DISPENSE DI

TEORIA DELLA RICORSIVITA'
E
CALCOLABILITA' EFFETTIVA

MARCELLO FRIXIONE

DIST – Università di Genova

1991

1. Il concetto intuitivo di algoritmo

1.1 Che cos'è un algoritmo

La *teoria della ricorsività* (o *teoria della computabilità effettiva*) è quel settore della logica matematica in cui vengono indagati concetti quali quello di algoritmo e di funzione computabile in modo algoritmico.

Intuitivamente, si dispone di un *algoritmo* (o di un *metodo effettivo*) per risolvere un problema se si dispone di un elenco *finito* di istruzioni tali che:

1) A partire dai dati iniziali, le istruzioni sono applicabili in maniera rigorosamente deterministica, in maniera cioè che ad ogni passo sia sempre possibile stabilire univocamente quale è l'istruzione che deve essere applicata al passo successivo;

2) Si dispone di un criterio univoco per stabilire quando si è raggiunto uno *stato finale*, quando cioè il processo deve considerarsi terminato e il risultato, se esiste, è stato ottenuto. Uno stato finale deve sempre essere raggiungibile in un numero finito di passi.

Chiameremo *input* i dati di partenza; *output* il risultato del calcolo.

Una *funzione* si dice *calcolabile in modo algoritmico* (o *calcolabile in modo effettivo*, o *effettivamente calcolabile*, o anche *calcolabile*) se esiste un algoritmo che consente di calcolarne i valori per tutti gli argomenti.

Esempi di algoritmi possono essere tratti dalle matematiche elementari: sono algoritmi, ad esempio, gli insiemi di regole che consentono di eseguire le quattro operazioni, come pure è un algoritmo il procedimento euclideo per la ricerca del massimo comun divisore di due numeri naturali. In logica, il metodo delle tavole di verità è un algoritmo che permette di stabilire se una formula del calcolo proposizionale è o meno una tautologia.

Per le caratteristiche di determinismo e di finitezza che abbiamo enunciato, ogni algoritmo si presta, almeno in linea di principio, ad essere automatizzato, ad essere eseguito cioè da una macchina opportunamente progettata. La teoria della computabilità effettiva ha dunque assunto grande importanza anche dal punto divista applicativo con lo sviluppo dei calcolatori digitali, e svolge il ruolo di teoria dei fondamenti per l'informatica teorica.

Durante la storia delle matematiche sono stati sviluppati algoritmi per risolvere classi di problemi sempre più estese. La teoria della computabilità nasce tuttavia soltanto nella terza decade del nostro secolo, con l'esigenza, nata nell'ambito degli studi di logica, di fornire un equivalente rigoroso del concetto intuitivo di algoritmo, e di indagare le possibilità ed i limiti dei metodi effettivi.

In teoria della ricorsività lo studio dei metodi algoritmici viene ricondotto allo studio delle funzioni aritmetiche computabili in modo effettivo. Ciò potrebbe sembrare troppo restrittivo, in quanto esistono algoritmi definiti su oggetti diversi dai numeri naturali. Vi sono algoritmi che stabiliscono se un certo oggetto matematico appartiene a un dato insieme o meno. Ve ne sono altri che eseguono operazioni simboliche sulle espressioni di un sistema formale, stabilendo ad esempio se una data formula gode o meno di una certe proprietà. In logica matematica tuttavia sono state sviluppate tecniche mediante le quali algoritmi di tipo diverso, quali quelli sopra citati, possono essere ricondotti a funzioni aritmetiche. L'appartenenza di un oggetto ad un insieme può essere espressa tramite una funzione caratteristica; le espressioni di un sistema formale possono essere

codificati nei numeri naturali tramite tecniche quali la gödelizzazione; e così via. I numeri naturali possono dunque essere visti come un mezzo per rappresentare dati generici di tipo discreto.

1.2 Funzioni computabili parziali e totali

Iniziamo introducendo alcune definizioni. Si dice *dominio* di una funzione l'insieme dei suoi possibili argomenti. Ad esempio, il dominio della funzione aritmetica "il quadrato di ... " è l'insieme $\mathbb{N}$ dei numeri naturali; il dominio della somma aritmetica è $\mathbb{N}^2$ (cioè l'insieme delle coppie di numeri naturali); e, in generale, il dominio di una funzione aritmetica a n argomenti è l'insieme $\mathbb{N}^n$ delle n-ple di numeri naturali.

Si dice *codominio* di una funzione l'insieme cui appartengono i valori della funzione stessa. Ad esempio, il codominio de "il quadrato di ... " e della somma aritmetica è l'insieme $\mathbb{N}$ dei numeri naturali. Non è necessario che il codominio coincida con l'insieme dei valori (o *rango*) di una funzione: ad esempio l'insieme dei valori de "il quadrato di ... " è un sottoinsieme proprio di $\mathbb{N}$.. Il rango di una funzione φ coincide col codominio *sse* φ è *surgettiva.*

Di una funzione φ che ha dominio D e codominio C si dice che è di tipo $D \to C$ (e si scrive $\varphi: D \to C$).

Una funzione $\varphi: D \to C$ si dice *totale sse* è definita per ogni elemento del dominio, vale a dire:

$$\forall x \in D\, \exists y \in C \text{ t.c. } \varphi(x) = y.$$

Ad esempio, la somma è una funzione totale di tipo $\mathbb{N}^2 \to \mathbb{N}$ in quanto essa associa ad ogni coppia (m, n) di numeri naturali presi come argomento il numero naturale $n + m$ come valore.

Una funzione $\varphi: D \to C$ si dice *parziale* se esiste almeno un elemento del dominio per il quale essa non è definita, al quale cioè essa non associa alcun elemento di C come valore. Ad esempio, la funzione "radice quadrata di ... " in quanto funzione di tipo $\mathbb{N} \to \mathbb{N}$ è parziale: essa assume un valore in $\mathbb{N}$ soltanto per quegli argomenti che sono quadrati perfetti (è invece totale se considerata funzione di tipo $\mathbb{R}^+ \to \mathbb{R}^+$, oppure di tipo $\mathbb{N} \to \mathbb{R}^+$). Se una funzione parziale φ è indefinita per l'argomento n, scriveremo:

$$\varphi(n) = \perp$$

($\perp$ si legge *bottom*) e diremo che φ *diverge* per n. Viceversa, se φ è definita per l'argomento n diremo che φ *converge* per n. Chiameremo *dominio di definizione* di φ il sottoinsieme del dominio per il quale φ è definita.

"Radice quadrata di ... " si comporta dunque nel modo seguente:

$$\sqrt{x} \quad \begin{cases} \text{converge se } \exists y \in \mathbb{N} \text{ t.c. } y^2 = x \\ \\ = \perp \quad \text{altrimenti} \end{cases}$$

e il suo dominio di definizione è l'insieme:

$$\{x \in \mathbb{N} \mid \exists y\, (y^2 = x)\}.$$

Graficamente, la situazione può essere schematizzata come nelle figure 1.1 e 1.2.

Caso limite di funzione parziale è la funzione *totalmente indefinita*, vale a dire quella funzione $\varphi: D \to C$ tale che:

$$\forall x \in \mathrm{D}\; \varphi(x) = \perp.$$

φ è effettivamente computabile in modo banale; il suo dominio di definizione e il suo rango coincidono con $\varnothing$.

All'estremo opposto, anche le funzioni totali possono essere viste come casi limite di funzioni parziali: per le funzioni totali il dominio di definizione coincide con l'intero dominio.

Nel caso di funzioni effettivamente computabili parziali, che la funzione φ diverga per un certo argomento n può essere dovuto a due motivi distinti: (i) l'algoritmo che computa φ, per l'input n, dà origine a un calcolo che termina ma che non produce alcun output; (ii) l'algoritmo che computa φ, per l'input n, dà origine ad un calcolo che non termina. Si considerino, ad esempio, i due algoritmi raffigurati in fig. 1.3, che calcolano (in maniera del tutto inefficiente, ma questo, nel nostro caso, è irrilevante) la radice quadrata come funzione di tipo $\mathbb{N} \to \mathbb{N}$. Se la funzione è indefinita, nel caso A l'algoritmo entra in *loop* e dà origine a un calcolo che non termina; nel caso B l'algoritmo termina non producendo però alcun risultato.

Se, per calcolare una funzione φ parziale, si dispone di un algoritmo che termina ogni qual volta la funzione è indefinita, è facile estendere φ ad una funzione φ' effettivamente computabile totale, assegnando convenzionalmente un valore a φ' quando φ è indefinita. Si consideri, ad esempio, l'usuale *differenza*. Come funzione di tipo $\mathbb{N}^2 \to \mathbb{N}$ essa è una funzione parziale, in quanto:

$$x - y \begin{cases} \text{è definita se } x \geq y \\ \\ = \bot \text{ altrimenti} \end{cases}$$

Essa può tuttavia essere estesa ad una funzione computabile totale (che chiameremo *differenza artimetica* e indicheremo con $\dot{-}$) ponendo:

$$x \dot{-} y = \begin{cases} x\text{-}y \text{ se } x \geq y \\ \\ 0 \text{ altrimenti} \end{cases}$$

Come vedremo, questo non è in generale possibile nel caso di algoritmi che non terminano.

1.3 Insiemi decidibili e insiemi effettivamente enumerabili

In questo paragrafo vengono introdotti alcuni concetti connessi al concetto intuitivo di algoritmo.

Dato un predicato n-adico $R(x_1, \ldots, x_n)$ (o dato un insieme R di n -ple) si dice *funzione caratteristica* di R la funzione $C_R(x_1, \ldots, x_n)$ tale che:

$$C_R(x_1, \ldots, x_n) = \begin{cases} 0 \text{ se } R(x_1, \ldots, x_n) \text{ è vero (oppure se } \langle x_1, \ldots, x_n \rangle \in \mathrm{R}) \\ \\ 1 \text{ se } R(x_1, \ldots, x_n) \text{ è falso (oppure se } \langle x_1, \ldots, x_n \rangle \notin \mathrm{R}) \end{cases}$$

Un *insieme* si dice *decidibile sse* la sua funzione caratteristica è effettivamente computabile (totale). (In altre parole, un insieme è decidibile sse esiste un algoritmo che permette sempre di stabilire se un oggetto vi appartiene o meno; è ovvio che ogni insieme finito è sempre decidibile.)

Un *insieme A* si dice *enumerabile in modo effettivo* (o *effettivamente enumerabile*) *sse A* = ∅ oppure A è il rango di una funzione φ effettivamente computabile totale. In tal caso si dice che φ *enumera A*, e, per ogni $x \in \mathbb{N}$, se

$$\varphi(x) = n$$

si dice che x è indice di n rispetto all'enumerazione data da φ. In generale, un'enumerazione φ può contenere delle ripetizioni, possono esistere cioè x, y tali che:

$$x \neq y \text{ e } \varphi(x) = \varphi(y).$$

Tuttavia (se A è infinito) si può sempre ottenere, a partire da φ, una funzione effettivamente computabile totale φ' che enumera A senza ripetizioni (φ' può essere calcolata calcolando $\varphi(x)$ per $x = 1, 2, ...$ e scartando via via i valori già apparsi in precedenza). Ovviamente, per ogni $x \in \mathbf{N}$ è possibile ottenere in modo algoritmico l'elemento di A di cui x è indice (basta calcolare $\varphi(x)$). Anche il passaggio inverso è però effettuabile in maniera algoritmica: dato un generico $n \in A$, è possibile ricavarne l'indice (o gli indici) j in modo effettivo, calcolando successivamente i vari $\varphi(i)$ per $i = 1, 2, ...$, sino a che non si trova j tale che $\varphi(j) = n$.

Un *predicato P* si dice *decidibile* (*enumerabile in modo effettivo*) *sse* l'insieme

$$A_P \equiv \{x \mid P(x)\}$$

è decidibile (enumerabile in modo effettivo).

Dimostreremo le seguenti proposizioni nel caso di insiemi di numeri naturali. Gli stessi risultati si possono tuttavia estendere ai sottoinsiemi di $\mathbf{N}^n$ (con $n = 2, 3, ...$).

PROPOSIZIONE: *(a) A è enumerabile in modo effettivo ⇔ (b) A è il dominio di definizione di una funzione computabile parziale ⇔ (c) A è il rango di una funzione computabile parziale.*

Dimostrazione.

1) (c) ⇒ (a). Sia φ la funzione computabile parziale di cui A è il rango. Se φ è totalmente indefinita, allora $A = \emptyset$. Altrimenti si analizzi il calcolo generato dall'algoritmo che computa φ in una successione di passi finiti distinti, facendo corrispondere, ad esempio, ogni passo all'applicazione di una delle istruzioni. Si immagini quindi una tabella in cui le colonne corrispondano alla serie di tutti gli argomenti di φ (cioè, alla serie dei numeri naturali), e le linee alla successione dei passi di calcolo, come in fig. 1.4.

Per ogni input, ogni volta che l'algoritmo giunge a uno stato finale producendo un output, si segni con l'output ottenuto la casella corrispondente delle tabella, e tutte le caselle successive della stessa colonna. Ad esempio, se $\varphi(1) = 9$, e se l'algoritmo che calcola φ ottiene tale risultato al quarto passo di calcolo, si marchi con 9 la quarta linea della prima colonna e tutte le sue linee successive (cfr. fig. 1.4).

È possibile esplorare l'intera tabella secondo un percorso come quello indicato nella figura, e costruire in questo modo un'enumerazione (con ripetizioni) di tutti gli output di φ, i quali possono quindi essere assegnati come valori a una funzione totale φ'. Ad esempio, nel caso di fig. 1.4, φ' si comporterebbe nel modo seguente:

$\varphi'(1) = 9$
$\varphi'(2) = 2$
$\varphi'(3) = 9$
$\varphi'(4) = 9$
$\varphi'(5) = 5$
$\varphi'(6) = 2$
$\varphi'(7) = 4$
$\varphi'(8) = 2$
$\varphi'(9) = 5$
. . . . .
. . . . .

φ' ha quindi lo stesso rango di φ, vale a dire l'insieme A. φ' inoltre è computabile in modo effettivo, in quanto è possibile specificare in maniera precisa un algoritmo che esegua il procedimento sopra descritto.

2) (a) ⇒ (b). Per $A \neq \emptyset$, sia φ la funzione computabile totale di cui A è rango. Definiamo allora una funzione φ' come segue:

$$\varphi'(x) = \text{il più piccolo } y \text{ tale che } \varphi(y) = x.$$

φ' è calcolata dall'algoritmo rappresentato in fig. 1.5. Per come è stata definita, $\varphi'(x) \neq \bot$ *sse* x appartiene al rango di φ (per quegli x che non appartengono al rango di φ l'algoritmo entra in *loop*); quindi A, in quanto rango di φ, coincide col dominio di definizione di φ'. Per $A = \emptyset$, A è dominio di definizione della funzione totalmente indefinita.

