Andrea Asperti

Teoria della Calcolabilità

Bologna, 2006

Indice

Capitolo 1

Nozioni preliminari e Notazione

Siano A e B insiemi, e consideriamo la loro *somma disgiunta* definita nel seguente modo:

$$A \oplus B = \{< 0, a > | a \in A\} \cup \{< 1, b > | b \in B\}$$

Si definisce la *funzione caratteristica* di A

$$f_A : A \to \{0, 1\} \quad t.c. \quad f_A(n) = \begin{cases} 1 & se\ n \in A \\ 0 & se\ n \notin A \end{cases}$$

Una funzione è abitualmente *definita* mediante una associazione tra i valori di input $x_1, \ldots, x_n$ e una espressione matematica che dipende da queste valori, come ad esempio nel seguente polinomio

$$(*) \qquad x, y \vdash x^2 + xy$$

In determinati contesti, tra cui in particolare l'informatica, è importante tuttavia disporre di una notazione che consenta di *esprimere* funzioni, oltre a poterle definire, trattandole quindi come un normale argomento di espressioni pìu' compesse. Adottando una notazione introdotta da Church negli anni trenta, la funzione definita dalla associazione $(*)$ è rappresentata dall'espressione

$$\lambda x, y.x^2 + xy$$

In una espressione della forma $\lambda x.M[x]$, l'operatore λ è un binder, che lega il *parametro formale* x all'interno del *corpo* M della funzione. Ad esempio , date due funzioni f e g, la funzione $\lambda x.f(g(x))$ è la funzione composta.

1.1 Ordinamenti parziali

Definizione 1.1. *Un insieme $\mathcal{D} \subseteq A$ si dice diretto se per ogni $a, b \in \mathcal{D}$ esiste $c \in \mathcal{D}$ tale che $a \leq c$ e $b \leq c$.*

Definizione 1.2. *Un ordinamento parziale si dice completo se ogni diretto $\mathcal{D} \subseteq A$ ammette un estremo superiore.*

Se F é una funzione monotona, l'immagine $F(\mathcal{D})$ di un diretto é ancora un diretto. Infatti, presi $F(a)$ e $F(b)$ in $\mathcal{D}$, $F(c)$ é maggiore di entrambi.

Definizione 1.3. *Una funzione monotona F si dice continua se per ogni insieme diretto $\mathcal{D}$*

$$F(\bigcup \mathcal{D}) = \bigcup F(\mathcal{D})$$

Si noti che per monotonia é sempre vero che $\bigcup F(\mathcal{D}) \leq F(\bigcup \mathcal{D})$.
Infatti, per ogni elemento $F(d) \in F(\mathcal{D})$, $F(d) \leq F(\bigcup \mathcal{D})$ poiché $d \leq \bigcup \mathcal{D}$; dunque $F(\bigcup \mathcal{D})$ é un maggiorante per il diretto $F(\mathcal{D})$ e siccome $\bigcup F(\mathcal{D})$ é per definizione il minimo dei maggioranti, si deve avere $\bigcup F(\mathcal{D}) \leq F(\bigcup \mathcal{D})$.

Teorema 1.4. *Ogni funzione continua $F : A \to A$ ammette un punto fisso, ovvero esiste $a \in A$ tale che $F(a) = a$.*

Dimostrazione. Sia $a_0 = \bot$ e $a_{i+1} = F(a_i)$. Per ogni i si ha $a_i \leq a_{i+1}$. Infatti, per $i = 0$, $a_0 = \bot \leq a_1$, e se per ipotesi induttiva $a_i \leq a_{i+1}$ allora, per la monotonia di F, $a_{i+1} = F(a_i) \leq F(a_{i+1}) = a_{i+2}$.
L'insieme $D = \{a_i\}_{i \in \mathcal{N}}$ é quindi un insieme diretto (in particolare, é una *catena*). Posto $a = \bigcup \mathcal{D}$, sfruttando la continuitá di F abbiamo:

$$F(a) = F \bigcup \mathcal{D} = \bigcup F(\mathcal{D}) = \bigcup \mathcal{D} = a$$

□

Capitolo 2

Numerabilità e diagonalizzazione

Un insieme si dice *finito* se il numero dei suoi elementi è un naturale, ed *infinito* altrimenti.

La nostra intuizione insiemistica sugli insiemi finiti non può essere estesa banalmente a quelli infiniti. Ad esempio, se B è un sottoinsieme proprio di un insieme finito A, B contieme meno elementi di A. Se tuttavia A è infinito possiamo trovare sottoinsiemi propri che sono in corrispondenza biunivoca con l'insieme stesso. Un semplice esempio è dato dai numeri naturali e dal sottoinsieme proprio dei numeri pari.

Un insieme A si dice *numerabile* se esiste una funzione suriettiva dall'insieme dei numeri naturali in A, detta appunto *funzione di enumerazione*. Intuitivamente, se un insieme è numerabile contiene un numero di elementi non superiore ai numeri naturali.

È naturale chiedersi se esistano insiemi infiniti più che numerabili. Consideremo quindi, nelle prossime pagine, alcune questioni di chiusura degli insiemi numerabili rispetto a semplici operazioni insiemistiche.

Il nostro primo problema consiste nel provare ad estendere un insieme numerabile aggiungendo un nuovo elemento.

Lemma 2.1. *Sia A numerabile. Allora $\{*\} \oplus A$ è ancora numerabile.*

Dimostrazione. Sia $f : \mathcal{N} \to A$ la funzione di enumerazione di A. definiamo

$g : \mathcal{N} \to \{*\} \oplus A$ nel modo seguente:

$$g(x) = \begin{cases} g(0) = * \\ g(x+1) = f(x) \end{cases}$$

Chiaramente g è suriettiva. $\square$

Il risultato precedente è strettamente collegato ad un noto "paradosso". Supponiamo di avere un albergo con infinite camere. Anche se l'albergo è al completo, se arriva un nuovo cliente posso comunque trovare una camera disponibile: basta chiedere ad ogni cliente che occupa la camera n di spostarsi nella camera $n+1$: ogni cliente ha una camera dove andare e in questo modo si libera la prima camera.

Iterando un numero finito di volte la tecnica del lemma 2.1 si ottiene il seguente corollario:

Corollario 2.2. *Se D è un insieme finito e A è numerabile, allora $D \oplus A$ è ancora numerabile.*

Cosa succede tuttavia se al nostro albergo con infinite stanze arriva all'improvviso una infinità numerabile di clienti? (si immagini ad esempio di dover chiudere all'improvviso un altro di questi alberghi e di doverne alloggiare tutti i clienti) La tecnica è appena più complicata: in questo caso basta chiedere ad ogni occupante della camera n di spostarsi nella camera $2n$, liberando contemporaneamente tutte le camere dispari. Questo è riassunto dal seguente risultato:

Teorema 2.3. *L'unione di due insiemi numerabili A e B è ancora numerabile.*

Dimostrazione. Siano f e g le due funzioni di enumerazione di A e B. $A \oplus B$ è allora enumerato dalla seguente funzione h:

$$h(x) = \begin{cases} h(2x) = f(x) \\ h(2x+1) = g(x) \end{cases}$$

$\square$

Anche il risultato precedente si generalizza banalmente ad una unione finita di insiemi numerabili:

Corollario 2.4. *Se A_i è numerabile per ogni $i \leq n$ allora $\bigcup_{i \leq n} A_i$ è ancora numerabile.*

2.1 Dovetailing

Definiamo il seguente insieme

$$\mathbf{2} = \{0, 1\}$$

È banale osservare che

$$\mathbf{2} \times A \equiv A \oplus A$$

Più in generale, dato un insieme D di n elementi,

$$D \times A \equiv \bigcup_{i<n} A_i$$

Si ha dunque il seguente risultato:

Corollario 2.5. *Se D è un insieme finito e A è numerabile, allora $D \times A$ è ancora numerabile.*

Dimostrazione. Segue dal corollario 2.4 per l'osservazione precedente. □

Ci chiediamo ora cosa si può dire del prodotto cartesiano di due insiemi numerabili.

Il caso più semplice è costituito dal prodotto cartesiano $\mathcal{N} \times \mathcal{N}$. Invece di definire una funzione suriettiva da $\mathcal{N}$ in $\mathcal{N} \times \mathcal{N}$ risulta più semplice definire una corrispondenza biunivoca da $\mathcal{N} \times \mathcal{N}$ in $\mathcal{N}$. Questa funzione biunivoca, che permette di rappresentare con un unico intero una coppia di interi, viene detta funzione di codifica delle coppie.

Un modo alternativo di descrivere il problema consiste nella definizione di una politica di visita delle coppie (i, j) del piano in modo tale che ogni coppia venga ispezionata una ed una sola volta. Se la coppia (i, j) viene ispezionata al passo k, quest'ultimo può essere interpretato come codifica della coppia. Il valore k, che codifica la coppia (i, j), verrà indicato con la notazione $< i, j >$[1].

Una semplice politica che risolve il problema è la seguente:

[1] Attenzione: (i,j) indica una coppia di interi, mentre $< i, j >$ è un unico intero.

	0		1		2		3		4	⋯	i
0	0		1		3		6		10		
		↙		↙		↙		↙			
1	2		4		7		11				
		↙		↙		↙					
2	5		8		12						
		↙		↙							
3	9		13								
		↙									
4	14										
⋮											
j											

Tecniche di questo tipo, che richiedono di muoversi alternativamente tra due o più potenziali infinità senza necessariamente perdersi lungo rami infiniti sono dette techiche di *dovetailing*.

Analizziamo più in dettaglio la funzione di codifica risultante dalla politica di visita illustrata nel disegno precedente.

La prima osservazione è che la somma delle componenti i ed j per i punti che giacciono su di una stessa diagonale è costante e pari al numero di diagonali già interamente percorse. Il numero di punti del piano già visitati in queste diagonali è dato dalla formula di Gauss

$$\sum_{k=0}^{i+j} k = \frac{(i+j)(i+j+1)}{2}$$

Per visitare l'elemento $< i, j >$ dovrò ancora percorrere j passi lungo l'ultima diagonale, che ci da la seguente funzione di codifica delle coppie:

$$< i, j >= j + \sum_{k=0}^{i+j} k = \frac{(i+j)^2 + i + 3j}{2}$$

Per come è stata costruita è evidente che la funzione precedente definisce una corrispondenza biunivoca da $\mathcal{N} \times \mathcal{N}$ in $\mathcal{N}$, e dunque $\mathcal{N} \times \mathcal{N}$ ènumerabile.

Corollario 2.6. *Il prodotto cartesiano di due insiemi numerabili è numerabile.*

Dimostrazione. Siano A e B due insiemi numerabili, e siano f e g le rispettive funzioni di enumerazione. La funzione $f \times g : \mathcal{N} \times \mathcal{N} \to A \times B$ è ovviamente suriettiva. Il risultato segue dunque dalla numerabilità di $\mathcal{N} \times \mathcal{N}$. □

Corollario 2.7. *L'unione di un insieme numerabile di insiemi numerabili è ancora numerabile.*

Dimostrazione. Sia A un insieme numerabile e sia f la sua funzione di enumerazione. $\{B_a | a \in A\}$ una collezione di insiemi numerabili, ciascuno enumerato da una funzione g_a. La funzione $h : \mathcal{N} \times \mathcal{N} \to \bigcup_{a \in A} \mathcal{B}_a$ definita da

$$h(n, m) = (f(n), g_{f(n)}(m))$$

è suriettiva. □

2.2 Sequenze e parti finite

Iterando un numero finito di volte il risultato del corollario 2.6 si dimostra banalmente che un prodotto finito di insiemi numerabili è ancora numerabile. In particolare, se A è numerabile, lo è anche il prodotto finito $A^n = A \times A \times \ldots \times A$ con n copie di A.

Anche la funzione binaria $< n, m >_2$ di codifica delle coppie di interi può essere estesa ad una funzione che opera su k argomenti $< n1, \ldots, n_k >_k$. Basta porre:

$$\begin{aligned} < n_1, n_2 \ldots n_k, n_{k+1} >_{k+1} &= < n_1, < n_2, \ldots, n_k, n_{k+1} >_k >_2 \\ &= < n_1, < n_2, \ldots < n_k, n_{k+1} >_2 \cdots >_2 >_2 \end{aligned}$$

Un insieme particolarmente interessante è costituito da tutte le sequenze finite costruite con elementi in un insieme dato A, espresso dalla seguente formula: $\bigcup_{i \in \mathcal{N}} A^i$.

Teorema 2.8. *Se A numerabile, anche $\bigcup_{i \in \mathcal{N}} A^i$ è numerabile.*

Dimostrazione. Banale conseguenza dei risultati precedenti. □

Nel caso in cui A sia finito, l'insieme $\bigcup_{i \in \mathcal{N}} A^i$ non è altro che l'insieme di tutte le stringhe definibili sull'alfabeto A. Dunque ogni linguaggio, inteso come sottoinsime di stringhe definite su di un dato alfabeto, è sempre numerabile.

In particolare, gli oggetti di cui possiamo dare una descrizione finitaria sono al più una inifinità numerabile.

È utile inoltre definire una funzione di codifica per tuple finite di interi di lunghezza aribitraria. Data una tupla $(n_1, \ldots, n_k)$ questa puó essere codificata con il valore $< k, < n_1, \ldots, n_k >_k >_2$, aggiungendo quindi esplicitamente al valore codificato la lunghezza della tupla stessa.

Lasciamo come semplice esercizio per il lettore la dimostrazione che in questo modo si definisce una corrispondenza biunivoca da $\bigcup_{k=1}^{\infty} \mathcal{N}^k$ in $\mathcal{N}$.

Esempio 2.9. *Consideriamo il numero 11. La coppia associata è* $< 3, 1 >$. *In base alla codifica precedente, il primo elemento (il numero 3) viene utilizzato per stabile quanti elementi devo estrarre dal secondo valore (il numero 1). Ora* $1 =< 1, 0 >_2 =< 1, < 0, 0 >_2 >_2$. *Per cui* 11 *codifica la tupla* $(1, 0, 0)$.

Osserviamo infine che l'insieme dei sottoinsiemi finiti di un insieme dato A, detto anche insieme delle sue parti finite e indicato con $\mathcal{P}_{fin}(A)$, può essere ottenuto dall'insieme delle sequenze quozientando in una stessa classe le sequenze con gli stessi elementi. La funzione che associa ad ogni ennupla $(a_1, \ldots, a_n)$ l'insieme $\{a_1, \ldots, a_n\}$ è ovviamente suriettiva, e dunque

Corollario 2.10. *L'insieme delle parti finite di un insieme numerabile è numerabile.*

2.3 Insiemi di funzioni e insieme delle parti

Se D è un insieme finito di n elementi, allora il prodotto cartesiano $A \times A \times \ldots \times A$ è isomorfo all'insieme delle funzioni da D in A, che indicheremo con A^D. Sia infatti $D = \{d_1, \ldots, d_n\}$. Allora ad ogni ennupla di elementi $(a_1, \ldots, a_n)$ possiamo far corrispondere la funzione finita da D in A che associa a d_i il valore a_i.

Abbiamo dunque dimostrato il seguente corollario.

Corollario 2.11. *L'insieme delle funzioni da un insieme finito in un insieme numerabile è numerabile.*

Ci chiediamo ora se vale anche il contrario, cioè se le funzioni da un insieme numerabile ad uno finito siano numerabili. Il caso pìu semplice (non banale) è il caso in cui l'insieme finito abbia due elementi, e sia quindi isomorfo a $\mathbf{2} = \{0, 1\}$.

Come è noto, ogni funzione $f : A \to \{0, 1\}$ pùo' essere vista come funzione caratteristica di un sottoinsieme di A (il sottoinsieme su cui f assume valore 1). Dunque esiste una corrispondenza biunivoca tra $\mathbf{2}^A$ e $\mathcal{P}(\mathcal{A})$.

Ora, un risultato fondamentale ottenuto da Cantor verso la fine dell'ottocento è il seguente:

Teorema 2.12. *(Cantor) Dato un insieme arbitrario A, non può esistere una funzione suriettiva da A in $\mathcal{P}(\mathcal{A})$.*

Dimostrazione. Supponiamo che esista una funzione suriettiva $g : A \to \mathcal{P}(A)$. Consideriamo il seguente insieme:

$$\Delta = \{a \in A \mid a \notin g(a)\}$$

Δ è un sottoinsieme di A e siccome abbimao supposto che g sia suriettiva deve esistere δ tale che $g(\delta) = \Delta$. Ci chiediamo ora se se δ appartiene a Δ. Si ha:

$$\begin{array}{rcll} \delta \in \Delta & \Leftrightarrow & \delta \notin g(\delta) & \text{per definizione di } \Delta \\ & \Leftrightarrow & \delta \notin \Delta & \text{per definizione di } \delta \end{array}$$

Siccome questo è assurdo, g non può essere suriettiva. □

Corollario 2.13. *$\mathcal{P}(\mathcal{N}) \cong \mathbf{2}^{\mathcal{N}}$ non è numerabile.*

Il teorema di Cantor utilizza una tecnica di importanza fondamentale per tutta la teoria della calcolabilità, nota come *tecnica diagonale* o *diagonalizzazione*.

La ragione di questo nome è dovuta alla seguente descrizione della prova teorema .

Costruiamo una tabella a doppia entrata $M[a_i, a_j]$ indicizzata su coppie di elementi $a_i, a_j \in A$ e a valori nell'insieme dei booleani. L'idea è che la riga di indice a_i descrive la funzione caratteristica dell'insieme $g(a_i)$, dove g è la funzione da A in $\mathcal{P}(\mathcal{A})$. In particolare, dunque, $M[a_i, a_j] = 1$ see $a_j \in g(a_i)$. La funzione caratteristica di *Delta* si ottiene dunque considerando la diagonale e complementadola, cioè sostituendo 1 a 0 e viceversa. Per costruzione è evidente che la funzione caratteristica così costruita differisce da tutte le righe della tabella, almeno per il valore lungo diagonale. Dunque la funzione caratteristica di *Delta* è diversa da ognuna delle $g(a_i)$, e g non può essere suriettiva.

Corollario 2.14. *L'insieme delle funzioni da $\mathcal{N}$ in $\mathcal{N}$ non è numerabile.*

Dimostrazione. Ovvio, dato che l'insieme delle funzioni da $\mathcal{N}$ in $\{0,1\}$ è un sottoinsieme delle funzioni da $\mathcal{N}$ in $\mathcal{N}$. □

Corollario 2.15. *Non tutte le funzioni da $\mathcal{N}$ in $\mathcal{N}$ sono definibili.*

Dimostrazione. Le funzioni definibili, in quanto esprimibili in un linguaggio con alfabeto finito (o numerabile), sono numerabili. □

Capitolo 3

Ricorsione primitiva

La definizione di una funzione f si dice *ricorsiva* se il valore di f su alcuni argomenti dipende dal valore di f su altri argomenti, solitamente pìu "semplici" rispetto ad una opportuna metrica da definire. Esempi famosi di funzioni ricorsive sono la funzione fattoriale e la funzione di Fibonacci.

Esempio 3.1.

- ***Fattoriale***

$$\begin{cases} fatt(0) = 1 \\ fatt(n+1) = (n+1) * fatt(n) \end{cases}$$

- ***Fibonacci***

$$\begin{cases} fibo(0) = 1 \\ fibo(1) = 1 \\ fibo(n+2) = fibo(n) + fibo(n+1) \end{cases}$$

3.1 Funzioni primitive ricorsive

La famiglia delle funzioni primitive ricorsive è la più piccola classe $\mathcal{C}$ di funzioni che contiene:

- Tutte le **funzioni costanti**

$$f(x_1, x_2, \ldots, x_k) = m \qquad con\ m \in \mathcal{N}$$

- Tutte le **proiezioni** (identità generalizzate)

$$f(x_1, x_2, \ldots, x_k) = x_i \qquad \textit{per qualche } 1 \leq i \leq k$$

- La funzione **successore**

$$f(x) = x + 1$$

ed è inoltre **chiusa** rispetto alle seguenti operazioni:

1. **Composizione**
   Sia h una funzione in $\mathcal{C}$ a n argomenti. Siano $g_1, g_2, \ldots, g_n$ funzioni in $\mathcal{C}$ a k argomenti. Allora anche la f definita come segue è in $\mathcal{C}$

$$f(\vec{x}) = h\left(g_1(\vec{x}), g_2(\vec{x}), \ldots, g_n(\vec{x})\right)$$

   con $\vec{x} = (x_1, x_2, \ldots, x_k)$

2. **Ricorsione primitiva**
   Sia g una funzione in $\mathcal{C}$ di k parametri, e h una funzione in $\mathcal{C}$ di $k+2$ parametri. Allora anche la f di $k+1$ parametri definita come segue è in $\mathcal{C}$

$$f \;:\; \begin{cases} f(0, \vec{x}) = g(\vec{x}) \\ f(y+1, \vec{x}) = h\left(y, f(y, \vec{x}), \vec{x}\right) \end{cases}$$

   con $\vec{x} = (x_1, x_2, \ldots, x_k)$

Quindi una funzione f è primitiva ricorsiva se e solo se esiste una sequenza di funzioni $f_1, f_2, \ldots, f_n = f$ e dove ogni f_i o è una funzione base oppure è ottenuta da funzioni f_j con $j < i$ mediante composizione o ricorsione primitiva.

Esempio

$$\begin{aligned} f_1(x) &= x \\ f_2(x) &= x + 1 \\ f_3(x, y, z) &= y \\ f_4(x, y, z) &= f_2(f_3(x, y, z)) \\ f_5(y, x) &= \begin{cases} f_5(0, x) = f_1(x) \\ f_5(y+1, x) = f_4(y, f_5(y, x), x) \end{cases} \\ f \equiv f_6(x) &= f_5(f_1(x), f_1(x)) \end{aligned}$$

Orientando le equazioni precedenti da sinistra destra otteniamo un insieme di regole di riscrittura che possono essere interpretate automaticamente per effettuare la valutazione di una espressione. Ad esempio, valutiamo $f(2)$:

$$\begin{aligned} f(2) &= f_6(2) \\ &= f_5(f_1(2), f_1(2)) \\ &= f_5(f_1(2), 2) \\ &= f_5(2, 2) \\ &= f_4(1, f_5(1, 2), 2) \\ &= f_4(1, f_4(0, f_5(0, 2), 2), 2) \\ &= f_4(1, f_4(0, f_1(2), 2), 2) \\ &= f_4(1, f_4(0, 2, 2), 2) \\ &= f_4(1, f_2(f_3(0, 2, 2)), 2) \\ &= f_4(1, f_2(2), 2) \\ &= f_4(1, 3, 2) \\ &= f_2(f_3(1, 3, 2)) \\ &= f_2(3) \\ &= 4 \end{aligned}$$

Si noti che questa forma di valutazione pone immediatamente importanti problemi relativi alla *strategia* di calcolo: ad esempio, al secondo passaggio posso espandere f_5 (strategia *lazy*) o procedere alla valutazione degli argomenti (stategia *eager*). Ci si chiede in particolare se la strategia adottata influisca o meno sul risultato della computazione e che impatto abbia sulla complessità della riduzione. Tutti questi interessanti quesiti sono oggetto di una branca dell'informatica teorica rivolta appunto allo studio dei *sistemi di riscrittura*, che esula tuttavia dall'ambito di questo corso.

