

Elementi di Calcolabilità e Complessità

Ahmad Shatti

2020-2021

Indice

I Calcolabilità	4
1 Definizioni	4
2 Approcci alla calcolabilità	4
3 Teoremi sulle funzioni	7
4 Insieme ricorsivo e ricorsivamente enumerabili	11
4.1 Insieme speciale K	12
4.2 Problema della fermata	13
5 Riducibilità	13
6 Classificare R e RE	15
II Complessità	18
7 Misure di complessità deterministiche	18
7.1 Complessità in tempo deterministico	18
7.2 Macchine di Turing I/O	20
7.3 Complessità in spazio deterministico	21
8 Misure di complessità non deterministiche	22
8.1 Complessità in tempo e spazio non deterministicici	23
9 Funzioni di Misura	24
10 \mathcal{P} e \mathcal{NP}	26
10.1 Problemi interessanti e riduzioni efficienti tra essi	27
10.1.1 Problemi completi per \mathcal{P} e \mathcal{NP}	29
10.2 \mathcal{P} -completezza	30
10.3 \mathcal{NP} -completezza	31

Calcolabilità

Definizioni

1.1	Funzione totale	4
1.2	Convergenza e divergenza di una funzione	4
1.3	Funzioni calcolabili	4
2.1	Idea intuitiva di algoritmo	4
2.2	Macchina di Turing	5
2.3	T -calcolabilità	5
2.4	<i>WHILE</i> -calcolabilità	5
2.5	Funzioni ricorsive primitive	5
2.6	Relazione	6
2.7	Numerazione di Gödel	6
2.8	Funzione di Ackermann	6
2.9	Diagonalizzazione	6
2.10	Funzioni ricorsive generali	7
2.11	Tesi di Church-Turing	7
3.1	Enumerazione effettiva	7
4.1	Insieme ricorsivo	11
4.2	Insieme ricorsivamente enumerabile	11
4.3	Insieme K	12
4.4	Problema della fermata	13
4.5	Insieme K_0	13
5.1	Riducibilità	13
5.2	Classificazione	14
5.3	Problema arduo e completo	14
6.1	Riduzione REC	15
6.2	Insieme di indici che rappresentano le funzioni	16

Teoremi

3.1	Cardinalità funzioni calcolabili, esistenza funzioni non calcolabili	7
3.2	Padding lemma	8
3.3	Forma normale, Kleene 1	8
3.4	Equivalenza tra MdT e funzioni μ -ricorsive	8
3.5	Enumerazione	9
3.6	Del parametro $s - m - n$	9
3.7	Espressività	9
3.8	Ricorsione, Kleene 2	10
4.1	Equivalenza fra caratterizzazioni	12
6.1	Relazione di riduzione REC	15
6.2	K è RE -completo	15
6.3	Pre-Rice	16
6.4	Rice	17

Complessità

Definizioni

7.1	Macchine di Turing a k nastri	18
7.2	Tempo richiesto esatto	18
7.3	Tempo richiesto approssimato	18
7.4	Classe di complessità in tempo deterministico	19
7.5	Classe \mathcal{P}	20
7.6	Macchine di Turing I/O	20
7.7	Spazio richiesto MdT I/O	21
7.8	PSPACE	21
7.9	LOGSPACE	21
8.1	Macchina di Turing non deterministica	22
8.2	Accettazione MdT non deterministica	22
8.3	Tempo non deterministico	23
8.4	NTIME	23
8.5	Classe \mathcal{NP}	23
8.6	Spazio non deterministico	24
8.7	NPSPACE	24
9.1	Funzioni di misura	24
9.2	\mathbf{EXP}	25
9.3	Axiomi di Blum	26
10.1	Tesi di Cook-Karp	26
10.2	Riduzione efficiente	26
10.3	Problema SAT	27
10.4	Problema HAM	27
10.5	Problema CRICCA	28
10.6	Circuito Booleano	28
10.7	Problema CIRCUIT SAT	29
10.8	Problema CIRCUIT VALUE	29
10.9	Tabella di computazione di una MdT	30
10.10	Problema MONOTONE CIRCUIT VALUE	31