3) (b) ⇒ (c). Sia φ la funzione computabile parziale di cui A è dominio di definizione. Ciò vuol dire che φ si comporta nel modo seguente:

$$\varphi(x) \begin{cases} \text{converge se } x \in A \\ = \bot \text{ se } x \notin A \end{cases}$$

Definiamo allora φ' come segue:

$$\varphi'(x) = \begin{cases} x \text{ se } \varphi(x) \text{ converge} \\ \bot \text{ se } \varphi(x) = \bot \end{cases}$$

φ' è a sua volta computabile, ed il suo rango coincide con A[1]. Q.E.D.

Di un insieme A, enumerabile in modo effettivo, si dice talvolta che è *semidecidibile*; da quanto dimostrato consegue infatti che si può disporre di un algoritmo che, preso in input un generico $n \in \mathbf{N}$, produca una risposta affermativa se $n \in A$, ma non è detto che termini o produca un output se $n \notin A$.

Sia $\overline{A}$ l'insieme complemento di A rispetto a $\mathbf{N}$, vale a dire:

$$\overline{A} \equiv \{ x \in \mathbf{N} \mid x \notin A \}.$$

Dimostriamo allora la seguente:

PROPOSIZIONE: *$\overline{A}$ è decidibile ⇔ A è decidibile ⇔ A e $\overline{A}$ sono enumerabili in modo effettivo.*

Dimostrazione.

1) A è decidibile ⇔ $\overline{A}$ è decidibile.

Data la funzione caratteristica C_A di A, la funzione caratteristica $C_{\overline{A}}$ di $\overline{A}$ può essere definita come segue:

---

[1] La dimostrazione diretta dell'implicazione *(a) ⇒ (c)*, benché superflua per la dimostrazione globale, può essere utile ai fini didattici. Si è quindi scelto di riportarla in nota.

Dimostrazione: sia $A \neq \emptyset$. Per la definizione di insieme effettivamente enumerabile, A è rango di una funzione computabile totale. Sia φ tale funzione. Allora A è rango della funzione computabile parziale φ' così definita:

$$\varphi'(x) = \begin{cases} \bot \text{ se } x = 0 \\ \varphi(x-1) \text{ altrimenti.} \end{cases}$$

Se $A = \emptyset$, A è il rango della funzione totalmente indefinita. Q.E.D.

$$C_{\overline{A}}(x) = \begin{cases} 0 \text{ se } C_A(x) = 1 \\ \\ 1 \text{ se } C_A(x) = 0 \end{cases}$$

È quindi ovvio che, se C_A è computabile, lo è anche $C_{\overline{A}}$, e viceversa.

2) A e $\overline{A}$ sono enumerabili in modo effettivo ⇒ A è decidibile.

Se $A = \varnothing$ oppure $\overline{A} = \varnothing$, ovvio. Altrimenti, si supponga di voler stabilire se un generico $n \in \mathbf{N}$ è membro o meno di A. Sia A che $\overline{A}$ sono il rango di una funzione computabile totale, rispettivamente φ_A e $\varphi_{\overline{A}}$. Si proceda, in parallelo, a calcolarne i valori per gli argomenti 1, 2, 3, ... Siccome n appartiene sicuramente ad A o ad $\overline{A}$, prima o poi n comparirà fra i valori di φ_A oppure fra quelli di $\varphi_{\overline{A}}$; si dispone quindi in questo modo di un algoritmo per decidere in un numero finito di passi se $n \in A$ o meno.

3) A è decidibile ⇒ A e $\overline{A}$ sono enumerabili in modo effettivo.

Definiamo la funzione φ come segue:

$$\varphi(x) = \begin{cases} x \text{ se } C_A(x) = 0 \\ \\ \perp \text{ se } C_A(x) = 1 \end{cases}$$

φ è computabile parziale, ed A ne è il rango. Quindi A è effettivamente enumerabile. Siccome abbiamo dimostrato che se A è decidibile lo è anche $\overline{A}$, allo stesso modo si dimostra che $\overline{A}$ è enumerabile in modo effettivo. Q.E.D.

Per esercizio, si dimostrino le seguenti

PROPOSIZIONI:

A e B sono decidibili ⇒ $A \cup B$ è decidibile.

A e B sono decidibili ⇒ $A \cap B$ è decidibile.

A e B sono effettivamente enumerabili ⇒ $A \cap B$ è effettivamente enumerabile.

In conseguenza delle definizioni di predicato decidibile e di predicato effettivamente enumerabile che abbiamo formulato, si possono ricavare come corollari le seguenti proposizioni.

PROPOSIZIONE: *il predicato P è effettivamente enumerabile ⇔ esiste φ computabile parziale tale che, per ogni x, $\varphi(x) \neq \perp$ sse $P(x)$ ⇔ esiste ψ computabile parziale tale che:*

$$\textit{rango di } \psi \equiv \{x \mid P(x)\}.$$

Sia $\overline{P}$ il predicato tale che, per ogni x:

$$\overline{P}(x) \text{ sse } \neg P(x).$$

Allora:

PROPOSIZIONE: *$\overline{P}$ è decidibile ⇔ P è decidibile ⇔ P e $\overline{P}$ sono effettivamente enumerabili.*

Dati i predicati P e Q, sia R il predicato tale che, per ogni x:

$$R(x) \textit{ sse } P(x) \wedge Q(x).$$

Allora:

PROPOSIZIONE: *P e Q sono decidibili (effettivamente enumerabili) ⇒ R è decidibile (effettivamente enumerabile).*

Sia R' il predicato tale che, per ogni x:

$$R'(x) \text{ sse } P(x) \vee Q(x).$$

Allora:

PROPOSIZIONE: *P e Q sono decidibili (effettivamente enumerabili) $\Rightarrow$ R' è decidibile (effettivamente enumerabile).*

Vale inoltre la seguente:

PROPOSIZIONE: *dato il predicato decidibile a due argomenti $P(x,y)$, il predicato a un argomento $\exists y\, P(x,y)$ è semidecidibile.*

Dimostrazione: si consideri l'algoritmo raffigurato nella fig. 1.6, che calcola i valori della funzione φ_P. Per ogni x, se $\exists y\, P(x,y)$, l'algoritmo termina e pone $\varphi_P = 0$. Se $\nexists y\, P(x,y)$, l'algoritmo entra in *loop*. Q.E.D.

Lo stesso risultato può essere facilmente esteso a predicati con un numero qualsiasi di argomenti. Nel § 6.2 vedremo che, in generale, il fatto che P sia decidibile, non implica che il predicato $\exists y$ P(x,y) sia a sua volta decidibile.

2. Ricorsività primitiva

2.1 Definizione induttiva di funzioni

La tecnica di definizione *induttiva* o *ricorsiva* consiste nel definire una funzione stabilendone innanzi tutto il valore per l'argomento 0 (base della definizione induttiva); poi, supponendo noto il valore che la funzione assume per un generico argomento n, si definisce il valore della funzione per il successore di n (in simboli: $s(n)$) (passo induttivo della definizione).

Un tipico esempio di definizione induttiva è offerto dalla definizione della funzione fattoriale. Il fattoriale di un numero naturale n diverso da 0 (in simboli $n!$) è il prodotto dei primi n numeri interi positivi: $n! = n * (n-1) * (n-2) * ... * 2 * 1$. Il fattoriale di 0 è per definizione 1. Si noti che, per ogni n diverso da 0, il fattoriale di n è uguale al prodotto di n moltiplicato per il fattoriale del suo antecedente n-1. Vale a dire, $n! = n * (n-1)!$ (oppure, in maniera equivalente, $(s(n))! = s(n) * n!$).

La funzione fattoriale può quindi essere definita ricorsivamente come segue:

$$\begin{cases} 0! = 1 \\ (s(n))! = s(n) * n! \end{cases}$$

dove la prima equazione costituisce la base dell'induzione, la seconda il passo induttivo.

L'interesse delle definizioni induttive risiede nel fatto che ogni definizione di questo tipo fornisce implicitamente un algoritmo che consente di calcolare in un numero finito di passi il valore della funzione per qualsiasi argomento. Per ogni argomento n, il passo induttivo infatti riconduce la valutazione della funzione alla valutazione della funzione per l'argomento n-1, fino a che, per l'argomento 0, il valore viene fornito esplicitamente dalla base della definizione.

Si calcoli ad esempio il fattoriale di 4. Per il passo induttivo si ha:

$$4! = 4 * 3!$$

e, sempre per il passo induttivo:

$$3! = 3 * 2!$$

Sostituendo si ottiene:

$$4! = 4 * 3 * 2!$$

Procedendo analogamente si avrà:

$$4! = 4 * 3 * 2 * 1!$$

$$4! = 4 * 3 * 2 * 1 * 0!$$

Utilizzando la base della definizione (0! = 1) si ottiene infine:

$$4! = 4 * 3 * 2 * 1 * 1 = 24.$$

Nell'esempio del fattoriale abbiamo assunto come nota la funzione prodotto. Anch'essa tuttavia, come la maggior parte delle usuali funzioni aritmetiche, è definibile ricorsivamente. Come esempi, riportiamo le definizioni induttive di somma, prodotto, esponenziazione, predecessore e differenza aritmetica. (Eccetto il predecessore, si tratta di funzioni a due argomenti, e l'induzione avviene sul secondo di essi.)

$$\begin{cases} x + 0 = x \\ x + s(y) = s(x + y) \end{cases}$$

$$\begin{cases} x * 0 = 0 \\ x * s(y) = (x * y) + x \end{cases}$$

$$\begin{cases} x^0 = s(0) \\ x^{s(y)} = x^y * x \end{cases}$$

$$\begin{cases} pr(0) = 0 \\ pr(s(x)) = x \end{cases}$$

$$\begin{cases} x - 0 = x \\ x - s(y) = pr(x - y) \end{cases}$$

Si noti che in ciascuna di queste definizioni, oltre allo 0, alla funzione successore s ed al *definiendum*, compaiono come costanti soltanto funzioni definite precedentemente.

2.2 La classe delle funzioni ricorsive primitive

Vediamo come caratterizzare in maniera rigorosa ed esplicita il procedimento di definizione ricorsiva descritto nel paragrafo precedente.

Definiamo la classe delle funzioni ricorsive primitive (*RP*) come la più piccola classe di funzioni da $\mathbf{N}^n$ a $\mathbf{N}$ (con $k = 1, 2, ...$) tale che le appartengano le seguenti *funzioni base*:

1) *funzione zero* $Z(x) = 0$
2) *funzione successore* $s(x) = x + 1$
3) *funzioni proiezione* $U_i^n(x_1,..., x_n) = x_i$
(con $n \geq 1$ e $1 \leq i \leq n$)

e tale che:

4) se le funzioni $\chi(x_1,..., x_k)$ e $\psi_1(x_1,..., x_n)$, ..., $\psi_k(x_1,..., x_n)$ appartengono a *RP*, e se la funzione $\varphi(x_1,..., x_n)$ è definita in questo modo:

$$\varphi(x_1,..., x_n) = \chi(\psi_1(x_1,..., x_n), ..., \psi_k(x_1,..., x_n))$$

allora $\varphi(x_1,..., x_n)$ appartiene a *RP* (chiusura rispetto alla *composizione*);

5) se le funzioni $\chi(x_1,..., x_n)$ e $\psi(x_1,..., x_{n+2})$ appartengono a *RP*, e se la funzione $\varphi(x_1,..., x_{n+1})$ è definita in questo modo:

$$\begin{cases} \varphi(x_1,..., x_n, 0) = \chi(x_1,..., x_n) \\ \varphi(x_1,..., x_n, s(y)) = \psi(x_1,..., x_n, y, \varphi(x_1,..., x_n, y)) \end{cases}$$

allora $\varphi(x_1,..., x_{n+1})$ appartiene a *RP* (chiusura rispetto alla *ricorsione primitiva*).

Lo schema di ricorsione primitiva rende esplicito in tutta la sua generalità lo schema di definizione induttiva esaminato nel paragrafo precedente. Si noti l'analogia con un sistema formale assiomatizzato. Le funzioni base sono analoghe ad assiomi, e i due schemi di sostituzione e di ricorsione possono essere visti come regole che, a partire dalle funzioni base/assiomi, permettono di derivare altre funzioni.

Definiamo la nozione di *derivazione* di una funzione: sia { φ_i }, con $i = 1, ..., n$, una successione di funzioni tali che:

1) Ogni funzione φ_i della successione è una funzione base oppure è ottenuta dalle funzioni precedenti per composizione o per ricorsione primitiva;

2) $\varphi_n = \varphi$.

Si dice allora che { φ_i } *è una derivazione per* φ.

Continuando il parallelismo con un sistema assiomatico, la nozione di derivazione è analoga a quella di dimostrazione.

Dalla definizione di *RP* e dalla definizione di derivazione segue che:

$$\varphi \in RP \Leftrightarrow \text{ esiste una derivazione per } \varphi.$$

In altre parole, una funzione è ricorsiva primitiva se e soltanto se è possibile ottenerla tramite successive applicazioni degli schemi di composizione e di ricorsione primitiva a partire dalla funzioni base.

Come esempio, traduciamo nella seguente derivazione in *RP* la definizione ricorsiva di somma data nel paragrafo precedente:

$\varphi_1 = U_1^1$ (funzione base)

$\varphi_2 = s$ (funzione base)

$\varphi_3 = U_3^3$ (funzione base)

$\varphi_4 = \varphi_2 (\varphi_3)$ (da φ_2 e φ_3 per composizione)

$\varphi_5(x, 0) = \varphi_1(x)$

$\varphi_5(x, s(y)) = \varphi_4(x, y, \varphi_5 (x, y))$

(da φ_1, φ_4 e φ_5 per ricorsione primitiva)

dove la funzione φ_5 definisce la somma: nella base $\varphi_5(x, 0)$ dà come risultato x (in quanto $U_1^1(x) = x$); nel passo il valore di $\varphi_5(x, s(y))$ viene definito come il successore di $\varphi_5(x, y)$.

In maniera analoga si può ottenere la derivazione in *RP* delle altre funzioni definite induttivamente nel paragrafo precedente.

Si può constatare che tutte le funzioni ricorsive sono computabili in modo effettivo: lo sono in modo ovvio le funzioni base, e i due schemi di composizione e ricorsione conservano la computabilità effettiva. La derivazione in *RP* di una funzione costituisce inoltre una lista di istruzioni che permette di calcolarne i valori. Come esempio, sommiamo il numero 2 a se stesso utilizzando la derivazione della somma in *RP*. Calcoliamo cioè:

$$\varphi_5(2, 2).$$

Dall'ulltima equazione della derivazione sappiamo che $\varphi_5(x, s(y)) = \varphi_4(x, y, \varphi_5 (x, y))$. Possiamo quindi passare da $\varphi_5(2, 2)$ a:

$$\varphi_4(2, 1, \varphi_5 (2, 1))$$

Applicando ora l'equazione che definisce φ_4 (e saltando un passaggio):

$$s(U_3^3(2,1, \varphi_5(2,1))).$$

Per la definizione di U_3^3 abbiamo allora:

$$s(\varphi_5 (2, 1))$$

Procedendo analogamente, si otterrà:

$$s(\varphi_4(2, 0, \varphi_5 (2, 0)))$$

$$s(s(U_3^3(2{,}0,\ \varphi_5(2{,}0))))$$
$$s(s(\varphi_5\ (2,\ 0)))$$
$$s(s(U_1^1(2)))$$
$$s(s(2))$$
$$s(3)$$
$$4$$

Il fatto che non sia più possibile applicare alcuna equazione costituisce un criterio per stabilire la terminazione del calcolo. Si noti tuttavia che non viene rispettata la richiesta di determinismo necessaria affinché un procedimento di calcolo si possa considerare algoritmico. Al terzo passo sarebbe stato infatti possibile sviluppare, anziché il simbolo di funzione più esterno, quello più interno, ottenendo una sequenza di questo genere:

$$\varphi_5(2{,}2)$$
$$\varphi_4(2{,}1,\ \varphi_5(2{,}1))$$
$$\varphi_4(2{,}1,\ \varphi_4(2{,}0,\ \varphi_5(2{,}0)))$$

e così via.