Il fatto di poter "eseguire" la funzione f non ci aiuta molto a comprenderne il comportamento. Proviamo a procedere in un modo leggermente diverso, operando una *riduzione parziale* all'interno delle definizioni delle varie f_i. Otteniamo in questo modo il seguente codice, molto più leggibile del precedente (anche se questa forma di scrittura non è formalmente corretta dal punto di visto strettamente sintattico, abuseremo spesso il formalismo ai fini di migliorare la comprensione)

$$
\begin{aligned}
f_1(x) &= x \\
f_2(x) &= x+1 \\
f_3(x,y,z) &= y \\
f_4(x,y,z) &= y+1 \\
f_5(y,x) &\begin{cases} f_5(0,x) = x \\ f_5(y+1,x) = f_5(y,x)+1 \end{cases} \\
f \equiv f_6(x) &= f_5(x,x)
\end{aligned}
$$

È facile a questo punto riconoscere la funzione di somma nella funzione f_5, e pertanto $f(x) = x + x = 2x$.

- *Moltiplicazione*

$$
\begin{aligned}
\texttt{mult}(0,x) &= 0 \\
\texttt{mult}(y+1,x) &= \texttt{add}(x, \texttt{mult}(y,x))
\end{aligned}
$$

- *Predecessore*

$$
\begin{aligned}
\texttt{pred}(0) &= 0 \\
\texttt{pred}(y+1) &= y
\end{aligned}
$$

- *Test*

$$
\begin{aligned}
\texttt{test}(0) &= 0 \\
\texttt{test}(x) &= 1
\end{aligned}
$$

- *Fattoriale*

$$
\begin{aligned}
\texttt{fatt}(0) &= 1 \\
\texttt{fatt}(y+1) &= \texttt{mult}(y+1, \texttt{fatt}(y))
\end{aligned}
$$

- *Sottrazione*

$$
\begin{aligned}
\texttt{sub}(0,m) &= m \\
\texttt{sub}(n+1,m) &= \texttt{pred}(\texttt{sub}(n,m))
\end{aligned}
$$

- *Confronto*

$$
\begin{aligned}
f(n,m) &\equiv \texttt{sub}(n,m) + \texttt{sub}(m,n) = \begin{cases} 0 & se\ n = m \\ \neq 0 & \text{se } n \neq m \end{cases} \\
\texttt{zero}(y) &= 1 - \texttt{test}(y) \Rightarrow \begin{cases} \texttt{zero}(0) = 1 \\ \texttt{zero}(y+1) = 0 \end{cases} \\
\underline{\texttt{compare}(n,m)} &\equiv \texttt{zero}(f(n,m)) = \begin{cases} 1 & se\ n = m \\ 0 & altrimenti \end{cases}
\end{aligned}
$$

Vediamo ora una semplice proprietá di chiusura delle funzioni primitive ricorsive.

Definizione 3.2. *Data una funzione $f(y, \vec{x})$, definiamo la somma ed il prodotto limitato relativi ad f nel modo seguente:*

- ***Somma Limitata***

$$\sigma_f(z, \vec{x}) \equiv \sum_{y \leq z} f(y, \vec{x})$$

- ***Prodotto Limitato***

$$\pi_f(z, \vec{x}) \equiv \prod_{y \leq z} f(y, \vec{x})$$

Lemma 3.3. *Le funzioni primitive ricorsive sono* chiuse *rispetto alla* somma limitata *e al* prodotto limitato.

Dimostrazione. Sia $f(y, \vec{x})$ primitiva ricorsiva. È sufficente osservare che i seguenti algoritmi per σ_f e π_f sono primitivi ricorsivi:

$$\sigma_f : \begin{cases} \sigma_f(0, \vec{x}) = f(0, \vec{x}) \\ \sigma_f(z+1, \vec{x}) = \sigma_f(z, \vec{x}) + f(z+1, \vec{x}) \end{cases}$$

$$\pi_f : \begin{cases} \pi_f(0, \vec{x}) = f(0, \vec{x}) \\ \pi_f(z+1, \vec{x}) = \pi_f(z, \vec{x}) * f(z+1, \vec{x}) \end{cases}$$

□

3.2 Predicati primitivi ricorsivi

Sia $P(n_1, n_2, \ldots, n_k)$ un predicato su interi. Dico che P è *primitivo ricorsivo* se e solo se la funzione caratteristica $_{\mathbf{c}P}$ è primitiva ricorsiva.

Esempio 3.4. *Le tre funzioni* `test`, `zero` *e* `compare` *della sezione precedente sono tre esempi di predicati primitivi ricorsivi. Come semplice esercizio il lettore puødimostrare che le relazioni d'ordine su interi $<, \leq, >, \geq$ sono tutte primitive ricorsive.*

Lemma 3.5. *I predicati primitivi ricorsivi sono* chiusi *rispetto ai* connettivi logici.

Dimostrazione. Siano P e Q predicativi primitivi ricorsivi. Dato che negazione e congiunzione costituiscono un insieme completo di connettivi, è sufficiente dimostrare che $\neg P$ e $P \wedge Q$ sono anch'essi primtivi ricorsive. A tal è sufficiente osservare che:

$$\mathbf{c}_{\neg P}(\vec{x}) = 1 - \mathbf{c}_P(\vec{x}) \equiv \mathtt{sub}(\mathbf{c}_P(\vec{x}), 1)$$

$$\mathbf{c}_{P \wedge Q}(\vec{x}) = \mathbf{c}_P(\vec{x}) * \mathbf{c}_Q(\vec{x}) \equiv \mathtt{mult}(\mathbf{c}_P(\vec{x}), \mathbf{c}_Q(\vec{x}))$$

□

Lemma 3.6. *I predicati primitivi ricorsivi sono* chiusi *rispetto alla quantificazione limitata. Ovvero, se $P(z, \vec{x})$ è un predicato primtivo ricorsivo lo sono anche $\forall_{z \leq y} P(z, \vec{x})$ e $\exists_{z \leq y} P(z, \vec{x})$.*

Dimostrazione. Sia $\mathbf{c}_P(z, \vec{x})$ la funzione caratteristica di P. Per dimostrare che il predicato $\forall_{z \leq y} P(z, \vec{x})$ è primitivo ricorsivo devo dimostrare che la sua funzione caratteristica $\mathbf{c}_{\forall(z \leq y) P(z, \vec{x})}$ è primitiva ricorsiva.
A questo fine basta ricordare che $\forall$ è una generalizzazione di $\wedge$, e come otteniamo la congiunzione per mezzo del prodotto, otteniamo la quantificazione universale limitata per mezzo di un prodotto limitato:

$$\mathbf{c}_{\forall_{z \leq y} P(z, \vec{x})} \equiv \prod_{z \leq y} \mathbf{c}_P(z, \vec{x})$$

Per quanto riguarda il quantificatore esistenziale, esso puó essere espresso utilizzando il quantificatore universale e la negazione (o direttamente mediante una iterazione del connettivo di disgiunzione). □

Le proprietà di chiusura relative ai connettivi logici e ai quantificatori limitati permettono uno stile di programmazione dicharativo estremamente divertente. Consideriamo ad esempio il problema della divisibilitá tra numeri naturali. In termini matematici,

$$\mathit{divide}(x, y) \quad \Longleftrightarrow \quad \exists n, nx = y$$

Osserviamo che se tale n esiste, è necessariamente inferiore ad x. Possimao dunque esprimere predicato di divisibilità mediante il seguente algoritmo primitivo ricorsivo:

$$\mathtt{divide}(x, y) \quad \Longleftrightarrow \quad \exists_{n \leq x} \mathtt{compare}(\mathtt{mult}(n, x), y)$$

In pratica si è ricondotta la scrittura del programma ad una semplice determinazione di un upper bound. Analogamente, considerimao il problema, non del tutto elementare, di scrivere un programma per testare la primalità di numeri naturali. In termini matematici, un numero maggiore o uguale di 2 è primo se e solo se è divisibile solo per 1 e per se stesso, ovvero:

$$prime(x) \quad \Longleftrightarrow \quad x \geq 2 \wedge \forall y, (divide(y, x) \Rightarrow y = 1 \vee y = x)$$

Anche in questo caso x fornisce un upper bound alla ricerca di y, e visto che i predicati elementari sono tutti primitivi ricorsivi, anche il test di primalità è banalmente esprimibile nel formalismo primitivo ricorsivo.

3.3 Minimizzazione limitata

Proviamo ora a definire la successione dei numeri primi.

$$P(x) = \text{“}x\text{-esimo numero primo”}$$

Traducendo in termini matematici:

$$\begin{cases} P(0) = 2 \\ P(x+1) = min\{y | prime(y) \wedge (y > P(x))\} \end{cases}$$

Affinchè questa funzione sia primitiva ricorsiva sarebbe sufficitente che min lo fosse. La ricerca progressiva di una minimo elemento che soddisfi una determinata proprietà è un costrutto di calcolo tradizionale nell'ambito della teoria della calcolabilità, noto come operatore di minimizzazione, e tradizionalmente indicato con la lettera μ.

In generale, la ricerca dell'elemento potrebbe non terminare (si pensi ad esempio al caso di un predicato $P(n)$ falso per ogni n). Siccome le funzioni primitive ricorsive sono tutte terminanti, dobbiamo accontentarci, come nel caso dei quantificatori, di una versione *limitata* dell'operatore di minimizzazione, che cerca l'elemento minimo al di sotto di un upper bound fissato z e restituisce un elemento di default (ad esempio z stesso) se tale elemento non esiste:

$$\mu_{y<z} R(y, \vec{x}) = \begin{cases} minimo\ n < z\ tale\ che\ R(n, \vec{x})\ se\ tale\ n\ esiste \\ z\ altrimenti \end{cases}$$

Lemma 3.7. *Le funzioni primitive ricorsive sono chiuse rispetto alla minimizzazione limitata.*

Dimostrazione. Dobbiamo dimostrare che se $R(y, \vec{x})$ è un predicato primitivo ricorsivo, allora lo è anche $\mu_{y<z}R(y, \vec{x})$.
Consideriamo il predicato primitivo ricorsivo

$$h(y, \vec{x}) = \forall_{w \leq y} \neg R(w, \vec{x})$$

Supponiamo che $y_0 = \mu_{y<z}R(y, \vec{x})$. Allora per ogni valore di $y < y_0$ la funzione $h(y, \vec{x})$ assume valore 1 e per ogni valore $y_0 \leq y < z$ la funzione $h(y, \vec{x})$ assume valore 0.
Dunque, y_0 è pari al numero di interi inferiori a z per cui $h(y, \vec{x})$ assume valore 1, ovvero:

$$\mu_{y<z}R(y, \vec{x}) = y_0 = \Sigma_{y<z} \forall_{w \leq y} \neg R(w, \vec{x})$$

□

Torniamo ora alla numerazione dei numeri primi. Devo riuscire a trovare un upper bound alla ricerca del minimo. Un limite superiore ragionevolmente stretto per $P(x+1)$ è fornito dal Teorema di Bertrand, che afferma che per ogni numero positivo n esiste sempre un numero primo compreso tra n e $2n$. Otteniamo dunque la seguente funzione primitiva ricorsiva di numerazione dei numeri primi:

$$\begin{cases} P(0) = & 1 \\ P(x+1) = & \mu_{y \leq 2P(x)} \left(\mathtt{pr}(x) \wedge (y > P(x)) \right) \end{cases}$$

3.4 Ricorsione multipla

Se il valore di input del parametro ricorsivo è $n+1$, lo schema di ricorsione primitiva permette solo di chiamarsi ricorsivamente su n. In alcuni casi, tuttavia, sarebbe conveniente poter effettuare al passo $n+1$ chiamate ricorsive su arbitrari valori di input inferiori o uguali a n. Ci si chiede se questo meccanismo di ricorsione sia ancora esprimibile all'interno del formalismo primitivo ricorsivo. A questo proposito dobbiamo preventivamente studiare la possibilità di simulare il tipo di dato coppia all'interno del nostro calcolo.

Osserviamo innanzi tutto che la funzione di codifica del piano della sezione 2.1, definita come segue

$$< i, j >= j + \sum_{k=0}^{i+j} k = \frac{(i+j)^2 + i + 3j}{2}$$

è primitiva ricorsiva in quanto composizione di funzioni primitive ricorsive.
Più interessante è il problema di capire se le due funzioni di proiezione sono anch'esse primitive ricorsive.
L'idea algoritmica per il calcolo, ad esempio, della prima proiezione, è piuttosto semplice. Dato in input p, cerco esaustivamente un j tale che $< i, j >= p$. L'unico problema consiste nel trovare degli upper bound che limitino opportunamente la ricerca.
A questo proposito è facile dimostrare che per ogni i e j,

$$i \leq < i, j > \quad \wedge \quad j \leq < i, j >$$

che porta alla seguente formulazione primitva ricorsiva per le due proiezioni:

$$\begin{aligned} \texttt{fst}(p) &= \mu_{x \leq p} \exists_{y \leq p} < x, y >= p \\ \texttt{snd}(p) &= \mu_{y \leq p} \exists_{x \leq p} < x, y >= p \end{aligned}$$

Vediamo ora come utilizzare le coppie per implementare un meccanismo di ricorsione multipla.
Inizieremo considerando l'esempio della funzione di Fibonacci:

$$\begin{cases} fibo(0) = 1 \\ fibo(1) = 1 \\ fibo(n+2) = fibo(n) + fibo(n+1) \end{cases}$$

La definizione precedente non rispetta lo schema primitivo ricorsivo. Posso tuttavia definire una funzione ausiliaria che, su input x restituisca in output la *coppia* dei valori $< fibo(x), fibo(x+1) >$:

$$\begin{aligned} \texttt{fibo'}(0) &= < 1, 1 > \\ \texttt{fibo'}(x+1) &= < \texttt{fibo}(x+1), \texttt{fibo}(x+2) > \\ &= < \texttt{snd}(\texttt{fibo'}(x)), \texttt{fst}(\texttt{fibo'}(x)) + \texttt{snd}(\texttt{fibo'}(x)) > \end{aligned}$$

La funzione `fibo'` è chiaramente primtiva ricorsiva, e dunque lo è anche fibonacci, definibile nel modo seguente:

$$\texttt{fibo}(n) = \texttt{fst}(\texttt{fibo'}(n))$$

Ci domandiamo se l'artificio usato per la successione di Fibonacci può essere generalizzato ad un numero arbitrario di parametri, cioè se può una funzione dipendere da *tutti* i valori precedenti e rimanere primitiva ricorsiva. A tal proposito introduciamo la seguente funzione.

Definizione 3.8. *Data una funzione* $f(y, \vec{x})$ *la* storia *di* f *è una funzione* $\hat{f}(y, \vec{x})$ *così definita:*

$$\begin{aligned} \hat{f}(y,\vec{x}) &= < f(0,\vec{x}), f(1,\vec{x}), \ldots, f(y,\vec{x}) >_{y+1} \\ &\equiv << f(0,\vec{x}), f(1,\vec{x}) >, \ldots >, f(y,\vec{x}) \underbrace{>_2 >_2 \ldots >_2}_{y} \end{aligned}$$

La storia di f restituisce dunque, su input n+1, la sequenza di tutti gli output $f(i)$ per ogni $i \leq n$. Possiamo ora estendere lo schema di ricorsione primitiva con uno più potente, che cattura l'idea di ricorsione multipla:

$$\begin{cases} f(0,\vec{x}) & = g(\vec{x}) \\ f(y+1,\vec{x}) & = h(y, \hat{f}(y,\vec{x}), \vec{x}) \end{cases}$$

Se riusciamo a dimostrare che questo schema è ancora riconducibile allo schema di ricorsione primitiva abbiamo dimostrato che la ricorsione multipla non estende effettivamente il potere espressivo del formalismo.

A tal fine, è sufficente osservare che la definizione della storia di f è banalmente esprimibile nello schema primitivo ricorsivo, infatti:

$$\begin{cases} \hat{f}(0,\vec{x}) & = f(0,\vec{x}) = g(\vec{x}) \\ \hat{f}(y+1,\vec{x}) & =< \hat{f}(y,\vec{x}), f(y+1,\vec{x}) >=< \hat{f}(y,\vec{x}), h(y,\hat{f}(y,\vec{x}),\vec{x}) > \end{cases}$$

Si noti che la definizione precedente non dipende da f ma unicamente da g e h.
Avendo definito $\hat{f}$ si può quindi definire f nel modo seguente

$$\begin{cases} f(0,\vec{x}) & = \hat{f}(0,\vec{x})) \\ f(y+1,\vec{x}) & = \texttt{snd}(\hat{f}(y+1,\vec{x})) \end{cases}$$

3.5 Ricorsione vs Iterazione

Una funzione ricorsiva si dice in forma ricorsiva *di coda* se la chiamata ricorsiva è l'ultima operazione eseguita dalla funzione chiamante.

Teorema 3.9. *Ogni funzione primitiva ricorsiva può essere espressa in forma ricorsiva (non necessariamente primitiva) di coda.*

Dimostrazione. Sia f la funzione definita dal seguente schema primitivo ricorsivo:

$$\begin{cases} f(0, \vec{x}) = g(\vec{x}) \\ f(y+1, \vec{x}) = h\left(y, f(y, \vec{x}), \vec{x}\right) \end{cases}$$

Definiamo la seguente funzione ausiliaria:

$$\begin{cases} f'(0, \vec{x}, i, acc) = acc \\ f'(y+1, \vec{x}, i, acc) = f'(y, \vec{x}, i+1, h(i, acc, \vec{x})) \end{cases}$$

Per definizione, f' è ricorsiva di coda. Dimostriamo per induzione su y che, per ogni i

$$f'(y, \vec{x}, i, f(i, \vec{x})) = f(i+y, \vec{x})$$

Se $y = 0$,

$$f'(0, \vec{x}, i, f(i, \vec{x})) = f(i, \vec{x}) = f(i+0, \vec{x})$$

Nel caso $y+1$, abbiamo

$$\begin{array}{rcl} f'(y+1, \vec{x}, i, f(i, \vec{x})) & = & f'(y, \vec{x}, i+1, h(i, f(i, \vec{x}), \vec{x})) \\ & = & f'(y, \vec{x}, i+1, f(i+1, \vec{x})) \\ & = & f(i+1+y, \vec{x}) \text{ per ipotesi induttiva} \\ & = & f(i+y+1, \vec{x}) \end{array}$$

In particolare, dunque, per ogni y

$$f(y, \vec{x}) = f'(y, \vec{x}, 0, g(\vec{x}))$$

che dimostra come f possa essere espressa mediante una funzione ricorsiva di coda. □

L'interesse delle funzioni ricorsive di coda consiste nel fatto che possono essere ricondotte banalmente a meccanismi di tipo iterativo. Ad esempio, la funzione f può essere calcolata dal seguente algoritmo ricorsivo:

```
f(y,x1,...,xn):=
   begin
      var acc = g(x1,...,xn);
      for i := 0 to y
```

```
      res := h(i,acc,x1,...,xn)
    return acc
end
```

In effetti è possibile dimostrare che le funzioni primitive ricorsive sono *tutte e solo* quelle `for`-calcolabili, cioè esprimibili in un qualunque linguaggio imperativo utilizzando come unici costrutti di controllo di flusso l'if-then-else, il for e la chiamata di funzione non ricorsiva.

3.6 Schema iterativo

Le considerazioni precedenti suggeriscono che sia possibile *indebolire* lo schema di ricorsione primitiva, riducendolo alla mera possibilità di *iterare* una determinata funzione h, come espresso dal seguente schema di iterazione:

$$\begin{cases} f(0, \vec{x}) = g(\vec{x}) \\ f(y+1, \vec{x}) = h(f(y, \vec{x}), \vec{x}) \end{cases}$$

Si noti che a differenza dello schema di ricorsione primitiva non si autorizza il passaggio del parametro di ricorsione y alla funzione da iterare. In particolare,

$$\begin{aligned} f(y+1, \vec{x}) &= h(f(y, \vec{x})) \\ &= h(h(f(y-1, \vec{x}))) \\ &= h(h(h(f(y-2, \vec{x})))) \\ &= \vdots \\ &= \underbrace{h(h(h(\ldots(h}_{y+1}(g(\vec{x})))\ldots) \end{aligned}$$

Chiaramente, lo schema iterativo è un caso particolare dello schema ricorsivo primitivo. La questioneè se lo schema iterativo sia abbastanza espressivo da catturare la ricorsione primitiva. In effetti, questo è vero a condizione di avere una nozione di coppia (intesa o come una funzione di codifica di coppie di interi in interi o come un tipo di dato primtivo).

Teorema 3.10. *Iterazione + Coppie = Ricorsione Primitiva.*

Dimostrazione. Dimostriamo il teorema nel caso un po' semplificato in dello schema di ricorsione con un vettore vuoto di parametri $\vec{x}$ lasciando al lettore

la generalizzazione al caso generale (concettualmente semplice ma sintatticamente un po' noiosa). Sia dunque f definita per ricorsione primitiva nel modo seguente:

$$\begin{cases} f(0) = a \\ f(y+1) = h(y, f(y)) \end{cases}$$

Vogliamo dimostrare che f può essere espressa mediante iterazione. A tal fine definiamo una funzione ausiliaria f', tale che

$$f'(y) =< y, f(y) >$$

Osserviamo che

$$\begin{array}{rcl} f'(y+1) & = & < y+1, f(y+1) > \\ & = & < y+1, h(y, f(y)) > \\ & = & < \mathtt{fst}(f'(y)) + 1, h(\mathtt{fst}(f'(y)), \mathtt{snd}(f'(y))) > \end{array}$$

Dunque, la funzione che consente di trasformare la coppia $f'(y)$ nella coppia $f'(y+1)$ è la seguente funzione `next`:

$$\mathtt{next}(p) =< \mathtt{fst}(p) + 1, h(\mathtt{fst}(p), \mathtt{snd}(p)) >$$

e la funzione f' può essere definita iterando `next` nel modo seguente:

$$\begin{cases} f'(0) =< 0, a > \\ f'(y+1) = \mathtt{next}(f'(y)) \end{cases}$$

Infine, $f(0) = \mathtt{snd}(f'(y))$. □

3.7 Funzionali di ricorsione e iterazione

Entrambi gli schemi di ricorsione primitiva e di iterazione permettono di costruire nuove funzioni a partire da funzioni date g e h, che definiscono in modo univoco il risultato. Questi schemi sono dunque due semplici esempi di *funzionali*, cioè funzioni di ordine superiore che operano su altre funzioni. Ad esempio, possiamo definire il funzionale `Rec` che, prese in input $g : N^k \to N$ e $h : N^{k+2} \to N$ restituisce la funzione $(\mathtt{Rec}\ g\ h) : N^{k+1} \to N$ definita per ricorsione primitiva a partire da g e h.
Similmente, si ha un funzionale `Ite` che, prese in input $g : N^k \to N$ e

$h : N \to N$ restituisce la funzione $(\texttt{Ite}\ g\ h) : N^k \to N$ ottenuta per iterazione dalla funzione base g e dalla funzione di iterazione h.