Teoremi

7.1	Riduzione del numero dei nastri	19
7.2	Accelerazione lineare MdT	20
7.3	Compressione lineare dello spazio	21
7.4	Gerarchia classi spazio deterministico	22
8.1	Simulazione non deterministica	23
8.2	Savitch	24
9.1	Gerarchia	25
9.2	Gerarchia	25
9.3	No richiesta di funzione appropriata	25
9.4	Accelerazione, Blum	25
9.5	Della lacuna, Borodin	26
10.1	LOGSPACE classifica \mathcal{P} e \mathcal{NP}	26
10.2	Riduzioni in spazio logaritmico	30
10.3	CIRCUIT VALUE è \mathcal{P} -completo	30
10.4	MONOTONE CIRCUIT VALUE è \mathcal{P} -completo	31
10.5	Cook	31

Parte I

Calcolabilità

1 Definizioni

Definizione 1.1: Funzione totale

Una funzione $f : A \rightarrow B$, sottoinsieme di $A \times B$ è una *funzione totale* se e solamente se

- i) $\forall a \in A, \exists b \in B : (a, b) \in f$ (la funzione è definita ovunque)
- ii) $(a, b), (a, c) \in f \Rightarrow b = c$ (unicità)

Se non vale il primo punto allora f è una **funzione parziale**.

Esempio: $doppio : \mathbb{N} \rightarrow \mathbb{N}$ è definita come l'insieme delle coppie $doppio = \{(0, 0), (1, 2), (2, 4), \dots\}$ da cui sappiamo per esempio che $doppio(5) = 10$

Definizione 1.2: Convergenza e divergenza di una funzione

Data una funzione $f : A \rightarrow B$ diremo che

- i) f converge su a (in simboli $f(a) \downarrow$) se $\exists b : (a, b) \in f$ cioè $f(a) = b$
- ii) f diverge su a (in simboli $f(a) \uparrow$) se $\nexists b : (a, b) \in f$

Definizione 1.3: Funzioni calcolabili

Una funzione è *calcolabile* se esiste un algoritmo che la calcola.

Una funzione f è un insieme di coppie, cioè f associa l'argomento con il risultato ($doppio(n) = n + n$) senza dire *come fare a calcolarlo*. Di conseguenza, in termini insiemistici non esistono due insiemi diversi che hanno gli stessi elementi. Un algoritmo (quando c'è) invece specifica *come si calcola* il risultato a partire dall'argomento. Quindi un algoritmo rappresenta in modo finito una funzione. È facile vedere che ci possono essere più algoritmi che calcolano la stessa funzione.

2 Approcci alla calcolabilità

Definizione 2.1: Idea intuitiva di algoritmo

Ci sono moltissimi formalismi che sono stati proposti per esprimere algoritmi. In ciascuno di questi gli algoritmi devono soddisfare i seguenti requisiti:

- I) un algoritmo è costituito da un insieme *finito* di istruzioni;
- II) le istruzioni sono in numero *finito*, su dati *discreti* e *finiti*, e hanno un effetto *limitato*;
- III) una computazione è una successione di passi *discreti*, senza ricorrere a sistemi analogici o metodi continui, ciascuno dei quali impiega un tempo *finito*;
- IV) ogni passo dipende *solo* dai precedenti e da una porzione *finita* dei dati, in modo *deterministico*;
- V) *non c'è limite al numero di passi* necessari all'esecuzione di un algoritmo, né alla *memoria* richiesta per contenere i dati iniziali, intermedi ed eventualmente finali.

Definizione 2.2: Macchina di Turing

Una macchina di Turing M è una quadrupla (Q, Σ, δ, q_0) dove:

- Q è l'insieme finito degli stati. Si indica con lo stato speciale $h \notin Q$ la fine corretta della computazione;
- $q_0 \in Q$ è lo stato iniziale;
- Σ è l'insieme finito dell'alfabeto, con $\# \in \Sigma$ il carattere bianco, $\triangleright \in \Sigma$ la marca di inizio stringa e $L, R, - \notin \Sigma$ per lo spostamento del nastro;
- $\delta \subseteq (Q \times \Sigma) \times (Q \cup \{h\})$ è la relazione di transizione tale che

$$\text{se } (q, \triangleright, q', \sigma, D) \in \delta \text{ allora } \sigma = \triangleright, D = R$$

Questa condizione dice che il carattere corrente, ovvero il cursore, non può mai trovarsi a sinistra della marca di inizio stringa \triangleright .