Può essere tuttavia dimostrato che, data una qualsiasi derivazione in *RP*, qualunque sia la strada seguita nel calcolo, il risultato ottenuto è sempre lo stesso. Si può dunque ovviare alla mancanza di determinismo specificando delle regole di esecuzione che impongano, ad esempio, di iniziare la valutazione dalle parentesi più interne, e procedendo da sinistra verso destra (in questo caso la sequenza di calcolo corretta sarebbe la seconda). Unitamente ad una convenzione di questo genere, ogni derivazione di una funzione in *RP* fornisce un algoritmo per il calcolo della funzione stessa.

Si può inoltre dimostrare che ognuno di tali algoritmi conduce sempre ad un calcolo che termina, fornendo sempre un risultato. Le funzioni ricorsive primitive sono dunque *totali*, associano cioè un valore ad ogni *n*-pla di argomenti. (La dimostrazione può essere condotta per induzione sulla lunghezza delle derivazioni: le funzioni base sono totali in modo ovvio (base della dim.), e si può dimostrare che composizione e ricorsione primitiva conservano la totalità (passo induttivo).)

Vediamo altri esempi di funzioni ricorsive primitive.

Un *predicato P* si dice *ricorsivo primitivo* se e soltanto se la sua funzione caratteristica C_P è ricorsiva primitiva. Si supponga di voler definire una funzione *CondP* che si comporti nel modo seguente:

$$\begin{cases} CondP(x, y_1,..., y_n) = \varphi(y_1,..., y_n) \text{ se } P(x) \\ CondP(x, y_1,..., y_n) = \psi(y_1,..., y_n) \text{ altrimenti (cioè, se } \neg P(x)) \end{cases}$$

(dove $\varphi(y_1,..., y_n)$ e $\psi(y_1,..., y_n) \in RP$, e *P* è un predicato ricorsivo primitivo). È evidente l'analogia di *CondP* con un costrutto condizionale del tipo IF ... THEN ... ELSE ... di un linguaggio di programmazione quale, ad esempio, il PASCAL. Un'istruzione della forma:

IF b THEN S_1 ELSE S_2

ha infatti questo significato intuitivo: se l'espressione booleana \fIb\fR è vera, allora deve essere eseguita l'istruzione S_1; altrimenti deve essere eseguita S_2.

Definiamo tramite ricorsione la seguente funzione ausiliaria:

$$\begin{cases} \theta(y_1,..., y_n, 0) = \varphi(y_1,..., y_n) \\ \theta(y_1,..., y_n, s(x)) = \psi(y_1,..., y_n) \end{cases}$$

Se C_P è la funzione caratteristica di P, definiamo *CondP* come segue:

$$CondP(x,y_1,..., y_n) = \theta\ (y_1,..., y_n, C_P).$$

Questa non è ovviamente una derivazione rigorosa in *RP*; tale derivazione può essere tuttavia facilmente ottenuta con un uso appropriato dello schema di composizione e delle funzioni proiezione.

Altro costrutto familiare in informatica che può essere espresso con gli strumenti della ricorsività primitiva è un'istruzione di tipo iterativo come il FOR del PASCAL. Un'espressione del tipo:

FOR x:=1 TO k DO S

sta ad indicare che l'istruzione S deve essere iterata un numero k prestabilito di volte (la variabile x può comparire in S, ma il suo valore non vi può essere modificato).

Data una funzione $\varphi(x)$ ricorsiva primitiva, definiamo allora una funzione $for_\varphi(x,y)$ che restituisca come il valore il risultato che si ottiene iterando yì volte la funzione φ a partire dall'argomento x; vale a dire:

$$for_\varphi(x,y) = \underbrace{\varphi(...(\varphi}_{y \text{ volte}}(x)...)$$

Questo può essere ottenuto tramite lo schema di ricorsione primitiva:

$$\begin{cases} for_\varphi\ (x,0) = x \\ for_\varphi\ (x,s(y)) = \varphi(for_\varphi\ (x,\ y)) \end{cases}$$

Anche in questo caso è solo questione di pazienza trasformare questa definizione informale in una derivazione in *RP*.

2.3 Limiti della ricorsività primitiva

In questo paragrafo vedremo come la classe delle funzioni ricorsive primitive non esaurisca la classe delle funzioni computabili in modo effettivo: esistono cioè funzioni effettivamente computabili che non appartengono a *RP*.

Storicamente, il primo esempio individuato di funzione computabile in modo effettivo non ricorsiva primitiva è la cosiddetta *funzione di Ackermann*. Essa fu elaborata dal logico W. Ackermann (Ackermann [1928]) sviluppando un'idea intuitiva di David Hilbert (Hilbert [1926]). Si considerino le definizioni ricorsive di somma, prodotto ed esponenziazione riportate nel § 2.1. La struttura di tali definizioni è simile, ed in ognuna di esse nel passo induttivo viene utilizzata la funzione definita immediatamente prima (nel caso della somma viene utilizzata la funzione base successore). La serie può essere proseguita indefinitamente in modo analogo: se indichiamo con f_0 la somma, con f_1 il prodotto e con f_2 l'esponenziazione, si può definire una funzione f_3 in questo modo:

$$\begin{cases} f_3\ (x,0) = x \\ f_3\ (x,s(y)) = f_2(f_3\ (x,\ y), x) \end{cases}$$

e, analogamente, per un generico n:

$$\begin{cases} f_n\ (x,0) = x \\ f_n\ (x,s(y)) = f_{n-1}(f_n\ (x,\ y), x) \end{cases}$$

Il prodotto cresce più rapidamente della somma, e l'esponenziazione più del prodotto. In generale, si può dimostrare che, per ogni n, la funzione f_n della serie cresce più rapidamente della funzione f_{n-1}. Si definisca una funzione in tre variabili, $\varphi(x, y, z)$, tale che, per ogni $x, y, z \in \mathbb{N}$:

$$\varphi(x, y, z) = f_z (x, y).$$

Per come la serie delle f_n è stata definita, la costruzione di ogni funzione della serie dato il suo indice è effettuabile in modo algoritmico. Inoltre ogni f_n è effettivamente computabile. Quindi la funzione $\varphi(x, y, z)$ risulta a sua volta computabile in maniera effettiva. Essa tuttavia cresce più rapidamente di ogni funzione della serie e quindi non vi può appartenere. Ackermann dimostrò che φ cresce più rapidamente di ogni funzione ricorsiva primitiva, e che quindi non è ottenibile tramite ricorsione primitiva. È possibile definire φ per ricorsione doppia, vale a dire mediante una ricorsione che proceda contemporaneamente su due variabili. Riportiamo la definizione di φ data da Ackermann. Definiamo innanzi tutto la funzione ausiliaria $\alpha \in PR$:

$$\begin{cases} \alpha(x,0) = 0 \\ \alpha(x,1) = 1 \\ \alpha(x, y) = x \ \ (\text{per } y > 1) \end{cases}$$

La definizione di φ è la seguente:

$$\begin{cases} \varphi(x, y, 0) = x + y \\ \varphi(x, 0, s(z)) = \alpha(x, z) \\ \varphi(x, s(y), s(z)) = \varphi(x, \varphi(x, y, s(z)), z) \end{cases}$$

φ è definita per ricorsione sulla variabile y e sulla variabile z.

Una dimostrazione alternativa a quella di Ackermann dell' esistenza di funzioni computabili in modo effettivo ma non ricorsive primitive è dovuta alla logica ungherese Rosza Peter. Riportiamo per esteso tale dimostrazione sia per la sua semplicità, sia perché si basa sul cosiddetto *metodo di diagonalizzazione*, ampiamente usato in teoria della ricorsività.

Il metodo di diagonalizzazione fu ideato da Georg Cantor per dimostrare che l'insieme dei numeri reali non ha la stessa cardinalità dell'insieme dei numeri naturali. Poiché dimostrare che due insiemi hanno cardinalità diverse è equivalente a dimostrare che non esiste alcuna biezione fra essi, Cantor dimostrò il seguente

TEOREMA: non esiste alcuna biezione $h: \mathbb{N} \rightarrow \mathbb{R}$

Dimostrazione: per assurdo, si supponga che tale biezione h esista; si supponga cioè che la successione:

$$h(0), h(1), ... , h(n), ...$$

includa tutti i numeri reali, senza ripetizioni. Ogni numero reale $h(k)$ sia scritto nell'usuale notazione decimale. Si costruisca un numero reale r che abbia 0 come parte intera, e la parte decimale costruita in questo modo: l'n-sima cifra a destra della virgola sia 0\ se l'n-sima cifra a destra della virgola di $h(n)$ì è diversa da \0 ; sia 1 altrimenti. Se, ad esempio, h si comportasse come in fig. 2.1, si avrebbe r = 0,010 ... 0 ... Il numero r così costruito differisce da ogni numero reale della sequenza per almeno una cifra a destra della virgola, e quindi non può appartenere ad essa. Ciò contraddice l'ipotesi di assurdo. Q.E.D.

È chiaro dalla figura che il termine *diagonalizzazione* deriva dal fatto che il controesempio r è costruito a partire dalle cifre che si trovano lungo la diagonale dello schema.

Per applicare la diagonalizzazione alla classe delle funzioni ricorsive primitive si tenga presente quanto segue. Una funzione appartiene a *RP sse* ne esiste una derivazione a partire dalle funzioni base. È possibile enumerare in modo effettivo le derivazioni di *RP*; è possibile cioè

metterle in corrispondenza biunivoca con l'insieme dei numeri naturali, in maniera che, dato qualsiasi numero naturale, si possa ottenere in maniera algoritmica la derivazione corrispondente. Una tale enumerazione si può ottenere, ad esempio, con una tecnica analoga a quella con cui si enumerano i teoremi di una teoria assiomatizzata: si inizi col considerare la derivazione consistente solamente nella prima delle funzioni base, quindi le derivazioni ottenibili da essa tramite una sola applicazione degli schemi: si prenda poi in considerazione la seconda funzione base e le derivazioni ottenibili tramite una applicazione degli schemi da quest'ultima e/o dalle funzioni corrispondenti alle derivazioni già ottenute; si passi poi alla terza funzione base, e così via. In questo modo si ottiene ad ogni passo un numero finito di derivazioni, e nessuna derivazione viene "lasciata indietro". L'enumerazione si ottiene specificando opportunamente l'ordine con cui applicare gli schemi a partire dalle funzioni derivate ai passi precedenti.

Disponendo di una enumerazione effettiva di tutte le derivazioni, si dispone allo stesso tempo di una enumerazione effettiva (con ripetizioni) di tutte le funzioni ricorsive primitive. In maniera analoga, è inoltre possibile ottenere una enumerazione effettiva di tutte le funzioni ricorsive primitive con un numero prefissato di argomenti. Fissata un'enumerazione effettiva, sia φ_n l'n-esima funzione dell'enumerazione.

Utilizzando il procedimento diagonale, dimostriamo che:

TEOREMA: esiste almeno una funzione computabile che non appartiene a *RP*.

Dimostrazione: Si costruisca una enumerazione effettiva di tutte le funzioni ricorsive primitive ad un argomento. Si consideri quindi la funzione $\psi(x)$ definita come segue:

$$\psi(x) = \varphi_x(x) + 1.$$

Dato il carattere effettivo dell'enumerazione, preso qualsiasi $n \in \mathbf{N}$ è possibile ottenere algoritmicamente la funzione φ_n ad esso corrispondente. Inoltre, ogni funzione ricorsiva primitiva (quindi ogni φ_n della enumerazione) è effettivamente computabile, così come lo è il passaggio al successore. La funzione $\psi(x)$ quindi, in quanto ottenuta per composizione da funzioni effettivamente computabili, è a sua volta effettivamente computabile. Tuttavia $\psi \notin RP$. Si supponga infatti per assurdo che $\psi \in RP$. Essa è una funzione ad un argomento, e deve quindi comparire nell'enumerazione, deve cioè esistere $z \in \mathbf{N}$ t.c., per ogni x:

$$\psi(x) = \varphi_z(x).$$

Ma, volendo calcolare il valore di φ con argomento z, si otterrebbe:

$$\psi(z) = \varphi_z(z) + 1 = \psi(z) + 1$$

il che è assurdo.Q.E.D.

$\psi(x)$ differisce da ciascuna $\varphi_n(x)$ dell'enumerazione almeno per il valore assunto per l'argomento n. Per ogni n infatti:

$$\varphi_n(n) \neq \psi(n) = \varphi_n(n) + 1$$

L'analogia con la dimostrazione di Cantor è evidente. Si consideri la tabella in fig. 2.2. Come nel caso del controesempio r, i valori che sicuramente contraddistinguono ψ da ogni funzione dell'enumerazione sono quelli che si trovano lungo la diagonale.

3. Le funzioni ricorsive generali

3.1 L'operatore di minimalizzazione [illegible] la ricorsività generale

In questo capitolo vedremo come sia possibile estendere la classe delle funzioni ricorsive primitive ad una classe più ampia di funzioni, la classe delle *funzioni ricorsive generali (parziali)*, in maniera da evitare i controesempi descritti nell'ultimo paragrafo del capitolo precedente.

Defininiamo innanzi tutto l'*operatore di minimalizzazione (illimitato)* μ. Intuitivamente, il significato di un'espressione del tipo μy può essere reso come: "il più piccolo y tale che ... ". Così, data una relazione $R(x_1 , ... , x_n, y)$ a n+1 argomenti, l'espressione $\mu y R(x_1 , ... , x_n, y)$ significa: "il più piccolo y che è nella relazione R con $(x_1 , ... , x_n)$". Se la relazione R è effettivamente decidibile (se cioé per ogni n+1-pla $(x_1 , ... , x_n, y)$ è possibile stabilire mediante un algoritmo se $x_1 , ... , x_n$ e y sono o no nella relazione R), allora $\mu y R(x_1 , ... , x_n, y)$ è una funzione parziale effettivamente calcolabile a n argomenti, e l'algoritmo che la calcola può essere schematizzato come in fig. 3.1. Che si tratti di una funzione parziale dipende dal fatto che non è detto che per ogni n-pla $(x_1 , ... , x_n,)$ esista un y tale che $R(x_1 , ... , x_n, y)$. Se tale y non esiste, l'algoritmo di fig. 3.1 entra in *loop*, e la funzione diverge.