I funzionali forniscono una sintassi estremamente compatta per indicare le funzioni costruite utilizzando i rispettivi schemi, senza peraltro essere costretti a dare un nome alle funzioni definite. La semantica di $(\texttt{Ite}\ g\ h)$ e $(\texttt{Rec}\ g\ h)$ è espressa dalle seguenti regole:

- **Iterazione**

$$\begin{cases} (\texttt{Ite}\ g\ h)(0, \vec{x}) = g(\vec{x}) \\ (\texttt{Ite}\ g\ h)(y+1, \vec{x}) = h((\texttt{Ite}\ g\ h)(y, \vec{x})) \end{cases}$$

- **Ricorsione primitiva**

$$\begin{cases} (\texttt{Rec}\ g\ h)(0, \vec{x}) = g(\vec{x}) \\ (\texttt{Rec}\ g\ h)(y+1, \vec{x}) = h(y, (\texttt{Rec}\ g\ h)(y, \vec{x}), \vec{x}) \end{cases}$$

Nel caso in cui la funzioni $(\texttt{Rec}\ g\ h)$ sia unaria, scriveremo semplicemente $(\texttt{Rec}\ g\ h\ n)$ invece di $(\texttt{Rec}\ g\ h)(n)$ per indicare l'applicazione della funzione $(\texttt{Rec}\ g\ h)$ all'argomento n. Similmente per `Ite`. In particolare, dunque, $(\texttt{Ite}\ a\ h\ n)$ indica il risultato di iterare n volte la funzione h a partire dal valore base a.

Osserviamo infine che `Ite` e `Rec` non appartengono al formalismo primitivo ricorsivo, per il semplice fatto che il formalismo primitivo ricorsivo non consente di astrarre rispetto a funzioni.

Capitolo 4

Ricorsione di ordine superiore

Il potere espressivo del formalismo primitivo ricorsivo è estremamente ampio. Tutti gli esempi di algoritmi totali e di costrutti computazionali che abbiamo considerato fino ad ora sono risultati essere esprimibili mediante funzioni primtive ricorsive.

È naturale chiedersi se qualunque funzione calcolabile totale sia esprimibile in tale linguaggio. In termini equivalenti, ci si può chiedere se ogni programma terminante possa essere riscritto utlizzando unicamente i comandi `for` e `if-then-else` come unici costrutti di controllo di flusso.

La risposta è negativa. Un esempio notevole di funzione totale e calcolabile ma non esprimibile nel formalismo primitivo ricorsivo è la *funzione di Ackermann*.

4.1 La funzione di Ackermann

La seguente funzione

$$\begin{aligned} \texttt{ack}(0,0,y) &= y \\ \texttt{ack}(0,x+1,y) &= \texttt{ack}(0,x,y)+1 \\ \texttt{ack}(1,0,y) &= 0 \\ \texttt{ack}(z+2,0,y) &= 1 \\ \texttt{ack}(z+1,x+1,y) &= \texttt{ack}(z,\texttt{ack}(z+1,x,y),y) \end{aligned}$$

è nota come funzione di Ackermann.

La funzione di Ackermann è ben definita, implementabile, e terminante; tuttavia la forma di ricorsione necessaria per esprimerla sfugge al potere espressivo della ricorsione primitiva.

Proviamo a capire meglio il senso della definizione precedente. Un modo conveniente per analizzare la funzione di ackermann è quello di considerare le sue istanze parziali

$$\mathtt{ack}_{z,y}(x) = \mathtt{ack}(z, x, y)$$

In particolare, è immediato osservare che, in base alle prime due equazioni, $\mathtt{ack}_{0,y}(x) = x + y$. In base all'ultima equazione abbiamo inoltre che

$$\begin{aligned}
\mathtt{ack}_{z+1,y}(x) &= \mathtt{ack}_{z,y}(\mathtt{ack}_{z+1,y}(x-1)) \\
&= \mathtt{ack}_{z,y}(\mathtt{ack}_{z,y}(\mathtt{ack}_{z+1,y}(x-2))) \\
&= \underbrace{\mathtt{ack}_{z,y}(\mathtt{ack}_{z,y}(\ldots \mathtt{ack}_{z,y}(\mathtt{ack}_{z+1,y}(0))\ldots))}_{x volte} \\
&= \mathtt{Ite}\ \mathtt{ack}_{z+1,y}(0)\ \mathtt{ack}_{z,y}\ x
\end{aligned}$$

Dunque il risultato di $\mathtt{ack}_{z+1,y}(x)$ si ottiene iterando x volte la funzione del livello sottostante $ack_{z,y}$ a partire dal valore base $\mathtt{ack}_{z+1,y}(0)$. La terza e quarta equazione della funzione di Ackermann fissano rispettivamente tale valore base a 0 per $z = 1$ e a 1 per $z \geq 2$.

Esplicitando i primi livelli della funzione abbiamo:

$$\begin{aligned}
\mathtt{ack}(0, x, y) &= x + y \\
\mathtt{ack}(1, x, y) &= x * y \\
\mathtt{ack}(2, x, y) &= y^x \\
\mathtt{ack}(3, x, y) &= \left.\begin{matrix} y^{y^{\cdot^{\cdot^{y}}}} \end{matrix}\right\} x\ volte
\end{aligned}$$

La funzione di Ackermann cresce dunque *molto* velocemente; ad esempio $\mathtt{ack}(3, x, y) = 3^{27} > 10^{14}$. Ackermann introdusse questa funzione proprio ai fini di studiare la complessità computazionale delle funzioni primitive ricorsive. In particolare, è possibile dimostrare che ogni funzione primitiva ricorsiva f è limitata in tempo da una opportuna istanza parziale della funzione di Ackermann per valori di z e y che dipendono dalla struttura sintattica di f. Come corollario, ne consegue che la funzione di Ackermann stessa non può essere espressa nel formalismo primitivo ricorsivo (altrimenti sarebbe bounded da una sua qualche istanza parziale).

4.2 Iterare un iteratore

Abbiamo visto nella sezione precedente che il calcolo di $\texttt{ack}_{z+1,y}$ si può ottenere iterando x volte la funzione del livello sottostante $\texttt{ack}_{z,y}$ a partire da $\texttt{ack}_{z+1,y}(0)$.

Per ogni z, $\texttt{ack}_{z+2,y}(0) = 1$ e la funzione che permette di costruire la funzione di livello $z+2$ a partire da quella di livello $z+1$ è la funzione

$$\texttt{next}(f) = \lambda x.\texttt{Ite}\ 1\ f\ x$$

Possiamo dunque ottenere la funzione $\texttt{ack}_{z+2,y}$ iterando $z+1$ volte la funzione `next` a partire da $ack_{1,y} = \lambda x.x * y$.
Si ottiene pertanto la seguente definizione alternativa per la funzione di Ackermann:

$$\begin{aligned} \texttt{ack}(0,x,y) &= x+y \\ \texttt{ack}(1,x,y) &= x*y \\ \texttt{ack}(z+2,x,y) &= \texttt{Ite}\ \lambda x.x*y\ \texttt{next}\ (z+1)\ x \end{aligned}$$

Abbiamo pertanto espresso, in modo abbastanza banale, la funzione di Ackermann utlizzando esclusivamente degli iteratori.
Perchè dunque Ackermann non è primitiva ricorsiva?
Il punto nodale è nel *tipo* dei dati coinvolti nell'iterazione.

La ricorsione primitiva permette unicamente di iterare funzioni da naturali in naturali. Tuttavia, nel caso della definizione precedente abbiamo bisogno di iterare il *funzionale* `next`, cioè una funzione di ordine superiore (e l'iterazione stessa parte da un valore base che è una funzione). Per quanto possa apparire sorprendente, l'arricchimento del linguaggio mediante la semplice possibilità di trattare funzioni come oggetti di prima categoria, passandoli e restituendoli come risultati di altre funzioni, arreca in questo caso un *aumento del potere espressivo* del linguaggio.

4.3 Il sistema T di Gödel

Proviamo a generalizzare il formalismo primitivo ricorsivo ad un linguaggio di ordine superiore. Il sistema che otterremo è noto in letteratura con il nome di *Sistema T di Gödel*.

Per prima cosa, dobbiamo arricchire il sistema di tipi del linguaggio in modo da poter passare e sintetizzare funzioni come possibili valori di input

e output del linguaggio; oltre ad avere un certo insieme di tipi base (consideremo i booleani $\mathcal{B}$ e i naturali $\mathcal{N}$) avremo dunque la possibilità, dati due tipi arbitrari T_1 e T_2, di considerare funzioni da T_1 in T_2, cioè espressioni di tipo $T_1 \to T_2$. Assumeremo anche di avere, per ogni T_1 e T_2, il prodotto cartesiano $T_1 \times T_2$.

Definizione 4.1. *L'insieme dei tipi del Sistema T è il più piccolo insieme definito dalle seguienti regoli:*

- $\mathcal{B}$ *(booleani) e* $\mathcal{N}$ *(naturali) sono tipi;*
- *se* T_1 *e* T_2 *sono tipi allora lo è anche* $T_1 \times T_2$*;*
- *se* T_1 *e* T_2 *sono tipi allora lo è anche* $T_1 \to T_2$*.*

Useremo la notazione $t : T$ per indicare che il termine t ha tipo T.

L'operatore $\to$ è tradizionalmente *associativo a destra*, ovvero l'espressione $A \to B \to C$ deve essere intesa come $A \to (B \to C)$. Dunque, una funzione $f : A \to B \to C$ è una funzione che prende in input un valore di tipo A e restituisce una funzione di tipo $B \to C$, da non confondere con $f : (A \to B) \to C$ che al contrario prende in input una funzione da A in B e restituisce un risultato di tipo C.

Passiamo a definire i termini del linguaggio. Definiremo dapprima l'universo dei termini *rozzi*, che potranno anche essere anche mal tipati. Daremo poi le *regole di tipizzazione* che ci permetteranno di determinare l'insieme dei termini *ben tipati* del linguaggio, che constuiscono i soli programmi corretti del nostro formalismo.

Definizione 4.2. *L'insieme dei termini del sistema T è il piú piccolo insieme definito dalle seguenti regole:*

- *le variabili sono termini;*
- *le seguenti costanti sono termini:0,S,True,False,Fst,Snd,If,Rec.*
- *se M ed N sono termini allora lo è anche* $< M, N >$ *(coppia);*
- *se M ed N sono termini allora lo è anche* $(M\ N)$ *(applicazione della funzione M all'argomento N);*
- *se M è un termine e T è un tipo, allora* $\lambda x : T.M$ *è un termine (astrazione della variabile x di tipo T nel corpo M della funzione* $\lambda x : T.M$*).*

Definiamo ora le regole di tipizzazione. Innazi tutto, attribuiamo un tipo a tutte le costanti del linguaggio.

$$\begin{array}{lcl} 0 & : & \mathcal{N} \\ S & : & \mathcal{N} \to \mathcal{N} \\ True & : & \mathcal{B} \\ False & : & \mathcal{B} \\ Fst & : & T_1 \times T_2 \to T_1 \\ Snd & : & T_1 \times T_2 \to T_2 \\ If & : & T \to T \to \mathcal{B} \to T \\ Rec & : & T \to (\mathcal{N} \to T \to T) \to \mathcal{N} \to T \end{array}$$

Nelle regole precedenti, T, T_1 e T_2 devono essere interpretate come variabili di tipo, e i tipi che le contengono come dei tipi *polimorfi*, istanziabili a piacimento. In altri termini si deve immaginare di avere una infinità di costanti, ciascuna corrispondente ad una particolare istanza dei loro tipi.

In generale, il tipo di un termine dipende dai tipi dichiarati per le sue variabili libere; se ad esempio x è una variabile di tipo $\mathcal{N} \times \mathcal{B}$, allora il termine $< x, Snd(x) >$ ha tipo $(\mathcal{N} \times \mathcal{B}) \times \mathcal{B}$. La dichiarazione dei tipi per le variabili prende il nome di *contesto*, e viene abitulamente descritto come una sequenza di coppie $x : T$ che attribuiscono il tipo T ad una variabile x.
Il predicato di buona tipizzazione assume la forma

$$\Gamma \vdash t : T$$

che asserisce che il termine t ha tipo T nel contesto Γ. Tutte le variabili libere in t devono essere dichiarate in Γ.
Le regole di tipizzazione sono le seguenti:

coppia

$$\frac{\Gamma \vdash M : T_1 \qquad \Gamma \vdash N : T_2}{\Gamma \vdash < M, N > : T_1 \times T_2}$$

applicazione

$$\frac{\Gamma \vdash M : T_1 \to T_2 \qquad \Gamma \vdash N : T_1}{\Gamma \vdash (MN) \ : T_2}$$

λ-astrazione

$$\frac{\Gamma, x : T_1 \vdash M : T_2}{\Gamma \vdash \lambda x : T_1 . M : T_1 \to T_2}$$

la regola relativa alla lambda astrazione è probabilmente l'unica che richiede qualche commento esplicativo. Per verificare la corretta tipizzazione di $\lambda x : T_1.M$ in un contesto Γ devo estendere il contesto assumendo che x abbia il tipo dichiarato T_1 e procedere quindi a verificare la corretta tipizzazione del corpo M. Se nel contesto esteso $\Gamma, x : T_1\ M$ ha tipo T_2, allora anche il termine di partenza è ben tipato ed ha tipo $T_1 \to T_2$.

La descrizione del Sistema T è completata dalla definizione delle regole di riscrittura, che ne definiscono il calcolo:

$$\begin{array}{ll}
(Fst) & (Fst < M, N >) \to M \\
(Snd) & (Snd < M, N >) \to N \\
(\beta) & (\lambda x : T.M\ N) \to M[N/x] \\
(If_true) & (If\ M\ N\ true) \to M \\
(If_false) & (If\ M\ N\ false) \to N \\
(Rec_O) & (Rec\ M\ N\ O) \to M \\
(Rec_S) & (Rec\ M\ N\ (S\ P)) \to (N\ P\ (Rec\ M\ N\ P))
\end{array}$$

Lasciamo al lettore la cura di verificare che:

1. tutti i termini coinvolti nelle regole di riscrittura sono ben tipati;
2. per ogni regola di riscrittura, il tipo del termine nella parte sinistra della regola coincide con il tipo del termine nella parte destra (ovvero, il tipo del termine è invariante per riduzione).

L'iteratore è un caso particolare di ricorsione. In particolare:

$$Ite = \lambda a : T.\lambda h : T \to T.(Rec\ a\ \lambda z : \mathcal{N}.h) : T \to (T \to T) \to \mathcal{N} \to T$$

Come semplici esempi di utilizzo dell'iteratore, definiamo le funzioni di somma e prodotto:

$$plus\ n\ m = (Ite\ m\ S\ n) : \mathcal{N} \to \mathcal{N} \to \mathcal{N}$$

$$times\ n\ m = (Rec\ O\ (plusm); n) : \mathcal{N} \to \mathcal{N} \to \mathcal{N}$$

È banale quindi definire la funzione di Ackermann dentro al Sistema T. Poniamo innanzi tutto

$$next = (Ite\ (SO)) : (\mathcal{N} \to \mathcal{N}) \to (\mathcal{N} \to \mathcal{N})$$

Realizziamo ora la funzione test che prende un elemento di tipo T, una funzione da $\mathcal{N}$ in T, un naturale n e restituisce M se $n = 0$ e $(N\ p)$ se $n = (S\ p)$:

$$Test\ a\ f = (Rec\ a\ \lambda x : \mathcal{N}.\lambda y : T.(f x))$$

Allora,

$$\begin{aligned} ack\ z\ x\ y \;=\; & (Test \\ & \quad (plus\ x\ y) \\ & \quad (Test \\ & \qquad (mult\ x\ y) \\ & \qquad \lambda w : \mathcal{N}.(Ite\ \lambda x : \mathcal{N}.(mult\ x\ y)\ next\ (S\ w)\ x)) \\ & \quad z) \end{aligned}$$

4.4 Incompletezza dei formalismi totali

Abbiamo visto nella sezione precedente che il Sistema T di Gödel estende strettamente il formalismo primitivo ricorsivo. É naturale a questo punto porsi la stessa domanda di completezza che ci eravamo posti per le funzioni primitive ricorsive, ovvero se esiste una qualche funzione totale, intuitivamente calcolabile, ma non esprimibile nel Sistema T.
Anche in questo caso la risposta é affermativa. L'argomentazione che utilizzeremo in questo caso si applica anche al sistema primitivo ricorsivo e piú in generale ad ogni formalismo in grado di calcolare solo funzioni totali.

Supponiamo infatti di avere un formalismo che permetta di esprimere solo programmi terminanti. Se il formalismo é dato in maniera effettiva (ad esempio mediante una grammatica che ne definisca i programmi) é possibile fornire una numerazione effettiva P_n di tutti i programmi (ad esempio ordinandoli lessicograficamente). Dato un qualunque intero n siamo in grado di identificare il programma P_n e di eseguirlo. Tra i programmi piú interessanti che si possono esprimere in un linguaggio di pogrammazione vi é il cosiddetto *interprete*. L'interprete é un programma in grado di simulare l'esecuzione di ogni altro programma; formalmente é un programma che presa in input una coppia di interi n ed m, simula l'esecuzione del programma P_n sul valore di input m:

$$I < n, m >= P_n(m)$$

Data la natura effettiva della numerazione dei programmi e il contenuto algoritmico dei programmi stessi, l'interprete (benché tutt'altro che banale) é

intuitivamente calcolabile. Consideriamo la seguente funzione:

$$f(x) = I < x, x > +1$$

Se l'interpete ćalcolabile, allora anche f lo /'e (/'e sufficiente che il formalismo sia chiuso per composizione e alcune funzioni primitive quali le proiezioni e la funzione successore). Dunque, se il formalismo permettesse di esprimere tutte le funzioni intuitivamente calcolabili dovrebbe esistere un qualche programma P_k in grado di calcolare f. Ricordando la definizione dell'interprete avremmo, per input k:

$$(*)P_k(k) = f(k) = I < k, k > +1 = P_k(k) + 1$$

che é ovviamente contradditorio! In atre parole, *nessun formalismo in grado di calcolare solo funzioni totali é algoritmicamente completo, ed in particolare non é in grado di esprimere il proprio interprete.*

Si noti la consueta struttura diagonale dell'argomentazione precedente. Osserviamo anche che il ragionamento sfrutta in maniera essenziale l'ipotesi che i programmi (e dunque anche il loro interprete) siano totali. Se il programma P_k su inut k divergesse non avremmo una contraddizione in quanto entrambi i lati di (*) sarebbero indefiniti.

Capitolo 5

La Tesi di Church

Dai risultati dei capitoli precedenti é difficile immaginare che, anche ammettendo costrutti linguistici che possano generare divergenza, quali minimizzazione illimitata, cicli infiniti o ricorsione generale, sia possibile raggiungere la completezza algoritmica. Sembra piuttosto plausibile che, anche in questo caso, come avviene per i formalismi totali, fissato un determinato linguaggio sia sempre possibile trovarne un altro di maggiore espressivitá.

In realtá questo non accade e sembra piuttosto esistere un limite invalicabile corrispondente alla nozione intuitiva di funzione calcolabile mediante una procedura algoritmica. In effetti, tutti i linguaggi considerati fino ad ora nel tentativo di oltrepassare tale limite sono risultati essere mutuamente traducibili, nel senso che ogni funzione esprimibile in un dato formalismo lo é anche negli altri (preciseremo alcuni di questi formalismi in questo capitolo).

Questo suggerisce che *la definizione intuitiva di funzione algoritimica o calcolabile, coincida esattamente con le funzioni esprimibili in uno qualunque di questi formalismi*

Il risultato precedente non é dimostrabile in quanto non é possibile fornire una definizione precisa del concetto di algoritmo; pertanto si parla piuttosto di una *Tesi*, che va sotto il nome di Tesi di Church o tesi di Turing a seconda del diverso formalismo di riferimento (rispettivamente, funzioni parziali ricorsive o macchine di Turing).

É opportuno sottolinere, come osservato da Gödel nel 46, il carattere miracoloso di questo fenomeno, che consente di dare una definizione assoluta, cioé indipendente dal formalismo adottato, di una interessante nozione epistemologica, quale la nozione di calcolabilitá effettiva. Questo contrasta con

molte altre nozioni logiche quali la dimostrabilitá o la definibilitá che sono invece fortemente dipendenti dal sistema formale e dal suo linguaggio.

La Tesi di Church-Turing é ormai universalmente accettata, al punto che le dimostrazioni di Turing-completezza di nuovi formalismi sono ormai considerate quasi prive di interesse, se non per enfatizzare alcuni costrutti linguistici innovativi, il loro uso inteso e la loro traduzione in termini di altre forme di calcolo.

Dal punto di vista della Teoria della CalcolabilitÍa tesi di Church permette di trattare le funzioni calcolabili senza dover precisare un particolare formalismo di riferimento: é sufficente fornire un algoritmo in termini astratti per poter concludere l'esistenza di un agente di calcolo di calcolo in un qualunque formalismo completo (cioé la traducibilitá dell'algoritmo in un programma). Viceversa, dimostrando la non esistenza di un programma per il calcolo di una determinata funzione in qualunque formalismo completo, si dimostra l'assoluta impossibilit'ị calcolare tale funzione, indipendentemente dal linguaggio o dalle tecniche di calcolo utilizzate.

5.1 Funzioni parziali ricorsive

Una funzione f si dice definita per *minimizzazione* a partire da g se

$$f(\vec{x}) = \mu y.(g(\vec{x}, y) = 0)$$

dove $\mu y.(g(\vec{x}, y) = 0)$ é il piú piccolo numero naturale n tale che $g(\vec{x}, n) = 0$. L'espressione $f(\vec{x})$ si intende indefinita se non esiste nessun n per cui $g(\vec{x}, n) = 0$, oppure se tale n esiste ma per un qualche m minore di n il calcolo di $g(\vec{x}, m)$ diverge (ovvero se durante la ricerca progressiva di n a partire da 0 la chiamata a g non termina).

