PROGRAMMI UNIVERSALI

Sia $\mathcal{P}$ un programma arbitrario tale che $\#(\mathcal{P}) = y$ e sia $\psi_{\mathcal{P}}^{(n)}(x_1, \ldots, x_n)$ la funzione (parzialmente calcolabile) calcolata da $\mathcal{P}$.

Per ogni $n > 0$, definiamo:

$$\Phi^{(n)}(x_1, \ldots, x_n, y) = \psi_{\mathcal{P}}^{(n)}(x_1, \ldots, x_n), \text{ con } y = \#(\mathcal{P})$$

TEOREMA

Per ciascun $n > 0$, la funzione $\Phi^{(n)}(x_1, \ldots, x_n, y)$ è PARZIALMENTE CALCOLABILE.

Osservazioni

L'esistenza di un tale teorema, che è uno dei risultati centrali della teoria della calcolabilità, non era affatto scontata. Ciò che ci dice è abbastanza sorprendente: PER OGNI n, TUTTE LE FUNZIONI DI n VARIABILI SONO CALCOLABILI MEDIANTE UN UNICO PROGRAMMA.

(Ricordando poi che dal punto di vista della calcolabilità possiamo limitarci a considerare soltanto funzioni monoargomentali, otteniamo il risultato che ESISTE UN PROGRAMMA CHE CALCOLA QUALSIASI FUNZIONE CALCOLABILE.

COMMENTO GENERALE

Nel campo della teoria della calcolabilità sembra esistere un livello di "complessità" (o di "ricchezza della teoria") tale che se viene superato non c'è alcuna necessità di arricchire ulteriormente la teoria, qualunque sia ciò che noi vogliamo esprimere in esso.

La "ricchezza della teoria" di cui parlavamo prima ha fondamentalmente a che fare con la possibilità di esprimere opportuni sistemi di codifica.

Sono proprio risultati e considerazioni di questo tipo quelli che sostengono e corroborano il programma di ricerca dell'Intelligenza Artificiale e quegli aspetti delle Scienze (e della Psicologia) Cognitive che cercano di costruire MODELLI COMPUTAZIONALI DELLA MENTE.

Per dimostrare il nostro teorema mostreremo come costruire, per ciascun $n > 0$, un programma universale $\mathcal{U}_n$ che calcola $\Phi^{(n)}$, cioè tale che

$$\psi_{\mathcal{U}_n}^{(n+1)}(x_1, \ldots, x_n, x_{n+1}) = \Phi^{(n)}(x_1, \ldots, x_n, x_{n+1})$$

$Z \leftarrow X_{n+1} + 1$

$$S \leftarrow \prod_{i=1}^{n} (p_{2i})^{X_i}$$

$K \leftarrow 1$

[C] IF $K = Lt(Z) + 1 \vee K = 0$ GOTO F

$U \leftarrow r((Z)_K)$

$P \leftarrow p_{r(U)+1}$

IF $1(U) = 0$ GOTO N

IF $1(U) = 1$ GOTO A

IF $\sim(P \mid S)$ GOTO N

IF $1(U) = 2$ GOTO M

$K \leftarrow \min_{i \leq Lt(Z)} [\, 1((Z)_i) + 2 = 1(U) \,]$

GOTO C

[M] $S \leftarrow \lfloor S/P \rfloor$

GOTO N

[A] $S \leftarrow S \cdot P$

[N] $K \leftarrow K + 1$

GOTO C

[F] $Y \leftarrow (S)_1$

PROGRAMMA UNIVERSALE $\mathcal{U}_m$

Prima di discutere istruzione per istruzione il programma che risolve il nostro problema consideriamo i passi principali che tale programma deve compiere:

I. Deve Memorizzare il numero del programma e le variabili di ingresso

II. Deve poter sapere quale istruzione del programma simulato sta effettuando

III. Deve poter effettuare l'istruzione del programma simulato in modo coerente con tutto il marchingegno

IV. Deve poter passare all'istruzione successiva del programma simulato

V. Deve fermarsi quando ha finito di eseguire tutte le istruzioni del programma simulato e fornire il valore corretto della variabile di uscita.