Definiamo la classe delle *funzioni ricorsive generali (parziali)* come la più piccola classe di funzioni da $\mathbf{N}^k$ a $\mathbf{N}$ (con k = 1, 2, ...) tale che le appartengano le seguenti *funzioni base*:

1) *funzione zero*

2) *funzione successore*

3) *funzioni proiezione*

definite come nel caso di *RP*; che sia chiusa rispetto agli schemi di 4) *composizione* e 5) *ricorsione primitiva*, definiti anch'essi come per *RP*, ed inoltre tale che:

6) se la funzione $\chi(x_1 , ... , x_n, y)$ è ricorsiva generale (parziale), allora la funzione: $\varphi(x_1 , ... , x_n,) = \mu y(\chi(x_1 , ... , x_n, y) = 0)$ è anch'essa ricorsiva generale (parziale) (chiusura rispetto all'operazione di *minimalizzazione*).

Si noti che una funzione $\varphi(x_1 , ... , x_n)$ definita per minimalizzazione a partire da $\chi(x_1 , ... , x_n, y)$ è, in generale, parziale anche nel caso che $\chi(x_1 , ... , x_n, y)$ sia totale. Infatti non è detto che, per ogni $x_1 , ... , x_n$, esista y tale che: $\chi(x_1 , ... , x_n, y) = 0$ (in tal caso $\varphi(x_1 , ... , x_n) = \perp$). Inoltre, se esiste y tale che: $\chi(x_1 , ... , x_n, y) = \perp$ e, per ogni $y' < y$, $\chi(x_1 , ... , x_n, y')$ converge ed è $\neq 0$, allora $\varphi(x_1 , ... , x_n) = \perp$.

La definizione di *derivazione* resta identica a quella data per *RP*, salvo il fatto che in una derivazione una funzione può essere ottenuta dalle precedenti anche tramite minimalizzazzione.

Analogamente a quanto accade in \fIRP\fI, si ha che, per ogni funzione φ:

$$\varphi \text{ è ricorsiva generale (parziale)} \Leftrightarrow \text{esiste una derivazione per } \varphi.$$

È ovvio che tutte le funzioni ricorsive primitive sono anche ricorsive generali, e che ogni funzione ricorsiva generale è effettivamente calcolabile. Come in *RP*, la derivazione di ogni funzione ricorsiva generale fornisce implicitamente un algoritmo per poterne calcolare i valori.

Vediamo ora come il metodo della diagonalizzazione non consenta di trovare un controesempio per la classe delle funzioni ricorsive generali, come cioè non sia possibile definire tramite diagonalizzazione una funzione effettivamente computabile non ricorsiva generale.

Premettiamo che, nel caso di funzioni parziali, $\varphi(x) = \psi(y)$ può indicare sia che entrambe le funzioni convergono per gli argomenti dati e producono lo stesso valore, sia che entrambe sono indefinite, cioè che:

$$\varphi(x) = \perp = \psi(y)$$

La classe delle funzioni ricorsive generali (parziali) è enumerabile in modo effettivo, con la stessa tecnica descritta per *RP*. Analogamente, è possibile costruire una enumerazione effettiva di tutte le funzioni ricorsive generali (parziali) ad un argomento. Sia φ_n l'n-esima funzione di una tale enumerazione. Anche in questo caso, la funzione $\psi(x)$ tale che, per ogni x:

$$\psi(x) = \varphi_x(x) + 1$$

è computabile in modo effettivo. Tuttavia, la sua appartenenza alla classe delle funzioni ricorsive generali (parziali) non comporta alcuna contraddizione. Sia infatti z l'indice di ψ, si abbia cioè, per ogni x:

$$\psi(x) = \varphi_z(x).$$

Anche in questo caso:

$$\psi(z) = \varphi_z(z) + 1 = \psi(z) + 1.$$

Nel caso di funzioni parziali, ciò non implica tuttavia una contraddizione, ma indica semplicemente che ψ è indefinita per l'argomento z. Infatti, poiché per calcolare $\psi(z) + 1$ bisogna avere prima calcolato $\psi(z)$, se il calcolo di $\psi(z)$ non termina, non può terminare neppure il calcolo di $\psi(z) + 1$. Quindi, se $\psi(z) = \perp$, allora $\psi(z)+1 = \perp = \psi(z)$.

Che ψ sia effettivamente indefinita per l'argomento z è evidente se si considera l'algoritmo raffigurato in fig. 3.2, che ne calcola i valori. Nel caso che l'input sia z, al blocco (*) del diagramma corrisponde a sua volta l'intero diagramma, e così via all'infinito.

Inoltre, è possibile dimostrare in maniera diretta che la funzione ψ è ricorsiva generale (parziale).

È possibile modificare la definizione della classe delle funzioni ricorsive generali in modo da restringerla a tutte e sole le funzioni ricorsive generali totali. Ciò si ottiene aggiungendo allo schema di minimalizzazione la clausola per cui, per la funzione χ sottopposta a minimalizzazione, deve esistere, per ogni n-pla $x_1, \ldots, x_n$ di argomenti, almeno un y tale che $\chi(x_1, \ldots, x_n, y) = 0$. Non è tuttavia possibile disporre di una enumerazione effettiva per la classe di funzioni così ottenuta. Anche in questo caso non è quindi possibile costruire un controesempio mediante diagonalizzazione.

3.2 Insiemi e predicati ricorsivi generali

Un *insieme* si dice *ricorsivo generale sse* la sua funzione caratteristica è ricorsiva generale (totale).

Un *insieme A* si dice *ricorsivamente enumerabile sse* $A = \varnothing$ oppure A è il rango di una funzione φ ricorsiva generale totale.

Un *predicato P* si dice *ricorsivo generale* (*ricorsivamente enumerabile*) *sse* l'insieme

$$A_P \equiv \{x \mid P(x)\}$$

è ricorsivo generale (ricorsivamente enumerabile).

In analogia con quanto dimostratro per insiemi e predicati decidibili o effettivamente enumerabili, si possono dimostrare le seguenti proposizioni.

PROPOSIZIONE: *A è ricorsivamente enumerabile* $\Leftrightarrow$ *A è il dominio di definizione di una funzione ricorsiva generale (parziale)* $\Leftrightarrow$ *A è il rango di una funzione ricorsiva generale (parziale).*

Sia $\overline{A}$ l'insieme complemento di A rispetto a $\mathbf{N}$, vale a dire:

$$\overline{A} \equiv \{ x \in \mathbf{N} \mid x \notin A \}$$

Vale allora la seguente:

PROPOSIZIONE: $\overline{A}$ *è ricorsivo generale* $\Leftrightarrow$ *A è ricorsivo generale* $\Leftrightarrow$ *A e* $\overline{A}$ *sono ricorsivamente enumerabili.*

Vale inoltre che:

PROPOSIZIONI: *A e B sono ricorsivi generali (ricorsivamente enumerabili)* $\Rightarrow A \cup B$ *è ricorsivo generale (ricorsivamente enumerabile).*

A e B sono ricorsivi generali (ricorsivamente enumerabili) $\Rightarrow A \cap B$ *è ricorsivo generale (ricorsivamente enumerabile).*

Analogamente, nel caso di predicati, vale quanto segue.

PROPOSIZIONE: *P è ricorsivamente enumerabile* $\Leftrightarrow$ *esiste* φ *ricorsiva generale (parziale) tale che, per ogni x,* $\varphi(x) \neq \bot$ *sse* $P(x)$ $\Leftrightarrow$ *esiste* ψ *ricorsiva generale (parziale) tale che: rango di* $\psi \equiv \{x \mid P(x)\}$.

Sia $\overline{P}$ il predicato tale che, per ogni x:

$$\overline{P}(x) \text{ sse } \neg P(x).$$

Allora:

PROPOSIZIONE: $\overline{P}$ *è ricorsivo generale* $\Leftrightarrow$ *P è ricorsivo generale* $\Leftrightarrow$ *P e* $\overline{P}$ *sono ricorsivamente enumerabili.*

Dati i predicati P e Q, sia R il predicato tale che, per ogni x:

$$R(x) \textit{ sse } P(x) \wedge Q(x)$$

Allora:

PROPOSIZIONE: *P e Q sono ricorsivi generali (ricorsivamente enumerabili)* $\Rightarrow$ *R è ricorsivo generale (ricorsivamente enumerabile).*

Sia R' il predicato tale che, per ogni x:

$$R'(x) \textit{ sse } P(x) \vee Q(x)$$

Allora:

PROPOSIZIONE: *P e Q sono ricorsivi generali (ricorsivamente enumerabili)* $\Rightarrow$ *R' è ricorsivo generale (ricorsivamente enumerabile).*

4. Le macchine di Turing

4.1 Il concetto di macchina di Turing

Turing affrontò il problema di fornire un equivalente rigoroso del concetto intutivo di algoritmo analizzando il comportamento di un essere umano che stia eseguendo un calcolo, e simulandolo mediante l'elaborazione di una macchina calcolatrice astratta. Tali macchine vennero dette in seguito *macchine di Turing*.

Seguiamo tale analisi come viene condotta da Turing stesso nell'articolo "On computable numbers, with an application to the Entscheidungsproblem" [1936-37], dove il concetto di macchina di Turing viene formulato per la prima volta. Un calcolo, osserva Turing, consiste nell'operare su di un certo insieme di simboli scritti su di un supporto fisico, che in genere è costituito da un foglio di carta. Per i fini che ci si propongono, il fatto che abitualmente venga usato un supporto bidimensionale è inessenziale. Si può quindi immaginare, senza nulla perdere in generalità, che la nostra macchina calcolatrice utilizzi per la "scrittura" un *nastro* unidimensionale di lunghezza virtualmente illimitata in entrambe le direzioni (tuttavia, come vedremo, in ogni fase del calcolo la macchina potrà disporre soltanto di una porzione finita di esso). Tale nastro sia inoltre suddiviso in celle, in "quadretti", "come un quaderno di aritmetica per bambini", ciascuna delle quali potrà ospitare un solo simbolo alla volta (cfr. fig. 4.1).

Quanto ai simboli da utilizzare per il calcolo, ogni macchina potrà disporre soltanto di un insieme finito di essi, che chiameremo l'*alfabeto* di quella macchina. Sia dunque $\Sigma \equiv \{s_1, s_2, \ldots, s_n\}$ l'alfabeto di una generica macchina di Turing. Ogni cella del nastro potrà contenere uno di tali simboli, oppure, in alternativa, restare vuota (indicheremo con s_0 la cella vuota). Il fatto che l'alfabeto di cui si può disporre sia finito non costituisce comunque una grave limitazione. È infatti sempre possibile rappresentare un nuovo simbolo mediante una sequenza finita di simboli dell'alfabeto, ed avere così la possibilità di esprimere un numero virtualmente infinito di simboli (come avviene usualmente nella numerazione decimale mediante cifre arabe).

Vi è senza dubbio un limite al numero di simboli che un essere umano può osservare senza spostare lo sguardo sul foglio su cui sta lavorando. Per semplicità, supporremo quindi che la macchina possa esaminare soltanto una cella alla volta, ed "osservare" quindi al più un singolo simbolo. A tal fine la macchina sarà dotata di una *testina di lettura*, che sarà posizionata, in ogni fase del calcolo, su di una singola cella (cfr. fig. 4.1). Essa, per poter accedere alle altre celle del nastro, dovrà quindi spostarsi verso destra o verso sinistra. Chi sta eseguendo un calcolo ha poi la possibilità di scrivere nuovi simboli, di cancellare quelli già scritti o di sostituirli con altri. La testina eseguirà anche tale compito di cancellazione e di scrittura. Anche in questo caso però essa potrà agire soltanto sulla cella "osservata", e, per accedere ad altre celle, dovrà prima spostarsi lungo il nastro. Poiché ogni cella può contenere un solo simbolo, scrivendo un nuovo simbolo in una cella il simbolo eventualmente presente in essa si deve ritenere cancellato.

Nell'eseguire un calcolo, un essere umano tiene conto delle operazioni già eseguite e dei simboli osservati mediante la propria memoria, cambiando cioè il proprio "stato mentale". Al fine di simulare ciò, supporremo che una macchina possa assumere, in dipendenza dagli eventi precedenti del processo di calcolo, un certo numero di *stati interni* (uno e non più di uno alla volta), che corrispondano agli "stati mentali" dell'essere umano. Tali stati saranno in numero finito, poiché "se ammettessimo un'infinità di stati di mente, alcuni di essi sarebbero 'arbitrariamente prossimi', e sarebbero quindi confusi" (Turing [1936-37], in Davis [1965], p. 136). Il limitarsi ad un numero finito di stati non costituisce tuttavia un vincolo, in quanto "l'uso di stati mentali più complicati può essere evitato scrivendo più simboli sul nastro" (ibid.). Siano allora $q_0, q_1, \ldots, q_m$ gli stati che una generica macchina di Turing può assumere. Nella rappresentazione grafica indicheremo sotto la testina di lettura/scrittura lo stato della macchina nella fase di calcolo rappresentata. Definiamo

configurazione di una macchina di Turing in una data fase di calcolo la coppia costituita dallo stato interno che essa presenta in quel momento e dal simbolo osservato dalla testina (la configurazione della macchina raffigurata nella fig. 4.1 è dunque (q_i, s_j)). Diremo inoltre *situazione* la configurazione unita alla descrizione completa del nastro.

Una macchina di Turing può dunque eseguire operazioni consistenti in spostamenti della testina lungo il nastro, scrittura e cancellazione di simboli, mutamenti dello stato interno. Immaginiamo di scomporre tali operazioni in un numero di *operazioni atomiche*, tali da non poter essere ulteriormente scomposte in operazioni più semplici. Nel tipo di macchina descritto ogni operazione può essere scomposta in un numero finito delle operazioni seguenti: (1) sostituzione del simbolo osservato con un altro simbolo (eventualmente con s_0; in tal caso si ha la cancellazione del simbolo osservato), e/o (2) spostamento della testina su di una delle celle immediatamente attigue del nastro. Ognuno di tali atti può inoltre comportare (3) un cambiamento dello stato interno della macchina. Nella sua forma più generale, ogni operazione atomica dovrà quindi consistere di un operazione di scrittura e/o di uno spostamento atomico, ed eventualmente di un mutamento di stato. Indicheremo d'ora in avanti rispettivamente con le lettere *L*, *R* e *C* il fatto che una macchina debba eseguire uno spostamento (di una cella) verso sinistra (*Left*), verso destra (*Right*), oppure non debba eseguire alcuno spostamento (*Center*). Grazie a ciò potremo rappresentare ogni operazione atomica mediante una terna, il primo elemento della quale starà ad indicare il simbolo che deve essere scritto sulla cella osservata, il secondo che spostamento deve essere eseguito (*L*, *R* o *C*), il terzo infine lo stato che la macchina deve assumere alla fine dell'operazione. Ad esempio, la terna:

$$s_i \; L \; q_j$$

significa che la macchina deve scrivere il simbolo s_i sulla cella osservata, spostarsi di una cella a sinistra, ed assumere infine lo stato q_j. Invece la terna:

$$s_0 \; C \; q_p$$

significa che la macchina deve cancellare il simbolo osservato, non eseguire alcun movimento ed assumere lo stato q_p.