Definizione 5.1. *La classe delle* **funzioni ricorsive** *é la piú piccola classe di funzioni che contiene le funzioni primitive ricorsive ed é chiusa per minimizzazione.*

Un predicato si dice ricorsivo quando lo é la sua funzione caratteristica.

É dunque sufficiente aggiungere al formalismo primitivo ricorsivo il meccanismo di minimizzazione (illimitato) per ottenere un linguaggio di programmazione algoritmicamente completo.

Intuitivamente, la minimizzazione ha il potere espressivo di un moderno costrutto di tipo *while*. Siccome la ricorsione primitiva corrisponde alla

iterazione non e' sorprendente che quest'ultimo costrutto sia sussunto dalla minimizzazione, in presenza di un numero sufficiente di funzioni primitive. La dimostrazione tuttavia non é del tutto banale. L'idea consiste nel dimostrare l'esistenza di una funzione σ tale che per ogni sequenza di numeri naturali $a_0, \ldots, a_n$ esista un a che codifica talesequenza via σ, tale cioé che, per ogni i minore o uguale ad n

$$\sigma(a, i) = a_i$$

Avendo a disposizione σ, se f é definita per ricorsione primitiva da g e h

$$\begin{cases} f(0, \vec{x}) = g(\vec{x}) \\ f(y+1, \vec{x}) = h\left(y, f(y, \vec{x}), \vec{x}\right) \end{cases}$$

possiamo codificare il grafo di f fino ad y semplicemente cercando la tupla di valori che rispetta la definizione ricorsiva, nel modo seguente:

$$t(\vec{x}, y) = \mu z[\sigma(z, 0) = g(\vec{x}) \wedge \forall i < y, \sigma(z, i+1) = h\left(y, \sigma(z, i), \vec{x}\right)$$

A questo punto si puó dunque calcolare f nel modo seguente:

$$f(\vec{x}, y) = \sigma(t(\vec{x}, y), y)$$

Il problema si riduce quindi ad esprimere σ utlizzando unicamente composizione e minimizzazione. Esistono varie costruzioni possibili; quella originaria di Gödel utilizzava il teorema dei resti cinesi, calcolando le proiezioni della tupla come resti rispetto ad una opportuna sequenza di numeri primi tra loro. Queste definizioni, benché ingegnose, non sono particolarmente istruttive e dunque le omettiamo.

5.2 Ricorsione generalizzata

Alternativamente all'aggiunta di una forma di iterazione illimitata (del tipo della minimizzazione) si puó generalizzare il meccanismo di ricorsione eliminando i vincoli che ne garantiscono la terminazione. Per semplicitá é opportuno ammettere un costrutto primitivo di tipo condizionale (if then else); inoltre, onde evitare di allargare il sistema dei tipi del linguaggio, ametteremo che l'unico test ammissibile sia sul valore nullo: l'espressione condizionale

$$if\ e_1 = 0\ then\ e_2\ else\ e_3$$

assume dunque il valore e_2 quando e_1 é nullo, ed il valore e_3 in tutti gli altri casi. Ammetteremo inoltre il successore ed il predecessore come altre funzioni primitive. Una funzione ricorsiva generalizzata, é dunque una funzione f il cui corpo puó essere espresso per composizione di funzioni base, altre funzioni giá definite in precedenza, o della funzione stessa.

Ad sempio, le funzioni di somma e prodotto possono essere scritta nella forma seguente:

$$add(x, y) := if\ x = 0\ then\ y\ else\ succ(add(x - 1, y))$$

$$mul(x, y) := if\ x = 0\ then\ 0\ else\ add(x, mul(x - 1, y))$$

Anche l'operazione di minimizzazione puó essere facilmente definita per ricorsione. Data la funzione g, definiamo una funzione $min\_from(n)$ che calcola il valore minimo per cui $g(x)$ si annulla a partire da un valore base n:

$$min\_from(n) := if\ g(n) = 0\ then\ n\ else\ min\_from(n + 1)$$

Il valore $\mu x.g(x) = 0$ si ottiene quindi calcolando l'espressione $min\_from(0)$.

5.3 Lambda calcolo

Un formalismo estremamente semplice ed elegante di calcolo di funzioni, ampiamente studiato in letteratura, é il λ-calcolo. L'insieme dei lambda termini é il piú piccolo insieme Λ che contiene un insieme infinito di variabili Var ed é chiuso rispetto ai costrutti di *applicazione* e λ-*astrazione*

applicazione se $M, N \in \Lambda$, allora l'applicazione di M ad N, denotata $(M\ N)$, appartiene ancora a Λ.

λ-astrazione se $x \in Var$ e $M \in \Lambda$, allora la funzione che si ottiene per astrazione di x in M, denotata $\lambda x.M$, appartieneancora a Λ.

La lambda astrazione *lega* la variabile x nel corpo M della funzione (che costituisce il campo di applicazione, o scope, del binder). Una occorrenza di una variabile x si dice *legata* se cade nel campo di applicazione di un binder per x, e libera altrimenti. Nel caso in cui una variabile x cada nel campo d'applicazione di piú binder relativi ad x si intende legata da quello piú interno. L'insieme delle variabili libere di un termine é l'insieme dei nomi

di variabili che ammettono delle occorrenze libere.
L'unica regola di riduzione del λ-calcolo é la cosiddetta β-riduzione:

$$(\lambda x.M\ N) \to M[N/x]$$

dove $M[N/x]$ denota il termine che si ottiene rimpiazzando tutte le occorrenze libere di x in M con l'argomento N. La regola corrisponde al procedimento di istanziazione di un un parametro formale con uno attuale al momento della applicazione di una funzione ad un argomento.

Una qualunqe (sotto-)espressione della forma $(\lambda x.M\ N)$ viene detta *redex*, termine che deriva dalla abbreviazione del termine inglese *reducible expression*. L'ordine e la posizione in cui possono essere eseguiti i redex dipende dalla strategia di riduzione del termine. Nel caso piú generale non si pone nessun vincolo, e ogni redex é potenzialmente riducibile.

Uno degli aspetti piú interessanti del *lambda*-calcolo é il modo in cui riesce a simulare il meccanismo di ricorsione generalizzata. Immaginiamo di avere una definizione ricorsiva di una funzione f, cioé una definizione della forma

$$(*) \qquad f = F(f)$$

dove F é una qualche espressione che richiama la funzione f al proprio interno. La definizione (*) caratterizza chiaramente f come un *punto fisso* di F. Dal punto di vista operazionale, ció che ci interessa e' che la chiamata ad f si espande al termine $F(f)$. Se dunque, dato un qualunque termine M fossimo in grado di calcolarne un punto fisso, ed in particolare un termine Θ_M tale che $\Theta_M \to (M\ \Theta_M)$, allora potremmo simulare nel *lambda*-calcolo il meccanismo di ricorsione.

Teorema 5.2. *Per ogni termine M esiste un termine Θ_M tale che $\Theta_M \to (M\ \Theta_M)$.*

Dimostrazione. Sia $A_M = \lambda x.(M\ (x\ x)$ e poniamo $\Theta_M = (A_M\ A_M)$. Abbiamo:

$$\begin{array}{rcll}
\Theta_M & = & (A_M\ A_M) & \text{per def. di } \Theta_M \\
 & = & (\lambda x.(M\ (x\ x))\ A_M) & \text{per def. di } A_M \\
 & \to & (M\ (x\ x))[A_M/x] & \text{per } \beta\text{-riduzione} \\
 & = & (M\ (A_M\ A_M)) & \text{per def.di sostituzione} \\
 & = & (M\ \Theta_M) & \text{per def.di } \Theta_M
\end{array}$$

□

5.4 Logica Combinatoria

Un formalismo di calcolo estremamente semplice e divertente, ma di grande importanza fondazionale é la cosiddetta Logica Combinatoria. Tutti i programmi della logica combinatoria sono scritti utilizzando due sole costanti S e K. L'unico meccanismo di costruzione di nuovi termini é l'applicazione $(M\ N)$ di due termini dati. In generale, si intende che l'applicazione sia associativa a sinistra, scrivendo dunque $(M_1\ M_2 \dots M_n)$ come abbreviazione di $(\dots (M_1\ M_2) \dots M_n)$. Due semplici regole di calcolo governano la riduzione:

$$(K\ M\ N) \to M$$

$$(S\ M\ N\ P) \to (M\ P\ (N\ P))$$

dove M, N e P sono termini generici del calcolo.
Ogni termine del calcolo é detto combinatore. Ad esempio, un utile combinatore é il termine I definito come $(S\ K\ K)$. I si comporta come una identitá (da cui il nome):

$$\begin{array}{rcl} (I\ M) & = & (S\ K\ K\ M) \\ & \to & (K\ M\ (K\ M)) \\ & \to & M \end{array}$$

Visto che il calcolo é chiuso rispetto alla applicazione, il trucco per dimostrarne la completezza algoritmica é quello di far vedere come si possa simulare il meccanismo di λ-astrazione, ottenendo quindi il potere espressivo del λ-calcolo.

A questo fine, estendiamo innanzi tutto il calcolo con un insieme infinito di variabili. Vogliamo ora dimostrare come, dato un qualunque combinatore M ne esista un altro, che denoteremo $\lambda^* x.M$ che si comporta come la lambda astrazione di M rispetto ad x, nel senso che per ogni termine N:

$$(*) \qquad (\lambda^* x.M\ N) \to M[N/x]$$

Il termine $\lambda^* x.M$ é definito per induzione strutturale su M:

- se $M = x$, poniamo $\lambda^* x.x = I = (S\ K\ K)$
- se M non contiene x (dunque, in particolare, se M é K, S o una variabile diversa da x), allora poniamo $\lambda^* x.M = (K\ M)$
- se infine $M = (P\ Q)$, allora $\lambda^* x.(P\ Q) = (S\ \lambda^* x.P\ \lambda^* x.Q)$

Dimostriamo $(*)$ per induzione strutturale su M:

- se $M = x$ allora $\lambda^* x.x = I$ ed abbiamo giá dimostrato che $(I\ M)$ riduce ad M
- se M non contiene x, allora $M[N/x] = M$, ed abbiamo $(\lambda^* x.MN) = (K\ M\ N) \to M$
- se $M = (P\ Q)$, allora per ipotesi induttiva possiamo supporre che $(\lambda^* x.P\ N) \to P[N/x]$ e $(\lambda^* x.Q\ N) \to Q[N/x]$; abbiamo quindi

$$\begin{array}{rcll} (\lambda^* x.(P\ Q)\ N) & = & (S\ \lambda^* x.P\ \lambda^* x.Q\ N) & \\ & = & ((\lambda^* x.P\ N)\ (\lambda^* x.Q\ N)) & \text{per riduzione di S} \\ & = & (P[N/x]\ Q[N/x]) & \text{per ip. induttiva} \\ & = & (P\ Q)[N/x] & \text{per def. di sostituzione} \end{array}$$

Calcoliamo, a titolo d'esempio, il termine $\lambda^* x.\lambda^* y.(x\ y)$. Innanzi tutto,

$$\lambda^* y.(x\ y) = (S\ \lambda^* y.x\ \lambda^* y.y) = (S\ (K\ x)\ I)$$

Dobbiamo ora calcolare $\lambda^* x.(S\ (K\ x)\ I)$:

$$\begin{array}{rcl} \lambda^* x.(S\ (K\ x)\ I) & = & (S\ \lambda^* x.(S\ (K\ x))\ \lambda^* x.I) \\ & = & (S\ (S\ \lambda^* x.S\ \lambda^* x.(K\ x))\ \lambda^* x.I) \\ & = & (S\ (S\ (K\ S)\ (S\ \lambda^* x.K\ \lambda^* x.x))\ (K\ I)) \\ & = & (S\ (S\ (K\ S)\ (S\ (K\ K)\ I))\ (K\ I)) \end{array}$$

Ci aspettiamo che il combinatore $(\lambda^* x.\lambda^* y.(x\ y)\ M\ N)$ riduca a $(M\ N)$; proviamo a verificarlo:

$$\begin{array}{rcl} (\lambda^* x.\lambda^* y.(x; y)\ M\ N & = & (S\ (S\ (K\ S)\ (S\ (K\ K)\ I))\ (K\ I)\ M\ N) \\ & \to & ((S\ (K\ S)\ (S\ (K\ K)\ I)\ M)\ (K\ I\ M)\ N) \\ & \to & ((S\ (K\ S)\ (S\ (K\ K)\ I)\ M)\ I\ N) \\ & \to & ((K\ S\ M)\ (S\ (K\ K)\ I\ M)\ I\ N) \\ & \to & (S\ (S\ (K\ K)\ I\ M)\ I\ N) \\ & \to & (S\ ((K\ K\ M)\ (I\ M))\ I\ N) \\ & \to & (S\ (K\ M)\ I\ N) \\ & \to & ((K\ M\ N)\ (I\ N)) \\ & \to & (M\ (I\ N)) \\ & \to & (M\ N) \end{array}$$

5.5 Macchine di Turing

Le macchine di Turing forniscono un modello di calcolo di natura sensibilmente differente da tutti quelli visti sino ad ora. L'enorme novitá introdotta da Turing śtata quella di affrontare il problema della calcolabilitá enfatizzando l'esistenza di un *agente di calcolo*, e di studiare quindi le potenzialitá e gli strumenti a disposizione di tale agente, ed in particolare l'esistenza di una struttura di memoria e i meccanismi che ne governano l'accesso.

L'obiettivo di Turing era quello di catturare tutte le potenzialitá operazionali insite nell'idea di *passo* algoritmico, enfatizzando in modo particolare la gestione *inevitabilmente finitaria* di un supporto di memoria *potenzialmente infinito*.

La Macchina di Turing (abbreviata in MdT d'ora in avanti) si compone di una unita' di controllo a stati finiti, un nastro di lunghezza infinita, ed una tesina di lettura e scrittura che consente l'interazione tra il nastro e la struttura di controllo.

Il nastro é diviso in celle in cui puó essere memorizzato un qualunque simbolo di un alfabeto fissato $\Sigma = a_1, \ldots, a_n$. Un simbolo particolare "*" (bianco) denota l'assenza di scrittura. Ad ogni istante della computazione solo una porzione finita di nastro sar'a scritta, mentre tutte le altre celle conterrano il carattere bianco (in termini moderni, le celle non saranno inizializzate).

La testina si intende posizionata su di una cella del nastro, detta cella in lettura. L'unitá di controllo puó trovarsi in uno solo di un insieme finito di stati $Q = q_1, \ldots, q_m$. Il nastro puø' essere spostato verso destra o verso sinistra, permettendo la scrittura e la lettura di nuovi simboli.

La procedura di calcolo procede per *passi discreti*. Ad ogni istante della computazione la testina legge un qualche carattere a dal nastro; in funzione del carattere letto e dello stato interno q la macchina decide, in accordo ad un insieme finito di istruzioni, di

- transire in un nuovo stato q'
- sostituire sul nastro il carattere appena letto con un nuovo carattere a'
- spostare la testina di lettura di una posizione verso destra (R) o verso sinsitra (L)

Ogni istruzione della macchina puó essere quindi descritta da una *quintupla* (q, a, q', a', m), dove $m \in \{R, L\}$, con la semantica intuitiva appena descritta.

Diamo ora la definizione formale:

Definizione 5.3. *Una macchina di Turing é definita da una quadrupla* (Q, Σ, t, q_0) *dove:*

- Q *é un insieme finito di stati*
- Σ *é un alfabeto finito (indichiamo con* $\hat{\Sigma}$ *l'insieme* Σ *esteso con il carattere "*", detto carattere bianco)*
- t *é un sottoinsieme finito di* $Q \times \hat{\Sigma} \times Q \times \hat{\Sigma} \times \{R, L\}$
- $q_0 \in Q$ *é lo stato iniziale.*

L'insieme t, detto matrice della MdT, é dunque costituito da un insieme *finito* di *quintuple* della forma (q, a, q', a', m), dove $q, q' \in Q$, $a, a' \in \hat{\Sigma}$ e $m \in \{R, L\}$.

Una configurazione istantanea di una MdT é caratterizzata dallo stato interno della macchina, dallo stato del nastro e dalla posizione della testina sul nastro di lettura. Le ultime due informazioni possono essere descritte fornendo lo stato del nastro a destra e a sinistra della testina (comprendendo arbitrariamente il simbolo in lettura nella porzione di destra). Siccome inoltre ad ogni istante solo una porzione finita di nastro puó essere scritta, il nastro é descritto da una stringa *finita* di caratteri sull'alfabeto $\hat{\Sigma}$, intendendo che la porzione restante del nastro contiene solo caratteri bianchi.

Definizione 5.4. *Una* configurazione istantanea *di una MdT é una tripla* $\sigma q \tau$ *dove* $q \in Q$ *e* $\sigma, \tau \in \hat{\sigma}^*$. q *é lo stato interno della macchina;* σ *e* τ *sono le stringhe finite che descrivono i due seminastri a destra e a sinistra della testina. Il simbolo in lettura é il primo simbolo di* τ*, oppure * se* σ_r *é la stringa vuota.*

Definiamo ora una funzione di transizione $\vdash$ tra configurazioni istantantaee che cattura la nozione di *passo atomico* di calcolo.

Definizione 5.5. *Data una MdT* (Q, Σ, t, q_0) *la relazione* $\vdash_t$ *é cosi definita:*

$$\begin{array}{ll} \sigma q a \tau \vdash_t \sigma a' q' \tau & \text{se } (q, a, q', a', R) \in t \\ \sigma b q a \tau \vdash_t \sigma q' b a' \tau & \text{se } (q, a, q', a', D) \in t \end{array}$$

Nelle regole precedenti si suppone che la porzione del nastro dscritta da σ e τ sia estesa opportunamente con caratteri bianchi qualora se ne abbia necessitá, per operazioni di lettura o scrittura. Ad esempio, come caso particolare della prima regola, otteniamo

$$\sigma q \vdash_t \sigma a' q' \tau \text{ se } (q, *, q', a', R) \in t$$

e cosí via.
Se per ogni coppia a, q esiste al piú una quintupla della forma $qaq'a'm \in t$, allora la relazione di transizione é univoca, e la MdT si dice deterministica.

Una configurazione istantanea della macchina si dice finale, o di terminazione, se non esiste nessuna transizione ammissibile per essa.

Indichiamo con $\vdash_t^*$ la chiusura riflessiva e transitiva della relazione $\vdash_t$. La funzione $\vdash_t^*$ é detta funzione di transizione multi-passo e mette in relazione ogni configurazione istantanea con le configurazione raggiungibili da essa durante la computazione.

Una computazione di una MdT é una sequenza di configurazioni

$$C_0 \vdash_t C_1 \cdots \vdash_t C_n$$

dove C_n úna configurazione finale. La macchina di Turing *diverge* per una data configurazione di partenza C_0 se nessuna configurazione finale ŕaggiungibile.

Per associare ad una macchina di Turing una funzione calcolata da essa si deve stabilire una *convenzione* di scrittura dell'input e di lettura del risultato. Ad esempio possiamo convenire che i dati di ingresso debbano essere scritti, in un qualche alfabeto, a destra della testina e separati da caratteri bianchi; il resto del nastro deve essere interamente bianco all'inizio della computazione. Similmente, ammetteremo che il valore di ritorno sia costituito dall'insieme delle celle del nastro a destra della testina fino al primo carattere bianco (senza richiedere quindi esplicitamente che la macchina ripulisca l'intero nastro prima del proprio arresto).

In base a queste convenzioni possiamo dare la seguente definizione formale:

Definizione 5.6. *Una funzione parziale $f(x_1, \ldots, x_n)$ é calcolata da una MdT (Q, Σ, t, q_0) se esiste una computazione*

$$q_0 x_1 * x_2 * \cdots * x_n \vdash_t \sigma q f(x_1, \ldots, x_n) * tau'$$

dove σ, tau e q sono arbitrari.

5.5.1 Simulazione di MdT mediante ricorsione

É relativamente semplice codificare MdT mediante funzioni parziali ricorsive. Supponiamo che ad ogni carattere a del nastro (e ad ogni stato $q \in Q$) sia univocamente associato un intero $\overline{a}$ ($\overline{q}$). Avendo a disposizione una funzione di codifica delle coppie, rappresentiamo una porzione di nastro $\epsilon = a_1 \dots a_{n-1} a_n$ mediante il numero $\overline{\epsilon} =< \overline{a}_1, \cdots < \overline{a}_{n-1}, < \overline{a}_n, 0 >> \cdots >$[1].

La configurazione istantanea $\sigma q \tau$ é descritta dalla tripla $< \overline{q}, < \overline{t}au, \overline{(}\sigma^r) >>$. Supponiamo che la MdT sia deterministica, e che l'insieme delle quintuple sia descritto da una funzione parziale $f : Q \times \hat{\Sigma} \to Q \times \hat{\Sigma} \times \{0, 1\}$ dove $0 = R$ e $1 = L$. Estendiamo la funzione ad una funzione totale f', restituendo, dove f non era definita, una tripla il cui stato é un nuovo simbolo q_F che rappresenta uno stato di terminazione (gli altri elementi non sono significativi). Siccome la funzione f' ha solo un numero finito di input significativi puó essere scritta banalmente nel formalismo ricorsivo (si tratta di una sequenza di if-then-else). Sia F tale funzione.

Per descrivere la funzione di transizione utilizziamo, per sempliciá di lettura, due importanti convenzioni:

destrutturazione dell'input: scriveremo $f(< x, y >) = e(x, y)$ come abbreviazione di $f(n) = e(fst(n), snd(n))$

let-in: scriveremo let $x = e_1$in $e_2(x)$ come sintassi alternativa di $e1(e_2)$).

Definiamo innanzi tutto una funzione $fill$ che inizializza una nuova cella del nastro, nel caso in cui si sia esaurito:

$$fill(l) = \text{if } l = 0 \text{ then } < \overline{(}*), 0 > \text{ else } l$$

La funzione di transizione in un passo T é allora definita nel modo seguente:

$$\begin{array}{l} T(< q, < r, l >>) = \\ \quad \text{let } a, rt = \text{fill}(r) \text{ in} \\ \quad \text{let } < q' < a', m >>= F(q, a) \text{ in} \\ \quad \text{if } m = 0 \text{ then } < q' < rt, < a', l >> \quad (*rightmove*) \\ \quad \text{else} \\ \quad\quad \text{let } b, lt = \text{fill}(l) \text{ in} \\ \quad\quad < q', < b, < a', rt >>, lt \end{array}$$

[1] Si noti lo 0 che termina la sequenza e che intuitivamente rappresenta la lista vuota.

