certa funzione determina la *tavola* della corrispondente macchina di Turing. Le istruzioni, come si evince dall'esempio, sono nella forma di quintuple del tipo:
stato-simbolo letto-nuovo stato-nuovo simbolo-azione da compiere.
A differenza di quanto visto nella presentazione precedente, ora ad ogni istruzione è possibile sia imprimere un nuovo simbolo sul nastro, che effettuare uno spostamento, mentre nel primo caso i due tipi di azione erano mutuamente esclusivi. Come accennato, facendo uso solo dei simboli 0 e 1, si può pensare di scrivere i numeri naturali in notazione unaria, quindi associando il simbolo `1` al numero 1, `11` al numero 2, `111` al numero 3 e così via. Per comprendere in modo più immediato l' azione di una tale macchina vediamo

l' esempio della macchina che calcola il successore di un numero naturale. L' insieme di istruzioni che costituisce la tavola della particolare macchina in questo caso è il seguente:

$$00\rightarrow 00\mathrm{R}, \qquad 01\rightarrow 11\mathrm{R}, \qquad 10\rightarrow 01\mathrm{STOP}, \qquad 11\rightarrow 11\mathrm{R}$$

Applichiamola ad un nastro su cui è impresso il numero 4, in notazione unaria:

$$\ldots 00000001111000000\ldots$$

Supponiamo che il dispositivo inizialmente si trovi a sinistra degli `1`. Esso è nello stato interno 0 e legge `0`, lo lascia così com' è e si sposta a destra di una casella, conservando lo stato interno 0. Ripete quest' istruzione finché non si trova su una casella contente un `1`: si muove a destra e conserva il simbolo `1`, ma passando allo stato 1. Ora, passa ad eseguire la quarta istruzione, fintanto che trova degli `1`, scorrendo il nastro e rimanendo nello stato 1. Quando si trova sulla prima cella contrassegnata dallo `0`, la sovrascrive con un `1`, torna nello stato 0 e si ferma. Ora, il nastro sarà diventato

$$\ldots 000000011111000000\ldots$$

Questa rappresentazione unaria dei numeri crea però problemi per la rappresentazione dello 0 ed inoltre per rappresentare numeri elevati risulta piuttosto inefficiente. Si può ovviare a questo problema introducendo una notazione binaria modificata, detta *sviluppata*. La modifica da apportare consiste in un processo di *contrazione* secondo cui ogni stringa di `0` e `1` (con un numero totale finito di `1`) non viene letta semplicemente come un numero binario, bensì viene sostituita da una stringa di `0,1,2,3`,... in modo tale che venga scritta la cifra corrispondente a quanti sono gli 1 tra due 0 consecutivi:

$$\ldots 0100010110101011010001110101011110011 0\ldots$$

diventa

$$\ldots 10012112100311402\ldots$$

Essa è una stringa di 0 e 1 separati da cifre più grandi. Le sequenze di 0 e 1 rappresentano i numeri in notazione binaria. Poi, stabiliamo il significato dei numeri 2,3,4,..., che possono essere simboli o istruzioni: il 2 rappresenti una virgola che segnala dove termina un numero, il 3 e il 4 possono essere delle istruzioni, come +,-, ecc. Dunque, nel nostro esempio, avremo:

$$9, 3, 4 \text{ (istruzione 3) } 3 \text{ (istruzione 4) } 0,$$

Vediamo anche il passaggio inverso, grazie a cui è possibile codificare una qualsiasi sequenza finita di numeri naturali e istruzioni in una sequenza di 1 e 0. Ad esempio 5,13,0,1 in binario equivale a: 101,1101,0,1. A questo punto, per ottenere la notazione binaria sviluppata si operano le seguenti sostituzioni:

$$0 \to 0$$
$$1 \to 10$$
$$, \to 110$$

e si aggiungono all' estremità destra e sinistra infiniti 0:

$$\ldots 0000100101101010010110011010110000 \ldots$$

Ultima regola da tener presente è che uno 0 tra due virgole, nella trasformazione in notazione binaria, può essere omesso e dunque essere scritto come ,, cosicché sul nastro viene codificato come `110110`. In questo modo, grazie alla notazione binaria sviluppata si possono codificare numeri, ma anche, stabilite le dovute corrispondenze, simboli e istruzioni, a cui si può poi applicare il concetto della macchina di Turing già visto. Fu questa l' intuizione di Turing quando si accostò al problema della decidibilità proposto da Hilbert, in cui si richiedeva una procedura algoritmica per risolvere problemi matematici di natura generale.
In [Penrose, 1992] si precisa, altresì, come già in [Turing, 1937], che la limitazione ad un solo nastro unidimensionale non è restrittiva: anche se avessimo considerato macchine con uno o più nastri a due, tre dimensioni, in linea

di principio, non si sarebbe guadagnato niente di più, se non una maggiore efficacia. Infine, si può dare in questo modo una descrizione intuitiva di una macchina universale $\mathcal{U}$, in cui l' idea fondante è quella di codificare l' elenco delle istruzioni della specifica macchina $\mathcal{M}$ (che si desidera simulare) in una sequenza di 0 e 1, nella notazione binaria sviluppata. Convertendo tale sequenza in notazione decimale si ottiene un numero n, detto numero della macchina $\mathcal{M}$. Scrivendo tale sequenza come input sul nastro di $\mathcal{U}$, quest' ultima è in grado di operare esattamente come farebbe $\mathcal{M}$. In tale codifica le azioni R, L, STOP vengono sostituite rispettivamente da 2, 3 e 4, che sono poi tradotti in 110, 1110 e 11110. Al posto di 0 e 1 invece si sostituiscono rispettivamente 0 e 10. La macchina universale $\mathcal{U}$ prende in input sia il numero n che identifica la macchina che deve simulare, sia l' input m che sarebbe necesario fornire alla macchina che viene simulata. Indicheremo ciò con $\mathcal{U}(n, m) = \mathcal{M}_n(m)$, per ogni (m, n) per cui $\mathcal{M}_n$ sia una macchina correttamente specificata. Questo verrà ripreso con maggiori dettagli in 2.5.4. Continuiamo ora con l' esposizione di [Hermes 1975], notando che se una funzione è Turing-computabile in modo standard, per il teorema 2.4.1 è anche Turing-computabile.

2.5.2 Turing-computabilità delle funzioni ricorsive

Teorema 2.5.1. *Ogni funzione ricorsiva è Turing-computabile in modo standard.*

Dimostrazione. La dimostrazione che faremo è costruttiva [Hermes 1975, Cap.4, §16]: difatti, esibiremo una descrizione effettiva di macchine di Turing che permettono di calcolare i valori delle funzioni iniziali e di quelle ottenibili con i tre schemi di sostituzione, ricorsione semplice e minimo. Le funzioni iniziali sono Turing-computabili in modo standard.

- Per la funzione successore abbiamo la macchina composta da K, una macchina che aggiunge | e l'applicazione di d, il tutto abbreviabile con K| d.

- Per la funzione proiezione U_i^n si usa la macchina copiatrice K_{n+1-i}.
- Per la funzione zero si usa $d \mid d$.

Sostituzione: sia f definita per sostituzione tramite g ed $h_1, \ldots, h_r$:

$$f({}^n x) = g(h_1({}^n x), \ldots, h_r({}^n x)).$$

Supponiamo che g ed $h_1, \ldots, h_r$ siano Turing-computabili in modo standard tramite le macchine $\mathcal{M}, \mathcal{M}_1, \ldots, \mathcal{M}_r$. Allora anche f è Turing-computabile in modo standard tramite la macchina composta da:

$$d|dK_{n+1}^n S^n s * \mathcal{D}\mathcal{M}_1 K_{n+1}^n \mathcal{M}_2 \ldots K_{n+1}^n \mathcal{M}_r K_{r+(r-1)n} K_{r+(r-2)n} \ldots K_r \mathcal{M} C.$$

Dapprima, tramite le $\mathcal{M}_j$ vengono calcolati i valori delle funzioni $h_j({}^n x)$ e con $\mathcal{M}$ viene calcolata $g({}^n x)$. Infine, la macchina di cancellazione C elimina tutta la procedura di calcolo, ad eccezione degli argomenti e del valore finale della funzione. Vediamo nei dettagli l' applicazione di tale macchina composta.* All' inizio sul nastro avremo impressa la n-upla di argomenti

$$*^n x \underline{*} \cdots$$

Tramite $d \mid d$ otteniamo

$$*^n x * |\underline{*} \cdots$$

poi, per mezzo di K_{n+1}^n copiamo la n-upla degli argomenti

$$*^n x * \mid *^n \underline{x*} \cdots$$

mediante $S^n s*$ cancelliamo la sbarra centrale

$$*^n x * \underline{*} *^n x * \cdots$$

e tramite la macchina di conclusione a destra $\mathcal{D}$ otteniamo

$$*^n x * * *^n \underline{x*} \cdots$$

*Ad ogni passo, la cella in esame viene evidenziata con una sottolineatura.

Possiamo ora applicare la macchina $\mathcal{M}_1$, che calcola in modo standard $h_1({}^n x)$

$$*^n x * * *^n x * h_1({}^n x)\underline{*} \cdots$$

Mediante K^n_{n+1}, copiamo l' argomento ${}^n x$ a destra, per poi calcolare $h_2({}^n x)$ tramite $\mathcal{M}_2$

$$*^n x * * *^n x * h_1({}^n x) *^n x * h_2({}^n x)\underline{*} \cdots$$

Analogamente facciamo per $h_3, \ldots, h_r$

$$*^n x * * *^n x * h_1({}^n x) *^n x * h_2({}^n x) \cdots *^n x * h_r({}^n x)\underline{*} \cdots$$

Tramite le macchine copiatrici

$$K_{r+(r-1)n} K_{r+(r-2)n} \ldots K_r$$

riportiamo i valori di $h_1, \ldots, h_r$ all'estremità destra del nastro e possiamo finalmente calcolare f mediante $\mathcal{M}$

$$*^n x * * *^n x * h_1({}^n x) *^n x * h_2({}^n x) \cdots *^n x * h_r({}^n x) * h_1({}^n x) * \cdots h_r({}^n x) * f({}^n x)\underline{*} \cdots$$

Infine, cancelliamo tutti i calcoli intermedi applicando la macchina di cancellazione C; rimangono sul nastro solo l' input e l' output

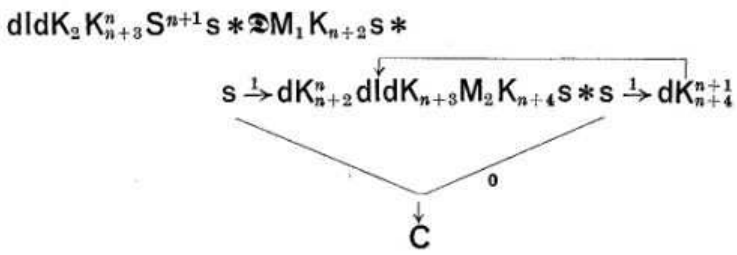
$$*^n x * f({}^n x)\underline{*} \ldots$$

Allora f è Turing-computabile in modo standard da $g, h_1, \ldots, h_r$.

Ricorsione semplice: sia f una funzione definita per ricorsione semplice:

$$\begin{cases} f({}^n x, 0) = g({}^n x) \\ f({}^n x, y') = h({}^n x, y, f({}^n x, y)). \end{cases}$$

con g e h Turing-computabili nel modo standard rispettivamente tramite $\mathcal{M}_1$ ed $\mathcal{M}_2$. Vediamo che anche f lo è. Per comodità poniamo $f({}^n x, y) = f_y$ e supponiamo $n > 0$. La macchina che permette di calcolare f è così articolata [Hermes, p.138]:



Nella figura $\mathfrak{D}$ è la macchina $\mathcal{D}$, come nel séguito.
Diamo in input ${}^n x$ ed y:

$$*^n x * y\underline{*}\cdots$$

e per mezzo di

$$d \mid dK_2 K^n_{n+3} S^{n+1} s * \mathcal{D}$$

otteniamo

$$*^n x * y * * * y *^n x\underline{*}\cdots$$

Applicando, poi, $\mathcal{M}_1$ troviamo il valore di $f_0 = f({}^n x, 0) = g({}^n x)$

$$*^n x * y * * * y *^n x * f_0\underline{*}\cdots$$

Ora, facciamo una copia di y, cancelliamo l'ultima sbarra della copia ottenendo $y - 1$, poi torniamo indietro di una cella tramite $K_{n+2} s * s$

$$*^n x * y * * * y *^n x * f_0 * y - \underline{1}*$$

Se la macchina si trova su una cella vuota, allora abbiamo $y = 0$ e il calcolo termina con la cancellazione dei simboli nelle celle comprese fra ${}^n x$ ed f_0. Al contrario, se la macchina si trova su una cella contenente una sbarra, proseguiamo nel calcolo tramite d, spostandoci a destra di una cella:

$$*^n x * y * * * y *^n x * f_0 * y - 1\underline{*}$$

Ora, calcoliamo mediante il procedimento di ricorsione $f_1, f_2, \ldots, f_y$: dapprima copiamo ${}^n x$, segniamo la cella successiva con una sbarra (che rappresenta lo 0) e infine copiamo f_0. A questo punto, utilizzando queste ultime copie, tramite $\mathcal{M}_2$, possiamo calcolare f_1:

$$*^n x * y * * * y *^n x * f_0 * y - 1 *^n x * 0 * f_0 * f_1\underline{*}\cdots$$

In questi passaggi abbiamo usato la combinazione delle macchine $K^n_{n+2}d|dK_{n+3}\mathcal{M}_2$. Proseguendo con le macchine $K_{n+4}s*s$ copiamo $y-1$, ne cancelliamo l'ultima sbarra e retrocediamo di una cella. Se ora la macchina si trova su una cella vuota, abbiamo che $y-1=0$ e $y=1$ e il calcolo è sostanzialmente terminato, tranne che per la cancellazione dei passaggi intermedi, che va effettuata con C. Altrimenti, se la macchina si trova su una cella segnata con una sbarra, allora $y-1\neq 0$ e dobbiamo continuare il procedimento con dK^{n+1}_{n+4}, che imprime nelle celle seguenti gli argomenti ${}^nx$ e 0. Quest'ultimo viene incrementato di una sbarra mediante $|d$ e viene ricopiato di séguito anche f_1 per mezzo di K_{n+3}

$$*^n x * y * * * y *^n x * f_0 * y-1 *^n x * 0 * f_0 * f_1 * y-2 *^n x * 1 * f_1 \underline{*} \cdots$$

D' ora in poi si ripete il procedimento già descritto, tornando ad applicare $\mathcal{M}_2$ per calcolare, questa volta, f_2:

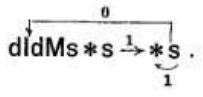
$$*^n x * y * * * y *^n x * f_0 * y-1 *^n x * 0 * f_0 * f_1 * y-2 *^n x * 1 * f_1 * f_2 \underline{*} \cdots$$

Iterando il procedimento, si giunge a calcolare f_y:

$$*^n x * y * * * \quad \text{come sopra, fino a} \quad * y-y *^n x * y-1 * f_{y-1} * f_y \underline{*} \cdots$$

Con un ultimo passaggio si calcola $f_{y'}$ e per concludere si cancellano calcoli intermedi tramite C.

μ-operatore: sia g una funzione $(m+1)$-aria, regolare in y, Turing-computabile nel modo standard tramite la macchina $\mathcal{M}$. Sia $f({}^mx) = \mu y g({}^mx, y) = 0$. Allora anche f è Turing-computabile nel modo standard e la macchina che permette di calcolarla è data da [Hermes, p.240]:



Come prima poniamo $g({}^mx, y) = g_y$.

