NUMERI DI GÖDEL

I – CODIFICA DI COPPIE DI NUMERI MEDIANTE UN SOLO NUMERO

Definiamo la funzione $<x, y> = 2^x(2y+1)-1$ che è ricorsiva primitiva.
Questa funzione è surgettiva perché tutti i numeri possono essere scomposti in fattori primi e poiché la scomposizione in fattori primi di un numero è univoca, allora la funzione è anche iniettiva.
Sia ora z un qualsiasi numero: l'equazione $<x, y> = z$ ammette una soluzione unica. Infatti:
$2^x(2y+1)-1 = z$
$2^x(2y+1) = z+1$
Sia x il massimo numero tale che $2^x|(z+1)$, cioè 2^x è divisore di z+1, e sia y la soluzione dell'equazione $2y+1 = (z+1)/2^x$ (questa equazione ammette un'unica soluzione perché essendo stati semplificati tutti i 2, il membro destro è dispari).

Per esempio sia z = 35. Si ha:
$<x, y> = 35$
$2^x(2y+1)-1 = 35$
$2^x(2y+1) = 36$
$2y+1 = 36/2^x$
$2y+1 = 9$ con $2^x = 4$, cioè $x = 2$
$2y = 8$
$y = 4$
quindi $<x, y> = z \Rightarrow <2, 4> = 35$

Operatore di minimalizzazione limitata:
$g = \min_{y\leq z} P(x) =$ { minimo valore di $y \leq z$ per cui il predicato P è vero se un tale g esiste
{ 0 altrimenti

PROPOSIZIONE: La minimalizzazione limitata è una funzione ricorsiva primitiva.
DIM (per induzione su y): Per y = 1 si ha che se P(x) è vero restituisco 1 altrimenti 0.
Supponiamo che vale per y = n e dimostriamolo per y = n+1:
- se $\min_y P(x)$ è vero per y = n allora sarà vero anche per n+1, perché a noi interessa trovare il minimo y per cui P(x) è vero;
- se $\min_y P(x)$ non è vero per y = n, allora si controlla se è vero per y = n+1: se lo è il risultato è n+1 altrimenti è 0.

Il vantaggio della minimalizzazione limitata è che genera sempre funzioni totali, lo svantaggio è che ci sono casi in cui questo limite non è trovato.

L'equazione $<x, y> = z$ definisce le due funzioni $x = l(z)$ e $y = r(z)$, e poiché sia x sia y sono minori di z+1 sia ha che: $l(z) < z+1$ e $r(z) < z+1$. Possiamo scrivere l(z) e r(z) in termini di operazioni di minimalizzazione limitata ed operatore esistenziale limitato:
$l(z) = \min_{x\leq z} [\exists y \leq z\ (z = <x, y>)]$
$r(z) = \min_{y\leq z} [\exists x \leq z\ (z = <x, y>)]$
Allora possiamo concludere che l(z) ed r(z) sono ricorsive primitive.

Riassumendo abbiamo che:
Le funzioni $<x, y> = z$, l(z), r(z) sono ricorsive primitive e sono tali che:
- $\mathbf{l(<x, y>) = x}$
- $\mathbf{r(<x, y>) = y}$
- $\mathbf{<l(z), r(z)> = z}$
- $\mathbf{l(z) \leq z}$
- $\mathbf{r(z) \leq z}$

II – CODIFICA DI SUCCESSIONI ARBITRARIE FINITE DI NUMERI

Chiamiamo **numero di Gödel** della successione $(a_1, \dots, a_n)$ il numero $[a_1, \dots, a_n] = \prod_{i=1}^{n} P_i^{a_i}$

Ad ogni successione finita di numeri associamo il prodotto dei primi n numeri primi (essendo n la lunghezza della successione) ciascuno dei quali ha come esponente il numero di posto corrispondente nella successione data.