Possiamo a questo punto scrivere un semplice programma ricorsivo che simula la computazione di una macchina di Turing, da una certa configurazione iniziale C in input, fino alla configurazione di arresto, se esite:

$$\text{eval}(C) = \text{ let } \ < q, < r, l >>= C \text{ in if } q = q_F \text{ then } C \text{ else eval}(T(C))$$

La discussione precedente permette di fare una osservazione estremamente interessante. La funzione T richiede molto poco potere espressivo: essenzialmente ragionamento per casi e coppie. D'altra parte, se conosciamo un bound temporale $t(n)$ al numero di passi della computazione (in funzione dell'input n), non abbiamo bisogno di utilizzare ricorsione nel corso della valutazione: possiamo semplicemente *iterare* la funzione T per $t(n)$ passi (in questo caso T deve essere modificata in modo che si coporti come l'identitá su configurazioni con stato finale). Dunque, se il formalismo é abbastanza espressivo da consertire l'iterazione, se la funzione t é esprimibile nel formalismo, lo é anche ogni funzione (Turing-)calcolabile *limitata temporalmente da* $t(n)$. Ad esempio, siccome la funzione esponenziale é banalmente esprimibile nel formalismo primitivo ricorsivo, lo é anche *ogni* funzione Turing-calcolabile in tempo esponenziale. Similmente, ogni esponenziale generalizzato (ogni valutazione parziale della funzione di ackermann $ack_z(x, y)$ dove si é fissato il primo parametro) é esprimibile nel formalismo primitivo ricorsivo e dunque lo sono anche *tutti* i programmi con un tempo di calcolo limitato da tali funzioni.

Capitolo 6

Funzioni non calcolabili

In base alla tesi di Church, non é possibile calcolare funzioni non esprimibili nel formalismo parziale ricorsivo. Resta aperto il problema di capire se esistano funzioni su naturali *matematicamente ben definite* ma non calcolabili.

Vedremo in questo capitolo che questo é effettivamente il caso.
Invece di lavorare su di uno specifico formalismo cercheremo di avere un approccio di natura piú assiomatica, discutendo di volta in volta le proprietá (tipicamente proprietá di esistenza di certe funzioni o di chiusura rispetto a determinati costrutti) che stiamo assumendo sul nostro sistema di calcolo.

Supponiamo quindi di avere una enumerazione φ_n di una classe di funzioni parziali. L'idea intuitiva é quella di immaginare di avere una enumerazione di agenti di calcolo o programmi P_i e che ϕ_i sia il la funzione calcolata dal programma dall'i-esimo programma P_i. In generale avremo che una stessa funzione puó essere calcolata da pi$\breve{}$ programmi, dunque la funzione φ_i di enumerazione non é in generale iniettiva: $\varphi_i = \varphi_j$ non implica $i = j$ (due programmi possono calcolare la stessa funzione ma essere differenti). Supporremo in generale che la classe di funzioni sia chiusa rispetto all'operazione di *composizione* e che contenga proiezioni, costanti, e costrutti di analisi per casi (if-then-else).

6.1 Il problema della terminazione

La funzione φ_i puó essere parziale. Useremo la notazione $\varphi_i(n)\downarrow$ per indicare che la funzione é definita (o converge) per input n, e $\varphi_i(n)\uparrow$ quando é

indefinita (o diverge) per tale input.
Una funzione di estremo interesse é il cosiddetto test di terminazione. Matematicamente, tale funzione definita nel modo seguente:

$$g(i,x) \quad = \quad \begin{cases} 1 & \text{se } \varphi_i(x) \downarrow \\ 0 & \text{se } \varphi_i(x) \uparrow \end{cases}$$

La funzione prende in input un indice i di un agente di calcolo e un valore n e vuole determinare se la computazione di P_i su input n termina oppure diverge, cioé se la funzione φ_i é o meno definita per input n.
Consideriamo ora la funzione f, definita come segue:

$$f(x) \quad = \quad \begin{cases} 1 & se\ g(x,x) = 0 \\ \uparrow & se\ g(x,x) = 1 \end{cases}$$

Se g é esprimibile nel nostro modello di calcolo, e il modello é chiuso per composizione, e contiene le proiezioni e l'analisi per casi, e un costrutto che induca divergenza, allora anche la funzione f deve necessariamente essere esprimibile. Esiste dunque un qualche m tale per cui $f = \varphi_m$.
Ci chiediamo quanto vale $\varphi_m(m)$:

$$\varphi_m(m) \quad = \quad \begin{cases} 1 & se\ g(m,m) = 0 \\ \uparrow & se\ g(m,m) = 1 \end{cases}$$

ma applicando la definizione di g notiamo che

$$\varphi_m(m) \quad = \quad \begin{cases} 1 & se\ \varphi_m(m) \uparrow \\ \uparrow & se\ \varphi_m(m) \downarrow \end{cases}$$

il che è assurdo. Questo dimostra che la funzione g di terminazione *non é esprimibile* in nessun modello di calcolo che ammetta divergenza, e con minime proprietá di chiusura. In particolare, non é una funzione parziale ricorsiva. Per la tesi di Church, concludiamo che *il problema della terminazione non é algoritmicamente risolubile.*

Si noti che, come corollario *della dimostrazione* del teorema precedente si ottiene anche che la cosiddetta funzione diagonale di terminazione, definita nel modo seguente

$$k(x) \quad = \quad \begin{cases} 1 & \text{se } \varphi_x(x)\downarrow \\ 0 & \text{se } \varphi_x(x)\uparrow \end{cases}$$

non é calcolabile.

Benché di minore interesse pratico, la funzione diagonale di terminazione cattura la vera essenza della dimostrazione precedente e, come vedremo, gioca un ruolo di primaria importanza nella teoria della Calcolabilitá.

6.2 La proprietá s.m.n.

Le funzioni parziali che consideriamo sono in generale funzioni a n-argomenti; utilizzeremo talvolta la notazione ϕ_i^n per enfatizzare l'arietá n della funzione di indice i.

Definizione 6.1. *Una classe di funzioni parziali gode della proprietá s.m.n. se, per ogni funzione f^{m+n} appartenente alla classe, esiste una funzione* totale s_n^m *tale che, per ogni $\vec{x}_m, \vec{y}_n$*

$$\varphi_{s_n^m(\vec{x}_m)}(\vec{y}_n) = f(\vec{x}_m, \vec{y}_n)$$

La proprietá s.m.n. afferma due cose:

1. la classe di funzioni é chiusa rispetto alla *valutazione parziale*: se si fissano i primi m parametri di f^{m+n}, ottengo ancora una funzione g interna alla classe (se la funzione f é parziale, g sara' anch'essa parziale, ma esiste sempre comunque siano fissati gli m parametri).

2. l'indice di questa funzione g é calcolabile in termini dei parametri fissati mediante una funzione della classe stessa

Se ad esempio la classe di funzioni che consideriamo e' la classe delle funzioni parziali ricorsive, il punto 1) é evidente, mentre il punto 2), invocando la tesi di Church, si riduce alla constatazione che possiamo effettivamente calcolare l'indice del programma istanziato in funzione dei parametri fissati (in quanto il programma istanziato é costruito in modo effettivo a partire dal programma originale, e la numerazione stessa dei programmi si suppone essere effettiva). Si puó dimostrare formalmente che tutti i formalismi considerati nel capitolo precedente godono della proprietá s.m.n. (in riferimento alle funzioni parziali calcolabili si parla dunque abitualmente di *teorema* s.m.n.).

La propriet'a s.m.n. é uno degli strumenti piú utili della teoria della calcolabilit‹

Come una prima applicazione, poviamo a considerare il problema della totalitá delle funzioni, ovvero la funzione *total* che permette di discriminare le funzioni parziali da quelle totali:

$$total(i) \quad = \quad \begin{cases} 1 & se\ \varphi_i\ è\ totale \\ 0 & altrimenti \end{cases}$$

Consideriamo una funzione ausiliaria definita nel modo seguente:

$$f(x,y) \quad = \quad \begin{cases} 0 & se\ \varphi_x(x)\downarrow \\ \uparrow & altrimenti \end{cases}$$

La funzione f é intuitivamente calcolabile: presi in input x e y basta lanciare la computazione e $\varphi_x(x)$ e attendere: se la computazione termina si restituisce in output 0, altrimenti il calcolo di $f(x,y)$ diverge. Per il teorema s.m.n. esiste dunque una funzione totale calcolabile h tale che

$$\varphi_{h(x)}(y) = f(x,y) \quad = \quad \begin{cases} 0 & se\ \varphi_x(x)\downarrow \\ \uparrow & altrimenti \end{cases}$$

La funzione di indice $h(x)$ é dunque la funzione costante 0 se $\varphi_x(x)\downarrow$ e la funzione ovunque divergente altrimenti. Avremmo allora

$$total(h(x)) \quad = \quad \begin{cases} 1 & \text{se } \varphi_x(x)\downarrow \\ 0 & \text{se } \varphi_x(x)\uparrow \end{cases}$$

Se *total* fosse calcolabile, anche $total \circ h$ lo sarebbe, contraddicendo il fatto che il problema della terminazione diagonale non é risolubile in modo algoritmico. Dunque *total* non é calcolabile.

6.2.1 Equivalenza di programmi

Come altro esempio di applicazione del teorema s.m.n consideriamo il problema della decisone della equivalenza tra due programmi. In termini matematici, ci si chiede se la seguente funzione **stessa** funzione.

$$eq(i,j) \quad = \quad \begin{cases} 1 & se\ \varphi_i \approx \varphi_j \\ 0 & altrimenti \end{cases}$$

sia calcolabile. La funzione $eq(i, j)$ testa la cosiddetta equivalenza estensionale tra le due funzioni, cioé se per ogni input n, $\varphi_i(n) = \varphi_j(n)$.
Consideriamo la funzione costante 0, e sia m un indice per essa (i.e. per ogni n, $\varphi_m(n) = 0$. Consideriamo la funzione h della sezione precedente:

$$\varphi_{h(x)}(y) \quad = \quad \begin{cases} 0 & se\ \varphi_x(x) \downarrow \\ \uparrow & se\ \varphi_x(x) \uparrow \end{cases}$$

In base alle sepcifiche date, avremmo

$$f(h(x), m) \quad = \quad \begin{cases} 1 & se\ \varphi_x(x) \downarrow \\ 0 & altrimenti \end{cases}$$

risolvendo il problema della terminazione diagonale.

6.3 La funzione universale

Una classe di funzioni φ_i contiene il proprio interprete (o funzione universale) se esiste una funzione u tale che per ogni i, x

$$u(i, x) = \varphi_i(x)$$

Abbiamo giá dimostrato che una classe di funzioni totali (con minime proprietá di chiusura) non puó contenere il proprio interprete; al contrario, la classe delle funzioni parziali calcolabili ammette funzioni universali.
Diamo ora una dimostrazione alternativa della non risolubilitá algoritmica del problema della totalitá che utilizza la nozione di interprete. L'idea é quella di usare il test di totalitá per filtrare gli indici delle funzioni parziali ottennedo una enumerazione di tutte e solo le funzioni totali calcolabili. Siccome avremmo un interprete, ne possiamo ricavare una contraddizione.
Supponiamo dunque per assurdo che la funzione definita *total* nella sezione precedente sia calcolabile, e consideriamo la seguente funzione:

$$g(x) \quad \equiv \quad \begin{cases} g(0) & = \mu i(total(i) = 1) \\ g(x+1) & = \mu i(y > i(x) \wedge total(i) = 1) \end{cases}$$

La funzione g ennumera *tutte e solo* le funzioni totali calcolabili. Avendo supposto *total* calolabile, anche g lo é; inoltre, g é totale in quanto esistono infiniti indici di funzioni totali e dunque la minimizzazzione nel corpo

della g ha sicuramente termine. Definiamo ora una nuova funzione h per diagonalizzazione nel modo seguente:

$$h(x) \quad = \quad \varphi_{g(x)}(x) + 1$$

La funzione h é totale e calcolabile. Siccome g enumerava *tutte* le funzioni parziali calcolabili, dovrebbe esistere un qualche indice m tale che $h = \varphi_{g(m)}$. Avremmo allora:

$$\varphi_{g(m)}(m) = h(m) = \varphi_{g(m)}(m) + 1$$

che é una contraddizione (in quanto g(m) é totale!).

6.4 Il predicato T di Kleene

Abbiamo giá osservato che non é possibile decidere la terminazione di un programma (parziale ricorsivo) su di un determinato input. D'altra parte é possibile determinare in modo effettivo se un determinato programma termina la propria computazione entro un tempo fissato t (o per meglio dire entro un certo numero di *passi* elementari di computazione). E' sufficiente infatti lanciare l'esecuzione del programma e osservare lo stato della computazione dopo il detto numero di passi. Il predicato $T(i, n, < t, m >)$ di Kleene afferma appunto che la funzione calcolata dal programma di indice i su input n temrina la propria computazione entro t passi e fornisce come output m. Avendo a disposizione il predicato di Kleene, possiamo facilmente definire un inteprete: per emulare il programma i su input n ci basta cercare il minimo y che realizza $T(i, n, y)$ e poi prendere la seconda componente.
Questo risultato viene abitualmente espresso nella seguente forma:

Teorema 6.2. *(della forma normale di Kleene) Esistono due funzioni totali calcolabili p e t tali che per ogni z*

$$u(z, x) = \varphi_z(x) = p(\mu y.(t(z, x, y) = 1))$$

Dimostrazione. t é la funzione caratteristica del predicato T di Kleene (totale e calcolabile), e p é la seconda proiezione della coppia. □

É possibile dimostrare che la funzione t non richiede una grande complessitćomputazionale e risulta esprimibile, ad esempio, nel formalismo primitivo ricosivo. In termini moderni, il risultato di kleene dice tra l'altro che ogni

programma puó essere scritto utlizzando unicamente cicli iteratitivi, ed al piu un unica applicazione esterna di un costrutto di tipo while.

Dato il comportamento triviale di p é legittimo domandarsi se non se ne possa fare a meno e semplificare l'enunciato del teorema precedente all'esistenza di una opportuna funzione t totale calcolabile tale che

$$\varphi_z(x) = \mu y(t(z, x, y) = 1)$$

Sorprendentemente, questo non é possibile. Vale infatti il seguente risultato

Teorema 6.3. *Esiste g parziale calcolabile tale che per nessuna funzione totale calcolabile f si ha che $\varphi_z(x) = \mu y(f(x, y) = 1)$*

Dimostrazione. Consideriamo la seguente funzione calcolabile

$$g(x) \quad = \quad \begin{cases} x & \textit{se } \varphi_x(x) \downarrow \\ \uparrow & \textit{altrimenti} \end{cases}$$

Supponiamo che $g(x) = \mu y(f(x, y) = 1)$ per qualche f totale calcolabile. Si presentano due casi:

1. se $\varphi_x(x) \downarrow$ allora $g(x) = \mu y(f(x, y) = 1) = x$, che implica $f(x, x) = 1$.
2. se $\varphi_x(x) \uparrow$ allora $g(x) = \mu y(f(x, y) = 1) \uparrow$ da cui $f(x, x) \neq 1$.

Quindi $f(x, x) = 1$ se e solo se $\varphi_x(x) \downarrow$ che permetterebbe di risolvere il problema della terminazione diagonale. □

6.4.1 Tre funzioni simili

Vogliamo ora considerare tre funzioni che hanno una specifica molto simile e investigarne la calcolabilitá.
La prima funzione vuole determinare o meno l'esistenza, per una funzione di indice i di almeno un valore di input n su cui la funzione converge:

$$g(i) \quad \equiv \quad \begin{cases} 1 & \text{se } \exists n, \varphi_i(n) \downarrow \\ 0 & \text{altrimenti} \end{cases}$$

Questa funzione non é calcolabile. Basta infatti considerare la solita funzione h della sezione e osservare che anche in questo caso la composta $g \circ h$ permetterebbe di risolvere il problema della terminazione diagonale.

Una interessante variante della funzione precedente é la funzione g' cosí definita:

$$g'(i) \quad \equiv \quad \begin{cases} 1 & \text{se } \exists n, \varphi_i(n)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

In questo caso si ammette la possibilitá che la funzione diverga se non si trova nessun valore di input su cui n converge. Ancora tuttavia non é del tutto banale convincersi che la funzione g' sia calcolabile, in quanto il semplice test di un dato di input per φ potrebbe provocare la divergenza del programma. Se ad esempio il calcolo di φ_i non termina per input 0, come accorgersi che invece termina per 1?

Il punto é che, come nel caso della numerazione del piano, bisogna giostrare tra due infinitá potenziali, senza perdersi lungo nessun ramo infinito (dovetailing). In questo caso le due infinitá sono i possibili valori di input e i possibili passi di computazione. Bisogna provare valori di input via via crescenti con tempi via via crescenti (interrompendo e riprendendo opportunamente le computazioni) : non bisogna ne' accanirsi su un valore di input ammettendo che il tempo possa crescere a dismisura, ne' pretendere di testare tutti gli input per un bound temporale fissato.

Un modo elegante per definire g' ́quello di utilizzare la funzione caratteristica t del predicato di Kleene:

$$g'(x) = \text{let } \_ := \mu < n, y > .t(i, n, y) = 1 \text{ in } 1$$

Se $\varphi_i(n) = m$ allora per un qualche t deve valere $t(i, n, < t, m >)$ e la minimizzazione nel corpo di g' deve terminare; in tal caso il risultto della minimizzaizone viene ignorato e si restituisce 1. Viceversa, per ogni n $\varphi_i(n)\uparrow$, la minimizzazione diverge, come richiesto dalla specifica.

Potremmo richiedere che, invece di restituire un valore booleano che indica l'esistenza o meno di un elemento nel dominio di convergenza di φ restituisca restituisca il *minimo* elemento su cui φ converge.

$$g''(i) \quad \equiv \quad \begin{cases} \mu n.\varphi_i(n)\downarrow & \text{se } \exists n, \varphi_i(n)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

La generalizzazione sembra, a prima vista, abbastanza innocua: dopo tutto se troviamo un valore n su cui la funzione φ_i converge, il problema si riduce poi ad analizzare un numero finito di casi nell'intervallo tra 0 e n. Cinonostante la funzione g" non é calcolabile.

Consideriamo infatti la seguente funzione calcolabile $f(i, x)$:

$$f(i, y) \quad \equiv \quad \begin{cases} 0 & \text{se } y > 0 \vee \varphi_i(i)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

Per il teorema s.m.n, esiste una funzione totale e calcolabile h tale che $\varphi_{h(i)}(y) = f(i, y)$. $\varphi_{h(i)}(0) \downarrow$ se e solo se $\varphi_i(i) \downarrow$. Inoltre $\varphi_i(1) \downarrow$. Dunque

$$g''(h(i)) \quad \equiv \quad \begin{cases} 0 & \varphi_i(i)\downarrow \\ 1 & \text{altrimenti} \end{cases}$$

Se g'' fosse calcolabile risolveremmo il problema della terminazione diagonale.

Capitolo 7

Insiemi Ricorsivi e Ricorsivamente Enumerabili

Molte delle funzioni non calcolabili viste nei capitoli precedenti sono funzioni caratteristiche di opportuni insiemi. La Teoria della Calcolabilitá tratta appunto di insiemi dal punto di vista della decidibilitá algoritmica della relazione di appartenenza.

7.1 Insiemi ricorsivi

Definizione 7.1. *Un insieme si dice* ***ricorsivo*** *se la sua funzione caratteristica è calcolabile (decidibile).*

L'insieme vuoto e l'insieme N di tutti i numeri naturali sono insiemi ricorsivi (la loro funzione caratteristica ŕispettivamente la funzione costante 0 e quella costante 1).
Ogni insieme finito é calcolabile: se $A = x_1, x_2, \ldots, X_m$ la sua funzione caratteristica c_A puó essere espressa mediante un numero finito di casi:

$$\begin{aligned} c_A(n) \quad = \quad & \text{if } n = x_1 \text{ then } 1 \\ & \text{else if } n = x_2 \text{ then } 1 \\ & \ldots \\ & \text{else if } n = x_m \text{ then } 1 \\ & \text{else } 0 \end{aligned}$$

L'insieme dei numeri pari, o l'insieme dei numeri primi sono esempi di insiemi ricorsivi infiniti non banali; in particolare, ogni insieme esprimibile mediante

un predicato *primitivo ricorsivo* é ricorsivo.

Teorema 7.2. *Esistono insiemi non ricorsivi.*

Dimostrazione. Sia

$$K = \{x|\ \varphi_x(x)\downarrow\}$$

Sappiamo che la funzione caratteristica di K non é calcolabile (risolve il problema diagonale di terminazione). □

Teorema 7.3. *Gli insiemi ricorsivi sono chiusi rispetto alle operazioni di unione, intersezione e complementazione.*

Dimostrazione. Siano A e B insiemi ricorsivi, e siano c_a e c_B le loro funzioni caratteristiche. Allora:

- $c_{\overline{A}}(n) = 1 - c_A(n)$
- $c_{A\wedge B}(n) = min\{c_A(n), c_B(n)\}$
- $c_{A\vee B}(n) = max\{c_A(n), c_B(n)\}$

□

Gli insiemi ricorsivi, ordinati rispetto alla relazione di inclusione insiemistica, formano quindi una *Algebra di Boole*.

7.2 Insiemi ricorsivamente enumerabili

Definizione 7.4. *Un insieme si dice* ***ricorsivamente enumerabile (r.e.)*** *se o è vuoto oppure è il codominio di una funzione totale calcolabile (detta funzione di enumerazione).*

Teorema 7.5. *Ogni insieme ricorsivo è anche r.e. In particolare, se non é vuoto (risp. non finito) puó essere enumerato mediante un funzione calcolabile monotona crescente (risp. strettamente crescente).*

Dimostrazione. Sia A ricorsivo. Allora

1. se $A = \emptyset$ è banalmente r.e.

2. sia $A = \{n_0, \ldots, n_k\}$ (finito), e supponiamo che pr ogni i, j tali che $i < j$ risulti $n_i < n_j$; poniamo

$$f(x) \quad = \quad \begin{cases} n_x & x \leq k \\ n_k & x > k \end{cases}$$

per definizione f é calcolabile e monotona crescente.

3. sia A ìnfinito, e sia g_A è la sua funzione caratteristica (calcolabile per ipotesi). Poniamo

$$\begin{cases} f(0) = & \mu y(g_A(y) = 1) \\ f(x+1) = & \mu y(g_A(y) = 1 \ \wedge \ y > f(x)) \end{cases}$$

per definizione f é calcolabile e *strettamente crescente*.