Ogni singola macchina di Turing deve essere attrezzata per eseguire un tipo di calcolo specifico, deve cioè disporre di una serie di regole, di istruzioni, che le permettano di eseguire il compito per il quale è stata progettata. In un calcolo algoritmico ogni passo deve essere completamente determinato dalla situazione precedente. Nel caso di un calcolatore umano, ogni sua mossa deve dipendere esclusivamente dal ricordo delle operazioni già eseguite e dai simboli che egli può osservare. Analogamente, in una macchina di Turing, ogni mossa deve essere determinata esclusivamente dallo stato attuale e dal simbolo nella cella osservata, vale a dire, dalla configurazione della macchina. Ogni istruzione deve specificare quale operazione atomica deve essere eseguita a partire da una deteminata configurazione. Poiché ogni configurazione è rappresentabile mediante una coppia, ed ogni operazione atomica mediante una terna, una istruzione potrà essere rappresentata mediante una *quintupla*, i primi due elementi della quale (uno stato interno ed un simbolo dell'alfabeto) indicheranno la configurazione di partenza, mentre gli ultimi tre elementi specificheranno l'operazione che deve essere eseguita. Ad esempio, la quintupla:

$$q_i \; s_j \; s_{j'} \; L \; q_{i'}$$

significa che, qualora la macchina si trovi nello stato q_i ed il simbolo osservato sia s_j, allora il simbolo $s_{j'}$ dovrà essere scritto al posto di s_j, la testina dovrà spostarsi di una cella verso destra e la macchina dovrà assumere lo stato $q_{i'}$ [2].

---

[2] Si noti che si potrebbero definire le operazioni atomiche in modo da comprendere un'operazione di scrittura o uno spostamento, ma non entrambi. In tal caso ogni operazione atomica sarebbe rappresentata da una coppia, ed ogni

Le istruzioni di cui dispone ogni singola macchina di Turing per eseguire il calcolo per il quale è stata progettata avranno quindi la forma di un opportuno insieme di quintuple (che verrà detto la *tavola* di quella macchina di Turing). Una volta fissato l'alfabeto, cio' che caratterizza ogni singola macchina di Turing rispetto a tutte le altre è appunto la tavola delle sue quintuple. Affinché un insieme di quintuple costituisca la tavola di una macchina di Turing è indispensabile che venga rispettata la seguente condizione. Poiché il calcolo deve esserre deterministico, a partire da una singola configurazione non devono essere applicabili istruzioni diverse. Il che corrisponde alla condizione che, nella tavola di una macchina, non possano comparire più quintuple con i primi due elementi uguali.

Affinché il calcolo possa terminare, è necessario che ad alcune delle configurazioni possibili non corrisponda alcuna quintupla, altrimenti, qualunque fosse il risultato di una mossa, esisterebbe sempre un'altra mossa che ad essa dovrebbe far seguito. Chiameremo tali configurazioni *configurazioni finali*. Che la tavola di una macchina di Turing comprenda configurazioni finali è una condizione necessaria ma non sufficiente perché la macchina termini il calcolo. Si consideri ad esempio la macchina di Turing seguente:

$$q_1\ s_0\ s_0\ R\ q_1$$

$$q_1\ /\ s_0\ C\ q_2$$

($q_2\ s_0$) è una configurazione finale; se tuttavia questa macchina viene attivatata col nastro completamente vuoto, il suo calcolo andrà avanti all'infinito. Data una macchina di Turing, è sempre possibile costruirne un'altra che esegua lo stesso calcolo, per la quale esista uno specifico stato interno che compare in tutte e sole le configurazioni finali. Chiameremo tale stato *stato finale*, e stabiliremo convenzionalmente di riservare ad esso il simbolo q_0. Data una macchina di Turing generica, per trasformarla in una che abbia q_0 come stato finale si proceda nel modo seguente. Sia (q_n, s_m) una generica configurazione finale della macchina di partenza. La tavola della nuova macchina si ottiene aggiungendo tutte le quintuple del tipo:

$$q_n\ s_m\ s_m\ C\ q_0$$

In ogni caso in cui la macchina di partenza giungeva in uno stato finale, la nuova macchina farà un'ulteriore mossa, assumendo lo stato q_0 (e lasciando inalterato tutto il resto).

I dati vengono forniti a una macchina di Turing sotto forma di una sequenza finita di simboli dell'alfabeto scritti sul nastro prima dell'inizio del calcolo (chiameremo *input* tale sequenza di simboli). Il risultato è costituito invece da ciò che è scritto sul nastro al momento dell'arresto (questo è l'*output* della macchina). Stabiliamo convenzionalmente che all'inizio del calcolo la testina debba essere collocata in *posizione standard* (vale a dire, in corrispondenza del primo simbolo a sinistra dell'input), e che lo stato interno della macchina debba essere q_1.

Vediamo un semplice esempio di macchina di Turing. Si consideri l'alfabeto $\Sigma \equiv \{|\}$, composto come unico simbolo da una barra verticale. Definiamo una macchina che, presa come input una successione di barre consecutive, restituisca come output tale successione aumentata di

---

istruzione da una quadrupla. Data un insieme di quintuple, è sempre possibile trasformarle in quadruple aggiungendo nuovi stati "intermedi". Ad esempio, la quintupla riportata nel testo può essere ricondotta alla due quadruple seguenti:

$$q_i\ s_j\ s_{j'}\ L\ q_{i''}$$

$$q_{i''}\ s_{j'}\ L\ q_{i'}$$

Ovviamente, si deve fare attenzione che i nuovi stati (in questo caso $q_{i''}$) siano tutti diversi fra loro e diversi da quelli che compaiono nell'insieme di quintuple di partenza. Il passaggio inverso, da quadruple a quintuple, è banale (ad esempio, una quadrupla $q_n\ s_m\ s_{m'}\ q_{n'}$ diventerà $q_n\ s_m\ s_{m'}\ C\ q_{n'}$; ed una quadrupla $q_n\ s_m\ R\ q_{n'}$ diventerà: $q_n\ s_m\ s_m\ R\ q_{n'}$). L'utilizzo di quintuple o di quadruple non comporta quindi alcuna modifica nella potenza computazionale di una macchina.

un elemento. A tal fine è sufficiente disporre del solo stato interno q_1 (oltre allo stato finale q_0); la tavola della macchina sarà la seguente:

$$q_1 \,|\, |\, R\, q_1$$
$$q_1\, s_0 \,/\, C\, q_0$$

Secondo le convenzioni stabilite, alla partenza la testina deve essere collocata sul primo simbolo a sinistra dell'input, e lo stato di partenza deve essere q_1 (fig. 4.2). Fintanto che la testina trova celle segnate con | allora, in virtù della prima quintupla, viene riscritto | sulla cella osservata (cioè, vengono lasciate le cose come stanno), e la testina si sposta a destra di una cella mantenendo lo stato q_1 (fig. 4.3). Quando la testina osserva una cella vuota viene attivata la seconda quintupla, in virtù della quale la macchina deve segnare con una barra la cella osservata, non eseguire alcuno spostamento, ed assumere lo stato finale q_0 (fig. 4.4).

4.2 Le funzioni T-computabili

Sin qui abbiamo considerato macchine di Turing che eseguono calcoli su alfabeti di simboli generici. Vediamo ora come si possano codificare i numeri naturali in modo da definire macchine di Turing che calcolino funzioni aritmetiche. Utilizziamo come alfabeto $\Sigma \equiv \{|\}$ (si potrebbe pensare che l'uso di un alfabeto di un solo simbolo comporti un decremento nella potenza di calcolo delle macchine; vedremo invece che la classe delle funzioni computabili dalle macchine di Turing è indipendente dal numero dei simboli dell'alfabeto). I numeri naturali vengono codificati come segue. Al numero 0 corrisponde la sequenza composta da una sola barra. In generale, ogni numero n viene codificato da una sequenza di $n + 1$ barre. Una n-pla di numeri naturali $(k_1, \ldots, k_n)$ viene codificata sul nastro scrivendo la sequenza di barre corrispondente ad ogni k_i (con $1 \leq i \leq n$), e lasciando una cella vuota come separatore fra ognuna di tali sequenze. Ad esempio, la terna (4, 1, 0) viene codificata come in fig. 4.5.

Diremo che una macchina M_φ *computa* una funzione φ a n argomenti (con $n \geq 1$) *sse* quanto segue vale per ogni n-pla $(x_1, \ldots, x_n)$ di numeri naturali. Sia $(x_1, \ldots, x_n)$ codificata nel modo sopra descritto e collocata in posizione standard rispetto alla testina (essendo vuota ogni altra cella del nastro). Allora $\varphi(x_1, \ldots, x_n) = y$ *sse*, al termine del calcolo, l'output di M_φ è costituito dalla codifica di y, e $\varphi(x_1, \ldots, x_n) = \perp$ *sse* il calcolo di M_φ non termina.

Diremo che una funzione φ è *T-computabile sse* esiste una macchina di Turing M_φ che la computa.

4.3 Equivalenza fra T-computabilità e ricorsività generale

Si tratta ora di individuare quali funzioni sono comprese nella classe delle funzioni T-computabili. Che si tratti di funzioni computabili in modo effettivo consegue in modo ovvio dalla definizione. Si dimostra che:

TEOREMA: sono T-computabili tutte e sole le funzioni ricorsive generali (parziali); tale risultato fu dimostrato per la prima volta da Turing stesso (Turing [1936-37] e [1937]).

Presentiamo una traccia della *dimostrazione*. Essa si divide in due parti:

(1) dimostrazione della T-computabilità delle funzioni ricorsive generali;

(2) dimostrazione della ricorsività generale delle funzioni T-computabili.

(1) La prima parte del teorema si dimostra per induzione nel modo seguente: (i) dimostrazione della T-computabilità delle funzioni base (base dell'induzione); (ii) i tre schemi di composizione, ricorsione primitiva e minimalizzazione conservano la T-computabilità (passo indutivo). Vale a dire, si dimostra che, per ognuna delle funzioni base, esiste una macchina di Turing che la computa (i), e che, date delle generiche funzioni T-computabili, è possibili costruire una macchina di Turing che computi ogni funzione ottenibile da esse tramite composizione, ricorsione primitiva e minimalizzazione (ii).

(2) Ricordiamo che in ogni fase del calcolo la *situazione* di una macchina di Turing consiste nella sua configurazione, unitamente alla descrizione completa del nastro. Si tenga presente che in ogni fase del calcolo soltanto una porzione finita del nastro risulta scritta. È dunque possibile rappresentare la situazione di una macchina mediante una tecnica analoga alla gödelizzazione. Si consideri l'esempio di situazione di una macchina di Turing rappresentato in fig. 4.6 (dove si assume che a destra e a sinistra dei simboli rappresentati il nastro sia completamente bianco). Il simbolo s_0 indica le eventuali celle vuote all'interno della parte scritta del nastro. Si noti che, nell'esempio, l'alfabeto è composto da più di un simbolo. Questa parte della dimostrazione è infatti effettuabile qualunque sia l'alfabeto impiegato. Descriviamo la parte scritta del nastro a sinistra della testina tramite il seguente numero di Gödel:

$$u = 5^1 * 3^0 * 2^2$$

Analogamente, il gödeliano della parte destra sarà:

$$v = 2^0 * 3^1$$

Si può ora descrivere l'intera situazione tramite il gödeliano:

$$2^u * 3^1 * 5^6 * 7^v$$

(dove gli esponenti di 2 e di 7 sono i gödeliani delle parti del nastro rispettivamente a sinistra e a destra della testina, l'esponente di 3 è l'indice del simbolo nella cella osservata, e l'esponente di 5 è l'indice dello stato).

La situazione di una macchina di Turing contiene, unitamente alla tavola delle quintuple, tutte le informazioni necessarie per il passaggio alla situazione successiva (qualora, ovviamente, essa esista). Conoscendo infatti la configurazione e la descrizione del nastro è possibile sitabilire, in base alle quintuple, configurazione e stato del nastro al passo successivo. Poiché per mezzo della gödelizzazione si possono rappresentare le situazioni sotto forma di numeri naturali, è possibile definire, per ogni macchina di Turing M, una funzione aritmetica ρ_M tale che:

$$\rho_M(w) = \begin{cases} w' \text{ (dove } w' \text{ è il gödeliano della situazione successiva a quella} \\ \text{rappresentata da } w) \textit{ sse } w \text{ è il gödeliano di una situazione non terminale;} \\ \\ w \text{ altrimenti} \end{cases}$$

Che ρ_M sia effettivamente una funzione è garantito dal determinismo delle macchine di Turing. Si dimostra che ρ_M è ricorsiva primitiva. Definiamo la funzione $\theta_M(w, z)$ come segue:

$$\begin{cases} \theta_M(w, 0) = w \\ \theta_M(w, s(z)) = \rho_M(\theta_M(w, z)) \end{cases}$$

Vale a dire: se w è il gödeliano di una certa situazione di M, allora $\theta_M(w, z)$ è il gödeliano della situazione che si ottiene dopo z mosse di calcolo a partire da w (se si giungesse in una situazione finale dopo un numero di mosse minore di z, allora, per come è definita ρ_M, il risultato di $\theta_M(w, z)$ sarebbe il gödeliano di tale situazione finale). Si noti che la funzione $\theta_M(w, z)$, essendo stata definita per ricorsione primitiva da ρ_M, è anch'essa ricorsiva primitiva.

Il seguito della dimostrazione può essere riassunto come segue. Data una macchina di Turing *M* che computa una certa funzione e dato il suo input, si tratta di ricavare, applicando l'operatore di minimalizzazione a partire da θ_M, il minimo numero z (cioè il minimo numero di passi) per cui si ottiene, a partire dal gödeliano della situazione di partenza, il numero di Gödel di una situazione finale, e ricavare quindi da esso il valore della funzione computata. Tutto ciò può essere ottenuto utilizzando, assieme all'operatore di minimalizzazione, opportune funzioni ricorsive primitive. Per ogni macchina di Turing *M* che computi una funzione φ, è quindi possibile esprimere la funzione da essa computata con gli strumenti della ricorsività generale (ricorsività primitiva ed operatore μ). Ogni funzione T-computabile è dunque ricorsiva generale. Q.E.D.

Abbiamo visto che la seconda parte della dimostrazione può essere condotta per macchine di Turing che operano con un alfabeto qualsiasi. Per la parte (1) della dimostrazione è invece sufficiente che le macchine di Turing utilizzino un alfabeto di un solo simbolo. È d'altra parte ovvio che se una funzione può essere computata da una macchina con un alfabeto di un solo simbolo, essere può essere computata da una macchina con un alfabeto più ricco. La dimostrazione di equivalenza nel suo complesso presenta quindi il seguente schema. Per ogni funzione aritmetica φ:

φ è ricorsiva generale $\Rightarrow$ φ è T-computabile (con Σ di un simbolo) (dim. (1));
φ è T-computabile $\Rightarrow$ φ è computabile da una macchina con Σ qualsiasi (ovvio);
φ è computabile da una macchina con Σ qualsiasi $\Rightarrow$ φ è ricorsiva generale (dim. (2)).