Per esempio:

Successione: $(a_1, \dots, a_n)$
Successione numeri primi: $P_1, \dots, P_n$
Numero di Gödel $[a_1, \dots, a_n] = P_1^{a1} \cdot \dots \cdot P_n^{an}$

Successione: (3, 1, 5, 4, 6)
Successione numeri primi: 2, 3, 5, 7, 11
Numero di Gödel $[3, 1, 5, 4, 6] = 2^3 \cdot 3^1 \cdot 5^5 \cdot 7^4 \cdot 11^6$

Un numero intero è esprimibile in un unico modo come prodotto di fattori primi e quindi la numerazione di Gödel gode della proprietà di unicità: se $[a_1, \dots, a_n] = [b_1, \dots, b_n]$ allora $a_1 = b_1, \dots, a_n = b_n$.
Se aggiungiamo uno 0 oltre il termine destro abbiamo che $[a_1, \dots, a_n] = [a_1, \dots, a_n, 0]$ (ambiguità).
Consideriamo 1 come il numero di Gödel della sequenza vuota.

Esempio: consideriamo $[2, 1, 0, 2] = 2^2 \cdot 3^1 \cdot 5^0 \cdot 7^2 = 4 \cdot 3 \cdot 1 \cdot 49 = 588$
Aggiungendo uno 0 a destra si ha: $[2, 1, 0, 2, 0] = 2^2 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^0 = 4 \cdot 3 \cdot 1 \cdot 49 \cdot 1 = 588$
Aggiungendo uno 0 a sinistra si ha: $[0, 2, 1, 0, 2] = 2^0 \cdot 3^2 \cdot 5^1 \cdot 7^0 \cdot 11^2 = 1 \cdot 9 \cdot 5 \cdot 1 \cdot 121 = 5445$

Definiamo una funzione **$(x)_i$** che applicata al numero di Gödel x ci fornisce l'i-esimo termine della successione codificata da x: cioè se $x = [a_1, \dots, a_n]$ allora $(x)_i = ([a_1, \dots, a_n])_i = a_i$.
Per trovare l'i-esimo termine dobbiamo cercare il massimo esponente t che si può dare al numero primo i-esimo P_i in modo tale che P_i^t sia divisore di x. Poiché questa operazione è esprimibile mediante minimalizzazione limitata (il limite superiore è il $\log_i(x)$), $(x)_i$ è ricorsiva primitiva.

Dimostriamo che la successione dei numeri primi P_i è una funzione ricorsiva primitiva.
Sia P_n: N → N una funzione totale che per ogni n ci fornisce l'n-esimo numero primo.
Poniamo $P_0 = 0$ e scriviamo le seguenti equazioni di ricorsione:
{ $P_0 = 0$
{ $P_{n+1} = \min_{t \leq H} [\text{primo}(t) \ \& \ t \geq P_n]$
Dobbiamo fissare un H tale che siamo sicuri di avere raggiunto il numero primo (n+1)-esimo partendo dal numero primo n-esimo.
Tecnica di Euclide per concludere che esistono infiniti numeri primi: dati i primi n numeri primi, consideriamo il numero primo $P_n = (P_n)! + 1$
P_n non è divisibile per nessuno degli n numeri primi, infatti per $0 < i \leq n$ si ha che:

$$\frac{(P_n)!+1}{P_i} = \frac{P_1 \cdots P_{i-1} \cdot P_i \cdot P_{i+1} \cdots P_n + 1}{P_i} = P_1 \cdots P_{i-1} P_{i+1} \cdots P_n + \frac{1}{P_i}$$

Concludiamo che o P_n è primo oppure che è divisibile per un numero primo > P_n.
Usiamo questo ragionamento per dire che P_n è un buon limite da imporre alla minimalizzazione limitata.