1. Applichiamo la macchina dopo l' ultima sbarra dell' argomento ${}^mx$: essa si muove a destra di una cella;

2. imprimiamo sul nastro una sbarra, che rappresenta il primo valore su cui calcoliamo g ($y = 0$) e ci spostiamo ulteriormente a destra:

$$*^m x * |\underline{*}.$$

Ora possiamo procedere col calcolo di $g(^m x, 0)$ tramite $\mathcal{M}$.

$$*^m x * | * g_0 \underline{*} \cdots$$

Per mezzo di $s*$ diminuiamo di un'unità il valore di g_0 e applichiamo s spostando la macchina a sinistra di una cella (quella contenente l' ultima sbarra di $g_0 - 1$):

$$*^m x * | * g_0 - \underline{1} * \cdots$$

3. Se la cella in esame non è vuota, significa che $g(^m x, 0) \neq 0$ e tramite $*s$ (ripetuto fintanto che si incontrano celle segnate con una sbarra) si procede alla cancellazione di $g_0 - 1$:

$$*^m x * |\underline{*} \cdots$$

A questo punto rieseguiamo il passo 2, il quale incrementa y di una unità ed esegue il calcolo di $g(^m x, 1)$ tramite $\mathcal{M}$. In generale, alla n-esima ripetizione del passo 2, avremo $g(^m x, n-1)$, anziché $g(^m x, 0)$.

4. Se al termine del passo 2 la cella in esame è vuota, allora $g(^m x, 0) = 0$ (in generale $g(^m x, y) = 0$) e il risultato si trova nella cella immediatamente precedente a quella in esame.

In ogni caso l' ipotesi di regolarità su g garantisce che questo calcolo abbia termine per ogni possibile m-upla $^m x$. □

2.5.3 Ricorsività delle funzioni Turing-computabili

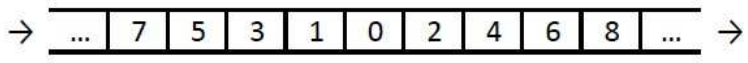
Dimostreremo in questa sottosezione che ogni funzione Turing-computabile è (μ)-ricorsiva (teorema 2.5.2). In preparazione a questa dimostrazione introduciamo la numerazione di Gödel delle macchine di Turing.

Caratterizzeremo le celle, l'espressione del nastro per il calcolo e le configurazioni mediante dei numeri naturali;

stabiliremo una corrispondenza biunivoca tra macchine di Turing e alcuni numeri;

introdurremo delle funzioni che ci permetteranno di determinare dai numeri di una configurazione i numeri delle configurazioni successive.

Numerazione delle celle del nastro. Sia $\mathcal{M}$ una macchina di Tuirng. Sul suo nastro fissiamo come prima una cella e le attribuiamo il numero 0. Alle celle alla sua destra attribuiremo i numeri pari in senso crescente e alle celle alla sua sinistra i numeri dispari, sempre in senso crescente, come in figura:



In questo modo usiamo soltanto numeri naturali. Introduciamo due funzioni $D(x)$ e $S(x)$ (r.p.) che, data una cella x, restituiscono quella alla sua destra e quella alla sua sinistra. [†]

$$D(x) = \begin{cases} x+2 & \text{se } Pa\ x \\ 0 & \text{se } x = 1 \\ x \dot{-} 2 & \text{se } Di\ x \wedge x \neq 1 \end{cases}$$

$$S(x) = \begin{cases} x+2 & \text{se } Di\ x \\ 1 & \text{se } x = 0 \\ x \dot{-} 2 & \text{se } Pa\ x \wedge x \neq 0 \end{cases}$$

Sulla base di queste funzioni definiamo il predicato ricorsivo primitivo

$$Dxy \equiv (Pa\ x \wedge Pa\ y \wedge x > y) \vee (Pa\ x \wedge Di\ y) \vee (Di\ x \wedge Di\ y \wedge x < y),$$

[†]Nella definizione di $D(x)$ ed $S(x)$ vengono utilizzati due predicati ricorsivi primitivi, $Pa\ x$ e $Di\ x$, che significano rispettivamente " x è un numero pari" ed "x è un numero dispari".

che significa "la cella x è a destra della cella y". Infine, consideriamo la funzione $Z(x, y)$, che rappresenta il numero di celle comprese tra la x e la y (x compresa, y esclusa) alla condizione che y sia a sinistra di x.

$$Z(x,y) = \begin{cases} \frac{\dot{x-y}}{2} & \text{se } Pa\ x \wedge\ Pa\ y \\ \frac{\dot{y-x}}{2} & \text{se } Di\ x \wedge\ Di\ y \\ \frac{x+y+1}{2} & \text{altrimenti} \end{cases}$$

Numerazione delle espressioni del nastro. Ora che abbiamo numerato le celle del nastro, possiamo caratterizzare, sempre mediante dei numeri, anche le espressioni del nastro del calcolo con alfabeto $\{a_0, \ldots, a_N\}$: il simbolo a_1 è | e i numeri vengono rappresentati in unario (pag. 22). Può essere $N = 1$. Poniamo

$$b = \prod_{j=0}^{\infty} p_j^{\beta(j)},$$

supposto che $a_{\beta(j)}$ sia il simbolo contenuto nella cella j. Ricordiamo che ogni cella vuota contiene il simbolo a_0 e dunque, non influisce sul prodotto b, in quanto determina un fattore p_j^0 =1. Se il nastro è caratterizzato dall' espressione b, allora nella cella j è impresso il simbolo $a_{exp(j,b)}$.

Supponiamo ora di avere calcolato il valore di una data funzione e quindi di avere il nastro caratterizzato da una certa espressione b. Sia a l' ultima cella presa in esame, posizionata all' immediata destra della fine del valore w della funzione. Il valore w, rappresentato per mezzo di sbarre ($n + 1$ se $w = n$), viene determinato da a e b, secondo una funzione che studieremo $w = W_0(a, b)$. A tal proposito consideriamo le funzioni $E_s(a, b)$ ed $E_d(a, b)$: esse indicano le celle che si trovano rispettivamente all'estremità sinistra e destra del valore della funzione. Si ha che $E_d(a, b) = S(a)$, mentre

$$E_s(a,b) = \mu_{(x \leq b)}\big[exp(S(x), b) = 0 \wedge \forall y_{\leq max(S(x),a))}(DyS(x) \wedge Day \Rightarrow exp(y, b) = 1)\big],$$

cioè la cella alla sua sinistra è vuota, mentre ogni cella frapposta tra quella alla sua sinistra e la cella a è impressa col simbolo a_1 (che possiamo pensare anche come una sbarra |). Anch' esse sono r.p.. Per come abbiamo definito queste funzioni abbiamo che $W_0(a, b) = Z(E_d(a, b), E_s(a, b))$, cioè il valore

della funzione w è dato dal numero di celle comprese tra $E_s(a,b)$ ed $E_d(a,b)$. Allora, anche W_0 è r.p., in quanto definita tramite sostituzione di funzioni r.p..

Numero di Gödel t associato ad una macchina di Turing $\mathcal{M}$. Sia $\mathcal{M}$ una macchina definita da una tavola. Supponiamo che $\mathcal{M}$ abbia $m+1$ stati $0,\ldots,m$ $(m \geq 0)$ e che operi coi simboli $a_0,\ldots,a_N$. Descriviamo la procedura che permette di associare un numero t ad ogni macchina $\mathcal{M}$. Ricordiamo che nella terza e quarta colonna della tavola troviamo rispettivamente le istruzioni ed i nuovi stati. Basta conoscere il numero N ed i simboli delle ultime due colonne, dato che determinati gli stati e i simboli su cui opera la macchina possiamo introdurre le prime due colonne della relativa tavola senza difficoltà. Sostituiamo, poi, i simboli della terza colonna nel modo seguente:

$$s \to 1, \quad d \to 2, \quad f \to 3, \quad a_0 \to 4,\ldots,a_N \to N+4.$$

In questo modo otteniamo una matrice numerica A_{ij} di $(N+1)(m+1)$ righe $(i=1,\ldots,(N+1)(m+1))$ e 2 colonne $(j=3,4)$, che possiamo identificare col numero t, che chiameremo *numero di Gödel* di $\mathcal{M}$:

$$t = p_0^N p_1^m \prod_{i=1}^{(N+1)(m+1)} \prod_{j=3}^{4} p_{\sigma_2(i,j)}^{A_{ij}}.$$

Dato il numero t è inoltre possbile ricavare la tavola di $\mathcal{M}$. Innanzitutto, $\sigma_2(i,j) = 2^i(2j+1)\dot{-}1 > 1$ per $j=3,4$, allora

$$N = exp(0,t)$$
$$m = exp(1,t).$$

Inoltre $A_{ij} = exp(\sigma_2(i,j),t)$. Allora, le $N+1$ righe della tavola di macchina riguardanti lo stato c $(c=0,\ldots,m)$, saranno della forma:

$$\begin{array}{cccc} c & a_0 & exp(\sigma_2((N+1)c+1,3),t) & exp(\sigma_2((N+1)c+1,4),t) \\ \vdots & \vdots & \vdots & \vdots \\ c & a_N & exp(\sigma_2((N+1)c+N+1,3),t) & exp(\sigma_2((N+1)c+N+1,4),t). \end{array}$$

Abbreviamo gli elementi della terza e quarta colonna ponendo

$$h(p,q,c,t) = exp(\sigma_2((N+1)c+q+1,p),t),$$

dove $0 \leq q \leq N$. Allora la riga della tavola di $\mathcal{M}$ che inizia per ca_q è data da

$$(*) \qquad c \qquad a_q \qquad h(3,q,c,t) \qquad h(4,q,c,t)$$

La funzione h è r.p. in (p,q,c)[‡]. Vediamo, ora, come determinare, dato un numero t, se questo è o meno il numero di Gödel di una macchina di Turing. In primo luogo, si calcolano i numeri N ed m. Poi, si costruisce in base a $(*)$ una matrice $(m+1)(N+1) \times 4$ e si verifica se sono soddisfatte le condizioni:

1. $N \geq 1$
2. $1 \leq h(3,q,c,t) \leq N+4$
3. $h(4,q,c,t) \leq m$

Se non sono soddisfatte, allora t non è numero di Gödel di nessuna macchina di Turing. Invece, se sono soddisfatte, la matrice costruita definisce una macchina di Turing, di cui va calcolato il numero di Gödel t_0 e va verificato che $t = t_0$.

Funzioni A,B,C per le configurazioni successive. Sia t il numero di Gödel di una macchina di Turing e $< a,b,c >$ la configurazione codificata dalla cella in esame a, dall'espressione del nastro b e dallo stato c ($a,b,c \in \mathbb{N}$). Se la macchina non cessa di operare dopo tale configurazione, ne viene determinata una nuova, in funzione di t,a,b,c. Indicheremo, allora, con $A(t,a,b,c)$ la nuova cella in esame, con $B(t,a,b,c)$ la nuova espressione del nastro e con $C(t,a,b,c)$ il nuovo stato. Vediamo nel dettaglio le tre funzioni A,B,C. La cella presa in esame inizialmente contiene il simbolo $a_{exp(a,b)} = a_q$ (dalla definizione di b). Quindi la riga decisiva per il passo successivo, in base a $(*)$ è

$$c \quad a_{exp(a,b)} \quad h(3,q,c,t) \quad h(4,q,c,t)$$

[‡]Nel sèguito t si considera un parametro e non una variabile naturale di funzioni.

e dunque il nuovo stato è dato da

$$C(t,a,b,c) = h(4, exp(a,b), c, t).$$

La nuova cella in esame sarà quella alla sinistra, o alla destra di quella precedente o rimarrà invariata:

$$A(t,a,b,c) = \begin{cases} S(a) & \text{se } \ h(3, exp(a,b), c, t) = 1 \\ D(a) & \text{se } \ h(3, exp(a,b), c, t) = 2 \\ a & \text{altrimenti} \end{cases}$$

La nuova espressione $B(t,a,b,c)$ è diversa da quella precedente solo se viene impresso un nuovo simbolo nella cella in esame a e l' alterazione di questa è resa dalla moltiplicazione di b per p_a elevato all' indice del nuovo simbolo aumentato di 4 (poichè si era stabilita la corrispondenza $a_0 \to 4, \ldots, a_N \to N+4$) e dalla divisione per p_a elevato all' indice del vecchio simbolo che viene sovrascritto:

$$B(t,a,b,c) = \begin{cases} \dfrac{b \cdot p_a^h}{p_a^{exp(a,b)}} & \text{se } \ 4 \leq h \\ b & \text{se } \ 4 > h \end{cases}$$

dove $h = h(3, exp(a,b), c, t)$.
Dalle definizioni appena date, segue che le tre funzioni A, B, C sono r.p..
Sia, ora, t il numero di Gödel di una macchina di Turing. Definiamo la relazione E_0tabc, la quale afferma che la configurazione $< a, b, c >$ è terminale per $\mathcal{M}$. Questo è possibile se e solo se la riga che inizia per $ca_{exp(a,b)}$ ha per terzo elemento il simbolo f, che, come abbiamo detto sopra, va rappresentato col numero 3, quindi, si ha che E_0tabc è dato dalla relazione:

$$h(3, exp(a,b), c, t) = 3.$$

Allora E_0 è anch' esso r.p..

Numeri di Gödel di configurazioni e relazioni: Sia $\mathcal{M}$ una macchina di Turing e $< a, b, c >$ una sua configurazione. Definiamo il numero di Gödel di tale configurazione come $k = \sigma_3(a,b,c) = \sigma_2(\sigma_2(a,b), c)$. Nel séguito

usiamo inoltre:
$\sigma_{31}(z) = \sigma_{21}(\sigma_{21}(z))$; $\sigma_{32}(z) = \sigma_{22}(\sigma_{21}(z))$; $\sigma_{33}(z) = \sigma_{22}(z)$;
$\sigma_{21}(z) = exp(0, z+1)$; $\sigma_{22}(z) = \frac{l}{2}$, dove $l = \frac{z+1}{2^{exp(0,z+1)}} \dot{-} 1$.
Sono tutte funzioni r.p., "inverse" di k:

$$\begin{aligned} a &= \sigma_{31}(k) \\ b &= \sigma_{32}(k) \\ c &= \sigma_{33}(k). \end{aligned}$$

Consideriamo la relazione che indichiamo con Etk, dove t è il numero di Gödel di una macchina di Turing e k è il numero di Gödel di una configurazione terminale della macchina il cui numero è t. Per come abbiamo definito la relazione E_0, abbiamo che

$$Etk = E_0 t \sigma_{31}(k) \sigma_{32}(k) \sigma_{33}(k).$$

Ora assumiamo che la configurazione identificata dal numero k abbia una configurazione consecutiva, il cui numero di Gödel sarà dato da una funzione

$$\begin{aligned} F(t,k) = \sigma_3\big(&A(t, \sigma_{31}(k), \sigma_{32}(k), \sigma_{33}(k)), \\ &B(t, \sigma_{31}(k), \sigma_{32}(k), \sigma_{33}(k)), \\ &C(t, \sigma_{31}(k), \sigma_{32}(k), \sigma_{33}(k))\big). \end{aligned}$$

Infine, sia k il numero di una configurazione in cui la cella in esame sia vuota e si trovi immediatamente a destra di una sequenza di sbarre, che rappresenta il valore di una funzione, prima di cui vi sia una cella vuota. Tale valore è dato da

$$W(k) = W_0(\sigma_{31}(k), \sigma_{32}(k)) = W_0(a, b)$$

È evidente che sia la relazione E, che le funzioni F e W sono r.p..