Abbiamo dunque come *corollario* che una funzione è T-computabile (quindi con alfabeto di un solo simbolo) *sse* essa è computabile da una macchina di Turing con alfabeto qualsiasi.

5. La Tesi di Church

Nel 1936 il logico americano Alonzo Church propose di identificare il concetto intuitivo di funzione calcolabile mediante un algoritmo con il concetto di funzione ricorsiva generale. Tale identificazione divenne in seguito nota col nome di *Tesi di Church*. L'enunciato della Tesi di Church è il seguente:

una funzione è effettivamente calcolabile sse è ricorsiva generale.

Che ogni funzione ricorsiva generale sia effettivamente computabile è facilmente verificabile, come abbiamo avuto modo di mettere in luce nei paragrafi precedenti. Ciò che invece è rilevante e problematico nella Tesi di Church è l'implicazione inversa, per la quale ogni procedimento algoritmico è riconducibile alla ricorsività generale. "Algoritmo" e "funzione computabile in modo effettivo" sono concetti intuitivi, non specificati in modo formale, per cui non è possibile una dimostrazione rigorosa di equivalenza con il concetto di funzione ricorsiva generale. La Tesi di Church non è dunque una congettura che, in linea di principio, potrebbe un giorno diventare un teorema. Tuttavia, la nozione intuitiva di funzione computabile in modo effettivo è contraddistinta da un insieme di caratteristiche (quali determinismo, finitezza di calcolo, eccetera) che possiamo considerare in larga misura "oggettive". Questo fa sì che sia praticamente sempre possibile una valutazione concorde nel decidere se un dato procedimento di calcolo debba essere considerato algoritmico o meno. Quindi, almeno in linea di principio, è ammissibile che venga "scoperto" un controesempio alla Tesi di Church: che si individui cioè una funzione effettivamente calcolabile secondo questi parametri informali, ma che non sia allo stesso tempo ricorsiva generale. In questo paragrafo esporremo le ragioni per cui si ritiene improbabile che un evento del genere si verifichi[3].

Raccoglieremo gli argomenti a favore della Tesi di Church nei tre gruppi *(a)*, *(b)* e *(c)*.

(a). Il primo gruppo di argomenti poggia su quella che si può chiamare *evidenza euristica*. Lo suddividiamo in tre ulteriori sottogruppi.

(a1). Per ogni singola funzione calcolabile che sia stata esaminata, è sempre stato possibile dimostrare la sua appartenenza alla classe delle funzioni ricorsive generali. Lo stesso si può dire delle operazioni per definire funzioni in modo effettivo a partire da altre funzioni. Tale indagine è stata condotta per un gran numero di funzioni, di classi di funzioni e di operazioni, con l'intento di essere il più esaustivi possibile.

(a2). I metodi per dimostrare che funzioni calcolabili sono ricorsive generali sono stati sviluppati con un grado di generalità tale da far ritenere improbabile che possa essere scoperta una funzione calcolabile cui tali metodi non possano essere applicati.

(a3). I vari metodi tentati per costruire funzioni effettivamente calcolabili non ricorsive generali hanno condotto tutti al fallimento, nel senso che le funzioni ottenute erano tutte a loro volta ricorsive generali, oppure non erano calcolabili in modo effettivo.

(b). Nel secondo gruppo di argomenti viene considerata l'*equivalenza delle diverse formulazioni* proposte. Tutti i tentativi che sono stati elaborati per caratterizzare in modo rigoroso la classe di tutte le funzioni effettivamente computabili si sono rivelati equivalenti. Ciò che è particolarmente rilevante ai fini di una "corroborazione" della Tesi di Church è la diversità degli strumenti e dei concetti impiegati nelle diverse formulazioni. In molti casi tali formulazioni traggono la loro origine da concetti matematici preesistenti. Nel caso della *ricorsività generale di*

[3] In questo paragrafo ci rifacciamo in parte al § 62 di Kleene [1952], cui rimandiamo per ulteriori informazioni.

Herbrand-Gödel si prendono le mosse dal concetto di sistema di equazioni, nella *λ-ricorsività* di Church si parte dall'idea di un calcolo di sole funzioni, il *λ-calcolo*. Delle macchine di Turing tratteremo ampiamente al punto *(c)*. Schönfinkel ([1924]) e Curry ([1929], [1930] e [1932]) elaborarono il cosiddetto *calcolo dei combinatori*. Rosser ne dimostrò l'equivalenza col *λ-calcolo*. Ad E. Post ([1943], [1946]) è dovuto l'approccio basato sui *sistemi normali* o *canonici.* Negli anni cinquanta, il logico sovietico A. A. Markov ([1951], [1954]) propose un'ulteriore formulazione tramite quelli che vennero poi detti appunto *algoritmi di Markov*.

Molte di tali formulazioni ammettono inoltre numerose varianti equivalenti. Riguardo alla formulazione mediante l'operatore di minimalizzazione μ, ad esempio, è possibile una variante che comprende esclusivamente gli schemi di composizione e di minimalizzazione, e, come funzioni base, le funzioni proiezione, somma e prodotto, e la funzione $\delta(x, y)$ così definita:

$$\boldsymbol{d}(x, y) = \begin{cases} 1 \text{ se } x = y \\ \\ 0 \text{ se } x \neq y \end{cases}$$

Una funzione $\varphi(x_1, \ldots, x_n)$ si dice *rappresentabile* in un sistema formale *SF sse* esiste una formula $\alpha_\varphi(x_1, \ldots, x_n, y)$ di *SF*, senza alcuna variabile libera eccetto $x_1, \ldots, x_n, y$, tale che, per ogni n+1-pla $(k_1, \ldots, k_n, w)$ di numeri naturali,

$$\varphi(x_1, \ldots, x_n) = w \Leftrightarrow \vdash_{SF} \alpha_\varphi(\overline{k}_1, \ldots, \overline{k}_n, \overline{w})$$

(dove, se k è un numero naturale, con $\overline{k}$ si rappresenta il numerale corrispondente, cioè il simbolo che rappresenta k nel sistema *SF*). Gödel (in Gödel [1936]) descrisse un sistema formale S_1 nel quale sono rappresentabili tutte e sole le funzioni ricorsive generali. S_1 dispone esclusivamente di variabili di tipo 0, cioè variabili che possono assumere i propri valori fra gli elementi dell'insieme **N** dei numeri naturali Il sistema S_1 può essere considerato come la base di una gerarchia di sistemi S_i (con i = 1, 2, 3, ...) che dispongono via via di variabili di tipo più elevato (cioè, variabili di tipo 1 che variano su insiemi di numeri naturali, variabili di tipo 2 che variano su insiemi di insiemi di numeri naturali, e così via). "Può essere dimostrato - Gödel stesso afferma - che una funzione che è rappresentabile in uno degli S_i, o persino in un sistema di ordine transfinito, è rappresentabile già in S_1, sicché il concetto di rappresentabile è in un certo senso 'assoluto', mentre quasi tutti i concetti matematici sin qui noti (ad esempio, dimostrabile, definibile, eccetera) dipendono essenzialmente dal sistema che viene preso come base" (citato in Kleene [1952], pag. 321).

Tale "indipendenza" dal sistema formale prescelto è ovviamente un forte elemento a favore della Tesi di Church. Le funzioni ricorsive generali sono inoltre tutte e sole le funzioni rappresentabili in un sistema formale del primo ordine che comprenda gli assiomi peaniani per l'aritmetica.

(c). Nel terzo punto consideriamo l'apporto all'evidenza della Tesi di Church fornito dall'analisi del concetto di calcolo compiuta da Turing. Il concetto di *macchina di Turing* si distingue dagli altri approcci sin qui considerati in quanto non si tratta di un concetto matematico elaborato per ragioni autonome e proposto in un secondo tempo come formulazione rigorosa del concetto di algoritmo, quanto piuttosto di un tentativo diretto di costruire un modello dell'attività di un essere umano che esegue un calcolo di tipo deterministico. Storicamente, furono proprio le macchine di Turing ad aumentare notevolmente la plausibilità della Tesi di Church.

In questo settore specifico della teoria della computabilità si è sviluppata una tendenza a considerare la Tesi di Church come una sorta di "legge empirica" piuttosto che come un enunciato a carattere "logico-formale". Emil Post, il quale propose contemporaneamente a Turing un concetto

di macchina calcolatrice fortemente analogo a quello sviluppato dal logico inglese (Post [1936]), sottolineava il suo disaccordo da chi tendeva ad identificare la Tesi di Church a un assioma o a una mera definizione. Essa dovrebbe piuttosto essere considerata, affermava Post, una "ipotesi di lavoro", che, se opportunamente "corroborata", dovrebbe assumere il ruolo di una "legge naturale", una "fondamentale scoperta circa le limitazioni del potere matematizzante dell'*Homo sapiens* " (Post [1936], pag. 105).

Sulla stessa linea sembrano procedere i successivi sviluppi dell'opera di Turing. Nel suo celebre saggio "Computing Machinery and Intelligence" (Turing [1950]) assistiamo ad una sorta di "radicalizzazione" di questo modo di intendere la Tesi di Church. Facendo riferimento a calcolatori reali, che tuttavia vengono caratterizzati in maniera analoga ad una macchina di Turing, Turing si dichiara fiducioso che macchine di questo tipo possano giungere a simulare, nel volgere di pochi decenni, non soltanto il "comportamento computazionale" di un essere umano, ma anche qualsiasi altra attività umana superiore.

In direzione opposta ma complementare si è sviluppato il lavoro di Robin Gandy (Gandy [1980]), secondo il quale il concetto di macchina di Turing è eccessivamente "antropomorfo" perché gli possa essere ricondotto ogni tipo di dispositivo di calcolo artificiale. Ad esempio, nelle macchine di Turing si assume che il calcolo proceda secondo una sequenza di passi elementari, elaborando un solo simbolo alla volta, mentre un calcolatore artificiale può procedere in parallelo, elaborando contemporaneamente un numero arbitrario di simboli. Per superare tali limitazioni, Gandy formula, utilizzando strumenti insiemistici, una caratterizzazione estremamente generale del concetto di macchina calcolatrice, in cui le macchine di Turing rientrano come caso particolare. Egli dimostra quindi che ogni funzione calcolabile da una di tali macchine è ricorsiva generale, a patto che vengano rispettate alcune condizzioni di "finitezza" (determinismo, possibilità di descrivere il calcolo in termini discreti, e così via). È interessante notare che, perché la Tesi di Church non sia falsificata, deve esistere un limite superiore della velocità di propagazione dei cambiamenti in un dispositivo di calcolo, in accordo con la teoria della relatività, secondo la quale tale limite è costituito dalla velocità della luce. Macchine "newtoniane", che ammettano la possibilità dell'azione istantanea a distanza, consentirebbero, secondo il modello di Gandy, di calcolare funzioni non computabili secondo la Tesi di Church.

6. Macchine universali e problemi indecidibili

In questo capitolo utilizzeremo la Tesi di Church per dimostrare alcuni enunciati relativi alle macchine di Turing e alla classe delle funzioni ricorsive generali. Mostreremo cioè come esistano degli algoritmi intuitivi che svolgono determinati compiti e, in base alla Tesi di Church, trarremo la conclusione che le funzioni calcolate da tali algoritmi sono ricorsive generali. Si tratta tuttavia di un espediente adottato esclusivamente a fini espositivi, per abbreviare le dimostrazioni: in ognuno di tali casi è sempre possibile dimostrare direttamente la ricorsività generale delle funzioni coinvolte costruendone la derivazione (oppure costruendo la tavola delle quintuple di una macchina di Turing che le calcola), senza fare appello alla Tesi di Church.

6.1 L'enumerabilità delle macchine di Turing e la macchina di Turing universale

PROPOSIZIONE: *la classe di tutte le macchine di Turing è enumerabile in modo effettivo (con ripetizioni).*

Dimostrazione: Poiché ogni macchina di Turing, qualunque sia il suo alfabeto, può essere ricondotta ad una macchina di Turing con alfabeto di un solo simbolo (cfr. § 4.3), ci limiteremo a prendere in considerazione macchine di quest'ultimo tipo, e vedremo come si possa costruire una enumerazione effettiva di tutte le macchine di Turing con $\Sigma \equiv \{|\}$. La stessa tecnica può tuttavia essere estesa a macchine con Σ qualsiasi.

Abbiamo visto che ogni macchina di Turing può essere identificata con la tavola delle sue quintuple. Per ogni numero fissato di quintuple, il numero di macchine di Turing con $\Sigma \equiv \{|\}$ è sempre finito. Si inizi dunque col costruire tutte le macchine di Turing la cui tavola è costituita da una sola quintupla, le si ordini secondo qualche criterio (ad esempio lessicografico[4]) e le si numeri seguendo tale ordine. Si costruiscano quindi tutte le macchine di Turing con due quintuple, le si ordini, e le si numeri proseguendo la numerazione precedente. Si proceda analogamente per macchine con un numero di quintuple sempre crescente. Specificandolo in tutti i dettagli, tale procedimento può essere trasformato in un algoritmo di enumerazione vero e proprio. Q.E.D.

Tale risultato implica la possibilità di associare ad ogni macchina di Turing un numero naturale n come suo indice (indicheremo con M_n la macchina il cui indice è n), in maniera tale che, dato n, sia possibile costruire in modo algoritmico le quintuple di M_n, e viceversa, data M_n, sia possibile individuare n.

Poter disporre di una enumerazione effettiva di tutte le macchine di Turing consente di dimostrare l'esistenza di una *macchina di Turing universale*, una macchina di Turing in grado cioè di simulare il comportamento di ogni altra macchina.

Premettiamo che una generica n-pla di numeri naturali $(k_1, \ldots, k_n)$ può essere codificata tramite gödelizzazione in un numero naturale $\overline{k}$ in maniera effettiva (cioè in maniera tale da poter ottenere in maniera algoritmica $(k_1, \ldots, k_n)$ a partire da $\overline{k}$ e viceversa).

Dimostriamo quindi la seguente:

---

[4] L'ordinamento di tipo lessicografico è quello che si usa abitualmente per ordinare le voci di dizionari, enciclopedie, elenchi telefonici, eccetera. Stabilito convenzionalmente un ordine fra i simboli dell'alfabeto, per ordinare due stringhe qualsiasi di simboli si procede come segue. Se le due stringhe iniziano con simboli diversi, sarà maggiore la stringa che inizia col simbolo maggiore. Se i primi due simboli sono uguali, l'ordine sarà deciso in base ai secondi sinboli; se anch'essi coincidono, in base ai terzi, e così via. Si tratta di un ordinamento totale: date due stringhe qualsiasi, o esse sono uguali, oppure una deve essere maggiore dell'altra. Se $Q \equiv \{q_0, q_1, q_2, \ldots\}$ è l'insieme di tutti gli stati delle macchine di Turing, l'"alfabeto" su cui si baserà l'ordinamento delle quintuple è cosituito dal seguente insieme: $\{|\} \cup \{L, R, C\} \cup Q$.