Tra le variabili ausiliarie ne sceglieremo due che, in deroga alla ferrea legge di chiamare le variabili col nome della loro categoria, verranno indicate con K e con S.

K ci dirà qual'è l'istruzione che sta per essere eseguita.

S conterrà lo stato attuale codificato mediante numeri di Gödel come segue:

- assumeremo che, come al solito, le variabili a cui non è assegnato un valore siano azzerate;
- seguiremo sempre l'ordinamento solito delle variabili:

  $$Y\ X_1\ Z_1\ X_2\ Z_2\ \ldots$$

  da cui si vede che le variabili di ingresso occupano i posti pari nella lista.

Invece di considerare un insieme di equazioni $V_i = v_i$, considereremo direttamente il numero di Gödel $[v_1, \ldots, v_m]$ per rappresentare lo stato attuale.

La situazione iniziale del programma $\mathcal{U}_n$ per calcolare la funzione $Y = \Phi^{(n)}(X_1, \ldots, X_n, X_{n+1})$ sarà dunque:

- $X_{n+1} = \#(\mathcal{P})$ (numero del programma)
- $[0, X_1, 0, \ldots, X_n]$ (stato iniziale)
- $K = 1$ (deve essere eseguita la prima istruzione)

Vediamo di trasformare queste informazioni in istruzioni di U_n.

Noi vogliamo conoscere in dettaglio le istruzioni di $\mathcal{P}$ per eseguirle quindi dobbiamo passare dal numero di $\mathcal{P}$ alla codifica delle istruzioni.

Ricordiamo che è:

$$\#(\mathcal{P}) = [\#(I_1), \#(I_2), \ldots, \#(I_m)] - 1$$

quindi avremo:

$$[\#(I_1), \#(I_2), \ldots, \#(I_m)] = X_{n+1} + 1$$

Diamo questo valore ad una variabile ausiliaria Z:

1. $Z \leftarrow X_{n+1} + 1$ (prima istruzione)

Dobbiamo adesso dare ad S come valore il numero di Gödel dello stato iniziale; abbiamo

$$[0, X_1, 0, X_2, \ldots, 0, X_n] = \prod_{i=1}^{n} (P_{2i})^{X_i}$$

quindi porremo:

2. $S \leftarrow \prod_{i=1}^{n} (P_{2i})^{X_i}$ (seconda istruzione)

Assegnamo infine a K il valore 1 perché stiamo adesso iniziando e deve essere eseguita la prima istruzione:

3. $K \leftarrow 1$

Passiamo adesso ad istruzioni che riguardano direttamente il programma specifico che stiamo simulando.

Questa parte sarà più difficile perché dovremo prendere in considerazione qualsiasi situazione, qualsiasi successione generica di istruzioni.

L'istruzione successiva è

4. [c] IF K=Lt(Z)+1 V K=0 GOTO F

Con quest'istruzione esaminiamo il caso in cui il programma simulato si ferma. Questo può avvenire perché, arrivato all'ultima istruzione, dovrebbe andare oltre oppure perché un salto condizionato lo rinvia ad una istruzione inesistente.

Il primo caso è rispecchiato dall'assegnare a K il valore Lt(Z)+1
Il secondo lo rispecchiamo assegnando a K un valore ancora non utilizzato.

In entrambi questi casi il programma viene rinviato ad una istruzione F (finale) che si prenderà cura di assegnare alla variabile di uscita Y il valore che il programma è andato calcolando.

5. $U \leftarrow r((\mathcal{Z})_k)$

Per quanto abbiamo detto prima, questa istruzione verrà attivata solo se $k < Lt(\mathcal{Z}) + 1$, cioè se vi sono istruzioni del programma $\mathcal{P}$ che si possono simulare.

Per prima cosa interpretiamo il simbolismo

Che cos'è $(\mathcal{Z})_k$?