La funzione $K(t,{}^n x, z)$. Sia $\mathcal{M}$ una macchina di Turing e t il suo numero di Gödel. Sia ${}^n x$ una n-upla di argomenti. Scriviamo ${}^n x$ sul nastro (altrimenti vuoto) e scegliamo come cella 0 quella che precede gli argomenti. La cella in esame nella configurazione iniziale sia quella che segue gli argomenti. Otteniamo una sequenza di configurazioni che può eventualmente

concludersi con un ultimo termine. I numeri di Gödel di tali configurazioni formano anch'essi una sequenza, che descriveremo con la funzione $K(t,^n x, z)$, $z = 0, 1, 2, \ldots$. Se dopo un certo passo z_0 si giunge alla configurazione terminale, allora il valore di $K(t,^n x, z)$ è determinato solo fino a $z \leq z_0$ e per $z > z_0$ poniamo $K(t,^n x, z) = K(t,^n x, z_0)$.
Si dimostra che la funzione K gode di alcune proprietà:

1) $K(t,^n x, z) = K(t,^n x, z') \Rightarrow K(t,^n x, z) = K(t,^n x, w),\ \forall w \geq z.$

2) Come conseguenza, se $\mathcal{M}$ cessa di operare dopo z_0 passi, allora
$z_0 = \mu z K(t,^n x, z) = K(t,^n x, z').$

Verifichiamo che $K(t,^n x, z)$ è r.p., ricavandone una sua definizione tramite lo schema di ricorsione semplice. $K(t,^n x, 0)$ è il numero di Gödel della configurazione iniziale. In questo caso $c = 0$ e l'espressione del nastro iniziale consiste dell' argomento ${}^n x$. Le celle a destra di quella col numero 0 sono tutte individuate da numeri pari. Allora l'espressione iniziale del nastro, che chiamiamo $b_0({}^n x)$, è data da [§]

$$\frac{\prod_{j=0}^{x_1+\ldots+x_n+2n} p(2j)}{p(2(x_1+\ldots+x_n+2n))p\big(2(x_1+\ldots+x_{n-1}+2(n-1))\big)\ldots p(2(x_1+2))p(0)}$$

La cella in esame nella configurazione iniziale ha il numero

$$a_0({}^n x) = 2(x_1 + \ldots + x_n + 2).$$

Se invece $n = 0$ poniamo $a_0({}^n x) = 0$ e $b_0({}^n x) = 1$.
Quindi, per $z = 0$ abbiamo

$$K(t,^n x, 0) = \sigma_3\big(a_0({}^n x), b_0({}^n x), 0\big) \qquad (\diamondsuit)$$

Inoltre, per quanto già visto in precedenza, abbiamo che se vale $EtK(t,^n x, z)$ (cioè se $K(t,^n x, z)$ è il numero di una configurazione terminale), allora $K(t,^n x, z) =$

[§]Per chiarezza, $p(l)$ indica l' l-esimo numero primo p_l.

$K(t,^n x, z')$, invece se la configurazione non è terminale, $K(t,^n x, z') = F\big(K(t,^n x, z)\big)$. Per abbreviare queste scritture, introduciamo una funzione r.p. φ, tale che

$$\varphi(t,y) = \begin{cases} y & \text{se } Ety \\ F(t,y) & \text{altrimenti} \end{cases}$$

Allora, abbiamo che la configurazione successiva sarà identificata dal numero

$$K(t,^n x, z') = \varphi\big(t, K(t,^n x, z)\big) \qquad (\diamondsuit\diamondsuit)$$

Dunque, dalle equazioni $(\diamondsuit)$ e $(\diamondsuit\diamondsuit)$ possiamo concludere che K è r.p..

Definizione 2.8. Il predicato $T_n t\ ^n xy$ ($(n+2)$-ario) è definito da

$$K\big(t,^n x, \sigma_{21}(y)\big) = K\big(t,^n x, (\sigma_{21}(y))'\big) \wedge \sigma_{22}(y) = K\big(t,^n x, \sigma_{21}(y)\big)$$

Per come è definito, esso risulta essere r.p.. Vediamo altre tre proprietà che caratterizzano K e T_n:

3) Se esiste un indice z tale che $K(t,^n x, z) = K(t,^n x, z')$, allora esiste un y tale che $T_n t\ ^n x, y$

4) Se esiste un y tale che $T_n t\ ^n xy$, allora esiste anche uno z tale che $K(t,^n x, z) = K(t,^n x, z')$ e precisamente $z = \sigma_{21}(y)$.

5) Sia t il numero di Gödel di una macchina di Turing $\mathcal{M}$. Se $\mathcal{M}$ cessa di operare dopo z_0 passi e se vale il predicato $T_n t\ ^n xz$, allora $\sigma_{22}(y) = K(t,^n x, z_0)$.

Grazie alla numerazione delle celle, del nastro per il calcolo, delle macchine e delle configurazioni, tramite le funzioni ed i predicati r.p. che abbiamo introdotto, possiamo ora affrontare la dimostrazione del

Teorema 2.5.2. *Esistono una funzione r.p. unaria U e, per ogni n, un predicato r.p. T_n, $(n+2)$-ario, tali che per ogni funzione n-aria Turing-computabile f esiste un numero k tale che*

1. *per ogni $^n x$ esiste un y tale che $T_n k\ ^n xy$*

2. $f({}^n x) = U(\mu y T_n k\ {}^n xy)$ *per ogni* ${}^n x$.

k *è il numero di Gödel di una macchina che calcola* f.

La ricorsività di f, che è Turing-computabile per ipotesi, segue dal punto 2, in cui viene applicato il μ-operatore nel caso regolare (condizione 1).

Dimostrazione. Sia f una funzione n-aria calcolabile tramite una macchina di Turing $\mathcal{M}$, che ha numero di Gödel t. Daremo una rappresentazione di f mediante il predicato T_n: sia z_0 il numero di passi dopo cui $\mathcal{M}$, applicata dopo gli argomenti ${}^n x$, cessa di operare. Per la proprietà **3)** esiste un y tale che vale $T_n t\ {}^n xy$. Per cui, applicando lo schema di minimo a T_n, $(\mu y T_n t\ {}^n xy)$, si ha un' applicazione del μ-operatore ad una funzione regolare. Evidentemente vale $T_n t\ {}^n x(\mu y T_n t\ {}^n xy)$ (cioè T_n vale anche per il minimo y per cui vale) e da 5) segue che $\sigma_{22}(\mu y T_n t\ {}^n xy) = K(t, {}^n x, z_0) =$ il numero di Gödel della configurazione terminale di $\mathcal{M}$. Allora, in base ai risultati precedenti, abbiamo che il valore della funzione f è dato da

$$f({}^n x) = W(K(t, {}^n x, z_0)) = W(\sigma_{22}(\mu y T_n t\ {}^n xy)).$$

Infine, introduciamo una funzione ricorsiva primitiva U, così definita:

$$U(u) = W(\sigma_{22}(u))$$

e per $u = \mu y T_n t\ {}^n xy$ otteniamo la relazione finale

$$f({}^n x) = U(\mu y T_n t\ {}^n xy).$$

Con $k = t$ numero di Gödel di $\mathcal{M}$.
Abbiamo così dimostrato che ogni funzione Turing-computabile è ricorsiva: nell'ultima relazione che abbiamo scritto, il μ-operatore viene applicato ad un predicato ricorsivo ed essendo U r.p. per ipotesi, segue la ricorsività di f. Dunque è dimostrato il teorema 2.5.2 [Hermes 1975 Cap.4, §18]. □

2.5.4 Macchina universale

Un risultato di vasta portata è inoltre quello dell' esistenza di una macchina di Turing universale $\mathcal{U}$, in grado di simulare l' operato di una qualunque macchina di Turing $\mathcal{M}$. Sia come prima $\{a_0, a_1, \ldots, a_N\}$ l' alfabeto di $\mathcal{M}$, vedremo che esiste una tale macchina $\mathcal{U}$ sull' alfabeto $\{a_0, a_1\} = \{*, |\}$.

Definizione 2.9. Siano B_0 una qualsiasi espressione finita del nastro e A_0 una sua qualsiasi cella. Applicando $\mathcal{M}$ a B_0 su A_0, otteniamo la configurazione iniziale $< A_0, B_0, C_0 >$, con $C_0 = c_{\mathcal{M}}$. Poi, $\mathcal{M}$ si troverà in altre successive configurazioni: $< A_1, B_1, C_1 >, < A_2, B_2, C_2 >, \ldots, < A_n, B_n, C_n >, \ldots$ e qualora cessi di operare dopo un numero finito di passi, si avrà una certa configurazione terminale $< A_{n_0}, B_{n_0}, C_{n_0} >$. Ora, applichiamo la macchina $\mathcal{U}$, il cui scopo è simulare $\mathcal{M}$, sull' espressione del nastro $t * k_0$, dove t è il numero di Gödel della macchina $\mathcal{M}$ e k_0 è il numero di Gödel della configurazione iniziale di $\mathcal{M} < A_0, B_0, C_0 >$. Ciò significa che dapprima va scritta sul nastro una sequenza di $t + 1$ sbarre per rappresentare il numero t, poi, separate da uno spazio, altre $k_0 + 1$ sbarre per rappresentare il numero k_0. Per la macchina $\mathcal{U}$ la cella inizialmente in esame è quella immediatamente a destra dell'ultima cella segnata. Se $C_0^{\mathcal{U}}$ è lo stato iniziale di $\mathcal{U}$, allora la configurazione iniziale di $\mathcal{U}$ è $< A_0^U, B_0^U, C_0^U >$. Torniamo a considerare la macchina $\mathcal{M}$ e supponiamo che dalla 0-esima configurazione $< A_0, B_0, C_0 >$ passi alla prima configurazione $< A_1, B_1, C_1 >$, allora $\mathcal{U}$ deve raggiungere, compiendo un numero finito di passi $r_1 \geq 1$, una configurazione $< A_{r_1}^U, B_{r_1}^U, C_{r_1}^U >$, dove la cella in esame sia quella immediatamente a destra dell'ultima cella segnata e l'espressione del nastro sia $t * k_1$, dove k_1 è il numero di Gödel della prima configurazione di $\mathcal{M}$. Diremo che la r_1-esima configurazione di $\mathcal{U}$ corrisponde alla prima configurazione di $\mathcal{M}$. Se $\mathcal{M}$ compie un altro passo e giunge alla configurazione $< A_2, B_2, C_2 >$, allora deve esistere una configurazione corrispondente per $\mathcal{U} < A_{r_2}^U, B_{r_2}^U, C_{r_2}^U >$ con $r_2 > r_1$ e così via. Se $\mathcal{M}$ dopo un numero finito di passi giunge ad una configurazione terminale $< A_{n_0}, B_{n_0}, C_{n_0} >$, allora anche $\mathcal{U}$ deve raggiungere una configurazione terminale corrispondente $< A_{r_{n_0}}^U, B_{r_{n_0}}^U, C_{r_{n_0}}^U >$. Nel caso particolare in cui si

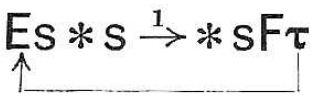
avesse che $< A_0, B_0, C_0 >$ fosse già la configurazione terminale di $\mathcal{M}$, cioè $n_0 = 0$, allora si avrebbe $r_{n_0} = r_0$. Una macchina $\mathcal{U}$ che opera verificando queste condizioni è detta *macchina di Turing universale*.

Riassumiamo in una tabella quanto detto finora:

Macchina $\mathcal{M}$	Macchina universale $\mathcal{U}$	
$< A_0, B_0, C_0 >$	$< A_0^U, B_0^U, C_0^U >$	$B_0^U = t * k_0$ e A_0^U=cella a destra di k_0
	$\vdots$	$\vdots$
$< A_1, B_1, C_1 >$	$< A_{r_1}^U, B_{r_1}^U, C_{r_1}^U >$	$B_{r_1}^U = t * k_1$ e $A_{r_1}^U$ =cella a destra di k_1
	$\vdots$	$\vdots$
$< A_2, B_2, C_2 >$	$< A_{r_2}^U, B_{r_2}^U, C_{r_2}^U >$	$B_{r_2}^U = t * k_2$ e $A_{r_2}^U$ =cella a destra di k_2
.....	.....	.....

Diamo ora una ulteriore descrizione della macchina universale $\mathcal{U}$ mediante un diagramma di flusso, ottenuto dalla combinazione di più macchine elementari. A tale scopo introduciamo il predicato ricorsivo primitivo $E'tk$: esso vale se e solo se t è il numero di Gödel di una macchina di Turing e k è il numero di Gödel della configurazione terminale di tale macchina. Dato che è r.p., per definizione esiste una funzione e, anch'essa r.p., che assume soltanto i valori 0 e 1 a seconda che esso valga o meno, per cui si avrà $E'tk \equiv e(t,k) = 0$. Allora, per il teorema 2.5.1, esiste una macchina di Turing $\mathcal{E}$ che calcola e in modo standard. Analogamente, dalla ricorsività della funzione $F(t,k)$ introdotta nella sezione 2.5.3, segue l'esistenza di una macchina di Turing $\mathcal{F}$ che la calcola in modo standard.

Sia $\mathcal{M}$ la macchina che $\mathcal{U}$ deve simulare. Se applichiamo $\mathcal{U}$ sulla cella immediatamente dopo $t*k_0$, bisogna verificare che $\mathcal{M}$ compia almeno un passo. Ciò non si verifica se e solo se vale $E'tk_0$, ossia $e(t,k_0) = 0$. Il diagramma della macchina universale $\mathcal{U}$ è il seguente [Hermes, p.270]:



Si inizia con l' applicazione di $\mathcal{E}$, che come abbiamo detto sopra, calcola il valore di $e(t, k_0)$. Sul nastro avremo allora

$$\ldots t * k_0 * e(t, k_0)\underline{*} \ldots$$

Proseguendo mediante l' applicazione di $s * s$ cancelliamo l'ultima sbarra impressa e ci spostiamo a sinistra di un'ulteriore cella. Secondo che sia $e(t, k_0) = 0$ oppure $e(t, k_0) = 1$, la nuova cella in esame sarà rispettivamente vuota o segnata. Se ci poniamo nel caso in cui $\mathcal{M}$ non compie alcun passo, cioè $e(t, k_0) = 0$, il calcolo è già giunto al termine. Se invece $e(t, k_0) = 1$, cioè $\mathcal{M}$ compie almeno un passo, tramite $\overset{1}{\rightarrow} *s$ completiamo la cancellazione del valore di $e(t, k_0)$ e riotteniamo un nastro con l'espressione

$$\ldots t * k_0\underline{*} \ldots$$

Dopodiché, bisogna calcolare $k_1 = F(t, k_0)$, il numero della prima configurazione di $\mathcal{M}$ e sovrascriverlo a k_0. Per fare ciò, dapprima, tramite $\mathcal{F}$ si ottiene l' iscrizione del nastro

$$\ldots t * k_0 * k_1\underline{*} \ldots$$

e mediante la macchina di traslazione τ si sovrascrive k_0 con k_1:

$$\ldots t * k_1\underline{*} \ldots$$

Da qui si ripete lo stesso procedimento, verificando se k_1 è il numero di Gödel di una configurazione terminale (tornando ad applicare $\mathcal{E}$), ecc. Una volta ottenuto il numero k_{n_0} della configurazione terminale di $\mathcal{M}$, il valore della funzione da calcolare, è dato da $f(^n x) = W\big(K(t,^n x, n_0)\big)$ (teorema 2.5.2).

Capitolo 3

Algoritmi di Markov

Introduciamo, ora, un'altra definizione rigorosa del concetto di computabilità effettiva: quella data dagli algoritmi di Markov.