PROPOSIZIONE: *Esiste una mcchina di Turing universale, vale a dire, una macchina di Turing U tale che, fissata una enumerazione effettiva di tutte le macchine di Turing, si comporti nel modo seguente. Sia* $\overline{k}$ *la codifica di una n-pla* $(k_1, \ldots, k_n)$. *Presa in input la coppia di numeri naturali* $(m, \overline{k})$, *U dia in output l'output della macchina* M_m *con input* $(k_1, \ldots, k_n)$ *(se, per tale input, il calcolo di* M_m *non termina, non termina neppure il calcolo di U).*

Dimostrazione: Poiché m è indice di M_m in un'enumerazione effettiva, dato m è possibile ottenere M_m in maniera algoritmica. Dato $\overline{k}$ si può inoltre ottenere in modo algoritmico $(k_1, \ldots, k_n)$. Infine, data M_m, il calcolo del suo output a partire da $(k_1, \ldots, k_n)$ è anch'esso di tipo algoritmico, per la definizione del concetto di macchina di Turing. Componendo tali procedimenti, si ottiene un algoritmo che permette di calcolare l'output di M_m per $(k_1, \ldots, k_n)$ a partire da $(m, \overline{k})$. In virtù della tesi di Church, a tale algoritmo deve corrispondere una macchina di Turing (che chiameremo U, appunto), che lo esegue. Q.E.D.

Si noti che, a differenza delle macchine di Turing viste sino ad ora, le quali erano in grado di eseguire un solo tipo di calcolo, U è in un certo senso "programmabile": l'indice m preso in input infatti può essere visto come il programma che specifica di volta in volta quale calcolo la macchina U deve eseguire a partire dai dati $\overline{k}$.

L'esistenza della macchina di Turing universale è dimostrabile in maniera diretta senza far riferimento alla tesi di Church, mostrando come se ne possano costruire le quintuple (tale dimostrazione è stata ottenuta per la prima volta in Turing [1936-37]). U può essere costruita in maniera tale da ricavare le quintuple di M_m a partire da m, e da tracriverle, opportunamante rappresentate, sul proprio nastro. Dopo di che, U ricava l'n-upla $(k_1, \ldots, k_n)$ a partire da $\overline{k}$ e simula il comportamento di M_m "leggendone" le quintuple sul proprio nastro ed applicando di volta in volta quelle opportune. (Per una descrizione esauriente della macchina di Turing universale si veda ad esempio [Minsky, 1967], pp. 137 e segg.)

Astraendo dalla formulazione in termini di macchine i Turing, la precedente proposizione può essere iformulata come segue:

PROPOSIZIONE: *Esiste una funzione* Φ, *ricorsiva generale (parziale), tale che, per ogni coppia* $(x, \overline{y})$ *di numeri naturali,*

$$\Phi(x, \overline{y}) = \varphi_x(y_1, \ldots, y_n)$$

(dove x *è l'indice di* φ_x *in una enumerazione effettiva di tutte le funzioni ricorsive generali (parziali), e* $\overline{y}$ *è la codifica effettiva di* $(y_1, \ldots, y_n)$.

La dimostrazione è in tutto analoga a quella della proposizione precedente. Si noti che Φ è parziale in senso proprio, in quanto, se $\varphi_x(y_1, \ldots, y_n) = \bot$, allora anche $\Phi(x, \overline{y}) = \bot$.

6.2 Problemi indecidibili: lo **halting problem**

In questo paragrafo vedremo come la tesi di Church implichi l'esistenza di funzioni non computabili in modo effettivo, e quindi di problemi matematici non decidibili in maniera algoritmica.

Una prima dimostrazione dell'esistenza di funzioni non computabili in modo effettivo può essere ottenuta come segue. Vale innanzi tutto la seguente:

PROPOSIZIONE: *la classe di tutte le funzioni ricorsive generali (parziali) è un insieme infinito numerabile.*

Dimostrazione: che esista almeno un'infinità numerabile di funzioni ricorsive generali segue dal fatto che tutte le funzioni costanti (vale a dire, tutte quelle funzioni φ per cui, per ogni x, $\varphi(x) = k$, con $k \in \mathbb{N}$) sono ricorsive generali in modo ovvio.

Che l'insieme di tutte le funzioni ricorsive generali (parziali) non sia più che numerabile consegue dal fatto che è possibile enumerarlo in maniera effettiva (con ripetizioni). Q.E.D.

D'altro canto, l'insieme di tutte le funzioni definite sui numeri naturali ha la potenza del continuo. Infatti, dal punto di vista insiemistico, una funzione aritmetica φ ad n argomenti può essere identificata con un opportuno insieme I_φ di n+1-ple ordinate, tali che:

$$\varphi(x_1, ..., x_n) = y \Leftrightarrow \langle x_1, ..., x_n, y \rangle \in I_\varphi$$

Si dimostra che l'insieme di tutti gli insiemi di n+1-ple che rappresentano una funzione ha la stessa cardinalità dell'insieme delle parti di $\mathbb{N}$, che coincide a sua volta con la cardinalità di $\mathbb{R}$

Esiste quindi un insieme più che numerabile di funzioni aritmetiche che non sono ricorsive generali, e che quindi, secondo la tesi di Church, non sono calcolabili in modo effettivo.

Esaminiamo ora un esempio particolarmente interessante di problema non decidibile in modo algoritmico, vale a dire il *problema dell'arresto* per le macchine di Turing: data la Tesi di Church, si dimostra che, in generale, dato l'indice m di una macchina di Turing M_m e un numero naturale k, non esiste un algoritmo per stabilire se il calcolo di M_m per l'input k termina o meno. Storicamente, si tratta del primo esempio di problema matematico indecidibile individuato. Il teorema che enuncia l'indecidibilità del problema dell'arresto viene usualmente indicato come *primo teorema di Church*, poiché Church fu il primo a fornirne una dimostrazione ([Church, 1936]) (anche se la dimostrazione originaria di Church non faceva riferimento direttamente alle macchine di Turing, ma ad un problema analogo formulato in termini di λ-calcolo).

Fissata una enumerazione effettiva delle macchine di Turing, definiamo anzi tutto il predicato a tre argomenti $T(m, k, y)$: sia m l'indice di una macchina di Turing M_m. Se M_m applicata all'input k si ferma dopo esattamente y passi di calcolo, allora $T(m, k, y)$ è vero. Altrimenti è falso.

Dimostriamo ora la seguente:

PROPOSIZIONE: *Il predicato T sopra definito è ricorsivo generale*.

Dimostrazione: Esiste un algoritmo (nel senso intuitivo) che, data una generica terna (m, k, y) di numeri naturali, consente di decidere della verità o della falsità di $T(m, k, y)$. Dato l'indice m è infatti possibile costruire in modo effettivo la macchina M_m. Dopo di che, si esegua il calcolo di M_m pplicata all'input k. Dopo esattamente y mosse di calcolo, è possibile decidere se M_m ha raggiunto uno stato finale o meno. Il predicato T è quindi decidibile in modo effettivo. In base alla tesi di Church esso è dunque ricorsivo generale. Q.E.D.

Partendo da T, definiamo il predicato a due posti $\exists y\, T(m, k, y)$. Come è facile constatare, esso è vero *sse* esiste un numero y di passi per cui la macchina M_m con input k si ferma. Altrimenti, se il calcolo di M_m con input k continua all'infinito, allora $\exists y\, T(m, k, y)$ è falso.

Definiamo poi il predicato ad un posto $\exists y\, T(x, x, y)$ come caso particolare del precedente in cui i primi due argomenti coincidono: esso è vero *sse* esiste un numero y di passi per cui la macchina M_x con input x si ferma. Dimostriamo ora il seguente:

LEMMA: *Il predicato* $\exists y\, \mathrm{T}(x, x, y)$ *non è decidibile, vale a dire, la sua funzione caratteristica:*

$$\boldsymbol{j}(x) = \begin{cases} 0 \text{ se } \exists y\, T(x, x, y) \\ \\ 1 \text{ altrimenti} \end{cases}$$

non è computabile in modo effettivo.

Dimostrazione: Per assurdo, si supponga che tale funzione sia computabile. Per la definizione di $\exists y$ T(x, x, y) abbiamo che:

$$\varphi(x) = \begin{cases} 0 \text{ se la macchina } M_m \text{ con input } x \text{ termina il calcolo} \\ 1 \text{ altrimenti} \end{cases}$$

Si costruisca allora una macchina di Turing H che si comporti nel modo seguente:

$$\text{dando } x \text{ in input ad } H \begin{cases} \text{se } \varphi(x) = 0, \text{ allora il calcolo di } H \text{ non termini} \\ \text{altrimenti, se } \varphi(x) = 1, H \text{ dia come output } 1 \end{cases}$$

Ammessa l'ipotesi di assurdo, costruire H è certamente possibile. Infatti, se φ è computabile, allora, per la tesi di Church, deve esistere una macchina M_φ che la computa. Supponendo data tale macchina, è facile definire H conformemente alla nostra definizione.

Vediamo come si può procedere. Per semplicità, supponiamo di lavorare con l'alfabeto {0,1}. Per ogni input x, M_φ assuma la configurazione finale (q_0, 0) *sse* φ(x) = 0, e assuma invece la configurazione finale (q_0, 1) *sse* φ(x) = 1, con il resto del nastro completamente vuoto. Sia inoltre q_j uno stato che non viene utilizzato nelle quintuple di M_φ. La tavola di H sarà allora la seguente:

....

quintuple di M_φ

....

q_0	0	0	R	q_j
q_j	s_0	s_0	R	q_j

Se M_φ raggiunge la configurazione finale (q_0, 1) (se cioè φ(x) = 1), nessuna nuova quintupla interverrà, e l'output di H sarà ancora 1. Altrimenti, se M_φ raggiunge la configurazione finale (q_0, 0), le due nuove quintuple faranno sì che H continui a muoversi all'infinito lungo il nastro.

H, se esiste, deve figurare nell'enumerazione di tutte le macchine di Turing, deve cioè corrispondere, per un certo indice k, alla macchina M_k dell'enumerazione. Per come è stata definita, accadrebbe allora quanto segue:

$$\begin{cases} \text{se } \varphi(k) = 0 \text{ (cioè, per la definizione di } \varphi\text{, se il computo di } H \\ \text{con input } k \text{ termina), allora il computo di } H \text{ non termina} \\ \text{se } \varphi(k) = 1 \text{ (cioè se il computo di } H \text{ con input } k \text{ non termina),} \\ \text{allora il computo di } H \text{ termina} \end{cases}$$

Avremmo cioè che, per l'input k, il computo di H termina *sse* il computo di H non termina. Supporre l'esistenza della funzione φ definita come sopra e computabile in modo effettivo conduce quindi all'assurdo. Q.E.D.

È banale ora dimostrare che:

TEOREMA (*primo teorema di Church*): *il predicato* $\exists y$ T(m, k, y) *non è decidibile in modo effettivo.*

Dimostrazione: il predicato $\exists y$ $T(m, k, y)$ non è, in generale, effettivamente decidibile in quanto, per il lemma precedente, si è visto che non è effettivamente decidibile il caso particolare in cui m e k coincidono. Q.E.D.

Si è dimostrato che quantificando esistenzialmente un argomento del predicato decidibile $T(m, k, y)$ si ottiene il predicato non decidibile $\exists y\, T(m, k, y)$. Nel § 1.3 avevamo visto come i predicati ad n-1 argomenti ottenuti quantificando esistenzialmente un argomento di un predicato decidibile ad n argomenti siano semidecidibili. Possiamo quindi enunciare il seguente:

COROLLARIO: *Dato un predicato decidibile* $P(x_1, ..., x_n)$ *ad n argomenti, i predicati ad* n-1 *argomenti* $\exists x_i\, P(x_1, ..., x_n)$ *(con* $1 \leq i \leq n$*) sono semidicidibili, ma non sono, in generale, decidibili.*

Bibliografia

Ackermann, W. (1928). Zum Hilbertschen Aufbau der reellen Zahlen. *Matematischen Annalen*, 93: 1-36 (trad. ingl. in van Heijenoort 1967).

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345-363.

Curry, H.B. (1929). An analysis of logical substitution. *American Journal of Mathematics*, 51: 363-384.

Curry, H.B. (1930). Grundlagen der Kombinatorischen Logik. *American Journal of Mathematics*, 52:509-536, 789-834.

Curry, H.B. (1932). Some additions to the theory of combinators. *American Journal of Mathematics*, 54:551-558.

Davis, M. (a cura di) (1965). *The Undecidable*. Raven Press, Hewlett, New york.

Gandy, R. (1980). Church's thesis and principles for mechanisms. In *The Kleene Symposium*, 123-148, North Holland, Amsterdam.

Gödel, K. (1936). Über die lange von Beweisen. *Ergebnisse eines math. Koll.*, 7:23-24.

van Heijenhoort, J. (a cura di) (1967). *From Frege to Gödel*. Harvard University Press, Harvard.

Hilbert, D. (1926). Über das Unendliche. *Mathematische Annalen*, 36:201-215 (trad. ingl. in van Heijenoort 1967; trad. it. in D. Hilbert, *Ricerche sui fondamenti della matematica*, Bibliopolis, Napoli, 1985).

Kleene, S.C. (1952). *Introduction to Metamathematics*. North Holland, Amsterdam.

Markov, A.A. (1951). Theory of algorithms. *American Mathematical Society Translations*, seconda serie, 15(1960):1-14 (trad. ingl. dell'originale russo).

Markov, A.A. (1954). *Theory of Algorithms.* National Science Foundation and Israel Program for Scientific Translation (1961) (trad. ingl. dell'originale russo).

Minsky, M. (1967). *Computation. Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs.

Post, E. (1936). Finite combinatory processes - formulation 1. *Journal of Symbolic Logic*, 1: 103-105.

Post, E. (1943). Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197-215.

Post, E. (1946). A variant of a recursively unsolvable problem. *Bullettin of the American Mathematical Society*, 52:284-316.

Schönfinkel, M. (1924). Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305-316.

Turing, A. (1936-7). On computable number, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, serie 2, 42:230-365; A correction, ibid., 43:544-546 (ristampato in Davis 1965).

Turing, A. (1937). Computability and λ-definability. *Journal of Symbolic Logic*, 2:153-163.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59:433-460.