- Poniamo perciò $H = (P_n)! + 1$ e avremo:
  { $P_0 = 0$
  { $P_{n+1} = \min_{t \leq (Pn)!+1} [\text{primo}(t) \ \& \ t \geq P_n]$
- Poniamo $h(y, z) = \min_{t \leq z} [\text{primo}(t) \ \& \ t > y]$ (h è ricorsiva primitiva)
- In questo caso avremo y = x e z = x!+1 e quindi h(x, x!+1)
- Poniamo K(x) = h(x, x!+1) (K è ricorsiva primitiva poiché lo sono h, la somma e x!)
- La funzione P_n può quindi scriversi come:
  { $P_0 = 0$
  { $P_{n+1} = K(P_n)$

Ora siamo certi che P_n è ricorsiva primitiva.

Associamo a ciascun programma P del linguaggio S un numero che indichiamo con **#(P)**, in modo tale che il programma può essere integralmente ricostruito a partire da #(P).
Ordiniamo le variabili Y, X_1, Z_1, X_2, Z_2, ... e le etichette A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, ...
Scriviamo **#(V)**, **#(L)** per la posizione di una variabile o di un'etichetta data nell'ordinamento appropriato. Per esempio avremo $\#(X_2) = 4$; $\#(Z_1) = \#(Z) = 3$; $\#(E) = 5$; $\#(B_2) = 7$ ecc..
Sia I un'istruzione (etichettata o non etichettata) del linguaggio S. Scriviamo allora
#(I) = <a, <b, c>> dove:

a. Se I non è etichettata allora a = 0
   Se I è etichettata L allora a = #(L)
b. Se l'enunciato in I è V ← V allora b = 0
   Se l'enunciato in I è V ← V+1 allora b = 1
   Se l'enunciato in I è V ← V-1 allora b = 2
   Se l'enunciato in I è IF V ≠ 0 GOTO L' allora b = #(L') + 2
c. Se la variabile V compare in I allora c = #(V) – 1

Esempio: programma che calcola la funzione non definita in nessun punto

I_1 [A] X ← X + 1
I_2 IF X ≠ 0 GOTO A

Calcoliamo il numero di codice dell'istruzione I_1:
a = 1 perché l'etichetta [A] ha posto 1
b = 1 perché l'istruzione è del tipo X ← X + 1
c = 1 perché la variabile che compare è X che si trova al posto 2 (c = #([X]) – 1)
Quindi abbiamo che $\#(I_1) = <1, <1, 1>>$
Ricordando che $<x, y> = 2^x(2y+1)-1$ abbiamo che:
$<1, 1> = 2^1(2\cdot1 + 1)-1 = 5$
$<1, 5> = 2^1(2\cdot5 + 1)-1 = 21$
Quindi $\#(I_1) = 21$

Calcoliamo il numero di codice dell'istruzione I_2:
a = 0 perché I_2 non è etichettata
b = #([A]) + 2 = 3
c = #([X]) – 1 = 1
Quindi abbiamo che $\#(I_2) = <0, <3, 1>>$
$<3, 1> = 2^3(2\cdot1 + 1)-1 = 23$
$<0, 23> = 2^0(2\cdot23 + 1)-1 = 46$
Quindi $\#(I_2) = 46$

Il numero del programma è quindi dato da
$\#(P) = [\#(I_1), \#(I_2)] - 1 = [21, 46] - 1 = 2^{21} \cdot 3^{46} - 1$

Esempio: calcoliamo il numero di codice dell'istruzione non etichettata Y ← Y:
a = 0 perché I non è etichettata;
b = 0 perché I è del tipo V ← V;
c = 0 perché #(Y) – 1 = 1 – 1 = 0.
Quindi il numero associato all'istruzione Y ← Y è <0, <0, 0>> = 0.

L'unica ambiguità nelle codifiche di Gödel sono gli zeri alla fine: possiamo eliminare questa ambiguità decretando che nessun programma può terminare con l'istruzione Y ← Y.