Ricordiamo che

- $\mathcal{Z} = [\#(I_1), \ldots, \#(I_m)]$ è un numero di Gödel
- la funzione $([\,])_i$ applicata ad un numero di Gödel ci restituisce l'i-esimo termine

Quindi $(\mathcal{Z})_k$ è il numero della k-esima istruzione del programma

Ricordiamo ancora che il codice di un'istruzione viene scritto come:

$$\langle a, \langle b, c \rangle \rangle$$

e che r prende la parte destra di tale scrittura.

Quindi: $r((\mathcal{Z})_k) = \langle b, c \rangle$

L'istruzione 5 ci dice di mettere in U il codice dell'enunciato che deve essere eseguito.

Abbiamo quindi

$$U = \langle b, c \rangle .$$

Ricordiamo che b codifica l'istruzione che deve essere eseguita mentre c codifica la variabile menzionata nell'istruzione.

Individuiamo prima la variabile.

Ricordiamo che $c = \#(V) - 1$

per cui $\#(V) = c + 1$

ma $c = r(U)$ e quindi:

$$\#(V) = r(U) + 1$$

Prendiamo allora l'$(r(U)+1)$-esimo numero primo e diamo tale valore a P:

6. $P \leftarrow p_{r(U)+1}$

(Ricordo:
Nel nostro meccanismo di codifica il valore di questa variabile viene ricordato come l'esponente al quale è elevato P nella decomposizione in numeri primi di S).

Avendo individuato la variabile presa in esame nella k-esima istruzione dobbiamo adesso prendere in esame l'istruzione che deve essere eseguita.
Questa è codificata da b, cioè da $l(U)$ essendo $U = \langle b, c \rangle$.

Il gruppo formato dalle quattro istruzioni seguenti esaminerà l'istruzione codificata da $b = l(U)$ e ci dirà come fare in corrispondenza dei vari casi.

7. IF $l(U) = 0$ GOTO N

Se $b = l(U) = 0$ allora l'istruzione è la banale $V \leftarrow V$ e non c'è nient'altro da fare se non passare all'istruzione successiva del programma simulato.
Il programma universale deve quindi rinviarci ad una successiva istruzione che aumenta di 1 il contenuto del "contatore" k.
Possiamo quindi preannunciare che l'istruzione N sarà:

[N] $K \leftarrow K+1$

la stabiliremo successivamente in una posizione opportuna.

8. IF $\ell(V)=1$ GOTO A

Se $b=\ell(V)=1$ allora l'istruzione è $V \leftarrow V+1$.

Ciò che deve fare il programma è di aumentare di una unità l'esponente al quale è elevato P nella decomposizione in fattori primi di S.

L'istruzione A alla quale veniamo rinviati deve quindi fare proprio ciò:

[A] $S \leftarrow S \cdot P$

OSSERVAZIONE

Mentre l'istruzione 7 ci ha rinviato direttamente all'istruzione che aumenta il valore dell'istruzione del programma simulato contenuto in K, la 8. non lo fa. Sia la 7. che la 8. dovrebbero poi rinviarci — dato che sono state effettuate — all'istruzione C per ricominciare il processo (se vi sono altre istruzioni da esaminare) ovvero per andare all'istruzione finale.

Di tutto questo dovremo tenere conto successivamente nell'ordinare opportunamente le istruzioni N, A ed altre che occorreranno.

9. IF $\nu(P|S)$ GOTO N

Se siamo giunti a quest'istruzione vuol dire che $l(V)$ non era uguale né a 0 né a 1 (perché le 7 e le 8 rinviano rispettivamente ad A ed N che saranno scritte successivamente).
Questo vuol dire che l'istruzione sarà una delle due seguenti:

V ← V−1

IF V≠0 GOTO L

Per entrambe queste due istruzioni distinguiamo due casi: V=0 e V≠0.