□

Teorema 7.6. *Un insieme A è ricorsivo se e solo se sia A che $\overline{A}$ sono r.e.*

Dimostrazione. L'implicazione destra é una conseguenza del teorema precedente (e del fatto che il complementare di un insieme ricorsivo ŕicorsivo). Per l'altro verso, il caso in cui A o $\overline{A}$ siano vuoti é banale. Supponiamo quindi che A e $\overline{A}$ siano rispettivamente enumerati dalle due funzioni totali calcolabili f e g. Poniamo

$$\begin{cases} h(2x) = & f(x) \\ h(2x+1) = & g(x) \end{cases}$$

La funzione h enumera alternativamente elementi di A e del suo complementare. Si noti che h śuriettiva su N. Sia $pari(n)$ la funzione caratteristica dell'insieme dei numeri pari. Allora, la funzione caratteristica c_A di A é:

$$c_A(n) = pari(\mu y(h(y) = n)$$

La funione c_A óvviamente calcolabile; é inoltre *totale* poiché, per la suirettivitá di h il calcolo $\mu y(h(y) = n$ deve necessarimente terminare; infine, puø' essere trovato in posizione pari nlla enumerazione di h se e solo se appartiene ad $n \in A$. □

Teorema 7.7.

1. *A è ricorsivo se e solo se $A = \emptyset$ oppure A è r.e. in modo crescente*
2. *A è ricorsivo se e solo se A è finito oppure A è r.e. in modo strettamente crescente*

Dimostrazione. $\Rightarrow$ dal Teorema precedente
$\Leftarrow$. Sia f la funzione di enumerazione per A. L'idea é di implementare la funzione caratteristica $c_A(x)$ semplicemente cercando l'elemento$x$ tra quelli enumerati da f; in particolare cerco il pi$\breve{}$ piccolo y tale che $x \leq f(y)$ e poi confronto tale valore con x:

$$c_A(x) = (\mu y.(x \leq f(y)) = x)$$

Il punto delicato consiste nell'argomentare la terminazione del calcolo di $\mu y.(x \leq f(y))$, qualunque sia x. Se la funzione f é strettamente crescente, come nel caso 2., non ci sono problemi (in questo caso potremmo addirittura limitare la ricerca ai valori di y piú piccoli di x). Il caso 1. é piú problematico, in quanto la funzione f potrebbe smettere di crescere ma restare costante da un certo punto in poi. Dobbiamo quindi distingure due casi: o l'insieme A é finito, ma allora é ricorsivo in quanto ogni insieme finito ŕicorsivo, oppure é infinito, e allora la funzione precedente c_A é sicuramente totale perché la funzione f enumera elementi arbitrariamente grandi. $\square$

Corollario 7.8. *Ogni insieme ricorsivamente enumerabile A infinito contiene almeno un sottoinsieme ricorsivo infinito.*

Dimostrazione. Sia f la funzione di enumerazione per A. Poniamo

$$\begin{cases} g(0) = & f(0) \\ g(x+1) = & f(\mu y(f(y) > g(x)) \end{cases}$$

g é totale in quanto f assume assume valori arbitrariamente grandi; inoltre, per definizione, eśtrettamente crescente. Dunque $B = cod(g) \subseteq A$ ŕicorsivo.
$\square$

7.3 Caratterizzazioni alternative: semidecidibilitá

Un modo alternativo di vedere gli insiemi ricorsivamente enumerabili é dal punto di vista della relazione di appartenenza, che risulta solo semi-decidibile: siamo in grado di rispondere in modo positivo se un elemento x appartiene all'insieme, ma possiamo divergere in caso contrario.
Questo importante risultato é espresso dal seguente teorema[1].

Teorema 7.9. *Sia A un insieme di numeri naturali. Le seguenti affermazioni sono equivalenti:*

1. $A = \emptyset \vee \exists f : A = cod(f)$, f totale *calcolabile*
2. $\exists g : A = dom(g)$, g parziale *calcolabile*
3. $\exists h : A = cod(h)$, h parziale *calcolabile*

Dimostrazione.

- $1 \Rightarrow 2$ Se $A = \emptyset$ basta considerare la funzione g ovunque divergente. Se $A = cod(f)$ per f totale calcolabile, poniamo

  $$g(x) = \mu y(f(y) = x)$$

  Chiaramente g é calcolabile e $g(x)\downarrow$ se e solo se $x \in cod(f)$.

- $2 \Rightarrow 3$ Sia $A = dom(g)$; basta porre

  $$h(x) \quad = \quad x + 0 * g(x)$$

  che converge a x se e solo se $g(x)$ converge (la funzione h é l'identitá ristretta al dominio di convergenza di g).

- $3 \Rightarrow 1$ Sia $A = cod(h)$, per h parziale calcolabile. Distinguiamo due casi. se A é vuoto, non c'é nulla dimostrare. Supponiamo quindi che esista un elmento $a \in A$ e supponiamo inoltre che $h = \varphi_i$;

  $$f(\langle x, s\rangle) \quad = \quad \begin{cases} m & \text{se } t(i, x, \langle s, m\rangle) = 1 \\ a & \text{altrimenti} \end{cases}$$

[1]Ricordiamo che data una funzione parziale $f : N \to N$ indichiamo con $dom(f)$ il dominio di convergenza della funzione e con $cod(f)$ il range di f; in simboli, $dom(f) = \{x | f(x)\downarrow\}$ e $cod(f) = \{x | \exists y, x = f(y)\}$.

dove t é la funzione caratteristica del predicato T di Kleene. Intuitivamente, $f(\langle x, s\rangle)$ restituisce in output il risultato (m) della computazione di $h(x)$ se questa é terminata entro s passi, ed il risultato di *default* a altrimenti.
La funzione f é totale e calcolabile. Inoltre, $cod(f) \subseteq A$ in quanto f restituisce in otuput o $a \in A$ oppure un possibile output di $\phi_i = h$. D'altra parte, é anche vero che $A \subseteq cod(f)$, perché se $m \in A = cod(h) = cod(\phi_i)$ deve esistere un input x ed un numero di passi s tali che $t(i, x, \langle s, m\rangle) = 1$, e dunque $m = f(\langle x, s\rangle) \in cod(f)$.

□

Vale la pena di fare un aclune osservazioni sulla dimostrazione del risultato precedente, ed in particolare sull'ultimo caso.
Innanzitutto, non é semplicemente possibile prendere, per f, una qualche estensione totale di h che assuma il valore a dove h divergeva, in quanto non é detto che tale funzione sia calcolabile.
In particolare, é possibile dimostrare il seguente risultato:

Teorema 7.10. *Esistono funzioni parziali calcolabili non estendibili a funzioni totali calcolabili.*

Dimostrazione. Sia $h(x) = \varphi_x(x) + 1$. Sia $\overline{h}$ una estensione totale di h e supponiamo che sia calcolabile, cioé che $\overline{h} = \varphi_j$ per qualche j. Siccome φ_j é totale deve convergere per ogni input. In particolare $h(j) = \varphi_j(j) + 1$ é un valore definito e siccome per ipotesi $\overline{(h)}$ estende h, dovremmo avere

$$\varphi_j(j) = \overline{h}(j) = h(j) = \varphi_j(j) + 1$$

che é una contraddizione. □

La seconda osservazione é che, benché siamo sicuri che ogni insieme esprimibile come codominio (o dominio) di una funzione parziale calcolabile φ_i sia o vuoto oppure enumerato da una qualche funzione totale calcolabile f, non é tuttavia possibile determinare in modo effettivo, in funzione di i, quale dei due casi suddetti si verifica. In altre parole l'insiseme $\{i|dom(\varphi_i) = \emptyset\}$ non é ricorsivo (il lettore lo dimostri per esercizio).

Infine, il fatto che ogni insieme ricorsivamente enumerabile sia esprimibile come dominio di una qualche funzione parziale calcolabile φ_i permette di dare una numerazione "effettiva" di tutti gli insiemi ricorsivamente enumerabili, che gode di proprietá molto simili a φ_i.

Definizione 7.11. $W_i = dom(\varphi_i)$.

7.4 Proprietá di chiusura degli insiemi r.e.

Teorema 7.12. *Gli insiemi r.e. sono chiusi rispetto alle operazioni di unione e intersezione.*

Dimostrazione.. Supponiamo che $A = dom(f)$ e $B = dom(g)$ per f, g parziali calcolabili. Allora $A \wedge B = domh$ per $h(x) = f(x) * g(x)$. Supponiamo inoltre che $A = cod(f')$ e $B = cod(g')$ con f' e g' parziali calcolabili. Allora $A \vee B = cod(h')$ dove

$$\begin{cases} h'(2x) & = f'(x) \\ h'(2x+1) & = g'(x) \end{cases}$$

□

Teorema 7.13. *Gli insiemi r.e. non sono chiusi per complementazione.*

Dimostrazione. Consideriamo l'insieme $K = \{x|\varphi_x(x)\downarrow\}$. Sappiamo che K é r.e. ma non ricorsivo. Dunque, $\overline{K} = \{x|\varphi_x(x)\uparrow\}$ non puó essere r.e. □

Corollario 7.14. *Esistono insiemi che non sono né ricorsivi né ricorsivamente enumerabili.*

7.4.1 Immagine e controimmagine

Lemma 7.15. *La controimmagine di un insieme ricorsivo via una funzione totale calcolabile é un insieme ricorsivo.*

Dimostrazione. Sia A ricosivo e sia c_A la sua funzione caratteristica. Siccome $x \in f^{-1}(A) \Leftrightarrow f(x) \in A$ la funzione caratteristica di $f^{-1}(A)$ é $c_A \circ f$. □

Si osservi invece che l'immagine di un insieme ricorsivo non é necessariamente ricorsivo. Infatti ogni insieme r.e. non vuoto puó essere espresso come codominio (immagine) dell'insieme ricorsivo di tutti i numeri naturali.

Lemma 7.16. *L'immagine e la controimmagine di un insieme r.e. via una funzione calcolabile é ancora r.e.*

Dimostrazione. Se A é r.e. esistono h e g parziali calcolabili tali che $A = cod(g) = dom(h)$. Allora $f(A) = cod(g \circ f)$ e $f^{-1}(A) = dom(g \circ f)$, e pertanto sono entrambi r.e. □

7.4.2 Unioni e intersezioni infinite

É inoltre interessante interrogarsi sulle propritá di chiusura rispetto ad operazioni insiemistiche di unione e intersezione infinita.

Osserviamo innanzi tutto che ogni insieme é sempre esprimibile come unione dei suoi sottoinsiemi finiti, e quindi come unione di insiemi ricorsivi. Dunque, lo studio di unioni e intersezioni *arbitrarie* non é significativo: ció che é interessante é interrogarsi sul risultato di unioni o intersezioni di famiglie ricorsive o r.e. di insiemi.

Teorema 7.17. *Una unione ricorsiva di insiemi ricosivi non é necessariamente ricorsiva.*

Dimostrazione. Consideriamo l'insieme K e sia k una funzione parziale calcolabile tale $K = dom(k)$. Considerimo la seguente classe di troncamenti progressivi di k:

$$\varphi_{h(n)}(x) = \text{if } x \leq n \text{ then } k(x) \text{ else } \uparrow$$

Possiamo inoltre definire la funzione h in modo tale da essere non solo calcolabile e totale, ma anche monotona crescente[2]. Dunque il codominio di h é un insieme ricorsivo, e siccome ovviamente

$$\bigcup_{i \in cod(h)} W_i = \bigcup_{\imath \in N} W_{h(n)} = K$$

la tesi é dimostrata. □

Teorema 7.18. *Una unione r.e. di insiemi r.e. é ancora r.e.*

Dimostrazione. Consideriamo la seguente unione

$$A = \bigcup_{i \in W_a} W_i$$

Vogliamo dimostrare che A é r.e. Abbiamo

[2]Questo é vero in generale per la funzione s_m^n del teorema s.m.n: basta pompare opportunamente gli indici dei pogrammi con istruzioni inutili.

$$\begin{aligned} z \in A &\Leftrightarrow \exists i \in W_a, z \in W_i \\ &\Leftrightarrow \exists i \in dom(\varphi_a), z \in dom(\varphi_i) \\ &\Leftrightarrow \exists i \in dom(\varphi_a), z \in dom(\varphi_i) \\ &\Leftrightarrow \exists i, \varphi_a(i)\downarrow \wedge \varphi_i(z)\downarrow \\ &\Leftrightarrow \exists i, s, s' t(a, i, s) = 1 \wedge t(i, z, s') = 1 \end{aligned}$$

dove t é la funzione caratteristica del predicato T di Kleene. Dunque A é esprimibile come dominio della seguente funzione parziale calcolabile:

$$f(z) = \mu\langle i, s, s'\rangle.(t(a, i, s) = 1 \wedge t(i, z, s') = 1)$$

□

Teorema 7.19. *Una intersezione ricorsiva di insiemi ricorsivi non é in generale neppure ricorsivamente enumerabile.*

Dimostrazione. Vogliamo esprimere $\overline{K}$ come intersezione di una famiglia ricorsiva di insiemi ricorsivi. Costruiamo la seguente famiglia di funzioni:

$$\varphi_{h(i)}(x) = \text{if } \forall m \leq i.t(x, x, \langle m, i\rangle) = 0 \text{ then 1 else } \uparrow$$

La funzione h é totale e calcolabile e come osservato in precedenza si puó assumere che sia anche strettamente monotona. Vogliamo ora dimostrare che

$$\bigcap_{i \in N} W_{h(i)} = \overline{K}$$

Osserviamo innanzitutto che per ogni i, $\overline{K} \subseteq W_{h(i)}$. Infatti, $x \in \overline{K}$ solo se per ogni m e ogni j, $t(x, x, \langle m, j\rangle) = 0$. Questo implica in particolare che per nessun m minore di i $t(x, x, \langle m, i\rangle) = 0$ e quindi che $x \in W_{h(i)}$ qualunque sia i.
D'altra parte se $x \in K$ allora esiste m ed esiste $i \geq m$ tale che $t(x, x, \langle m, i\rangle) = 1$. Questo implica $x \notin W_{h(i)}$.

□

Capitolo 8

I Teoremi di Rice e Rice-Shapiro

Gran parte della ricerca nel campo dei linguaggi di programmazione é focalizzata sullo studio delle proprietá delle funzioni calcolate dai programmi (se e dove convergono, quale e' il range dei possibili valori di output, se sono equivalenti a funzioni date, etc.). Proprietá di questo tipo, che riguardano unicamente ció che il programma calcola e non il modo in cui avviene il calcolo (come ad esempio la sua complessitá) sono dette *estensionali*.

Definizione 8.1. *Sia $A \subseteq \mathcal{N}$. A é detto estensionale (rispetto a φ) se per ogni i, j*

$$i \in A \wedge \varphi_i \approx \varphi_j \rightarrow j \in A$$

Quindi un insieme di indici estensionale é l' insieme di *tutti* gli indici dei programmi che calcolano una determinata classe di funzioni.

Esempio 8.2. *I seguenti insiemi sono estensionali:*

1. $\{\ i\ |\varphi_i$ *è totale*$\}$
2. $\{\ i\ |5 \in cod(\varphi_i)\}$
3. $\{\ i\ |dom(\varphi_i)$ *é finito*$\}$
4. $\{\ i\ |\varphi_i(0)\uparrow\}$
5. $\{\ i\ |\exists n, \varphi_i(n)\downarrow \wedge \varphi_i(n+1)\downarrow\}$

I seguenti insiemi non sono estensionali:

1. $\{\ i\ |\varphi_i(i)\downarrow\}$
2. $\{\ i\ |\varphi_i(0) < i\}$
3. $\{\ i\ |t(i,0,8) = 1$, *dove* t *é la f. caratteristica del predicato* T *di Kleene*$\}$
4. $\{\ i\ |\varphi_i \approx \varphi_{i+1}\}$
5. $\{\ i\ |i$ *é pari*$\}$

Si noti che il complementare di un insieme estensionale é anch'esso estensionale. Inoltre un insieme estensionale non vuoto non puó essere finito, in quanto se $i \in A$ allora A contiene anche tutti gli indici delle funzioni equivalenti a i, che sono infiniti.

Teorema 8.3. *(di Rice) Sia* $A \subseteq \mathcal{N}$ *estensionale. Allora,* A *é ricorsivo se e solo se é banale, cioé* $A = \emptyset$ *oppure* $A = \mathcal{N}$*.*

Dimostrazione. Supponiamo che A non sia banale, cioé che né A né $\overline{A}$ siano vuoti. Consideriamo un indice m per la funzione ovunque divergente; necessariamente, $m \in A$ oppure $m \in \overline{A}$. Consideriamo il secondo caso (il primo é del tutto analogo e viene lasciato al lettore per esercizio).
Preso $a \in A$, costruiamo la seguente funzione:

$$f(x,y) \quad = \quad \begin{cases} \varphi_a(y) & se\ \varphi_x(x)\ \downarrow \\ \uparrow & se\ \varphi_x(x)\ \uparrow \end{cases}$$

Per il teorema s.m.n. esiste h totale e calcolabile, tale che $\varphi_{h(x)}(y) = f(x,y)$. Abbiamo quindi

$$\varphi_{h(x)} \quad \approx \quad \begin{cases} \varphi_a & se\ \varphi_x(x)\downarrow \\ \varphi_m & se\ \varphi_x(x)\uparrow \end{cases}$$

Quindi, se $\varphi_x(x)\downarrow$, siccome $a \in A$ e A é estensionale, abbiamo $h(x) \in A$. Viceversa, se $\varphi_x(x)\uparrow$, siccome $m \in \overline{A}$ e $\overline{A}$ é estensionale, abbiamo $h(x) \in \overline{A}$. In conlusione

$$h(x) \in \overline{A} \Leftrightarrow \varphi_x(x)\downarrow$$

e se A fosse ricorsivo avremmo risolto il problema diagonale della terminazione (basta calcolare $1 - c_A(h(x))$). $\square$

Il teorema di Rice ha due applicazioni tipiche: la prima é quella diretta, per dimostrare che determinati insiemi essendo estensionali non possono essere ricorsivi. La seconda é quella che permette di concludere che determinati insiemi estensionali, essendo complementari di insiemi r.e., non sono neppure r.e. (altrimenti, avendo un complementare r.e. dovrebbero essere ricorsivi).

Ad esempio, consideriamo il seguente insieme estensionale

$$A = \{\ i\ |\varphi_i(0)\downarrow\}$$

Ovviamente A non é banale, e dunque, per Ric, non puó essere ricorsivo (uso diretto); d'altra parte A é r.e. (il lettore lo dimostri per esercizio), dunque

$$\overline{A} = \{\ i\ |\varphi_i(0)\uparrow\}$$

non é neppure r.e. altrimenti sia A che $\overline{A}$ sarebbero ricorsivi, contraddicendo il risultato di Rice (uso indiretto).

Tuttavia, il caso non é sempre cos'i favorevole. Consideriamo ad esempio

$$A = \{\ i\ |dom(\varphi_i) \text{ é finito}\}$$

ed il suo complementare

$$A = \{\ i\ |\varphi_i(0) \text{ é infinito}\}$$

Per Rice possiamo concludere che non sono ricorsivi, ma siccome non risulta evidente che né A né $\overline{A}$ sia r.e. (ed in effetti, come vedremo, nessuno dei due lo é) non riusciamo ad applicare indirettamente Rice per concludere qualche cosa di pi$\check{}$.

Risulta pertanto interessante investigare separatamente le proprietd́egli insiemi estensionali r.e., oggetto della prossima sezione.

8.1 Il teorema di Rice-Shapiro

Definizione 8.4. *Un insiemi di numeri naturali A si dice* monotono *(rispetto a φ) se per ogni i e j*

$$i \in A \wedge \varphi_i \subseteq \varphi_j \rightarrow j \in A$$

Definizione 8.5. *Un insiemi estensionale di numeri naturali A si dice* compatto *(rispetto a φ) se per ogni $i \in A$ esiste $j \in A$ tale che*

1. *il grafo di φ_j é finito*

2. $\varphi_j \subseteq \varphi_i$

Si noti che ogni insieme monotono é necessariamente estensionale.

Lemma 8.6. *Il complementare di un insieme A monotono é monotono se e solo se A é banale.*

Dimostrazione. Basta osservare che se un insieme monotono A contiene l'indice m della funzione ovunque divergente, allora $A = \mathcal{N}$. Ovviamente, $m \in A$ o $m \in \overline{A}$. □

Lemma 8.7. *Ogni insieme A estensionale che contiene un indice della funzione ovunque divergente é compatto.*

Dimostrazione. Ovvio, dato che la funzione vuota ha un grafo finito ed é contenuta in ogni altra funzione. □

Consideriamo i seguenti insiemi:

1. A = $\{ i \mid \varphi_i$ è totale$\}$
2. B = $\{ i \mid \varphi_i(0) \downarrow \}$
3. C = $\{ i \mid dom(\varphi_i)$ e $\overline{dom(\varphi_i)}$ sono infiniti$\}$

Il primo insieme é banalmente monotono: se $\varphi_i \subseteq \varphi_j$ e φ_i é totale, allora $\varphi_j \approx \varphi_i$ e dunque é anch'essa totale. Non é tuttavia compatto, in quanto A non contiene nessun indice di funzione con grafo finito e dunque la proprietá 1) della definizione di compattezza é necessariamente violata. Il complementare di A non é monotono, ma é compatto, in quanto contiene la funzione ovunque divergente.
L'insieme B é chiaramente monotono: se $\varphi_i(0) \downarrow$ e $\varphi_i \subseteq \varphi_j$ allora necessariamente $\varphi_j(0) \downarrow$. É anche compatto, infatti data una funzione $\varphi_i(0) \downarrow$, e supposto $\varphi_i(0) = a$, consideriamo la funzione

$$\varphi_j(x) := \text{if } x = 0 \text{ then } a \text{ else } \uparrow$$

Chiaramente $j \in A$, il grafo di φ_j é finito e $\varphi_j \subseteq \varphi_i$. Il complementare di B non é monotono, ma é ancora compatto (in quanto contiene la funzione

vuota). Questo mostra tra l'altro che, in questo ambito, il complementare di un insieme compatto puó essere compatto.
L'insieme C non é monotono: se una funzione diverge su un numero infinito di punti non é detto che ogni sua estensione continui a farlo. Non é neppure compatto, in quanto per definizione C non puó contenere indici di funzioni finite. Il complementare di C é l'insieme degli indici delle funzioni che sono o finite o quasi-totali (che cioé divergono su un numero al piú finito di punti). Tale insieme non é monotono, poiché ogni funzione finita ammette estensioni non finite ma neppure quasi-totali; é banalmente compatto in quanto contiene la funzione ovunque divergente.

Introduciamo ora una nozione utile alla dimostrazione del prossimo teorema.