3.1 Cenni biografici su Markov [8]

Figura 3.1: Andrej Andreevič Markov

Gli algoritmi di Markov prendono il loro nome dal matematico sovietico Andrej Andreevič Markov (San Pietroburgo, 22 settembre 1903 - Mosca, 11 ottobre 1979), figlio dell' omonimo Andrej Andreevič Markov (Rjazan, 14

giugno 1856 - San Pietroburgo, 20 luglio 1922) il quale, invece, è conosciuto universalmente per i suoi importanti contributi alla teoria della probabilità (catene di Markov e processi markoviani). Markov jr., fin da giovane, nutrì un profondo interesse per la chimica che approfondì dapprima presso la scuola di Matematica e Fisica di S. Pietroburgo, poi all'università di Leningrado. Già a partire dal 1920 prese parte ad alcune ricerche che lo portarono alla pubblicazione del suo primo articolo nel 1924, anno in cui si laureò in fisica presso l' università di Leningrado, dove in séguito fu professore (1933-1955). Dopo le sue prime pubblicazioni, dedicò i suoi studi al problema dei tre corpi, ai sistemi dinamici ed alla meccanica quantistica (1926-1937). Dopo la seconda guerra mondiale rivolse i suoi interessi alla teoria assiomatica degli insiemi, alla logica ed ai fondamenti della matematica, che lo portarono ad essere tra i fondatori della Scuola Russa di Matematica costruttiva e Logica; in séguito diede importanti contributi alla teoria degli algoritmi, ai quali il suo nome è associato. Un rilevante risultato che ottenne in questo campo, nel 1947, fu la dimostrazione dell' indecidibilità del problema della parola per semigruppi, risolto sempre nello stesso periodo, ma indipendentemente, anche da Emil Post. Risale, invece, al 1954 la pubblicazione di *The Theory of Algorithms*, in cui sono esposti alcuni degli argomenti a cui fa riferimento questa tesi. Nel 1960, mostrò che la classificazione delle varietà 4-dimensionali è indecidibile (non esiste un algoritmo generale che distingua due varietà qualsiasi di dimensione superiore a 4). Dal 1959 fino all' anno della sua morte fu direttore del Dipartimento di Logica matematica presso l' Università di Mosca.

3.2 Definizioni preliminari

Daremo, ora, alcune definizioni utili per inquadrare l' argomento. La seguente esposizione è tratta da [Mendelson 1972, Cap.5, §1].
Il formalismo degli algoritmi di Markov prevede l' uso di linguaggi in cui ogni espressione è costituita da una sequenza di simboli.

Definizione 3.1. Si definisce *alfabeto* un insieme finito non vuoto di simboli.

Generalmente si utilizzano linguaggi con un numero finito di simboli, tra cui va incluso anche lo *spazio vuoto*, usato di solito per separare le parole l' una dall' altra. (Infatti, anche partendo da un alfabeto con infiniti simboli $a_1, a_2, \ldots$, ci si può sempre ricondurre ad uno con due soli simboli $\{b, c\}$, sostituendo all' elemento generico a_n, $b\underbrace{cc\ldots cc}_{n\ volte}b$).

Definizione 3.2. Una *parola* in un alfabeto A è data da una sequenza finita di simboli di A. La sequenza vuota di simboli viene detta *parola vuota* e si indica con Λ.

Definizione 3.3. Se $P = a_{j_1} \ldots a_{j_k}$ e $Q = a_{r_1} \ldots a_{r_m}$ sono due parole in un alfabeto A, chiamiamo $PQ = a_{j_1} \ldots a_{j_k} a_{r_1} \ldots a_{r_m}$ la *giustapposizione* delle due parole. In particolare, vale $\Lambda P = P\Lambda = P$ e la giustapposizione è associativa: $(P_1P_2)P_3 = P_1(P_2P_3)$.

Definizione 3.4. Siano dati due alfabeti A e B, si dice che B è l' *estensione* di A se e solo se $A \subseteq B$. Ne segue che se B è estensione di A, ogni parola di A è una parola di B.

Definizione 3.5. Si definisce *algoritmo in* un alfabeto A una funzione effettivamente computabile $\mathfrak{A}$ il cui dominio è un sottoinsieme dell' insieme delle parole di A e i cui valori sono parole in A. Se P è una parola in A, $\mathfrak{A}$ si dice *applicabile* a P se P è nel dominio di $\mathfrak{A}$.
Si definisce *algoritmo sopra* un alfabeto A un algoritmo $\mathfrak{A}$ in un' estensione B di A.
I passi di un algoritmo non possono essere eseguiti ad arbitrio ma in un ordine prestabilito.

Definizione 3.6. Si dice che una parola T *occorre* in una parola Q se esistono due parole U, V (eventualmente vuote) tali che $Q = UTV$.

Descriviamo ora le *produzioni*, che sono l' unità fondamentale di costruzione degli algoritmi.

Definizione 3.7. Se P e Q sono parole di un alfabeto A, allora $P \to Q$ e $P \to \cdot Q$ vengono dette rispettivamente *produzione semplice* e *produzione terminale* nell' alfabeto A, dove " $\to$" e "$\cdot$" non fanno parte di A.

Definizione 3.8. Se $P \to (\cdot)Q$ sta per $P \to Q$ o $P \to \cdot Q$, allora uno *schema d' algoritmo* è dato da una lista finita di produzioni in A

$$\begin{aligned} P_1 &\to (\cdot)Q_1 \\ P_2 &\to (\cdot)Q_2 \\ &\vdots \\ P_r &\to (\cdot)Q_r. \end{aligned}$$

Data una parola P in A,

1. se nessuna delle parole $P_1, \ldots, P_r$ presenti nello schema dell' algoritmo occorre in P scriveremo $\mathfrak{A} : P\square$;

2. se m è il più piccolo numero intero, $1 \leq m \leq r$, tale che P_m occorra in P e se R è la parola che risulta dal rimpiazzamento dell' occorrenza più a sinistra di P_m in P con Q_m allora scriveremo:

   a) $\mathfrak{A} : P \vdash R$ se $P_m \to (\cdot)Q_m$ è semplice, e diremo che $\mathfrak{A}$ trasforma semplicemente P in R;

   b) $\mathfrak{A} : P \vdash \cdot R$ se $P_m \to (\cdot)Q_m$ è terminale, e diremo che $\mathfrak{A}$ trasforma in modo terminale P in R.

3.3 Algoritmi di Markov ed alcuni esempi

Definizione 3.9. (Algoritmo di Markov) Scriveremo $\mathfrak{A} : P \models R$ ("l' algoritmo $\mathfrak{A}$ da P *finisce* in R") quando esiste una sequenza $R_0, \ldots, R_k$ tale che

$$P = R_0,$$

$$R = R_k,$$

$$\mathfrak{A} : R_j \vdash R_{j+1} \quad \text{se } \ 0 \leq j \leq k-2,$$

e, o si ha

$$\mathfrak{A} : R_{k-1} \vdash R_k,$$

oppure

$$\mathfrak{A} : R_{k-1} \vdash \cdot R_k$$

Nel secondo caso scriviamo $\mathfrak{A} : P \models \cdot R$ ("l'algoritmo $\mathfrak{A}$ da P *termina* in R"). Si pone $\mathfrak{A}(P) = R$ se e solo se $\mathfrak{A} : P \models \cdot R$, oppure $\mathfrak{A} : P \models R$ e $R\square$.
Per convenzione, per segnalare l' input P e l' output R (risultato dell' applicazione dell' algoritmo), all' inizio e alla fine di queste due parole, si scrive uno spazio vuoto: $\sqcup$. Tuttavia, per non appesantire la notazione, lo ometteremo. L' algoritmo così definito è chiamato *algoritmo di Markov* o *algoritmo normale* nell' alfabeto A.

Descriviamo, quindi, come si esplica, data una parola P, l' applicazione di un suddetto algoritmo:

- si cerca la *prima* produzione $P_m \to (\cdot)Q_m$ nello schema tale che P_m occorra in P;
- si sostituisce Q_m al posto dell' occorrenza più a sinistra di P_m in P; sia R_1 la nuova parola ottenuta tramite tale sostituzione;
- se $P_m \to (\cdot)Q_m$ è una produzione terminale, il processo termina e il valore dell' algoritmo è R_1;
- se $P_m \to (\cdot)Q_m$ è semplice, si applica lo stesso processo ad R_1 come era stato applicato a P e così via;
- nel corso dell' applicazione di tale processo, nel caso in cui si ottenga una parola R_i che non contiene alcuna occorrenza di P_m per $1 \leq m \leq r$, il processo finisce ed il valore di $\mathfrak{A}$ è $R_i\square$;
- se si ottiene $\cdot R_i$ il processo *termina* in R_i.

È possibile che il processo descritto non abbia mai termine e dunque si entri in una situazione di *loop*: in questo caso $\mathfrak{A}$ non risulta applicabile alla parola P.

Per familiarizzarci con gli algoritmi di Markov presentiamo alcuni primi semplici esempi.

Esempio 3.1 (Cancellazione di un simbolo)**.** Sia A l' alfabeto $\{b, c\}$, allora

$$b \to \cdot\Lambda$$
$$c \to c$$

è lo schema di un algoritmo normale che trasforma ogni parola contenente almeno un'occorrenza di b nella parola ottenuta cancellando l' occorrenza più a sinistra di b. $\mathfrak{A}$ trasforma Λ in se stessa, ma non è applicabile a parole non vuote che non contengono occorrenze di b.

Esempio 3.2 (Parola vuota)**.** Sia A l' alfabeto $\{a_0, a_1, \ldots, a_n\}$, allora

$$a_0 \to \Lambda$$
$$a_1 \to \Lambda$$
$$\vdots$$
$$a_n \to \Lambda$$

è lo schema di un algoritmo normale che trasforma ogni parola nella parola vuota. Esso si può abbreviare con $\xi \to \Lambda$ (ξ in A).
Se $P = a_1a_2a_1a_3a_0$, l' applicazione di $\mathfrak{A}$ avviene nel modo seguente:

$$\mathfrak{A} : a_1a_2a_1a_3a_0 \vdash a_1a_2a_1a_3 \vdash a_2a_1a_3 \vdash a_2a_3 \vdash a_3 \vdash \Lambda$$

e se $P = \Lambda$

$$\mathfrak{A} : \Lambda\Box.$$

Esempio 3.3 (Successore)**.** Per ogni numero naturale n, rappresentiamo il numerale $\overline{n}$ nel modo seguente: $\overline{0} = |$ e $\overline{n+1} = \overline{n}\,|$. Dunque $\overline{1} =||$, $\overline{2} =|||$ ecc.
Sia A l' alfabeto $\{|\}$. Lo schema $\Lambda \to \cdot\,|$ definisce un algoritmo normale $\mathfrak{A}$, tale che per ogni parola P in A, $\mathfrak{A}(P) = |\,P$, infatti Λ occorre all' inizio di ogni parola P, dato che, come avevamo già visto $P = \Lambda P$. Allora, in

particolare per ogni numero naturale n, l' algoritmo $\mathfrak{A}$ calcola il successivo: $\mathfrak{A}(\overline{n}) = \overline{n+1}$:

$$\mathfrak{A} :| \underbrace{| \ldots |}_{n-1} | \vdash | \underbrace{| \ldots |}_{n} |$$

Esempio 3.4 (Zero)**.** Siano H=$\{|\}$ ed M=$\{|, \alpha\}$ alfabeti. Ogni numero naturale è rappresentato dal suo numerale $\overline{n}$, che è una parola in H. Allora lo schema:

$$\begin{gathered} \alpha \, ||\rightarrow \alpha \, | \\ \alpha \, |\rightarrow \cdot \, | \\ \Lambda \rightarrow \alpha \end{gathered}$$

definisce un algoritmo normale $\mathfrak{A}$ che ad ogni numerale $\overline{n}$ associa $\overline{0}$, per esempio $\mathfrak{A}(\overline{3}) = \overline{0}$: $|||| \vdash \alpha \; |||| \vdash \alpha \; ||| \vdash \alpha \; || \vdash \alpha \; | \vdash |$.

D'ora in poi useremo 1 al posto di |: ciò non dovrebbe causare confusione.

Esempio 3.5 (Proiezione)**.** Sia M=$\{1, *\}$ un alfabeto, $\alpha_1, \ldots, \alpha_{2k}$ simboli non in M. Sia $\mathcal{S}_i$, $0 \leq i \leq k-1$, la sequenza di produzioni:

$$\begin{gathered} \alpha_{2i-1} * \rightarrow \alpha_{2i-1} * \\ \alpha_{2i-1} 1 \rightarrow \alpha_{2i} 1 \\ \alpha_{2i} 1 \rightarrow \alpha_{2i} \\ \alpha_{2i} * \rightarrow \alpha_{2i+1}. \end{gathered}$$

Sia $1 \leq j \leq k$, allora a seconda del valore di j si hanno tre schemi d' algoritmo distinti [Mendelson, p.258]:

$1 < j < k$	$j = 1$	$j = k$
$\mathcal{S}_1$	$\alpha_1* \to \alpha_1*$	$\mathcal{S}_1$
$\vdots$	$\alpha_1 1 \to \alpha_2 1$	$\vdots$
$\mathcal{S}_{j-1}$	$\alpha_2 1 \to 1\alpha_2$	$\mathcal{S}_{k-1}$
$\alpha_{2j-1}* \to \alpha_{2j-1}*$	$\alpha_2* \to \alpha_3$	$\alpha_{2k-1}* \to \alpha_{2k-1}*$
$\alpha_{2j-1}1 \to \alpha_{2j}1$	$\mathcal{S}_2$	$\alpha_{2k-1}1 \to \alpha_{2k}1$
$\alpha_{2j}1 \to 1\alpha_{2j}$	$\vdots$	$\alpha_{2k}1 \to 1\alpha_{2k}$
$\alpha_{2j}* \to \alpha_{2j+1}$	$\mathcal{S}_{k-1}$	$\alpha_{2k}* \to \alpha_{2k}*$
$\mathcal{S}_{j+1}$	$\alpha_{2k-1}* \to \alpha_{2k-1}*$	$\alpha_{2k}* \to \alpha_{2k}*$
$\vdots$	$\alpha_{2k-1}1 \to \alpha_{2k}1$	$\alpha_{2k} \to \cdot\Lambda$
$\mathcal{S}_{k-1}$	$\alpha_{2k}1 \to \alpha_{2k}$	$\Lambda \to \alpha_1.$
$\alpha_{2k-1}* \to \alpha_{2k-1}*$	$\alpha_{2k}* \to \alpha_{2k}*$	
$\alpha_{2k-1}1 \to \alpha_{2k}1$	$\alpha_{2k} \to \cdot\Lambda$	
$\alpha_{2k}1 \to \alpha_{2k}$	$\Lambda \to \alpha_1.$	
$\alpha_{2k}* \to \alpha_{2k}*$		
$\alpha_{2k} \to \cdot\Lambda$		
$\Lambda \to \alpha_1.$		

Questi schemi danno un algoritmo $\mathfrak{A}_j^k$ tale che $\mathfrak{A}_j^k(\overline{(n_1, \ldots, n_k)}) = \overline{n_j}$, dove $\overline{(n_1, \ldots, n_k)} = \overline{n_1} * \overline{n_2} * \ldots * \overline{n_k}$.

3.4 Computabilità secondo Markov

Definizione 3.10. Siano $\mathfrak{A}$ e $\mathfrak{B}$ due algoritmi, allora scriveremo $\mathfrak{A}(P) \approx \mathfrak{B}(P)$ se e solo se:

- o $\mathfrak{A}$ e $\mathfrak{B}$ sono entrambi applicabili a P e $\mathfrak{A}(P) = \mathfrak{B}(P)$
- o né $\mathfrak{A}$ né $\mathfrak{B}$ sono applicabili a P.

Definizione 3.11. Siano $\mathfrak{A}$ e $\mathfrak{B}$ due algoritmi sopra un alfabeto A, diremo che $\mathfrak{A}$ e $\mathfrak{B}$ sono *equivalenti* rispetto ad A $\Leftrightarrow \forall P \in A$, se $\mathfrak{A}(P)$ o $\mathfrak{B}(P)$ esiste ed è in A, allora si ha $\mathfrak{A}(P) \approx \mathfrak{B}(P)$.