Se V=0 in entrambi i casi non dobbiamo fare nulla (e quindi passare all'istruzione successiva del programma simulato P).
Ma V=0 significa che P non compare nella decomposizione in fattori primi di S la formula $\nu(P|S)$.
L'istruzione 9 quindi ci dice che se V=0 non c'è niente da fare se non aumentare di 1 la variabile K, con che, sappiamo già, fa l'istruzione N.

Consideriamo adesso il caso in cui $V \neq 0$ e sia $\ell(V) = 2$.
In questo caso l'istruzione considerata è la $V \leftarrow V - 1$.
Ciò che bisogna quindi fare è diminuire la variabile di una unità e questo può essere ottenuto diminuendo di 1 l'esponente col quale P compare nella decomposizione in fattori primi di S.
Ciò è rappresentato dall'istruzione

10. IF $\ell(V) = 2$ GOTO M

che rinvia all'istruzione M:

[M] $S \leftarrow \lfloor S/P \rfloor$

Rimane da esaminare il caso in cui l'istruzione in P da considerare è della forma IF $V \neq 0$ GOTO L con $V \neq 0$

Come fare a rinviare il programma all'istruzione L ?
Assegnando al contatore k il valore numerico corrispondente alla prima istruzione etichettata L in P.

Dobbiamo quindi assegnare a k il più piccolo i tale che

$$\ell((Z)_i) = \#(L)$$

(Nota: il più piccolo perché potremmo avere più istruzioni con la stessa etichetta).

L'istruzione IF V≠0 GOTO L è codificata da $b = \#(L) + 2$ quindi si ha

$$\#(L) = b - 2 = \ell(V) - 2$$

e quindi $\ell((Z)_i) = \ell(V) - 2$

Possiamo quindi scrivere:

11. $K \leftarrow \min_{i \leq Lh(Z)} \left[\ell((Z)_i) + 2 = \ell(V) \right]$

Facciamo seguire alla 11 un'istruzione che fa ricominciare il ciclo

12. GOTO C

OSSERVAZIONE

Prima di andare oltre chiediamoci cosa succede se non esiste un'istruzione con l'etichetta richiesta. Intuitivamente il programma dovrebbe fermarsi. Ed è ciò che fa il nostro programma universale, nei tempi e nei modi dovuti.
In questo caso infatti l'operatore di minimalizzazione limitata assume il valore 0; la 11 quindi assegna a k il valore 0, la 12 rimanda a C che, a sua volta rimanda ad F, essendo k=0.
La F, come vedremo, non fa altro che assegnare ad Y

Noi ci rimane adesso che sistemare in ordine opportuno le tre istruzioni scritte ma lasciate in sospeso:

Minus	[M]	$S \leftarrow [S/P]$
Aggiungi	[A]	$S \leftarrow S \cdot P$
	[N]	$k \leftarrow k+1$

Sia dopo la M che dopo la A bisogna passare alla N e subito dopo ritornare a C per ricominciare il ciclo.

Tali istruzioni possono perciò ordinarsi come segue:

13.	[M]	$S \leftarrow [S/P]$
14.		GOTO N
15.	[A]	$S \leftarrow S \cdot P$
16.	[N]	$k \leftarrow k+1$
17.		GOTO C

Manca solo un'ultima istruzione:

18.	[F]	$Y \leftarrow (S)_1$

Quest'ultima viene attivata solo dalla C quando il calcolo è finito e cioè o quando abbiamo esaurito tutte le istruzioni o quando un'istruzione di salto condizionato ci ha rinviato ad un'istruzione che non esiste.
Ciò che fa la 18 è quindi di fornire ad Y, la variabile di uscita, il valore conservato in $(S)_1$, cioè nel primo posto dello stato che è quello che contiene, temporaneamente, il valore della variabile di uscita.

Consideriamo il programma P
il cui numero è $2^{45} \cdot 3^2 \cdot 5^{46} - 1$

Vediamo cosa calcola per $x = 2$.