Per ogni numero dato q vi è un'unica istruzione I con #(I) = q.
Per trovare l'etichetta che compare in I calcoliamo l(q):

- se l(q) = 0, I non è etichettata
- se l(q) = j, I ha la j-esima etichetta della nostra lista.

Per trovare la variabile che compare in I calcoliamo i = r(r(q)) + 1 e localizziamo l'i-esima variabile V nella nostra lista.
Per trovare l'enunciato che compare in I calcoliamo l(r(q)):

- se l(r(q)) = 0, allora l'enunciato sarà $V \leftarrow V$
- se l(r(q)) = 1, allora l'enunciato sarà $V \leftarrow V + 1$
- se l(r(q)) = 2, allora l'enunciato sarà $V \leftarrow V - 1$
- se l(r(q)) > 2, allora l'enunciato sarà IF $V \neq 0$ GOTO L
  dove L è la j-esima etichetta della nostra lista con j = l(r(q)) – 2

Sia P un programma formato dalle istruzioni $I_1, I_2, \dots, I_k$.
Poniamo allora $\#(P) = [\#(I_1)\ \#(I_2) \dots, \#(I_k)] - 1$.

Esempio di decodifica: #(P) = 199
Abbiamo $\#(P) = [I_1, \dots, I_n] - 1 = 199$ quindi $[I_1, \dots, I_n] = 200 = 2^3 \cdot 5^2$
Quindi abbiamo [3, 0, 2] = 200
Dobbiamo trovare le istruzioni i cui numeri sono rispettivamente 3, 0, 2:
$I_1 = 3 = <2, 0> = <2, <0, 0>>$
$I_2 = 0 = <0, 0> = <0, <0, 0>>$
$I_3 = 2 = <0, 1> = <0, <1, 0>>$
Si ha che:
I_1 [B] $Y \leftarrow Y$
I_2 $Y \leftarrow Y$
I_3 $Y \leftarrow Y + 1$
Calcola in modo non semplice la funzione y=1.

Dato un y, sia P il programma tale che #(P) = y, allora HALT(x, y) è vero se $\Psi_P^{(1)}(x)$ è definito, mentre è falso se $\Psi_P^{(1)}(x)$ non è definito, cioè HALT prende in input x e il numero y del programma P e ci dice se il programma P con input x si ferma oppure no.

TEOREMA DELLA FERMATA: HALT(x, y) non è un predicato calcolabile.
DIM (per assurdo): supponiamo che HALT(x, y) sia calcolabile, quindi esisterà un programma Q che lo calcola: [A] IF HALT(X, X) GOTO A. Q è la macro espansione di questo programma ed è stato costruito in modo tale che:
$\Psi_Q^{(1)}(x) = \{$ indefinito se HALT(x, x) è vero
$\{$ 0 se HALT(x, x) è falso
Supponiamo che $\#(Q) = y_0$. Usando la definizione del predicato HALT si ha che:
$HALT(x, y_0) \Leftrightarrow \sim HALT(x, x)$
Poiché questa equivalenza è vera per ogni x, possiamo porre $x = y_0$ e si ha:
$HALT(y_0, y_0) \Leftrightarrow \sim HALT(y_0, y_0)$
Questa è una contraddizione, quindi il teorema è dimostrato.

Questo teorema ci fornisce un esempio di funzione che non è calcolabile mediante nessun programma del linguaggio S. Inoltre dato un programma S ed un ingresso a tale programma, non esiste nessun algoritmo in grado di determinare se il programma dato si fermerà o no, prima o poi, sull'ingresso dato (*insolubilità del problema della fermata*).

CONGETTURA DI GOLDBACH
Ogni numero pari ≥ 4 è la somma di due numeri primi.
È facile verificarla per numeri piccoli per esempio: 4 = 2+2; 6 = 3+3; 20 = 7+13; 36 = 17+19…
È possibile scrivere un programma che verifichi la congettura cercando controesempi. Il programma si ferma quando ha trovato un n pari che non soddisfa la congettura.