Definizione 3.12. Siano $\mathfrak{A}$ e $\mathfrak{B}$ due algoritmi sopra un alfabeto A, diremo che $\mathfrak{A}$ e $\mathfrak{B}$ sono *pienamente equivalenti* rispetto ad A $\Leftrightarrow \forall P \in A$, $\mathfrak{A}(P) \approx \mathfrak{B}(P)$.

Definizione 3.13. Consideriamo l' alfabeto M= $\{1, *\}$, l' insieme $\mathbb{N}$ dei numeri naturali e una funzione $\varphi : \mathbb{N}^k \to \mathbb{N}, k \geq 1$, di cui $\mathfrak{B}_\varphi$ è il corrispettivo algoritmo nell' alfabeto M, ossia $\mathfrak{B}_\varphi(\overline{(n_1, \ldots, n_k)}) = \overline{\varphi(n_1, \ldots, n_k)}$. $\mathfrak{B}_\varphi$ sia applicabile solo a k-uple di numerali $\overline{(n_1, \ldots, n_k)}$. Allora si dice che la funzione φ è *computabile secondo Markov* o *Markov-computabile* se e solo se esiste un algoritmo di Markov normale $\mathfrak{A}$ sopra M, pienamente equivalente a $\mathfrak{B}_\varphi$ rispetto ad M.

Definizione 3.14. Un algoritmo di Markov $\mathfrak{A}$ si dice *chiuso* se e solo se nel relativo schema una delle produzioni è della forma $\Lambda \to \cdot Q$. In tal caso $\mathfrak{A}$ finisce soltanto con l' applicazione di una produzione terminale.

Dato un qualunque algoritmo normale, se alle produzioni che costituiscono il suo schema aggiungiamo (per ultima) la produzione $\Lambda \to \cdot\Lambda$, otteniamo un nuovo algoritmo normale che denotiamo con $\mathfrak{A}\cdot$, che è pienamente equivalente ad $\mathfrak{A}$ rispetto all' alfabeto di $\mathfrak{A}$.

Proposizione 3.4.1. *Siano $\mathfrak{A}, \mathfrak{B}$ algoritmi normali in un alfabeto A, allora la loro composizione è ancora un algoritmo normale.*

Dimostrazione. Per ogni simbolo b dell' alfabeto A sia $\overline{b}$ un nuovo simbolo ad esso correlato, in modo da formare l' insieme dei correlati $\overline{A}$. Siano α, β altri due simboli, che non appartengono all' alfabeto A$\cup\overline{A}$, sia $\mathcal{S}_\mathfrak{A}$ lo schema di $\mathfrak{A}\cdot$, con l' accorgimento che ogni punto terminale che compare nelle relative produzioni sia rimpiazzato da α. Analogamente sia $\mathcal{S}_\mathfrak{B}$ lo schema di $\mathfrak{B}\cdot$, con l' accorgimento che ogni simbolo di A sia rimpiazzato dal suo correlato, ogni punto terminale da β, le produzioni della forma $\Lambda \to Q$ da $\alpha \to \alpha Q$ e quelle della forma $\Lambda \to \cdot Q$ da $\alpha \to \alpha\beta Q$. Allora si vede che, per $a, b \in$A, lo schema:

$$a\alpha \to \alpha a$$
$$\alpha a \to \alpha \overline{a}$$

$$\overline{a}b \to \overline{a}\overline{b}$$
$$\overline{a}\beta \to \beta\overline{a}$$
$$\beta\overline{a} \to \beta a$$
$$a\overline{b} \to ab$$
$$\alpha\beta \to \cdot\Lambda$$
$$\mathcal{S}_{\mathfrak{B}}$$
$$\mathcal{S}_{\mathfrak{A}}$$

determina un algoritmo di Markov $\mathfrak{C} = \mathfrak{B} \circ \mathfrak{A}$ sopra A, che è la composizione di $\mathfrak{A}$ e $\mathfrak{B}$ ed è tale che per ogni parola $P \in$A, $\mathfrak{C}(P) \approx \mathfrak{B}(\mathfrak{A}(P))$. □

Definizione 3.15 (Propagazione)**.** Sia $\mathfrak{D}$ un algoritmo normale in un alfabeto A, sia B$\supseteq$A una estensione di A. Se alle produzioni che formano lo schema di $\mathfrak{D}$ anteponiamo produzioni della forma $b \to b$ per ciascun simbolo $b \in$B\A, otteniamo uno schema di un nuovo algoritmo normale $\mathfrak{D}_{\mathfrak{B}}$ in B, che è detto *propagazione* di $\mathfrak{D}$ su B ed è tale che $\forall P \in$A $\mathfrak{D}_{\mathfrak{B}}(P) \approx \mathfrak{D}(P)$. Si ha che $\mathfrak{D}_{\mathfrak{B}}$ non è applicabile a parole in B che contengono simboli di B\A ed è pienamente equivalente a $\mathfrak{D}$ rispetto ad A.

Definizione 3.16 (Composizione)**.** Siano $\mathfrak{A}, \mathfrak{B}$ algoritmi normali rispettivamente in un alfabeto A_1 e A_2, con $A = A_1 \cup A_2$. Siano $\mathfrak{A}_A$ e $\mathfrak{B}_A$ le propagazioni di $\mathfrak{A}$ e $\mathfrak{B}$ sopra A. Allora la composizione $\mathfrak{C}$ di $\mathfrak{A}_A$ e $\mathfrak{B}_A$ viene definita *composizione normale* di $\mathfrak{A}$ e $\mathfrak{B}$. $\mathfrak{C}$ è applicabile solo alle parole P di A, con P parola di A_1, $\mathfrak{A}$ è applicabile a P e $\mathfrak{B}$ è applicabile ad $\mathfrak{A}(P)$.

Definizione 3.17. (Proiezione) Sia A un alfabeto, B una sua estensione. Data una parola P in B, si definisce *proiezione* P^A di P su A la parola che si ottiene da P, cancellando tutti i simboli in B\A, mediante l' *algoritmo di proiezione* $\mathfrak{P}$ su A, il cui schema è dato da: $\xi \to \Lambda$, dove $\xi \in B \setminus A$ e quindi $\mathfrak{P}(P) = P^A, \forall P \in B$.

Definizione 3.18 (Estensione)**.** Sia $\mathfrak{A}$ un algoritmo normale in un alfabeto A, sia B una estensione di A. Allora si definisce *estensione naturale* $\mathfrak{B}$ dell' algoritmo $\mathfrak{A}$ a B, l' algoritmo normale in B determinato dallo stesso schema

di $\mathfrak{A}$. Si ha che $\mathfrak{B}(P) \approx \mathfrak{A}(P), \forall P \in$A, inoltre, per $P \in$A e $Q \in$B\A, $\mathfrak{B}(PQ) \approx \mathfrak{A}(P)Q$. $\mathfrak{B}$ si indica con $\mathfrak{A}^\sharp$.

3.5 Equivalenza tra funzioni Markov-computabili e ricorsive

Ci apprestiamo a dimostrare l' equivalenza fra ricorsività e computabilità secondo Markov, restringendoci al caso di funzioni computabili totali, che vedremo corrisponde al caso in cui $\mathfrak{A}$ è applicabile ad ogni k-upla di numerali. Tuttavia, a questo scopo, ci serviremo anche di funzioni parziali.

3.5.1 Markov-computabilità delle funzioni ricorsive

Proposizione 3.5.1 (Mendelson, p.262)**.** *Siano* $\mathfrak{A}_1, \ldots, \mathfrak{A}_k$ *algoritmi normali, rispettivamente negli alfabeti* $A_1, \ldots, A_k$*. Sia* $A{=}A_1 \cup \ldots \cup A_k$*. Allora esiste un algoritmo di Markov* $\mathfrak{B}$ *sopra* A*, tale che* $\mathfrak{B}(P) \approx \mathfrak{A}_1^\sharp(P)\mathfrak{A}_2^\sharp(P)\ldots\mathfrak{A}^\sharp{}_k(P)$ $\forall P \in A$*, dove* $\mathfrak{A}_i^\sharp(P)$ *è l' estensione naturale di* $\mathfrak{A}_i$ *ad* A*.*
Esso è detto giustapposizione *degli algoritmi* $\mathfrak{A}_1, \ldots, \mathfrak{A}_k$*.*

Corollario 3.5.2. *Siano* $\mathfrak{A}_1, \ldots, \mathfrak{A}_k$ *algoritmi normali, rispettivamente negli alfabeti* $A_1, \ldots, A_k$*. Sia* $A{=}A_1 \cup \ldots \cup A_k$*. Allora esiste un algoritmo di Markov* $\mathfrak{B}$ *sopra* $A \cup \{*\}$ *tale che* $\mathfrak{B}(P) \approx \mathfrak{A}^\sharp{}_1(P) * \mathfrak{A}^\sharp{}_2(P) * \ldots * \mathfrak{A}^\sharp{}_k(P)$ $\forall P \in A$*.*

Dimostrazione. Consideriamo lo schema

$$\begin{aligned} a &\to \Lambda \quad (a \in A) \\ \Lambda &\to \cdot * \end{aligned}$$

Esso definisce un algoritmo normale $\mathfrak{D}$ in A$\cup\{*\}$ tale che $\mathfrak{D}(P) = *, \forall P \in$A$\cup\{*\}$.
Facendo riferimento alla proposizione precedente, sia $\mathfrak{B}$ la giustapposizione (sopra A$\cup\{*\}$) di $\mathfrak{A}_1, \mathfrak{D}, \mathfrak{A}_2, \mathfrak{D}, \ldots, \mathfrak{D}, \mathfrak{A}_k$.
Allora $\mathfrak{B}(P) \approx \mathfrak{A}_1^\sharp(P) * \mathfrak{A}_2^\sharp(P) * \ldots * \mathfrak{A}_k^\sharp(P)$, $\forall P \in$A$\cup\{*\}$. □

Lemma 3.5.3. ***a)*** *Sia* $\mathfrak{C}$ *un algoritmo normale in un alfabeto* A *ed* α *un simbolo qualsiasi. Allora esiste un algoritmo normale* $\mathfrak{D}$ *sopra* $A \cup \{\alpha\}$ *tale che*

$$\mathfrak{D}(P) = \begin{cases} \alpha P & \text{se } P \in A \wedge \mathfrak{C}(P) = \Lambda \\ P & \text{se } P \in A \wedge \mathfrak{C}(P) \neq \Lambda \end{cases}$$

ed è applicabile soltanto alle parole a cui si applica $\mathfrak{C}$*.*

b) *Siano* $\mathfrak{A}$ *e* $\mathfrak{B}$ *algoritmi normali nell' alfabeto* A*,* α *un simbolo non in* A*. Allora esiste un algoritmo normale* $\mathfrak{E}$ *sopra* $A \cup \{\alpha\}$ *tale che se* $P \in A$

$$\mathfrak{E}(P) \approx \mathfrak{A}(P)$$

$$\mathfrak{E}(\alpha P) \approx \mathfrak{B}(P)$$

Dimostrazione. **a)** Sia β un simbolo non in A$\cup\{\alpha\}$. Costruiamo un algoritmo normale $\mathfrak{D}_1$ sopra A$\cup\{\alpha\}$:

$$\begin{array}{ll} a \to \beta & a \in A \cup \{\alpha\} \\ \beta\beta \to \beta & \\ \beta \to \cdot\Lambda & \\ \Lambda \to \cdot\alpha & \end{array}$$

che trasforma ogni parola dell' alfabeto A$\cup\{*\}$ nella parola vuota e Λ in α.
Poniamo $\mathfrak{D}_2 = \mathfrak{D}_1 \circ \mathfrak{C}$.
Per ogni $P \in A$, se $\mathfrak{C}(P) = \Lambda$, allora, per come abbiamo definito $\mathfrak{D}_1$, $\mathfrak{D}_2(P) = \mathfrak{D}_1(\Lambda) = \alpha$. Se invece $\mathfrak{C}(P) \neq \Lambda$, allora $\mathfrak{D}_2(P) = \Lambda$.
Sia $\mathfrak{I}$ l' algoritmo identico in A, il cui schema è dato da $\Lambda \to \cdot\Lambda$ e sia $\mathfrak{D}$ la giustapposizione di $\mathfrak{D}_2$ ed $\mathfrak{I}$: $\mathfrak{D}(P) \approx \mathfrak{D}_2^\sharp(P)\mathfrak{I}^\sharp(P)$.
Ora:

- se $\mathfrak{C}(P) = \Lambda$, allora $\mathfrak{D}(P) = \mathfrak{D}_2^\sharp(P)\mathfrak{I}^\sharp(P) = \alpha P$
- se $\mathfrak{C}(P) \neq \Lambda$, allora $\mathfrak{D}(P) = \mathfrak{D}_2^\sharp(P)\mathfrak{I}^\sharp(P) = \Lambda P = P$,

per tutte le parole a cui si applica $\mathfrak{C}$.
b) Facciamo corrispondere ad ogni simbolo $a \in$A un simbolo $\overline{a}$ e sia $\overline{A}$ l' alfabeto costituito da questi ultimi simboli.

Poniamo B=A$\cup\overline{A}\cup\{\alpha,\beta\}$, dove $\beta\notin$A$\cup\overline{A}\cup\{\alpha\}$.
Costruiamo un nuovo schema di algoritmo $\mathcal{S}_{\overline{\mathfrak{B}}}$ ottenuto dallo schema dell' algoritmo $\mathfrak{B}\cdot$($\mathfrak{B}$ è dato inizialmente e gli si aggiunge come ultima produzione $\Lambda\to\cdot\Lambda$) rimpiazzando ogni simbolo a con il corrispondente $\overline{a}$, sostituendo tutti i punti terminali con β, le produzioni della forma $\Lambda\to Q$ con $\alpha\to\alpha Q$ e le produzioni $\Lambda\to\cdot Q$ con $\alpha\to\alpha\beta Q$. Sia, inoltre, $\mathcal{S}_{\mathfrak{A}\cdot}$ lo schema di $\mathfrak{A}\cdot$.
Infine, formiamo l' ulteriore schema:

$$\begin{array}{ll}\alpha a\to\alpha\overline{a} & a,b\in\text{A}\\ \overline{a}b\to\overline{a}\overline{b}&\\ \overline{a}\beta\to\beta\overline{a}&\\ \beta\overline{a}\to\beta a&\\ a\overline{b}\to ab&\\ \alpha\beta\to\cdot\Lambda&\\ \mathcal{S}_{\overline{\mathfrak{B}}}&\\ \mathcal{S}_{\mathfrak{A}\cdot}\ .&\end{array}$$

Esso determina un nuovo algoritmo di Markov $\mathfrak{E}$ sopra A$\cup\{\alpha\}$ tale che per ogni parola $P\in$A:

- $\mathfrak{E}(P)\approx\mathfrak{A}(P)$, infatti, le uniche produzioni applicabili a P in questo caso sono quelle che compaiono in $\mathcal{S}_{\mathfrak{A}\cdot}$, che è equivalente ad $\mathcal{S}_{\mathfrak{A}}$.

- $\mathfrak{E}(\alpha P)\approx\mathfrak{B}(P)$, infatti, in questo caso vengono applicate inizialmente le produzioni di $\mathfrak{E}$ che precedono $\mathcal{S}_{\overline{\mathfrak{B}}}$ e trasformano ogni simbolo a in $\overline{a}$ poi, quelle di $\mathcal{S}_{\overline{\mathfrak{B}}}$ e, per finire, alcune delle produzioni che precedono $\mathcal{S}_{\overline{\mathfrak{B}}}$, che trasformano ogni simbolo $\overline{a}$ in a, cosicché si ottiene come risultato proprio $\mathfrak{B}(P)$.