Avremo che

$Z \leftarrow 2^{45} \cdot 3^2 \cdot 5^{46}$

$S \leftarrow 3^2 \qquad \left(\prod_{i=1}^{n} p_{2_i}^{x_i} \text{, con } x_i = x_1 = x \right)$

$K \leftarrow 1$

$U \leftarrow r((Z)_1) \quad \left[= r\left(2^{45} \cdot 3^2 \cdot 5^{46}\right)_1 = r(45) = 11 \right]$

$P \leftarrow p_{r(U)+1} \quad \left[= p_{r(45)+1} = p_{11+1} = p_2 = 3 \right]$

IF $\ell(U) = 0$ GOTO N $\quad [\ell(11) = 2]$

IF $\ell(U) = 1$ GOTO A

IF $\nu(P|S)$ GOTO N $\quad [\nu(P|S) = \text{falso}]$

IF $\ell(U) = 2$ GOTO M

[M] $S \leftarrow LS/P\rfloor \quad [= 3]$

GOTO N

[N] $K \leftarrow K+1 \quad [= 2]$

GOTO C

poiché $k = 2$ $(\neq 4 \text{ e } \neq 0)$

$U \leftarrow r((z)_2)$ $\left[= r\left(2^{45} \cdot 3^2 \cdot 5^{46}\right)_2 = r(2) =\right.$

$P \leftarrow p_{r(U)+1}$ $\left[= p_{r(1)+1} = p_{0+1} = p_1 = 2\right]$

IF $\ell(U) = 0$ GOTO N $[\ell(1) = 1]$

IF $\ell(U) = 1$ GOTO A

[A] $S \leftarrow S \cdot P$ $[= 2 \cdot 3]$

$k \leftarrow k+1$ $[= 3]$

GOTO C

poiché $k = 3$ $(\neq 4 \text{ e } \neq 0)$

$U \leftarrow r((z)_3)$ $[= r(46) = 23]$

$P \leftarrow p_{r(U)+1}$ $\left[= p_{r(23)+1} = p_{1+1} = p_2 = 3\right]$

. . . . . .

$\ell(23) = 3$ quindi si passa a

$k \leftarrow \min_{i \leq 3} [\ell((z)_i) + 2 = 3] = 1$

$S \leftarrow \lfloor S/P \rfloor$ $[= 2 \cdot 3/3 = 2]$

$K \leftarrow 2$

$P \leftarrow p_{r(U)+1}$ $[= p_1 = 2]$

- - - - -

IF $\ell(U) = 1$ GOTO A

[A] $S \leftarrow S \cdot P$ $[= 2^2]$

$k \leftarrow 3$

- - - - - -

IF $N(P|S)$ GOTO N $[N(P|S)$ è vero $P=3;\ S=2^2.]$

[N] $k \leftarrow k+1$ $[= 4]$

GOTO C

- - - - -

GOTO F

[F] $Y \leftarrow (S)_1 = 2$

Programma

[A] $X \leftarrow X-1$ (I_1)

$Y \leftarrow Y+1$ (I_2)

IF $X \neq 0$ GOTO A (I_3)

$\#(I_1) = \langle a, \langle b, c \rangle \rangle = \langle 1, \langle 2, 1 \rangle \rangle =$

$= \langle 1, 11 \rangle = 45$

perché $\langle 2,1 \rangle = 2^2(2 \cdot 1 + 1) - 1 = 2^2 \cdot 3 - 1 = 11$

e $\langle 1, 11 \rangle = 2^1(2 \cdot 11 + 1) - 1 = 2 \cdot 23 - 1 = 45$

$\#(I_2) = \langle 0, \langle 1, 0 \rangle \rangle = \langle 0, 2^1(0+1) - 1 \rangle =$

$= \langle 0, 2-1 \rangle = \langle 0, 1 \rangle = 2^0(2 \cdot 1 + 1) - 1 = 3 - 1 = 2$

$\#(I_3) = \langle 0, \langle 3, 1 \rangle \rangle = \langle 0, 2^3(2 \cdot 1 + 1) - 1 \rangle = \langle 0, 23$

$= 2^0(2 \cdot 23 + 1) - 1 = 46$

$\#(P) = [\#(I_1), \#(I_2), \#(I_3)] - 1 =$

$= [45, 2, 46] - 1 =$

$= 2^{45} \cdot 3^2 \cdot 5^{46} - 1.$

2. Decodifiche

Scriviamo esplicitamente il programma il cui numero è : 199.