Definizione 8.8. *La composizione parallela $f|g$ di due funzioni f e g é quella funzione calcolabile che, su input x, restituisce in output $f(x)$ o $g(x)$ a seconda di quale delle due funzioni termini prima la propria computazione.*

Si noti che la composizione parallela puó essere definita in modo formale a partire dal predicato T di kleene.

Teorema 8.9. *Rice-Shapiro (monotonia) Ogni insieme estensionale A ricorsivamente enumerabile é monotono.*

Dimostrazione. Supponiamo che esistano due indici i e j tali che $i \in A$, $j \notin A$ and $\varphi_i \leq \varphi_j$.
Sia φ_k la funzione di semidecisione per K, i.e. $dom(\phi_k) = \mathcal{K}$. Consideriamo la funzione

$$\varphi_{f(x)}(y) = \varphi_i(y)|(\varphi_k(x); \varphi_j(y))$$

dove il punto e virgola denota la composizione sequenziale. Se $x \notin K$ allora $\varphi_k(x) \uparrow$ e dunque la funzione $\varphi_{f(x)}$ si comporta esattamente come φ_i. In caso contrario, non sappiamo chi tra le due funzioni parallele terminerá prima la propria computazione, ma siccome $\varphi_i \leq \varphi_j$ anche se $\varphi_i(y)$ risultasse piu' veloce, l'output coinciderebbe comunque con quello di $\varphi_j(y)$ e dunque, $\varphi_{f(x)}(y) \approx \varphi_j$.
In conclusione

$$f(x) \in A \Leftrightarrow x \in \overline{K}$$

e $\overline{K}$ sarebbe r.e., il che é assurdo. $\square$

Come corollario del risultato precedente, otteniamo anch una dimostrazione alternativa del teorema di Rice.
Supponiamo infatti che A sia un insieme ricorsivo estensionale. Allora sia A che $\overline{A}$ sono r.e. e dunque entrambi monotoni. Ma per il lemma 8.6, A é allora banale.

Teorema 8.10. *Rice-Shapiro (compattezza) Ogni insieme estensionale A ricorsivamente enumerabile é compatto.*

Dimostrazione. Sia A un insieme estensionale ricorsivamente enumerabile. Supponiamo che $i \in A$ e che per ogni j tale che $\varphi_j \subseteq \varphi_i$ e φ_j é finito si abbia $j \notin A$. Consideriamo la funzione f totale calcolabile definita come segue (per s-m-n)

$$\varphi_{f(x)}(y) = \begin{cases} \uparrow & \text{if } \varphi_x(x) \downarrow \text{ in meno di } y \text{ passi} \\ \varphi_i(x) & \text{otherwise} \end{cases}$$

Se $x \in \overline{K}$ allora $\varphi_x(x)\uparrow$ e chiaramente $\varphi_{f(x)} \approx \varphi_i$. Se invece $x \in K$ allora la computazione di $\varphi_x(x)$ terminerá in un numero finito t di passi, e la funzione $\varphi_{f(x)}$ convergerá solo per valori di input $y \leq t$. Dunque $f(x)$ é l'indice di una sottofunzione finita di φ_i, e per ipotesi $f(x) \notin A$.
In conclusione

$$f(x) \in A \Leftrightarrow x \in \overline{K}$$

e $\overline{K}$ sarebbe r.e., il che é assurdo. □

8.2 Il Teorema di Myhill-Shepherdson

Sia $\mathcal{PR} = \{\varphi_i \mid i \in \mathcal{N}\}$ l'insieme delle funzioni parziali ricosive.

Definizione 8.11. *Una funzione $F : \mathcal{PR} \to \mathcal{PR}$ é effettiva se esiste una funzione totale calcolabile $f : \mathcal{N} \to \mathcal{N}$ tale per ogni i, $F(\varphi_i) = \varphi_{f(i)}$, ovvero il seguente diagramma commuta:*

$$\begin{array}{ccc} \omega & \xrightarrow[\text{totale calcolabile}]{f} & \omega \\ {\scriptstyle\varphi}\downarrow & & \downarrow{\scriptstyle\varphi} \\ \mathcal{PR} & \xrightarrow[F]{} & \mathcal{PR} \end{array}$$

Diremo in tale caso che f realizza F.

Se f realizza F, allora f é *estensionale*, nel senso che trasforma indici di funzioni equivalenti in indici equivalenti. Infatti

$$\varphi_i = \varphi_j \Rightarrow \varphi_{f(i)} = F(\varphi_i) = F(\varphi_j) = \varphi_{f(j)}$$

D'altra parte, ogni funzione $f : \mathcal{N} \to \mathcal{N}$ totale, calcolabile ed estensionale, induce una trasfomazione effettiva $F : \mathcal{PR} \to \mathcal{PR}$. Basta porre $F(\varphi_i) = \varphi_{f(i)}$ e osservare che questa é una buon definizione in uanto indipendente dall'indice i.

L'insieme $\mathcal{PR}$ é un ordinamento parziale rispetto all'operazione di inclusione di funzioni. L'ordinamento é anche completo rispetto ai diretti ricorsivamente enumerabili (in quanto una unione r.e. di insiemi r.e. é ancora r.e.). Il teorema di Myhill-Shepherdson afferma alcune importanti proprietá topologiche su F, nel caso in cui F sia una funzione effettiva: in particolare, F risulta *monotona* e *continua*.

Teorema 8.12. *Myhill-Shepherdson (monotonoia) Ogni funzione effettiva $F : \mathcal{PR} \to \mathcal{PR}$ é monotona, ovvero per ogni i e j*

$$\varphi_i \subseteq \varphi_j \Rightarrow F(\varphi_i) \subseteq F(\varphi_j)$$

Dimostrazione. Supponiamo che F sia realizzata da f, ovvero che per ogni i $F(\varphi_i) = \varphi_{f(i)}$. Supponimo inoltre che $\varphi_i \subseteq \varphi_j$, ma per assurdo $\varphi_{f(i)} \not\subseteq \varphi_{f(j)}$ Dunque esistono a e b tali che $\varphi_{f(i)}(a) = b$ ma $\varphi_{f(j)}(a) \neq b$.
Sia

$$C = \{x \mid \varphi_x(a) = b\}$$

C è un insieme estensionale e r.e. di indici; inoltre, per definizione $f(i) \in C$ e $f(j) \notin C$.
Siccome f é una funzione estensionale, anche $f^{-1}(C)$ é estensionale, infatti se $\varphi_x = \varphi_y$ allora

$$x \in f^{-1}(C) \Leftrightarrow f(x) \in C \Leftrightarrow f(y) \in C \Leftrightarrow y \in f^{-1}(C)$$

Inoltre $f^{-1}(C)$ é totale e calcolabile in quanto controimmagine di un insieme r.e. via una funzione totale calcolabile. Dunque, per il teorema di Rice-Shapiro, dovrebbe essere monotono.
Tuttavia, $i \in f^{-1}(C)$ ma $j \notin f^{-1}(C)$. $\square$

L'insieme delle funzioni parziali ricorsive $\mathcal{PR}$ non é un CPO, in quanto non tutti i diretti ammettono estremo superiore. Infatti una qualunque funzione (anche non calcoloabile) é unione di tutte le sue sottofunzioni finite (tutte ovviamente calcolabili).
Con un piccolo abuso di linguaggio, diremo che un diretto $\mathcal{D} \subset \mathcal{PR}$ é r.e. se esiste una collezione di indici A *non necessariamente estensionale*, tale che $\mathcal{D} = \{\varphi_i \mid i \in A\}$.

Lemma 8.13. *Data una funzione parziale calcolabile φ_i, l'insieme delle sue sottofunzioni finite $\hat{\varphi}_i$ é r.e.*

Dimostrazione. Osserviamo innanzitutto che non possiamo semplicemente considerare l'insieme

$$A = \{x \mid \varphi_x \text{ finito}, \varphi_x \subseteq \varphi_i\}$$

in quanto tale insieme, essendo estensionale ma non monotono, non puó essere r.e.
Fissiamo dunque una numerazione D_i di tutti gli insiemi finiti. Consideriamo la funzione

$$g(i,d,s,x) = \begin{cases} m & \text{se } x \in D_d \wedge t(i,x,\langle m,s\rangle) = 1 \\ \uparrow & \text{altrimenti} \end{cases}$$

dove t é la funzione caratterisitca del predicato T di Kleene. La funzione g é calcolabile, dunque esiste h totale e calcolabile tale che $\varphi_{h(i,d,s)}(x) = g(i,d,s,x)$. Mostriamo che $A = cod(h)$ é una numerazione di tutte le sottofunzioni finite di φ_i.
Sia infatti $varphi_j \subseteq \varphi_i$ e suppondiamo che $dom(\varphi_j) = D_d$. Preso s sufficientemente grande, la computazione di $\varphi_i(x)$ terminerá entro s passi per ogni $x \in D_d$, e dunue $t(i,x,\varphi_i(x),s) = 1$. Questo mostra che h enumera tutte le sottofunzioni finite di φ_i. Siccome é evidente che h enumera solo sottofunzioni di φ_i l'asserto é dimostrato. □

Il seguente teorema asserisce che possiamo sempre calcolare il valore di una funzione effettiva F su di un input φ_i come limite dei valori di F sulle approssimazioni finite di φ_i.

Teorema 8.14. *Per ogni funzione effettiva $F : \mathcal{PR} \to \mathcal{PR}$*

$$F(\varphi_i) = \bigcup F(\hat{\varphi}_i))$$

Dimostrazione. Supponiamo che F sia realizzata da f, ovvero che per ogni i $F(\varphi_i) = \varphi_{f(i)}$. Per monotonia $\bigcup F(\hat{\varphi}_i)) \subseteq F(\varphi_i)$. Supponiamo per assurdo che $F(\varphi_i) \not\subseteq \bigcup F(\hat{\varphi}_i))$, Dunque esistono a e b tali che $\varphi_{f(i)}(a) = b$, ma per ogni indice j di una sottofunzione finita di φ_i, $\varphi_{f(j)}(a) \neq b$.
Sia

$$C = \{x \mid \varphi_x(a) = b\}$$

C è un insieme estensionale e r.e. di indici; inoltre, per definizione $f(i) \in C$ e $f(j) \notin C$.
Siccome f é una funzione estensionale, anche $f^{-1}(C)$ é estensionale, infatti se $\varphi_x = \varphi_y$ allora

$$x \in f^{-1}(C) \Leftrightarrow f(x) \in C \Leftrightarrow f(y) \in C \Leftrightarrow y \in f^{-1}(C)$$

Inoltre $f^{-1}(C)$ é totale e calcolabile in quanto controimmagine di un insieme r.e. via una funzione totale calcolabile. Dunque, per il teorema di Rice-Shapiro, dovrebbe essere compatto.
Tuttavia, $i \in f^{-1}(C)$ ma $j \notin f^{-1}(C)$ per nessun j tale che $\varphi_j \subseteq \varphi_i$ e φ_j é finito. □

Il teorema precedente puó essere generalizzato ad arbitrari diretti ricorsivamente enumerabili.

Lemma 8.15. *Sia F una funzione monotona. Se $F(\varphi_i) = \bigcup F(\hat{\varphi}_i)$ allora per ogni diretto (D) tale che $\varphi_i = \bigcup \mathcal{D}$ si ha $F(\varphi_i) = \bigcup F((D))$.*

Dimostrazione. Per monotonia di F é sempre vero che $\bigcup F((D)) \subseteq F(\varphi_i)$. Ci basta dunque dimostrare l'inclusione inversa. Supponiamo che $\langle a, b\rangle \in F(\varphi_i) = \bigcup F(\hat{\varphi}_i)$. Allora deve esistere una qualche sottofunzione finita φ_j di φ_i tale che $\langle a, b\rangle \in F(\varphi_j)$. Sia $\{\langle x_1, y_1\rangle, dots\langle x_n, y_n\rangle\}$ il grafo finito di ‘φ_j. Sicome $varphi_j \subseteq \varphi_i = \bigcup \mathcal{D}$, per ogni $langlex_k, y_k\rangle$ nel grafo di φ_j deve esistere un elemento $d_k \in \mathcal{D}$ tal che $\langle x_k, y_k\rangle \leq d_k$. Siccome inoltre $\mathcal{D}$ é un diretto deve esistere un elemento $d \in \mathcal{D}$ tale che per ogni k $d_k \leq d$. Questo implica che $\varphi_j \subseteq d$ e per monotonia $F(\varphi_j) \subseteq F(d)$. Pertanto $\langle a, b\rangle \in F(d) \subseteq \bigcup F(\mathcal{D})$. □

Possiamo quindi formulare il teorema di Myhill-Shepherdson nella sua forma piú generale:

Teorema 8.16. *Myhill-Shepherdson (continuitá)*
Ogni funzione effettiva $F : \mathcal{PR} \to \mathcal{PR}$ é continua, ovvero per ogni diretto $\mathcal{D} \subset \mathcal{PR}$ ricorsivamente enumerabile

$$F(\bigcup \mathcal{D}) = \bigcup F(\mathcal{D}))$$

Dimostrazione. Sia $\varphi_i = \bigcup \mathcal{D}$. Si ha:

$$\begin{array}{rcll} F(\bigcup \mathcal{D}) & = & F(\varphi_i) & \text{per def. di } \varphi_i \\ & = & \bigcup F(\hat{\varphi}_i) & \text{per il teorema 8.14} \\ & = & \bigcup F(\mathcal{D})) & \text{per il lemma 8.15} \end{array}$$

□

Un importante corollario del teorema precedente é l'esistenza di punti fissi per funzioni totali ricorsive estensionali:

Teorema 8.17. *Ogni funzione totale ricorsiva estensionale f ammette un punto fisso, nel senso che esiste un indice m tale che*

$$\varphi_{f(m)} \approx \varphi_m$$

Dimostrazione. Sia $F : \mathcal{PR} \to \mathcal{PR}$ la funzione effettiva indotta da f, ovvero definita dall'equazione $F\varphi_i = \varphi_{f(i)}$. Per il teorema di Myhill-Shepherdson, F é continua, e per il teorema del punto fisso ammette un punto fisso, ovvero esiste una funzione parziale ricorsiva φ_m tale che

$$\varphi_m = F(\varphi_m) = \varphi_{f(m)}$$

□

Si noti che il punto fisso puó essere costruito in modo effettivo come limite della catena $F^i\bot$. In particolare, sia d un indice per la funzione ovunque divergente (che é l'elemento minimo $\bot$ in $\mathcal{PR}$); allora $F^i(\bot) = \varphi_{f^i(d)}$, e

$$m = \bigcup_{i \in \mathcal{N}} F^i(\bot) = \bigcup_{i \in \mathcal{N}} \varphi_{f^i(d)}$$

Dunque il grafo di m é una unione r.e. di insiemi r.e. e puó essere enumerato in modo effettivo.

Capitolo 9

I teoremi di ricorsione

Teorema 9.1. *Per ogni funzione totale calcolabile f esiste m tale che*

$$\varphi_{f(m)} \approx \varphi_m$$

Dimostrazione. Per il teorema $s.m.n$ esiste h totale e calcolabile tale che

$$\varphi_{h(x)}(y) = g(x, y) = \varphi_{f(\varphi_x(x))}(y)$$

Sia p un indice per h e poniamo $m = \varphi_p(p) = h(p)$ (che é sicuramente definito in quanto h é totale). Allora, per ogni y

$$\varphi_m(y) = \varphi_{h(p)}(y) = g(p, y) = \varphi_{f(\varphi_p(p))}(y) = \varphi_{f(m)}(y)$$

□

In effetti, il punto fisso m per una funzione totale e calcolabile puó essere calcolato in modo effettivo in funzione di un indice per f. Un modo generale per enunciare questo risultato é fornito dal cosiddetto secondo teorema di ricosione di Kleene:

Teorema 9.2. *Per ogni funzione binaria totale calcolabile f esiste una funzione calcolabile s tale che, per ogni y*

$$\varphi_{f((s(y),y)} \approx \varphi_{s(y)}$$

Dimostrazione. Per il teorema $s.m.n$ esiste h totale e calcolabile tale che

$$\varphi_{h(x,y)}(z) = g(x, y, z) = \varphi_{f(\varphi_x(x),y)}(z)$$

Applicando nuovamente il teorema s-m-n alla funzione h, otteniamo una funzione totale e calcolable r tale che $\varphi_{r(y)}(x) = h(x, y)$. Posto $s(y) = \varphi_{r(y)}(r(y))$ abbiamo, per ogni z

$$\varphi_{s(y)}(z) = \varphi_{\varphi r(y)(r(y)}(z) = \varphi_{h(r(y),y)} = \varphi_{f(\varphi_{r(y)}(r(y)),y)}(z) = \varphi_{f(s(y)),y)}(z)$$

□

Come corollario del teorema precedente, otteniamo:

Teorema 9.3. *Esiste una funzione totale e calcolabile s tale che per ogni i*

$$\varphi_{\varphi_i(s(i))} \approx \varphi_{s(i)}$$

Dimostrazione. Si consideri la funzione universale $u(x, i) = \varphi_i(x)$. Per il secondo teorema di ricorsione esiste s totale e calcolabile tale che

$$\varphi_{\varphi_i(s(i))} = \varphi_{u((s(i),i)} \approx \varphi_{s(i)}$$

□

9.1 Applicazioni

Il teorema del punto fisso di Kleene puó essere utilizzato per concludere alcune interessanti proprietá della funzione di enumerazione φ (cosa abbastanza sorprendente in quanto si sono fatte pochissime assunzioni su tale enumerazione). Ad esempio, una banale consegenza del teorema del punto fisso é che, nella nostra enumerazione di programmi, esistono necessariamente due programmi consecutivi che calcolano la stessa funzione, in quanto, considerata la funzione successore, deve esistere m tale che

$$\varphi_m \approx \varphi_{m+1}$$

Ovvero, se la funzione di enumerazione é effettiva, la distribuzione estensionale delle funzioni calcolate é talmente casuale da non poter impedire che due funzioni equivalenti appaiano in posizione consecutiva!

Un altro risultato interessante, che avremo modo di utilizzare in seguito, é il seguente:

Lemma 9.4. *Esiste necessariamente un indice i tale che $\varphi_i(0) = i + 1$.*

Dimostrazione. Per il teorema *s.m.n.* esiste h totale e calcolabile tale che

$$\varphi_h(x)(y) = g(x, y) = x + 1$$

Per il teorema di Kleene esiste m tale che $\varphi_m \approx \varphi_{h(m)}$ e dunque

$$\varphi_m(0) = \varphi_h(m)(0) = g(m, 0) = m + 1$$

□

In altri termini, esiste sicuramente un numero $i + 1$ che puó essere definito in modo piú compatto come output di un programma di indice $j < i$ (in particolare, di indice i). Si noti che al posto della funzione successore posso prendere qualunque funzione totale calcolabile, mostrando che il guadagno in spazio nella descrizione del numero puó essere arbitrariamente grande.

Come altro esempio di uso del teorema del punto fisso, diamo una ulteriore dimostrazione del teorema di Rice.
Supponiamo per assurdo che A sia ricorsivo, ma non banale. Esistono dunque i e j tali che $i \in A$ e $j \in \overline{A}$.
Considero la seguente funzione:

$$g(x, y) \quad = \quad \begin{cases} \text{ı} & se\ x \in \overline{A} \\ \text{ȷ} & se\ x \in A \end{cases}$$

Essendo A ricorsivo, il test è effettivo, e la funzione $g(x, y)$ è quindi calcolabile. Per il teorema s-m-n esiste h totale e calcolabile tale che $\varphi_{h(x)}(y) = g(x, y)$; inoltre, per costruzione,

$$h(x) \in A \Leftrightarrow x \in \overline{A}$$

Per il teorema del punto fisso di Kleene, esiste un indice b tale che $\varphi_b = \varphi_{h(b)}$. Avremmo quindi

$$b \in A \Leftrightarrow h(b) \in A \Leftrightarrow b \in \overline{A}$$

che é una contraddizione.

Capitolo 10

Riducibilitá

Definizione 10.1. *Siano $A, B \subseteq \mathcal{N}$; A si dice* m-riducibile *B se esiste una funzione totale e calcolabile f tale che*

$$x \in A \Leftrightarrow f(x) \in B$$

In questo caso scriveremo $A \leq_m B$.
Due insiemi si dicono m-equivalenti *se $A \leq_m B$ e $B \leq_m A$; in questo caso scriveremo $A =_m B$.*

La relazione $\leq_m$ é un preordine (i.e. é riflessiva e transitiva). La relazione $= m$ é una relazione di equivalenza.
Si noti inoltre che per definizione $A \leq_m B$ se e solo se $\overline{A} \leq_m \overline{B}$.

Lemma 10.2. *Siano $A, B \subseteq \mathcal{N}$ tali che $A \leq_m B$. Se B é ricorsivo allora A é ricorsivo; se B é r.e. allora A é r.e.*

Dimostrazione. La facile dimostrazione é lasciata al lettore. □

Come esempio delle definizioni precedenti dimostriamo il seguente risultato:

Lemma 10.3. *Sia $K_0 = \{\langle i, n\rangle \mid n \in W_i\}$. $K_0 =_m K$.*

Dimostrazione.
Siccome

$$i \in K \Rightarrow i \in W_i \Rightarrow \langle i, i\rangle \in K_0$$

la funzione $f(x) = \langle x, x\rangle$ permette di ridurre K a K_0.
D'altra parte, consideriamo la funzione totale calcolabile h per cui

$$\varphi_{h(i,x)}(y) = g(i, x, y) = \varphi_i(x)$$

Abbiamo

$$\langle i,n\rangle \in K_0 \;\Leftrightarrow\; n \in W_i \;\Leftrightarrow\; \forall y, \varphi_{h(i,n)}(y)\downarrow \Leftrightarrow\; \varphi_{h(i,n)}h(i,n)\downarrow \Leftrightarrow\; h(i,n) \in K$$

Quindi la funzione h riduce K_0 a K. $\square$

Definizione 10.4. *Un insieme si dice m-completo se é r.e. ed ogni insieme r.e. é riducibile ad esso.*

Lemma 10.5. *K_0 e K sono insiemi completi.*

Dimostrazione. Dato che $K_0 =_m K$ é sufficiente dimostrare la proprietá per K_0. Abbiamo giá dimostrato che se $A \leq_m K_0$ allora A é r.e.
Supponiamo quindi che A sia r.e. Allora esiste i tale che $A = W_i$ e per ogni n

$$n \in A \Leftrightarrow n \in W_i \Leftrightarrow \langle i,n\rangle \in K_0$$

Pertanto la funzione $f(x) = \langle i,x\rangle$ riduce A a K_0. $\square$

Lemma 10.6. *A é completo se e solo se $A =_m K$.*

Dimostrazione. Se $A =_m K$ allora A é r.e. e m-completo perché lo é K.
Viceversa se A é m-completo, allora é r.e. e per la completezza di K, $A \leq_m K$; inoltre, siccome K é r.e. e $K \leq_m A$ per la m-completezza di A. $\square$

Definizione 10.7. *Sia $A \subseteq \mathcal{N}$.*

1. *A si dice* produttivo *se esiste una funzione totale e calcolabile f tale che per ogni i*
$$W_i \subseteq A \Rightarrow f(i) \in A\backslash W_i$$
2. *A si dice* creativo *se é r.e. ed il suo complemento $\overline{A}$ é produttivo.*

Si osservi che un insieme produttivo non puó essere r.e. Infatti, se $A = W_i$ allora preso $W_i \subseteq A$ avremmo che $A\backslash W_i = \emptyset$ e quindi $f(i) \notin A\backslash W_i$.