□

Proposizione 3.5.4. *Siano $\mathfrak{A},\mathfrak{B},\mathfrak{C}$ algoritmi normali ed A l' unione dei rispettivi alfabeti. Allora esiste un algoritmo normale $\mathfrak{F}$ sopra A, chiamato*

la ramificazione *di* $\mathfrak{A}$ *e* $\mathfrak{B}$ *regolata da* $\mathfrak{C}$*, tale che*

$$\mathfrak{F}(P) \approx \begin{cases} \mathfrak{B}(P) & se\ P \in A \wedge \mathfrak{C}(P) = \Lambda \\ \mathfrak{A}(P) & se\ P \in A \wedge \mathfrak{C}(P) \neq \Lambda \end{cases}$$

ed $\mathfrak{F}$ *si applica solo alle parole di A a cui è applicabile* $\mathfrak{C}$*.*

Dimostrazione. Siano $\mathfrak{A}_1, \mathfrak{B}_1, \mathfrak{C}_1$ le propagazioni di $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$ ad A ed α un simbolo non in A.
Applicando la parte *a*) del Lemma 3.5.3 a $\mathfrak{C}_1$, che è un algoritmo su A, esiste un altro algoritmo normale $\mathfrak{D}$ sopra A$\cup\{\alpha\}$ tale che

$$\mathfrak{D}(P) = \begin{cases} \alpha P & se\ P \in A \wedge \mathfrak{C}(P) = \Lambda \\ P & se\ P \in A \wedge \mathfrak{C}(P) \neq \Lambda \end{cases}$$

e $\mathfrak{D}$ si applica solo a quelle parole a cui si applica $\mathfrak{C}$.
Dalla parte *b*) del Lemma 3.5.3, applicato ad $\mathfrak{A}_1$ e $\mathfrak{B}_1$, discende l' esistenza di un algoritmo normale $\mathfrak{E}$ sopra A$\cup\{\alpha\}$ tale che per ogni $P \in$A:

$$\mathfrak{E}(P) \approx \mathfrak{A}_1(P) \approx \mathfrak{A}(P)$$

$$\mathfrak{E}(\alpha P) \approx \mathfrak{B}_1(P) \approx \mathfrak{B}(P).$$

Allora $\mathfrak{F} = \mathfrak{E} \circ \mathfrak{D}$. □

Definizione 3.19 (Iterazione)**.** Siano $\mathfrak{A}$ e $\mathfrak{C}$ algoritmi in un alfabeto A e sia P_0 una qualsiasi parola in A. Se dall' applicazione di $\mathfrak{A}$ a P_0 risulta una parola P_1, applichiamo $\mathfrak{C}$ a P_1. Ora, se $\mathfrak{C}(P_1) = \Lambda$ ci fermiamo, altrimenti, se $\mathfrak{C}(P_1) \neq \Lambda$, applichiamo $\mathfrak{A}$ a P_1. Se $\mathfrak{A}(P_1) = P_2$, proviamo ad applicare $\mathfrak{C}$ a P_2. Anche questa volta, possiamo avere o $\mathfrak{C}(P_2) = \Lambda$ (nel qual caso ci fermiamo) o $\mathfrak{C}(P_2) \neq \Lambda$, allora applichiamo $\mathfrak{A}$ a P_2, e si continua analogamente, in modo da ottenere un algoritmo $\mathfrak{B}$, che viene definito *iterazione* di $\mathfrak{A}$ regolata da $\mathfrak{C}$.
Dunque, $\mathfrak{B}(P_0) = Q \Leftrightarrow$ esiste una sequenza di parole $P_0, P_1, \ldots, P_n$ $(n > 0)$ con $P_n = Q$, $\mathfrak{C}(P_n) = \Lambda$, $P_i = \mathfrak{A}(P_{i-1})$ per $1 < i \leq n$ e $\mathfrak{C}(P_i) \neq \Lambda$ per $0 < i < n$.

Proposizione 3.5.5. *Siano $\mathfrak{A}$ e $\mathfrak{C}$ due algoritmi di Markov, sia A l'unione dei loro alfabeti e siano $\mathfrak{A}_1$ e $\mathfrak{C}_1$ le propagazioni di $\mathfrak{A}$ e $\mathfrak{C}$ ad A. Allora la iterazione di $\mathfrak{A}_1$ regolata da $\mathfrak{C}_1$ è un algoritmo normale sopra A.*

Dimostrazione. Non è restrittivo dimostrare il risultato per $\mathfrak{A}$ e $\mathfrak{C}$ nello stesso alfabeto. Allora abbiamo $\mathfrak{A}_1 = \mathfrak{A}$ e $\mathfrak{C}_1 = \mathfrak{C}$.
Consideriamo α, simbolo non in A. Applicando la prima parte del Lemma 3.5.3 a $\mathfrak{C}$, è garantita l' esistenza di un algoritmo normale $\mathfrak{D}$ sopra B=A$\cup\{\alpha\}$ tale che, come prima in 3.5.4:

$$\mathfrak{D}(P) = \begin{cases} \alpha P & se\ P \in A \wedge \mathfrak{C}(P) = \Lambda \\ P & se\ P \in A \wedge \mathfrak{C}(P) \neq \Lambda. \end{cases}$$

Poniamo $\mathfrak{F} = \mathfrak{D} \circ \mathfrak{A}$. $\mathfrak{F}$ è un algoritmo normale (perché composizione di algoritmi normali) in un' estensione F di B.
Sia β un simbolo non in F. Definiamo un nuovo algoritmo di Markov $\mathfrak{M}$, tramite lo schema:

$$\begin{gathered} \xi\beta \to \beta\xi, \quad \xi \in \mathrm{F} \\ \beta\alpha \to \cdot\alpha \\ \beta \to \Lambda \\ \mathcal{S}_{\mathfrak{F}^\beta} \end{gathered}$$

dove $\mathcal{S}_{\mathfrak{F}^\beta}$ è lo schema di $\mathfrak{F}\cdot$, in cui tutti i punti terminali sono sostituiti da β. $\mathfrak{M}$ è tale che $\mathfrak{M}(P_0) = Q \Leftrightarrow$ esiste una sequenza di parole $P_0, P_1, \ldots, P_n$ con $P_n = Q$, $P_i = \mathfrak{F}(P_{i-1})$ per $1 < i \leq n$ e P_n è l'unica parola tra $P_0, P_1, \ldots, P_n$ che inizi con α, cioè P_n è la prima parola per cui $\mathfrak{C}$ applicato ad essa restituisca la parola vuota.
I punti terminali in $\mathfrak{F}\cdot$ sono rimpiazzati da β in modo che $\mathfrak{F}$ produca parole che contengano il simbolo β e si possano, poi, applicare le produzioni di $\mathfrak{M}$ che precedono $\mathcal{S}_{\mathfrak{F}^\beta}$.
L' applicazione a P_n di $\mathfrak{D}$ e delle produzioni che precedono $\mathcal{S}_{\mathfrak{F}^\beta}$ nello schema, introduce il simbolo α nel risultato dato da $\mathfrak{M}$. Per cancellare le occorrenze di α, componiamo $\mathfrak{M}$ con l' algoritmo di proiezione $\mathfrak{P}$ di F su F$\backslash\{\alpha\}$. Allora $\mathfrak{B} = \mathfrak{P} \circ \mathfrak{M}$ è l' iterazione di $\mathfrak{A}$ regolata da $\mathfrak{C}$ ed è normale sopra A. □

Corollario 3.5.6. *Siano $\mathfrak{A}$ e $\mathfrak{C}$ due algoritmi di Markov, sia A l' unione dei loro alfabeti. Allora esiste un algoritmo normale $\mathfrak{R}$ sopra A tale che per qualsiasi parola P_0 in A, $\mathfrak{R}(P_0) = Q \Leftrightarrow$ esiste una sequenza di parole $P_0, P_1, \ldots, P_n$ $(n \geq 0)$, $P_n = Q$, $\mathfrak{C}(P_n) = \Lambda$, $P_{i+1} = \mathfrak{A}(P_i)$ e $\mathfrak{C}(P_i) \neq \Lambda$ per $0 \leq i < n$. L' algoritmo $\mathfrak{R}$ viene detto l'* iterazione piena *di $\mathfrak{A}$ regolata da $\mathfrak{C}$.*

Dimostrazione. Consideriamo l' algoritmo identico $\mathfrak{I}$ e l' iterazione $\mathfrak{B}$ di $\mathfrak{A}$ regolata da $\mathfrak{C}$. Allora $\mathfrak{R}$ è dato dalla ramificazione di $\mathfrak{B}$ ed $\mathfrak{I}$ regolata da $\mathfrak{C}$:

$$\mathfrak{R}(P) \approx \begin{cases} \mathfrak{I}(P) & se\ P \in A \wedge \mathfrak{C}(P) = \Lambda \\ \mathfrak{B}(P) & se\ P \in A \wedge \mathfrak{C}(P) \neq \Lambda. \end{cases}$$

□

Proposizione 3.5.7. *Sia $\mathfrak{A}$ un algoritmo normale in un alfabeto A. Allora esiste un algoritmo $\mathfrak{A}^I$ sopra l' alfabeto B=A∪M (dove M=$\{1, *\}$) tale che, per ogni parola $P_0 \in A$ e per ogni numero $n \in \mathbb{N}$ si abbia $\mathfrak{A}^I(\overline{n} * P_0) = Q$ se e solo se esiste una sequenza $P_0, \ldots, P_n (n \geq 0)$ con $P_n = Q$ e $P_i = \mathfrak{A}(P_{i-1})$ per $0 < i \leq n$.*

Dimostrazione. Fissiamo α, simbolo non in B. Sia C=B∪$\{\alpha\}$.
Consideriamo i seguenti algoritmi normali in C:

- $\mathfrak{R}_1$, il cui schema è dato da

$$\begin{cases} \alpha 11 \to \cdot 1 \\ \alpha 1 * \to \alpha * \\ \alpha * \xi \to \alpha * \qquad (\xi \in B) \\ \Lambda \to \alpha. \end{cases}$$

  Si vede che $\mathfrak{R}_1(\overline{0} * P) = \Lambda$ e $\mathfrak{R}_1(\overline{n} * P) \neq \Lambda$, per $n > 0$ e per ogni $P \in$B.

- $\mathfrak{R}_2$, il cui schema è dato da

$$\begin{cases} * \xi \to * \qquad (\xi \in B) \\ * \to \Lambda. \end{cases}$$

  Si vede che per ogni parola $P \in$B, che non contiene $*$, $\mathfrak{R}_2(P * Q) = P$.

- $\mathfrak{R}_3$, il cui schema è dato da

$$\begin{cases} \alpha 1 \to \alpha \\ \alpha * \to \cdot\Lambda \\ \Lambda \to \alpha. \end{cases}$$

  Si vede che $\mathfrak{R}_3(\overline{n} * P) = P$, per ogni $P \in$B.

- $\mathfrak{R}_4$, il cui schema è dato da

$$1 \to \cdot\Lambda.$$

  Si vede che $\mathfrak{R}_4(\overline{n} * P) = \overline{(n-1)} * P$, per $n > 0$ e $\mathfrak{R}_4(\overline{0} * P) = *P$, per ogni $P \in$B.

- $\mathfrak{R}_5$, il cui schema è dato da

$$1* \to \cdot\Lambda.$$

  Si vede che $\mathfrak{R}_5(\overline{0} * P) = P$, per ogni $P \in$B.

Ora, sia $\mathfrak{B}$ l' algoritmo normale sopra C ottenuto come giustapposizione di $\mathfrak{R}_2 \circ \mathfrak{R}_4$ e $\mathfrak{A} \circ \mathfrak{R}_3$ (Corollario 3.5.2): $\mathfrak{B}(P) = (\mathfrak{R}_2 \circ \mathfrak{R}_4)(P) * (\mathfrak{A} \circ \mathfrak{R}_3)(P)$, per ogni parola $P \in$C.
Si ha che per ogni parola $P \in$A

$$\mathfrak{B}(\overline{n} * P) \approx \begin{cases} \overline{n-1} * \mathfrak{A}(P) & se\ n > 0 \\ *\mathfrak{A}(P) & se\ n = 0 \end{cases}$$

Sia E l' alfabeto di $\mathfrak{B}$. Applicando il corollario 3.5.6 agli algoritmi $\mathfrak{B}$ e $\mathfrak{R}_1$, allora esiste un algoritmo $\mathfrak{F}$ normale sopra E, tale che $\mathfrak{F}(P_0) = Q \Leftrightarrow$ esiste una sequenza $P_0, \ldots, P_k$ $(k \geq 0)$ con $P_k = Q$, $\mathfrak{R}_1(P_k) = \Lambda$, $P_i = \mathfrak{B}(P_{i-1})$ per $0 < i \leq k$ e $\mathfrak{R}_1(P_i) \neq \Lambda$ per $0 \leq i < k$.
Infine, componendo $\mathfrak{R}_5$ con $\mathfrak{F}$, si ottiene l' algoritmo normale richiesto $\mathfrak{A}^I = \mathfrak{R}_5 \circ \mathfrak{F}$. □

Proposizione 3.5.8. *Ogni funzione ricorsiva è computabile nel senso di Markov.*

Dimostrazione. 1) Le **funzioni iniziali** (successore, zero, proiezione), come abbiamo visto negli esempi 3.3, 3.4, 3.5, sono computabili tramite algoritmi normali.
2) **Sostituzione**: sia $\psi(x_1,\ldots,x_n) = \tau(\varphi_1(x_1,\ldots,x_n),\ldots,\varphi_k(x_1,\ldots,x_n))$ lo schema di sostituzione, dove $\tau,\varphi_1,\ldots,\varphi_k$ sono ricorsive.
Per ipotesi esistano algoritmi di Markov $\mathfrak{A}_\tau,\mathfrak{A}_{\varphi_1},\ldots,\mathfrak{A}_{\varphi_k}$ sopra M=$\{1,*\}$ che calcolano le funzioni $\tau,\varphi_1,\ldots,\varphi_k$.
Per il corollario 3.5.2 esiste un algoritmo normale $\mathfrak{B}$ sopra M tale che

$$\mathfrak{B}(P) \approx \mathfrak{A}_{\varphi_1}(P) * \mathfrak{A}_{\varphi_2}(P) * \ldots * \mathfrak{A}_{\varphi_k}(P)$$

per ogni parola P in M.
In particolare, per $P = \overline{(x_1,\ldots,x_n)}$, dove $x_i \in \mathbb{N}$ si ha:

$$\mathfrak{B}(\overline{(x_1,\ldots,x_n)}) \approx \overline{\varphi_1(x_1,\ldots,x_n)} * \overline{\varphi_2(x_1,\ldots,x_n)} * \ldots * \overline{\varphi_k(x_1,\ldots,x_n)}.$$