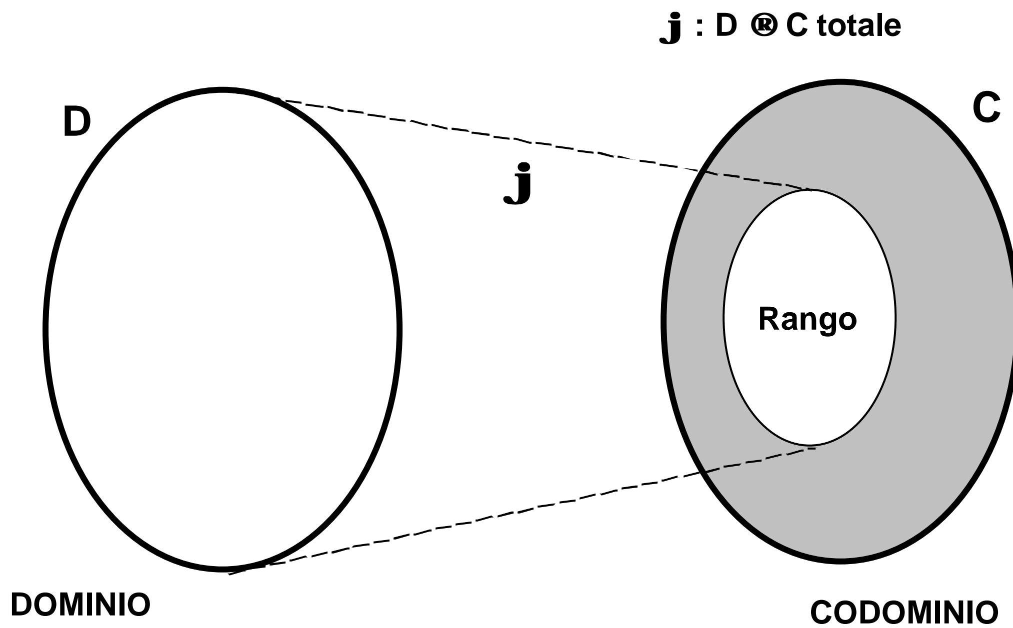


Fig. 1.1

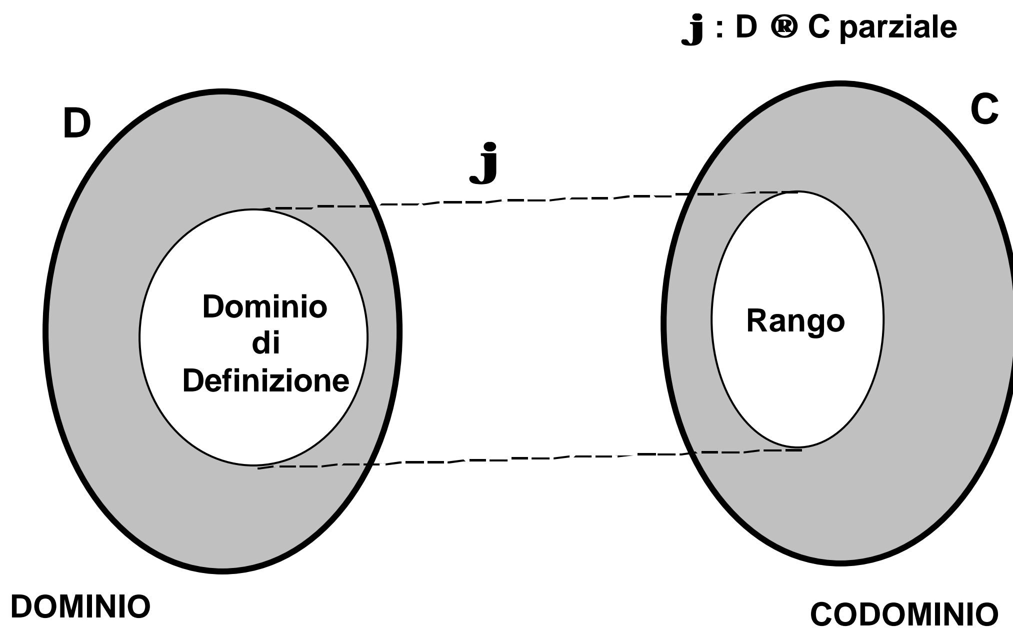
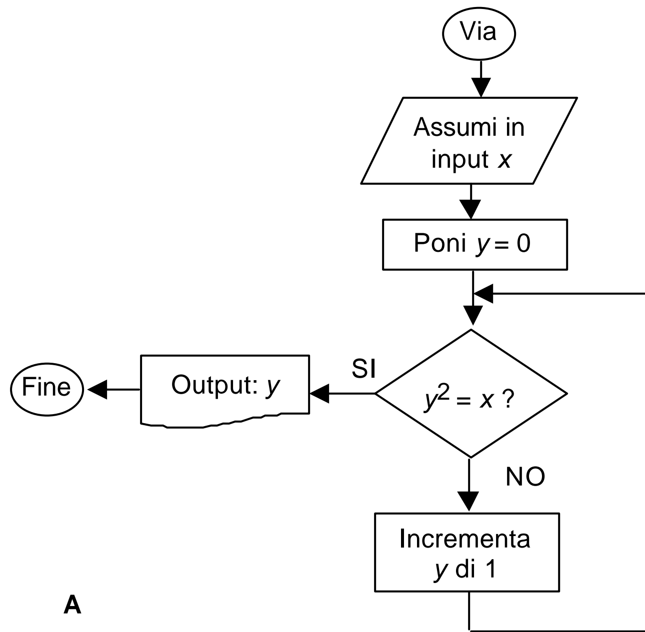


Fig. 1.2



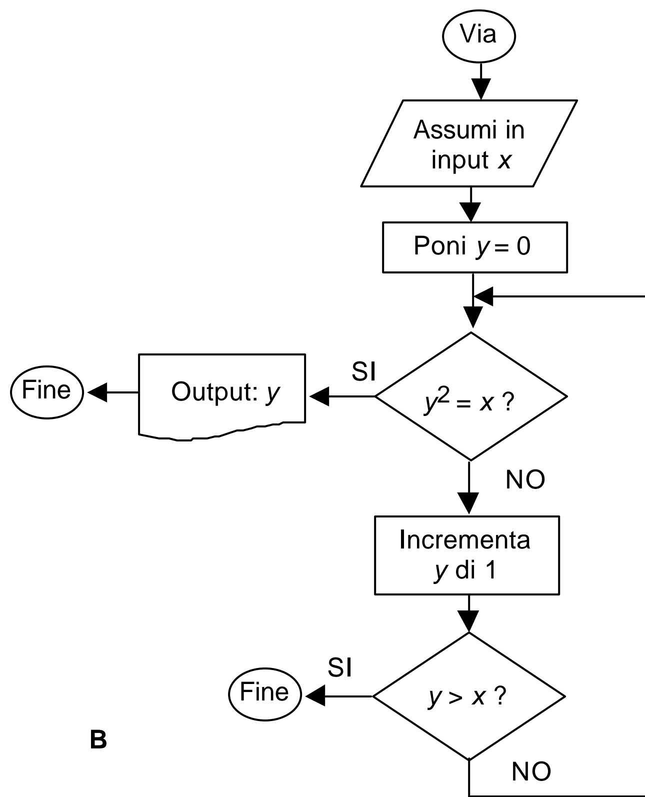


Fig. 1.3

	1	2	3	4	5	6	7	…
passo 1	-	-	-	-	-	[illegible]	-	…
passo 2	-	-	-	-	-	4	-	…
passo 3	-	-	2	-	-	4	-	…
passo 4	9	-	2	-	-	4	-	…
passo 5	9	5	2	-	7	4	-	…
passo 6	9	5	2	-	7	4	0	…
…	…	…	…	…	…	…	…	…

Fig. 1.4

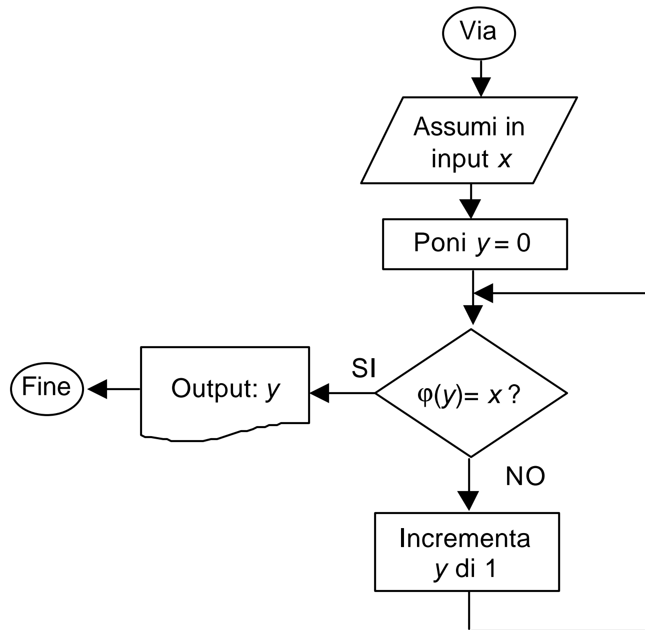


Fig. 1.5

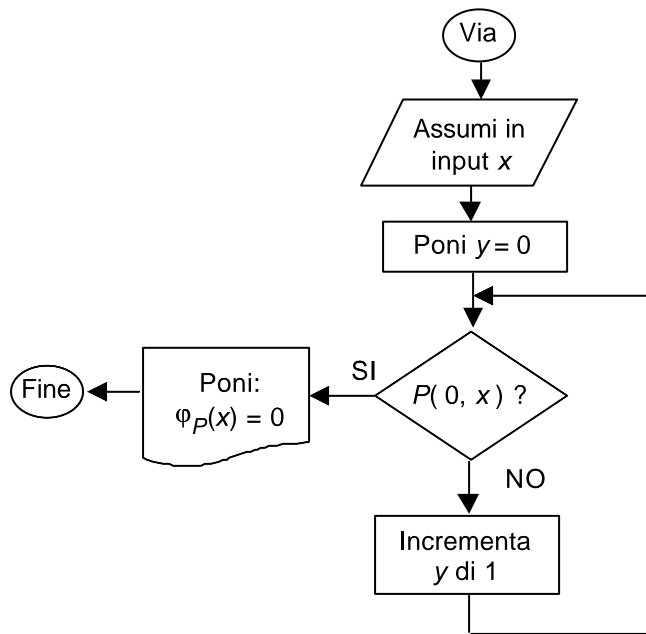


Fig. 1.6

$h(1) =$	0,	**3**	7	9	8	0	4	…
$h(2) =$	137,	0	**0**	1	9	2	8	…
$h(3) =$	9,	7	0	**1**	4	6	0	…
…	…	…	…	…	**…**	…	…	…
$h(n) =$	12,	4	8	…	1	**5**	0	…

Fig. 2.1

$\varphi_1(1)$	$\varphi_1(2)$	$\varphi_1(3)$	...	$\varphi_1(n)$
$\varphi_2(1)$	**$\varphi_2(2)$**	$\varphi_2(3)$	...	$\varphi_2(n)$
$\varphi_3(1)$	$\varphi_3(2)$	**$\varphi_3(3)$**	...	$\varphi_3(n)$
...	...	...	**...**	...
$\varphi_n(1)$	$\varphi_n(2)$	$\varphi_n(3)$	...	**$\varphi_n(n)$**

Fig. 2.2

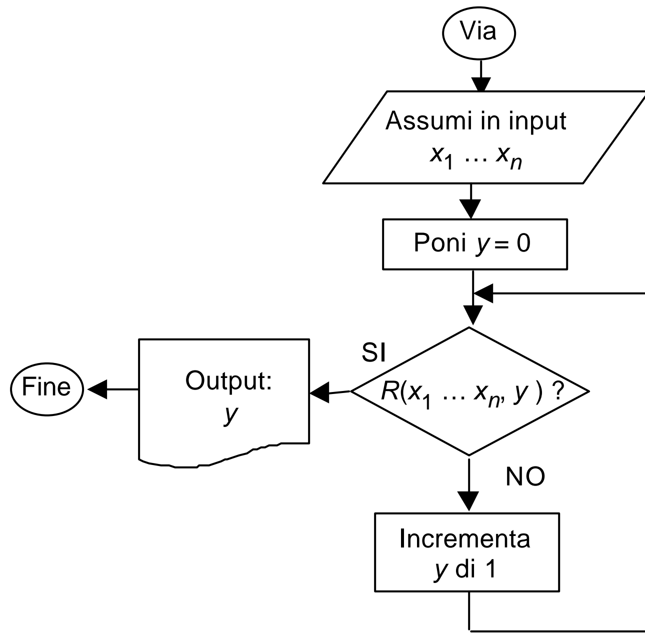


Fig. 3.1

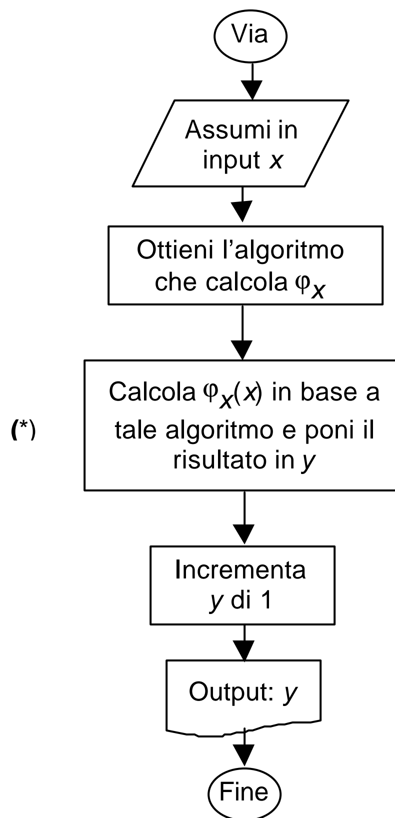


Fig. 3.2

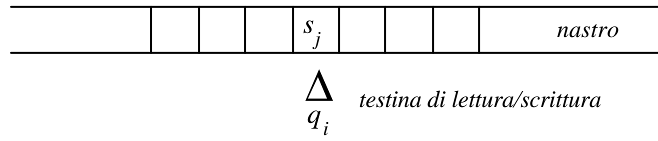


Fig. 4.1

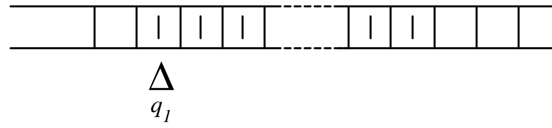


Fig. 4.2

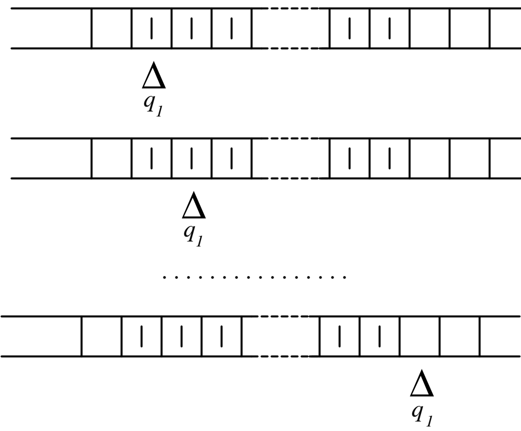


Fig. 4.3

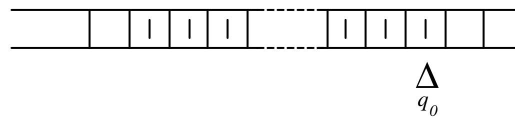


Fig. 4.4

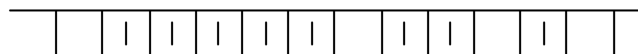


Fig. 4.5