Abbiamo : $\#(P) = 199 = [I_1, \dots, I_n] + 1$

quindi $[I_1, \dots, I_n] = 200 = 2^3 \cdot 5^2$

allora $[3, 0, 2] = 200$

Dobbiamo trovare le istruzioni i cui numeri sono rispettivamente 3, 0, 2 :

I_1 : $3 = \langle 2, 0 \rangle = \langle 2, \langle 0, 0 \rangle \rangle$

I_2 : $0 = \langle 0, 0 \rangle = \langle 0, \langle 0, 0 \rangle \rangle$

I_3 : $2 = \langle 0, 1 \rangle = \langle 0, \langle 1, 0 \rangle \rangle$

si ha che

I_1 [B] $Y \leftarrow Y$

$Y \leftarrow Y$

$Y \leftarrow Y + 1$

(calcola, in modo non semplice, la funzione $y = 1$)

have:

$$U(n,m) = T_n(m)$$

for each (n, m) for which T_n is a correctly specified Turing machine.[6] The machine U, when first fed with the number n, precisely imitates the n^{th} Turing machine!

Since U is a Turing machine, it will itself have a number; i.e. we have

$$U = T_u,$$

for some number u. How big is u? In fact we can take *precisely*

u = 7244855335339317577198395039615711237952360672556559631108144796606505059404241090310483613632359365644443458382226883278767626556144692814117715017842551707554085657689753346356942478488597046934725739988582283827795294683460521061169835945938791885546326440925525508205559894518907165374148960330967530204315536250349845298323206515830476641421307088193297172341510569802627346864299218381721573334828230734537134214750597403451843723595930906400243210773421788514927607975976344151230795863963544922691594796546147113457001450481673375621725734645227310544829807849651269887889645697609066342044779890219144379328300194935709639217039048332708825962013017737272027186259199144282754374223513556751340842222998937441053430547104436869587640517812801943753081387063994277282315642528923751456544389905278079324114482614235728619311833261065612275553181020751108533763380603108236167504563585216421486954234718742643754442879006248582709124042207653875426445413345174856629157429990950262300973373813772416217274772361020678685400289356608569682262014198248621698902609130940298570600174300670086896759034473417412787425581201549366393899690581773859165405535670409282133222163141097871081459978669599704509681841906299443656015145490488092208448003482249207730403043188429899393135266882349662101947161910701461968523192847482034495897709553561107027581748733327296678998798473284098190764851272631001740166787363477605857245036964434897992034489997455662402937487668839751404451665707750060513883991668814072545544665222050724262392379211525318162512536305093172863142200406457130527580230766518335199568913974813750492642960501001365198018694563949 8

(or some other possibility of at least that kind of size). This number no doubt seems alarmingly large! Indeed it *is* alarmingly large, but I have not been able to see how it could have been made significantly smaller. The coding procedures and specifications that I have given for Turing machines are quite reasonable and simple ones, yet one is inevitably led to a number of this kind of size for the coding of an actual universal Turing machine.[7]

I have said that all modern general purpose computers are, in effect, universal Turing machines. I do not mean to imply that the logical design of such computers need resemble at all closely the kind of description for a universal Turing machine that I have just given. The point is simply that, by supplying any universal Turing machine first with an appropriate program (initial part of the input tape), it can be made to mimic the behaviour of any Turing machine whatever! In the description above, the program simply takes the form of a single number (the number n), but other procedures are possible, there being many variations on Turing's original theme. In fact in my own descriptions I have deviated somewhat from those that Turing originally gave. None of these differences is important for our present needs.