Consideriamo ora $\mathfrak{C} = \mathfrak{A}_\tau \circ \mathfrak{B}$: esso è un algoritmo normale ed è tale che

$$\mathfrak{C}(\overline{(x_1,\ldots,x_n)}) \approx \mathfrak{A}_\tau(\overline{\varphi_1(x_1,\ldots,x_n)} * \overline{\varphi_2(x_1,\ldots,x_n)} * \ldots * \overline{\varphi_k(x_1,\ldots,x_n)}) \approx$$

$$\approx \overline{\tau(\varphi_1(x_1,\ldots,x_n),\ldots,\varphi_k(x_1,\ldots,x_n))},$$

per ogni numero naturale $x_1,\ldots,x_n$. Dunque, abbiamo provato che ψ è computabile secondo Markov.
3) **Ricorsione** Sia ψ definita per ricorsione semplice:

$$\begin{cases} \psi(x_1,\ldots,x_k,0) = \tau(x_1,\ldots,x_k) \\ \psi(x_1,\ldots,x_k,y+1) = \varphi(x_1,\ldots,x_k,y,\psi(x_1,\ldots,x_k,y)) \end{cases}$$

con τ,φ ricorsive e $\mathfrak{A}_\tau$ e $\mathfrak{A}_\varphi$ i corrispondenti algoritmi normali sopra M. Sia $\mathfrak{A}_Z$ l' algoritmo normale che calcola la funzione zero, $\mathfrak{A}_N$ l' algoritmo normale che calcola la funzione successore e $\mathfrak{A}_j^k$ l' algoritmo normale che calcola la funzione di proiezione U_j^k. Applicando il corollario 3.5.2 agli algoritmi $\mathfrak{A}_i^{k+1}$,

per $i = 1, 2, \ldots, k$ è garantita l' esistenza di un nuovo algoritmo normale $\mathfrak{B}_1$ sopra M tale che $\mathfrak{B}_1(\overline{x_1} * \ldots * \overline{x_k} * \overline{y}) = \overline{x_1} * \ldots * \overline{x_k}$.
Sia $\mathfrak{T} = \mathfrak{A}_\tau \circ \mathfrak{B}_1$. Applicando nuovamente il corollario 3.5.2 a $\mathfrak{A}_{k+1}^{k+1}, \mathfrak{A}_1^{k+1}, \ldots, \mathfrak{A}_k^{k+1}$, $\mathfrak{A}_Z, \mathfrak{T}$, esiste un algoritmo normale di giustapposizione $\mathfrak{B}_2$ sopra M tale che $\mathfrak{B}_2(\overline{x_1} * \ldots * \overline{x_k} * \overline{y}) \approx \overline{y} * \overline{x_1} * \ldots * \overline{x_k} * \overline{0} * \overline{\tau(x_1, \ldots, x_n)}$.
Sia $\mathfrak{B}_3 = \mathfrak{A}_N \circ \mathfrak{A}_{k+1}^{k+2}$. Esso è tale che $\mathfrak{B}_3(\overline{x_1} * \ldots * \overline{x_k} * \overline{y} * \overline{x}) = \overline{y+1}$.
Applicando il corollario 3.5.2 a $\mathfrak{A}_1^{k+2}, \ldots, \mathfrak{A}_k^{k+2}, \mathfrak{B}_3, \mathfrak{A}_\varphi$, otteniamo un algoritmo normale $\mathfrak{B}_4$ sopra M di giustapposizione tale che
$\mathfrak{B}_4(\overline{x_1} * \ldots * \overline{x_k} * \overline{y} * \overline{z}) \approx \overline{x_1} * \ldots * \overline{x_k} * \overline{y+1} * \overline{\varphi(x_1, \ldots, x_k, y, x)}$.
Applichiamo ora la proposizione 3.5.7 a $\mathfrak{B}_4$: allora esiste un algoritmo di Markov $\mathfrak{B}_4^I$ sopra M tale che se $n \geq 0$, $\mathfrak{B}_4^I(\overline{n} * P_0) = Q \Leftrightarrow$ esiste una sequenza di parole $P_0, \ldots, P_n$ tale che $Q = P_n$ e $P_i = \mathfrak{B}_4(P_{i-1})$, per $0 < i \leq n$.
Allora l' algoritmo (sopra M) $\mathfrak{B} = \mathfrak{A}_{k+2}^{k+2} \circ \mathfrak{B}_4^I \circ \mathfrak{B}_2$ è normale (perché composizione di algoritmi normali) e calcola ψ. Infatti, si applica dapprima $\mathfrak{B}_2(\overline{x_1} * \ldots * \overline{x_k} * \overline{y}) \approx \overline{y} * \overline{x_1} * \ldots * \overline{x_k} * \overline{0} * \overline{\tau(x_1, \ldots, x_n)}$ poi $\mathfrak{B}_4^I$ a $\overline{y} * \overline{x_1} * \ldots * \overline{x_k} * \overline{0} * \overline{\tau(x_1, \ldots, x_n)}$. Esso produce una ripetizione per y volte di $\mathfrak{B}_4$, iniziando con $\overline{x_1} * \ldots * \overline{x_k} * \overline{0} * \overline{\tau(x_1, \ldots, x_k)}$:

$$\mathfrak{B}_4(\overline{x_1} * \ldots * \overline{x_k} * \overline{0} * \overline{\tau(x_1, \ldots, x_k)}) = \overline{x_1} * \ldots * \overline{x_k} * \overline{1} * \overline{\psi(x_1, \ldots, x_k, 1)}$$

$$\mathfrak{B}_4(\overline{x_1} * \ldots * \overline{x_k} * \overline{1} * \overline{\tau(x_1, \ldots, x_k)}) = \overline{x_1} * \ldots * \overline{x_k} * \overline{2} * \overline{\psi(x_1, \ldots, x_k, 2)}$$

$$\vdots$$

$$\mathfrak{B}_4(\overline{x_1} * \ldots * \overline{x_k} * \overline{y-1} * \overline{\tau(x_1, \ldots, x_k)}) = \overline{x_1} * \ldots * \overline{x_k} * \overline{y} * \overline{\psi(x_1, \ldots, x_k, y)}.$$

Infine, applicando $\mathfrak{A}_{k+2}^{k+2}$, cioé proiettando sull' ultima componente, si ottiene $\overline{\psi(x_1, \ldots, x_k, y)}$.

4) **μ-operatore (nel caso regolare).**

Supponiamo che $\psi(x_1, \ldots, x_n) = \mu y(\varphi(x_1, \ldots, x_n, y) = 0)$, dove φ è computabile secondo Markov ed esiste un y tale che $\varphi(x_1, \ldots, x_n, y) = 0$ per ogni $(x_1, \ldots, x_n)$. Vogliamo vedere che anche ψ è computabile secondo Markov. Sia $\mathfrak{A}_\varphi$ l' algoritmo normale sopra M che calcola φ. Per il corollario 3.5.2 applicato agli algoritmi $\mathfrak{A}_1^{n+1}, \ldots, \mathfrak{A}_n^{n+1}, \mathfrak{A}_N \circ \mathfrak{A}_{n+1}^{n+1}$, esiste un algoritmo normale

$\mathfrak{W}$ sopra M tale che $\mathfrak{W}(\overline{x_1} * \ldots * \overline{x_n} * \overline{y}) = \overline{x_1} * \ldots * \overline{x_n} * \overline{y+1}$.
Sia $\mathfrak{D}$ l' algoritmo normale sopra M dato dallo schema

$$11 \to \cdot 11$$
$$1 \to \Lambda.$$

Allora

$$\mathfrak{D}(\overline{n}) \begin{cases} = \Lambda & se\ n = 0 \\ \neq \Lambda & se\ n > 0 \end{cases}$$

Sia $\mathfrak{C} = \mathfrak{D} \circ \mathfrak{A}_\varphi$, allora

$$\mathfrak{C}(\overline{x_1} * \ldots * \overline{x_n} * \overline{y}) \begin{cases} = \Lambda & se\ \varphi(x_1, \ldots, x_n, y) = 0 \\ \neq \Lambda & se\ \varphi(x_1, \ldots, x_n, y) \neq 0. \end{cases}$$

Sia $\mathfrak{Z}$ un nuovo algoritmo normale sopra M tale che

$$\mathfrak{Z}(\overline{x_1} * \ldots * \overline{x_n}) = \overline{x_1} * \ldots * \overline{x_n} * \overline{0}.$$

Per il corollario 3.5.6 applicato a $\mathfrak{W}$ e $\mathfrak{C}$, esiste un algoritmo normale $\mathfrak{N}$ sopra M tale che $\mathfrak{N}(P_0) = Q \Leftrightarrow$ esiste una sequenza $P_0, \ldots, P_n$ $(n \geq 0)$ tale che $P_n = Q$, $\mathfrak{C}(P_n) = \Lambda$, $P_{i+1} = \mathfrak{W}(P_i)$ e $\mathfrak{C}(P_i) \neq \Lambda$ per $0 \leq i < n$.
Sia $\mathfrak{B} = \mathfrak{A}_{n+1}^{n+1} \circ \mathfrak{N} \circ \mathfrak{Z}$. Allora

$$\mathfrak{B}(\overline{x_1} * \ldots * \overline{x_n}) \approx \overline{\mu y(\varphi(x_1, \ldots, x_n, y) = 0)} \approx \overline{\psi(x_1, \ldots, x_n)}.$$

5) Dai punti 1)-4) se ψ è una funzione ricorsiva k-aria, allora esiste un algoritmo normale sopra M $\mathfrak{A}_\psi(\overline{x_1} * \ldots * \overline{x_k}) \approx \overline{\psi(x_1, \ldots, x_k)}$ che calcola ψ.
Sia $\mathfrak{K}$ un algoritmo normale sopra M, definito solo per parole di M della forma $\overline{x_1} * \ldots * \overline{x_k}$, con $x_1, \ldots, x_k \in \mathbb{N}$, tale che $\mathfrak{K}(\overline{x_1} * \ldots * \overline{x_k}) = \overline{x_1} * \ldots * \overline{x_k}$.
Se ψ è (μ-)ricorsiva, $\mathfrak{A}_\psi$ sia l' algoritmo ottenuto componendo gli algoritmi 1-4. Consideriamo $\mathfrak{G}_\psi = \mathfrak{A}_\psi \circ \mathfrak{K}$. Allora $\mathfrak{G}_\psi \approx \overline{\psi(x_1, \ldots, x_k)}$ e $\mathfrak{G}_\psi$ è definito solo per le parole di M della forma $\overline{x_1} * \ldots * \overline{x_k}$, le stesse per cui è definita $\psi(x_1, \ldots, x_n)$. Pertanto, ogni funzione ricorsiva ψ è computabile nel senso di Markov con $\mathfrak{G}_\psi$. □

3.5.2 Ricorsività delle funzioni Markov-computabili

Proseguiamo, ora, introducendo una gödelizzazione dei simboli, delle parole e delle sequenze di parole che ci sarà utile per dimostrare l' altra parte dell' equivalenza tra funzioni ricorsive e computabili secondo Markov.
Siano $S_0, S_1, S_2, \ldots, S_m$ simboli di un alfabeto A. Allora a ciascuno di essi assegniamo un numero $g(S_i) = 2i + 3$.
Sia $P = S_{j_0} \ldots S_{j_k}$ una qualsiasi parola nell' alfabeto A. Ad essa assegniamo il numero $g(P) = 2^{g(S_{j_0})} \cdot 3^{g(S_{j_1})} \cdot \ldots \cdot p_k^{g(S_{j_k})} = 2^{2j_0+3} 3^{(2j_1+3)} \ldots p_k^{(2j_k+3)}$, dove p_k è il k-esimo numero primo. Inoltre, poniamo $g(\Lambda) = 1$. Sia $P_0, \ldots, P_k$ una sequenza di parole; ad essa associamo il numero $g(P_0, \ldots, P_k) = 2^{g(P_0)} \cdot 3^{g(P_1)} \ldots p_k^{g(P_k)}$. Per convenzione, poniamo $\begin{cases} S_1 = 1 \\ S_2 = * \end{cases}$. Se consideriamo i numerali come parole, allora $g(\overline{0}) = 2^5$, $g(\overline{1}) = 2^5 \cdot 3^5$, $g(\overline{2}) = 2^5 \cdot 3^5 \cdot 5^5$, ecc. In generale, quindi, si ha: $g(\overline{n}) = \prod_{i=0}^{n} p_i^5$.

Proposizione 3.5.9. *Esistono algoritmi normali $\mathfrak{L}_1, \mathfrak{L}_2$ sopra $A \cup M$ tali che $\mathfrak{L}_1(P) = \overline{g(P)}$ e $\mathfrak{L}_2(\overline{g(P)}) = P$ per tutte le parole $P \in A$.*

Cioè $\mathfrak{L}_1$ associa ad ogni parola il numerale del suo numero di Gödel, $\mathfrak{L}_2$ invece associa ad ogni numerale di un numero di Gödel, la parola corrispondente.

Definizione 3.20. Dato un algoritmo di Markov $\mathfrak{A}$ sopra un alfabeto A definiamo una funzione $\psi_{\mathfrak{A}}$ associata all' algoritmo $\mathfrak{A}$ come segue:
$\psi_{\mathfrak{A}}(n) = m \Leftrightarrow$ o n non è il numero di Gödel di nessuna parola di A, e in questo caso $m = 0$, oppure n ed m sono numeri di Gödel di due parole in A, rispettivamente P e Q, tali che $\mathfrak{A}$ è applicabile a P e $\mathfrak{A}(P) = Q$.

Definizione 3.21. Data $P \to Q$, produzione semplice, definiamo il suo *indice* come il numero $2^1 \cdot 3^{g(P)} \cdot 5^{g(Q)}$. Analogamente, l' indice di una produzione terminale $P \to \cdot Q$ è dato da $2^2 \cdot 3^{g(P)} \cdot 5^{g(Q)}$. Se $P_0 \to (\cdot)Q_0, \ldots, P_r \to (\cdot)Q_r$ è uno schema di algoritmo, definiamo suo indice il numero $2^{k_0} \cdot 3^{k_1} \cdot \ldots \cdot p_r^{k_r}$, dove k_i è l' indice della i-esima produzione.

Definizione 3.22. Definiamo, ora, una serie di predicati ricorsivi che utilizzeremo nella prossima proposizione.

- **Par(u):** $u = 1 \vee \forall z(z < lh(u) \ ^{*}\Rightarrow \exists y(y < u \wedge (u)_z \ ^{\dagger}= 2y + 3))$.
  Vale se e solo se u è il numero di Gödel di una parola.

- **SI(u):** $lh(u) = 3 \wedge (u)_0 = 1 \wedge Par(u)_1 \wedge Par(u)_2$.
  Vale se e solo se u è l' indice di una produzione semplice.

- **TI(u):** $lh(u) = 3 \wedge (u)_0 = 2 \wedge Par(u)_1 \wedge Par(u)_2$.
  Vale se e solo se u è l' indice di una produzione terminale.

- **Ind(u):** $u > 1 \wedge \forall z\big(z < lh(u) \Rightarrow SI((u)_z) \vee TI((u)_z)\big)$.
  Vale se e solo se u è l' indice di uno schema di algoritmo.

La sequenza di naturali $(\alpha_0 \dots \alpha_n)$ si può codificare con il numero $x = \prod_{i=0}^{n} p_i^{\alpha_i}$ e così per la sequenza $(\beta_0 \dots \beta_m)$ con $y = \prod_{i=0}^{m} p_i^{\beta_i}$. Sia $x\Box y$ una funzione (ricorsiva) definita come la giustapposizione di parole per rappresentare $(\alpha_0 \dots \alpha_n \beta_0 \dots \beta_m)$: se $x = \prod_{i=0}^{n} p_i^{\alpha_i}$ con $\alpha_i > 0$, $\forall i$ ed $y = \prod_{i=0}^{m} p_i^{\beta_i}$, $x\Box y = \prod_{i=0}^{n} p_i^{\alpha_i} \cdot \prod_{i=0}^{m} p_{i+n+1}^{\beta_i}$. Se x, y non sono della forma di sopra $x\Box 1 = 1\Box x = x$ e analogamente.

- **Sso(x,y,e):** $Par(x) \wedge Par(y) \wedge \big(SI(e) \vee TI(e)\big) \wedge (\exists u)_{\leq x}(\exists v)_{\leq x}(x = u\Box(e)_1\Box v \wedge y = u\Box(e)_2\Box v \wedge \neg(\exists w)_{\leq x}(\exists z)_{\leq x}(x = w\Box(e)_1\Box z \wedge w < u))$.
  Vale se e solo se e è l' indice di una produzione $P \to (\cdot)Q$ ed x e y sono i numeri di Gödel di parole U e V tali che P occorre in U e V è il risultato della sostituzione Q al posto dell' occorrenza più a sinistra di P in U.

- **Occ(x,y):** $Par(x) \wedge Par(y) \wedge (\exists v)_{v \leq x}(\exists z)_{z \leq x}(x = v\Box y\Box z)$.
  Vale se e solo se x e y sono numeri di Gödel di parole U e V, dove V occorre in U.