Teorema 10.8. *K é creativo.*

Dimostrazione. Sappiamo che K é r.e. e dunque dobbiamo solo dimostrare che $\overline{K}$ é produttivo.
La funzione di creazione f per K é la funzione identitá. Vogliamo dimostrare che

$$W_i \subseteq \overline{K} \Rightarrow i \in \overline{K} \backslash W_i$$

Dimostriamo innanzitutto che $i \in \overline{K}$. Infatti, se $i \in K$ allora per definizione di K avremmo $i \in W_i$ e siccome $W_i \subseteq \overline{K}$ avremmo anche $i \in \overline{K}$ che é una contraddizione. Dunque $i \notin K$ che implica $i \in \overline{K}$.
Dimostriamo ora che $i \notin W_i$. Infatti, se $i \in W_i$ allora i dovrebbe appartenere a K, ma abbiamo appena dimostrato il contrario. □

Teorema 10.9. *Sia $A \subseteq \mathcal{N}$. A é produttivo se e solo se $\overline{K} \leq_m A$.*

Dimostrazione.

(⇐) Supponiamo che $\overline{K} \leq_m A$, e sia g la corrispondente funzione di riduzione. Per il teorema s-m-n esiste h totale e calcolabile tale che $\varphi_{h(i)}(x) = \varphi_i(g(x))$. Dunque

$$W_{h(i)} = g^{-1}(W_i)$$

Posto $f(x) = g(h(i))$ vogliamo dimostrare che f é una funzione di produzione per A, ovvero che per ogni i

$$W_i \subseteq A \Rightarrow f(i) \in A \backslash W_i$$

Supponiamo che $f(x) = g(h(i)) \in \overline{A}$; allora per riducibilitá $h(i) \in K$ e quindi $h(i) \in W_{h(i)} = g^{-1}(W_i)$; questo implica che $f(i) = g(h(i)) \in W_i \subseteq A$, che é assurdo.
Se inoltre $f(x) = g(h(i)) \in W_i$; allora $h(i) \in g^{-1}(W_i) = W_{h(i)}$ e quindi $h(i) \in K$ e per riducibilitá $f(i) = g(h(i)) \notin A$, ma abbiamo apena dimostrato il contrario.

(⇒) Sia A produttivo, e sia f la relativa funzione di produzione. Consideriamo la seguente funzione calcolabile:

$$g(x, y, z) \quad = \quad \begin{cases} 0 & \textit{se } f(x) = z \wedge y \in K \\ \uparrow & \text{altrimenti} \end{cases}$$

Per il teorema s-m-n esiste h totale e calcolabile tale che $g(x,y,z) = \varphi_{h(x,y)}(z)$, ed in particolare

$$W_{h(x,y)} = \begin{cases} \{f(x)\} & \text{se } y \in K \\ \emptyset & \text{altrimenti} \end{cases}$$

Per il secondo teorema di ricorsione esiste s totale e calcolabile tale che $\varphi_{h(s(y),y)} = \varphi_{s(y)}$, e quindi

$$W_{h(s(y),y)} = W_{(s(y)} = \begin{cases} \{f(s(y))\} & \text{se } y \in K \\ \emptyset & \text{altrimenti} \end{cases}$$

Vogliamo dimostrare che $f \circ s$ é una funzione di riduzione da K a $\overline{A}$:

- Se $y \in K$ allora $W_{s(y)} = \{f(s(y))\}$. Se $f(s(y)) \in A$ allora $W_{s(y)} \subseteq A$ e quindi per la produttiviá di A, $f(s(y)) \in A \backslash W_{s(y)}$, ed in particolare $f(s(y)) \notin W_{s(y)} = \{f(s(y))\}$ che é assurdo. Dunque $f(s(y)) \in \overline{A}$.
- Se $y \notin K$, allora $W_{s(y)} = \emptyset \subseteq A$ e dunque, per la produttivitá di A, $f(s(y)) \in A \backslash W_{s(y)}$, ed in particolare $f(s(y)) \in A$.

□

Teorema 10.10. *Un insieme A é creativo se e solo se $A =_m K$.*

Dimostrazione. Per definizione, A é creativo se e solo se é r.e. e $\overline{A}$ é produttivo. Ma A é r.e se e solo se $A \leq_m K$, e per il teorema precedente, $\overline{A}$ é produttivo se e solo se $K \leq_m A$. □

10.1 Insiemi Immuni e insiemi semplici

Applicando ripetutamente la funzione di produzione f per un insieme produttivo A é facile costruire un sottoinsieme infinito r.e. di A.

Lemma 10.11. *Ogni insieme produttivo contiene un sottoinsieme infinito r.e.*

Dimostrazione. Sia f una funzione di produzione per A. Utilizzando il teorema s-m-n é facile costruire una funzione totale e calcolabile r tale che, per ogni i:

$$W_{r(i)} = W_i \cup \{f(i)\}$$

Si noti che siccome $f(i) \notin W_i$, $W_{r(i)}$ estende strettamente W_i. Inoltre, se $W_i \subseteq A$ anche $W_{r(i)} \subseteq A$. Preso dunque m tale che $W_m = \emptyset$, l'unione

$$\bigcup_{n \in \mathcal{N}} W_{r^n(m)}$$

é infinita, r.e. e contenuta in A. $\square$

Un modo per dimostrare l'esistenza di insiemi r.e. non creativi, sarebbe quello di dimostrare l'esistenza di insiemi r.e. il cui complementare non contiene nessun insieme infinito r.e.
Questo giustifica la seguente definizione:

Definizione 10.12. *Sia $A \subseteq \mathcal{N}$.*

- *A si dice* immune *se é infinito e non contiene nessun sottoinsieme infinito r.e.*
- *A si dice* semplice *se é r.e. e $\overline{A}$ é immune.*

Si noti che un insieme *immune* non puó essere r.e. in quanto altrimenti conterrebbe se stesso, infinito e r.e.
Vogliamo ora dimostrare l'esistenza di un insieme semplice. A tal fine, introduciamo la seguente nozione, dovuta a Kolmogorov.

Definizione 10.13. *La complessitá di Kolmogorov di un numero n, indicata con $k(n)$ é il piú piccolo indice i tale che $\varphi_i(0) = n$.*

L'idea é che, invece di dare una descrizione esplicita di un certo numero n, si puó fornire una procedura che permetta di *generarlo* (ad esempio, eseguendo un certo programma i su di un input predeterminato, che possiamo scegliere convenzionalmente a 0). In taluni casi, la procedura risulterá piú compatta del numero esplicito, ovvero avremo $i < n$; si supponga ad esempio di dover *trasmettere* il numero: trasmettendo i al posto di n possiamo ottenere un chiaro risparmio di spazio (e dunque di tempo di trasmissione).

Osserviamo anche che il valore $k(n)$ é sempre definito per ogni n; in particolare $k(n) \neq \mu i.\varphi_i(0) = n$ che potrebbe essere indefinito a causa della possibile divergenza di φ_i per valori di i piú piccoli di $k(n)$. In effetti, la funzione k *non é calcolabile.*

Lemma 10.14. *Per ogni i, se $\varphi_i(0)\downarrow$, $k(\varphi_i(0)) \leq i$.*

*Dimostrazione.*Ovvio, per la definizione di k. □

Definizione 10.15. *Un numero n si dice* random *se $n \leq k(n)$. Indicheremo con $\mathcal{R}$ l'insieme dei numeri random.*

Intuitivamente, un numero random é un numero di cui non é possibile fornire una descrizione piú succinta: lui stesso é la sua descrizione minima.

Lemma 10.16. $\overline{R} \neq \emptyset$.

Dimostrazione. Il lemma 9.4 asseriva l'esistenza di un numero i tale che $\varphi_i(0) = i + 1$. Dunque $i + 1$ non é random. □

Lemma 10.17. *L'insieme $\overline{R}$ é r.e.*

Dimostrazione. Un numero a non é random se e solo se esiste un indice $i < a$ tale che $\varphi_i(0) = a$. Dunque $\overline{R}$ é codominio della funzione parziale calcolabile

$$g(i) \quad = \quad \begin{cases} \varphi_i(0) & \text{se } i < \varphi_i(0) \\ \uparrow & \text{altrimenti} \end{cases}$$

□

Teorema 10.18. *L'insieme R dei numeri random é immune.*

Dimostrazione. Sia A un insieme infinito r.e. e sia f una sua funzione di enumerazione, totale e calcolabile.
Per il teorema s-m-n esiste h totale e calcolabile tale che

$$\varphi_{h(i)}(x) = f(\mu n.f(n) > i)$$

Si osservi che la miminimizzazione $\mu n.f(n) > i$ termina per ogni i in quanto f é totale e $cod(f) = A$ é infinito. Per definizione, $f(\mu n.f(n) > i) \in A$, e se supponiamo $A \subseteq R$ deve trattarsi di un numero random; inoltre

$$(*) \quad \varphi_{h(i)}(x) = f(\mu n.f(n) > i) > i$$

Per il teorema del punto fisso, esiste m tale che $\varphi_m \approx \varphi_{h(m)}$. Dunque avremmo

- $\varphi_m(0) = \varphi_{h(m)}(0) \in R$, che implica $\varphi_m(0) \leq k(\varphi_m(0))$; inoltre, per il Lemma 10.14, $k(\varphi_m(0)) \leq m$, e per transitiviá $\varphi_m(0) \leq m$.
- d'altra parte, per $(*)$, $\varphi_m(0) = \varphi_{h(m)}(0) > m$, che porta a contraddizione.

□

Capitolo 11

Calcolabilitá e completezza

11.1 Aritmetica

Definizione 11.1. *Il linguaggio dell'aritmetica é il linguaggio del primo ordine basato sulla seguente segnatura:*

$$0, S, +, \cdot, =$$

Dato un numero n indichiamo con $\overline{n}$ il termine dell'aritmetica che lo rappresenta.

Definizione 11.2. *Un insieme $A \subseteq \mathcal{N}^k$ si dice aritmetico se esiste una formula $\psi(x_1, \ldots, x_k)$ esprimibile nel linguaggio aritmetico tale che*

$$(n_1, \ldots, n_k) \in A \Leftrightarrow \mathcal{N} \models \psi[\overline{n}_1/x_1, \ldots, \overline{n}_k/x_k]$$

ovvero la formula é vera nel modello dei numeri naturali.
Diremo in questo caso che ψ é una descrizione aritmetica *di A.*

Lemma 11.3. *Gli insiemi aritmetici sono chiusi rispetto alle operazioni di unione, intersezione e complementazione.*

Dimostrazione. Basta utilizzare i connettivi di disgiunzione, congiunzione, e negazione. □

Definizione 11.4. *Una funzione f si dice aritmetica se il suo grafo é un insieme aritmetico.*

Lemma 11.5. *Le seguenti funzioni sono aritmetiche:*

1. $0 : \mathcal{N}^0 \to \mathcal{N}$
2. $S : \mathcal{N}^0 \to \mathcal{N}$
3. $+ : \mathcal{N}^0 \to \mathcal{N}$
4. $\cdot : \mathcal{N}^0 \to \mathcal{N}$
5. $\pi_i^k : \mathcal{N}^0 \to \mathcal{N}$

Dimostrazione. I grafi delle funzioni sono rispettivamente rappresentati dalle seguenti formule:

1. $\psi_0(y) := \quad y = 0$
2. $\psi_S(x, y) := \quad y = S(x)$
3. $\psi_+(x, y, z) := \quad z = x + y$
4. $\psi_\cdot := \quad z = x \cdot y$
5. $\psi_{\pi_i^k}(x_1, \ldots, x_k, z) := \quad z = x_i$

□

Lemma 11.6. *La composizione di funzioni aritmetiche é aritmetica.*

Dimostrazione. Trattiamo per semplicitá il caso di composizone unaria. Supponiamo dunque che $f(x) = g(h(x))$ e siano $\psi_g(x, y)$ e $\psi_h(x, y)$ le descrizioni aritmetiche di g e h. Definiamo

$$\psi_f(x, z) = \exists y, g(x, y) \wedge h(y, z)$$

é facile dimostrare che ψ_f é una descrizione aritmetica di f. □

Lemma 11.7. *Una funzione definita per minimizzazione di una funzione aritmetica é ancora aritmetica.*

Dimostrazione. Sia

$$f(x) = \mu y.(g(x, y) = 0)$$

e supponiamo che ψ_g sia una descrizione aritmetica di g. Definiamo

$$\psi_f(x, y) = \psi_g(x, y, 0) \wedge \forall i, i < y \rightarrow \exists m, (\psi_g(x, i, m) \wedge m \neq 0)$$

Anche in questo caso lasciamo al lettore la facile verifica che ψ_f é una descrizione aritmetica di f. □

Si osservi che nella definizione precedente sarebbe scorretto rimpiazzare

$$\exists m, (\psi_g(x, i, m) \wedge m \neq 0)$$

con

$$\neg\psi_g(x, i, 0)$$

in quanto quest'ultima formula é vera anche in caso di divergenza di $g(x, i)$.

Teorema 11.8. *Tutte le funzioni calcolabili sono aritmetiche.*

Dimostrazione. Come mostrato nella sezione 5.1 ogni funzione calcolabile puó essere espressa in un formalismo che contiene somma, prodotto, costanti, proiezioni ed é chiuso per composizione e ricorsione primitiva. In base ai risultati appena enunciati, ogni funzione calcolabile é quindi aritmetica. □

Teorema 11.9. *Ogni insieme ricorsivamente enumerabile é aritmetico.*

Dimostrazione. Se A é r.e. esiste f totale e calcolabile tale che $A = dom(f)$. Sia $\psi_f(x, y)$ una descrizione aritmetica di f. Allora

$$n \in A \Leftrightarrow \mathcal{N} \models \exists y, \psi_f(\overline{(}n), y)$$

e dunque $\psi_A(x) = \exists y, \psi_f(x, y)$ é una descrizione aritmetica di A. □

Corollario 11.10. *L'insieme K é aritmetico.*

Siamo ora nella posizione di dimostrare alcuni famosi risultati risalenti agli anni trenta del secolo scorso.

Teorema 11.11. *L'insieme delle formule aritmetiche* vere *non é decidibile.*

Dimostrazione. Sia $\{\psi_n\}_{n\in\mathcal{N}}$ una enumerazione effettiva delle formule aritmetiche in una variabile. Consideriamo l'insieme A cosí definito

$$n \in A \Leftrightarrow \models \neg\psi_n\overline{n}$$

Se la veritá aritmetica fosse decidibile, allora A sarebbe ricorsivo. Dunque dovrebbe essere aritmetico e dovrebbe esistere una qualche formula ψ_a nella nostra enumerazione che ne fornisce una descrizione aritmetica, ovvero

$$n \in A \Leftrightarrow \models \psi_a(\overline{n})$$

Ma per $n = a$ otteniamo una contraddizione. □

In base alla dimostrazione precedente, non solo l'insieme delle formule aritmetiche vere non é decidibile, ma non é neppure r.e. (altrimenti anche A sarebbe r.e., che é sufficiente per ottenere la contraddizione). In effetti possiamo ulteriormente rafforzare il risultato:

Teorema 11.12. *L'insieme delle formule aritmetiche vere é un insieme produttivo.*

Dimostrazione. Sia $\{\psi_n\}_{n\in\mathcal{N}}$ una enumerazione effettiva di tutte le fomrule aritmetiche. Per la natura effettiva della enumerazione, esiste una funzione totale e calcolabile $\ni$ tale che $\neg\psi_n = \psi_{\ni(n)}$, ovvero possiamo calcolare in modo uniforme ed effettivo l'indice della formula negata in funzione dell'indice della formula di partenza (la stessa proprietá vale per tutti i connettivi e tutti i quantificatori). Possiamo inoltre supporre che anche l'operazione di sostituzione di variabili con numerali sia effettiva, nel senso che esiste una funzione σ tale che

$$\psi_n[\overline{m}/x] = \psi_{\sigma(n,m)}$$

Poiché K é un insieme r.e. esiste una qualche formula ψ_k nella nostra enumerazione tale che

$$n \in K \Leftrightarrow \mathcal{N} \models \psi_k[\overline{n}/x]$$

o anche

$$n \in \overline{K} \Leftrightarrow \mathcal{N} \models \neg\psi_k[\overline{n}/x] = \psi_{\ni(\sigma(k,n))}$$

Pertanto l'appartenenza a $\overline{K}$ 'e riducibile in modo effettivo alla veritá di una formula aritmetica, il che mostra che l'insieme delle formule aritmetiche vere é produttivo. □

Teorema 11.13. *Ogni sistema formale aritmetico formale, se consistente é incompleto, nel senso ch esistono formule aritmetiche valide ma non dimostrabili.*

Dimostrazione. Un sistema formale é definito da un insieme ricorsivo di assiomi e un insieme di regole di inferenza che permettono di dedurre nuovi teoremi a a partire dagli assiomi in un numero finito di applicazioni. Pertanto le formule aritmetiche dimostrabili costituiscono un insieme ricorsivamente enumerabile. Poiché le formule vere costituiscono un insieme produttivo, esistono necessarimente delle formule vere ma non dimostrabili. □

11.2 Indecidibilitá della logica del primo ordine

Il risultato di incompletezza stabilito dal teorema 11.13 non permette di concludere nulla sulla decidibiliá della nozione di dimostrabilitá. Ad esempio, il sistema formale potrebbe essere incompleto proprio perché dimostra un sottoinsieme relativamente piccolo (ricosivo) di formule aritmetiche.

Al fine di investigare la natura ricorsiva o meno dell'insieme delle formule dimostrabili in un dato sistema formale $\mathcal{F}$ é necessario studiare la nozione di *rappresentabilitá*, che é essenzialmente il corrispettivo *sintattico* della nozione semantica di *descrivibilitá aritmetica*.

Definizione 11.14. *Un insieme $A \subseteq \mathcal{N}^k$ si dice debolmente rappresentabile in un sistema formale $\mathcal{F}$ se esiste una formula $\psi(x_1, \ldots, x_k)$ esprimibile nel linguaggio aritmetico tale che*

$$(n_1, \ldots, n_k) \in A \Leftrightarrow \mathcal{F} \vdash \psi[\overline{n}_1/x_1, \ldots, \overline{n}_k/x_k]$$

ovvero la formula é dimostrabile in $\mathcal{F}$. Diremo in questo caso che ψ é una rappresentazione debole *di A.*

Si parla di rappresentazione *debole* poiché potremmo richiedere qualche cosa di piú: nel caso in cui $(n_1, \ldots, n_k) \notin A$ la definizione precedente assicura solo che $\mathcal{F} \nvdash \psi[\overline{n}_1/x_1, \ldots, \overline{n}_k/x_k]$, mentre potremmo desiderare che $\mathcal{F} \vdash \neg\psi[\overline{n}_1/x_1, \ldots, \overline{n}_k/x_k]$:

Definizione 11.15. *Un insieme $A \subseteq \mathcal{N}^k$ si dice* rappresentabile *in un sistema formale $\mathcal{F}$ se esiste una formula $\psi(x_1, \ldots, x_k)$ esprimibile nel linguaggio aritmetico tale che*

$$(n_1, \ldots, n_k) \in A \Rightarrow \mathcal{F} \vdash \psi[\overline{n}_1/x_1, \ldots, \overline{n}_k/x_k]$$

e

$$(n_1, \ldots, n_k) \notin A \Rightarrow \mathcal{F} \vdash \neg\psi[\overline{n}_1/x_1, \ldots, \overline{n}_k/x_k]$$

Diremo in questo caso che ψ é una rappresentazione *di A.*

11.2.1 Aritmetica di Robinson

Volgiamo ora introdurre un semplice sistema aritmetico formale sufficientemente espressivo da permettere di rappresentare tutte le funzioni parziali iorsive (ovvero, i loro grafi): l'aritmetica di Robinson.

Definizione 11.16. *L'aritmetica di Robinson $\mathcal{R}$ é quel sistema formale del primo ordine basato sui seguenti assiomi (tutte le variabili libere si intendono quantificate universalmente)*

A1 $0\neg S(x)$

A2 $S(x) = S(y) \rightarrow x = y$

A3 $x\neg 0 \rightarrow existsy, x = S(y)$

A4 $x + 0 = x$

A5 $x + S(y) = S(x + y)$

A6 $x \cdot 0 = 0$

A7 $x \cdot S(y) = x \cdot y + x$

L'aritmetica di Peano é una estensione consistente dell'aritmetica di Robinson che si ottiene aggiungendo lo *schema di induzione*:

$$P(0) \wedge (\forall x.P(x) \rightarrow P(S(x))) \rightarrow \forall x.P(x)$$

In questo caso, l'assioma $A3$ diviene dimostrabile e puó essere rimosso.

Usando tecniche simili a quelle impiegate per il teorema 11.8 é possibile dimostrare che il grafo di ogni funzione parziale ricorsiva é rappresentabile nell'aritmetica di Robinson. Questo permette di stabile il seguente risultato:

Teorema 11.17. *In una qualunque estensione consistente dell'aritmetica di Robinson*

1. *Un insieme é rappresentabile se e solo se é ricorsivo*
2. *Un insieme é debilmente rappresentabile se e solo se é ricorsivamente enumerabile.*

*Dimostrazione.*Per la dimostrazione si veda ad esempio [**?**]. □

Teorema 11.18. *Ogni estensione consistente $\mathcal{F}$ dell'aritmetica di Robinson é indecidibile.*

Dimostrazione. La dimostrazione é identica a quella del teorema **??** sostituendo la nozione di descrivibilitá con quella di rappresentabilitá. □

Teorema 11.19. *La logica del primo odine é indecidibile.*

*Dimostrazione.*Sia R la congiunzione degli assiomi $A1 - A7$ dell'aritmetica di Robinson. Allora $R \vdash \psi$ se e solo se la formula $R \to \psi$ é dimostrabile al primo ordine. □