*Notazione: $lh(x)$ è una funzione ricorsiva primitiva che ad ogni numero x associa il numero di esponenti diversi da zero nella scomposizione in fattori primi di x.

†$(u)_z$ è una funzione ricorsiva primitiva che associa ad u l'esponente di posto z nella scomposizione in fattori primi di u.

- **End(e,z):** $Ind(e) \wedge Par(z) \wedge (\forall w)_{w<lh(e)}(\neg Occ(z, ((e)_w)_1))$.
  Vale se e solo se z è il numero di Gödel di una parola P, e è l' indice di uno schema di algoritmo e nessun algoritmo definito da tale schema può essere applicato a P.

- **SCons(e,y,x):** $Ind(e) \wedge Par(x) \wedge Par(y) \wedge (\exists v)_{v<lh(e)}(SI((e))_v) \wedge SSo(x, y, (e)_v) \wedge (\forall z)_{z<v}(\neg Occ(x, ((e)_z)_1)))$
  Vale se e solo se e è l' indice di uno schema d' algoritmo ed y e x sono numeri di Gödel di parole V e U tali che V deriva da U per una produzione semplice dello schema.

- **TCons(e,y,x):** è analogo al precedente ma con una produzione terminale invece che semplice.

- **Der(e,x,y):** $Ind(e) \wedge Par(x) \wedge (\forall z)_{z<lh(y)}(Par((y)_z) \wedge (y)_0 = x \wedge (\forall z)_{z<lh(y)-2}(SCons(e, (y)_{z+1}, (y)_z)) \wedge ((lh(y) = 1 \wedge End(e, (y)_0)) \vee (lh(y) > 1 \wedge (TCons(e, (y)_{lh(y)-1}, (y)_{lh(y)-2}) \vee (SCons(e, (y)_{lh(y)-1}, (y)_{lh(y)-2}) \wedge End(e, (y)_{lh(y)-1})))))$.
  Vale se e solo se e è l' indice di uno schema di algoritmo, x è il numero di Gödel di una parola U_0 ed y è il numero di Gödel di una sequenza di parole $U_0, \ldots, U_k (k \geq 0)$ tali che per $0 \leq i < k \dot{-} 1$, U_{i+1} deriva da U_i per mezzo di un algoritmo determinato dallo schema di cui sopra e U_k deriva da $U_{k \dot{-} 1}$ o per mezzo di una produzione terminale, o per mezzo di una semplice e nel qual caso si ha $\mathfrak{A} : U_k \Box$.

- $P_A(u)$**:** $u = 1 \vee (\forall z)_{z<lh(u)}((u)_z = 3 \vee \ldots \vee (u)_z = 2m + 3)$.
  Vale se e solo se u è il numero di Gödel di una parola di A=$\{S_0, \ldots, S_m\}$.

Proposizione 3.5.10. *Sia $\mathfrak{A}$ un algoritmo normale sopra un alfabeto A, sia $\mathfrak{A}$ applicabile a tutte le parole in A. Allora $\psi_{\mathfrak{A}}$ è ricorsiva.*

Dimostrazione. Sia $\mathfrak{A}$ un qualsiasi algoritmo normale sopra A ed e l' indice del suo schema d' algoritmo. Poniamo $\varphi(x) = \mu y((P_A(x) \wedge (\exists e \leq y) Der(e, x, y)) \vee \neg P_A(x))$. Per ipotesi, $\mathfrak{A}$ è applicabile ad ogni parola in

A, allora φ, definita per mezzo del $\mu-$operatore nel caso regolare, è ricorsiva come relazione, ma non ovunque definita come funzione.
Ora, sia $\psi_{\mathfrak{A}}(x) = (\varphi(x))_{lh(\varphi(x))\dot{-}1}$, ossia $\psi_{\mathfrak{A}}$ associa ad x, che è il numero di Gödel della prima parola U_0 della sequenza, il numero di Gödel dell' ultima parola della sequenza, cioè del risultato dell' algoritmo $\mathfrak{A}$ applicato ad U_0; allora anche $\psi_{\mathfrak{A}}$ è ricorsiva come φ. □

I risultati ottenuti da queste proposizioni valgono, in particolare, considerando come alfabeto A l' alfabeto M=$\{1, *\}$ e dunque nel caso in cui le parole siano k-uple di numeri naturali $(n_1 * \ldots * n_k)$

Corollario 3.5.11. *Sia φ una funzione aritmetica $\mathbb{N}^k \to \mathbb{N}$. Se φ è computabile nel senso di Markov e ovunque definita, allora φ è ricorsiva.*

Si considera il numero di Gödel x di $(n_1 \ldots n_k)$ a cui viene applicata $\psi_{\mathfrak{A}}$.

Abbiamo, così, cercato di chiarire, esponendo tre descrizioni rigorose, il concetto di algoritmo e funzione effettivamente computabile. In ogni caso, data la genericità di tali concetti, non esiste una vera e propria dimostrazione della tesi di Church (pag.12). In particolare, risulta evidente soltanto una delle due implicazioni (ricorsività ⇒ effettiva computabilità), mentre l' altra (effettiva computabilità ⇒ ricorsività) è stata solo confermata per ogni funzione effettivamente computabile conosciuta, ma non dimostrata. A sostegno di questa Tesi, però, resta un fatto peculiare, cioé che i diversi approcci, su alcuni dei quali ci siamo concentrati, nonostante presentino caratteristiche diverse tra loro, sono stati dimostrati essere tutti equivalenti e quindi adatti a descrivere la stessa classe di funzioni.

Capitolo 4

λ-K-calcolo

Presentiamo un'ultima descrizione della classe delle funzioni computabili: il λ-K-calcolo, un sistema formale introdotto da A. Church per analizzare le funzioni e il loro calcolo.

4.1 Osservazioni preliminari

Questo calcolo è motivato dal calcolo con funzioni piuttosto che con numeri: siano f una funzione unaria ed x un suo argomento, allora indicheremo il valore della funzione f con (fx). Se anche x è una funzione e se è definito anche (xf), in generale si avrà $(fx) \neq (xf)$. Una delle particolarità del sistema formale del λ-K-calcolo è che è sufficiente considerare funzioni di un argomento. Vediamolo con l' esempio dell' addizione: consideriamo il + come una funzione unaria che, applicata all' argomento a, dà come valore quella funzione $(+a)$ che applicata all' argomento b ha a sua volta come valore la somma di a e b: (+a)b. Così facendo, possiamo ricondurci da funzioni di n argomenti $f(x_1, x_2, \ldots, x_n)$ all' applicazione successiva di funzioni unarie: $\big(\ldots((fx_1)x_2)\ldots x_n\big)$, scrittura che coincide con $(fx_1 \ldots x_n)$.

Chiariamo anche la notazione corrispondente all' usuale $f(x)$: consideriamo la variabile x come se fosse vincolata, premettendo ad (fx) l' operatore λ, allora la scrittura $\lambda x(fx)$ indica la funzione f che associa ad x, $f(x)$. Al-

lora si ha $\lambda x(fx) = \lambda y(fy)$, mentre in generale $(fx) \neq (fy)$. Il λ-operatore è comodo per definire funzioni tramite sostituzione in altre funzioni. Ad esempio, per definire la funzione cot, partendo dalle funzioni $\sin, \cos$ si ha $\cot = \lambda x(\cos x / \sin x)$ ed introducendo un simbolo di funzione Q per il quoziente, $\cot = \lambda x(Q(\cos x)(\sin x)) = \lambda y(Q(\cos y)(\sin y))$.
Regola di calcolo che deriva dal significato del $\lambda-$operatore è la seguente:

$$\lambda x(fx)a = (fa)$$

e in generale, per un termine funzionale λxF, dove F è un' espressione composta, vale la regola:

$$\lambda xFa = G,$$

dove G è ottenuto da F sostituendo con a ogni occorrenza libera di x in F.
In particolare, riprendendo l'esempio della cotangente, avremo

$$(\cot 2) = \big(\lambda x(Q(\cos x)(\sin x))2\big) = \big(Q(\cos 2)(\sin 2)\big).$$

Abbiamo fatto riferimento finora all' applicazione di una funzione ad un argomento e all' operatore λ. A partire da queste nozioni è possibile sviluppare il cosiddetto λ-K-calcolo. Consideriamo un insieme numerabile di variabili: $x_1, x_2, \ldots, x_n$ e di simboli: $(,\ ),\ \lambda,\ =$. Come simboli per le variabili usiamo $x, y, z, \ldots$ ed $f, g, h, \ldots$ quando tali variabili sono delle funzioni.
Utilizzando variabili e simboli, si ottengono delle parole: $W_1, W_2, \ldots$. Per indicare che W_1 e W_2 sono la stessa parola scriveremo $W_1 \approx W_2$.
Diamo ora una definizione "ricorsiva" di cosa intendiamo con *termini* e quali variabili occorrono *libere* in essi.

Definizione 4.1. - Ogni variabile x è un termine; x occorre libera in x e nessun'altra variabile occorre libera in x;

- Se T_1 e T_2 sono termini, allora anche (T_1T_2) è un termine; una variabile x occorre libera in (T_1T_2) se e solo se x occorre libera in almeno uno fra T_1 e T_2;

- Se T è un termine e x è una variabile qualunque, allora anche λxT è un termine. Una variabile y occorre libera in λxT se e solo se y occorre libera in T e $y \neq x$.

Se $T_1, T_2, \ldots, T_n$ sono termini, convenzionalmente si usa l' abbreviazione $(T_1T_2T_3 \ldots T_n)$ al posto di $\big(\ldots((T_1, T_2)T_3)\ldots\big)$ (associazione delle parentesi sulla sinistra). Non vale in generale l' associatività $(T_1T_2T_3) = ((T_1T_2)T_3) \neq (T_1(T_2T_3))$.

Definizione 4.2. Se T_1 e T_2 sono termini, allora $T_1 = T_2$ è un' *equazione*.

Definizione 4.3. Indicheremo l' *operazione di sostituzione* con ${}^x/U$. Essa si può applicare a qualsiasi termine T e indichiamo con T^x/U la sostituzione del termine U al posto di x all' interno del termine T.

- Se T è una variabile, allora $T^x/U \approx U$ o T, a seconda che si abbia rispettivamente $T \approx x$ oppure $T \not\approx x$.

- $(T_1T_2)^x/U \approx (T_1^x/U, T_2^x/U)$

- $[\lambda yT]^x/U \approx \lambda yT$ o $\lambda y[T^x/U]$ a seconda che $y \approx x$ o $y \not\approx x$.

Nella sezione successiva vedremo come poter derivare le equazioni del λ-K-calcolo tramite alcune regole sintattiche che esporremo.
Se è derivabile $T_1 = T_2$, allora T_1 e T_2 rappresentano una stessa funzione. Una funzione però può essere espressa da diversi termini.

4.2 Regole per la derivazione

1) Rinomina delle variabili vincolate:

$$\lambda xT = \lambda y[T^x/y], \quad \text{purché } y \text{ non occorra in } T.$$

In sostanza si possono scegliere etichette diverse per le variabili vincolate tra quelle non presenti nell'espressione.

2) Eliminazione del λ-operatore:

$$(\lambda xTU) = T^x/U,$$

purché per nessuna variabile y che occorra libera in U la parola λy sia una parte di T. Questa regola afferma che λ è un operatore che indica funzioni.

3) È lecito passare dalle equazioni $T_1 = T_2$ e $U_1 = U_2$ all' equazione $(T_1U_1) = (T_2U_2)$.

4) È lecito passare dall' equazione $T_1 = T_2$ all' equazione $\lambda xT_1 = \lambda xT_2$.

5) Riflessività della relazione di identità: $T = T$.

6) Simmetria della relazione di identità: è lecito passare dall' equazione $T_1 = T_2$ all'equazione $T_2 = T_1$.

7) Transitività della relazione di identità: è lecito passare dalle equazioni $T_1 = T_2$ e $T_2 = T_3$ all' equazione $T_1 = T_3$.

4.3 Rappresentazione dei numeri naturali nel λ-calcolo

Introduciamo dapprima il concetto di *n-esima iterazione* di una funzione f:

$$\textbf{1}^{\textbf{a}}\textbf{iterazione di} f \approx \lambda x(fx)$$

$$\textbf{2}^{\textbf{a}}\textbf{iterazione di} f \approx \lambda x(f(fx))$$

$$\textbf{3}^{\textbf{a}}\textbf{iterazione di} f \approx \lambda x\big(f(f(fx))\big)$$

$$\textbf{4}^{\textbf{a}}\textbf{iterazione di} f \approx \lambda x\big(f((f(f(fx)))\big)$$

ecc.

Ovviamente si ha: **0ªiterazione di**$f \approx \lambda xx$.
Consideriamo i seguenti termini: $\underline{0}$,$\underline{1}$,$\underline{2}$,...:

$$\underline{0} \approx \lambda f \lambda x x$$

$$\underline{1} \approx \lambda f \lambda x (f x)$$

$$\underline{2} \approx \lambda f \lambda x (f (f x))$$

$$\underline{3} \approx \lambda f \lambda x \big(f (f (f x))\big)$$

. . . . . .

Diremo che il termine $\underline{n}$ rappresenta il numero naturale n.
Nel λ-K-calcolo sono inoltre derivabili le seguenti equazioni (a fianco le regole applicate):

a) $(\underline{n} f) = \lambda x (\underbrace{f \ldots (f}_{n \ volte} x) \ldots)$ 2)

b) $(\underline{n} f x) = (\underbrace{f \ldots (f}_{n \ volte} x) \ldots)$ $a), 5), 3), 2), 7)$

c) $(f(\underline{n} f x) = (\underbrace{f \ldots (f}_{n+1 \ volte} x) \ldots)$ $b), 5), 3)$

d) $\lambda f \lambda x (f (\underline{n} f x)) = \underline{n+1}$ $c), 4)$

4.4 Definibilità di funzioni nel λ-K-calcolo

Sia φ una funzione definita per tutte le k-uple di numeri naturali ($k \geq 0$) a valori sempre nei numeri naturali. Diciamo che φ è λ-K-definibile se nel λ-K-calcolo esiste un termine F tale che, per ogni k-upla $r_1, \ldots, r_k$ di numeri naturali, l' equazione

$$(F \underline{r_1} \ldots, \underline{r_k}) = \underline{\varphi(r_1, \ldots, r_k)}$$

è derivabile nel λ-K-calcolo. F è detto il *termine che definisce* φ.

Concludiamo la presentazione sul λ-K-calcolo col seguente

Teorema 4.4.1. *Ogni funzione aritmetica f è ricorsiva se e solo se è λ-K-definibile.*

di cui tralasciamo la dimostrazione [Hermes, p.279].
Questa è una ulteriore conferma del fatto che le funzioni effettivamente computabili sono una classe ben precisa, che si può definire in molte maniere alternative, come abbiamo visto.

Bibliografia

[1] [Hermes 1975] Hans Hermes, *Enumerabilità, decidibilità, computabilità*, Bollati Boringhieri, Torino 1975.

[2] [Mendelson 1972] Elliot Mendelson, *Introduzione alla logica matematica*, Bollati Boringhieri, Torino 1972.

[3] [Hodges 2006] Hodges Andrew, *Alan Turing. Storia di un enigma*, Bollati Boringhieri, Torino 2006.

[4] [Penrose 1992] Roger Penrose, *La mente nuova dell' imperatore*, Rizzoli, 1992.

[5] [Penrose 1996] Roger Penrose, *Le ombre della mente*, Rizzoli, 1996

[6] [Turing 1937] Alan Turing, *On computable numbers with an application to Entscheidung problem*, *Proc. Lond. Math. Soc. 42(II), pp. 230-265 (1937)*; una correzione 43, pp. 544-546.

[7] http://matematica-old.unibocconi.it

[8] http://www.jstor.org.