Definizione 2.3: T -calcolabilità

Dati Σ alfabeto di M , Σ_0 alfabeto di input e Σ_1 alfabeto di output con $\#, \triangleright \notin \Sigma_0 \cup \Sigma_1$, $\Sigma_0 \cup \Sigma_1 \subset \Sigma$ e $f : \Sigma_0^* \rightarrow \Sigma_1^*$ una funzione. Allora si dice che una MdT $M = (Q, \Sigma, \delta, q_0)$ è T -calcolabile se e solo se

$$\forall w \in \Sigma_0^* : f(w) = w' \text{ se e solamente se } M(w) \xrightarrow{*} (h, \triangleright w')$$

Intuizione: cioè esiste una MdT che per ogni stringa finita w in input arriva, con un numero finito di passi, all'arresto lasciando sul nastro la stringa di output corretta. Quindi f è T -calcolabile se e solo se $\exists M$ che la calcola.

Definizione 2.4: WHILE-calcolabilità

f è WHILE-calcolabile se e solamente se

$$\forall \sigma \in (Var \rightarrow \mathbb{N}) : f(x) = n \text{ se e solamente se } \langle C, \sigma \rangle \xrightarrow{*} \sigma' \text{ e } \sigma'(x) = n$$

Intuizione: cioè per ogni possibile memoria σ , $f(x) = n$ se e solamente se l'esecuzione di C con la memoria σ porta ad una nuova memoria σ' e $\sigma'(x) = n$. Quindi f è WHILE-calcolabile se esiste una computazione che la calcola.

Definizione 2.5: Funzioni ricorsive primitive

La classe \mathcal{C} delle funzioni ricorsive primitive è la minima classe che obbediscono alle seguenti regole di inferenza, regole di sintassi per definire le funzioni.

Casi base

- I **Zero**: $\lambda x_1, \dots, x_k. 0$ con $k \geq 0$
prende un vettore di argomenti e restituisce 0
- II **Successore**: $\lambda x. x + 1$
prende un valore e restituisce il suo successore
- III **Identità**: $\lambda x_1, \dots, x_k. x_i$ con $1 \leq i \leq k$
dato un vettore di k variabili, restituisce l' i -esimo

Casi iterativi

- IV **Composizione**: se $g_1, \dots, g_k \in \mathcal{C}$ sono funzioni in m variabili ed $h \in \mathcal{C}$ è una funzione in k variabili, allora appartiene a \mathcal{C} anche la loro composizione $\lambda x_1, \dots, x_m. h(g_1(x_1, \dots, x_m), \dots, g_k(x_1, \dots, x_m))$
- V **Ricorsione primitiva**: se $h \in \mathcal{C}$ è una funzione in $k+1$ variabili, $g \in \mathcal{C}$ è una funzione in $k-1$ variabili, allora appartiene a \mathcal{C} anche la funzione f in k variabili definita da

$$\begin{cases} f(0, x_2, \dots, x_k) = g(x_2, \dots, x_k) \\ f(x_1 + 1, x_2, \dots, x_k) = h(x_1, f(x_1, x_2, \dots, x_k), x_2, \dots, x_k) \end{cases}$$

Definizione 2.6: Relazione

La relazione $P(x_1, \dots, x_k) \subseteq \mathbb{N}^k$ è ricorsiva primitiva se lo è la sua funzione caratteristica χ_P definita come:

$$\chi_P(x_1, \dots, x_k) = \begin{cases} 1 & \text{se } (x_1, \dots, x_k) \in P \\ 0 & \text{se } (x_1, \dots, x_k) \notin P \end{cases}$$

Esempi di funzioni ricorsive primitive notevoli:

- $R = \{x \in \mathbb{N} \mid x \text{ numero primo}\}$ è ricorsiva primitiva;
- **unica fattorizzazione**, se $p_0 < \dots < p_k \dots$ sono i numeri primi, cioè $R = \{p_0, \dots, p_k \dots\}$ allora per ogni $x \in \mathbb{N}$ esiste un numero *finito* di esponenti $x_i \neq 0$ tali che $x = p_0^{x_0} p_1^{x_1} \dots p_n^{x_n} \dots$;
- la funzione $(x)_i$ che restituisce l'esponente dell' i -esimo fattore p_i della fattorizzazione di x è ricorsiva primitiva.

Definizione 2.7: Numerazione di Gödel

La conseguenza dell'unica fattorizzazione è la *Gödelizzazione*. Ogni sequenza di numeri naturali può essere codificata univocamente come un singolo numero $n = p_0^{n_0+1} p_1^{n_1+1} \dots p_k^{n_k+1}$ (con p_i primi), ovvero come il prodotto di un numero *finito* di fattori e viceversa. In altre parole, data la sequenza $n_0 n_1 \dots n_k$ esiste un unico n che verifica l'uguaglianza $n = p_0^{n_0+1} + p_1^{n_1+1} \dots p_k^{n_k+1}$ e viceversa. Quindi è possibile rappresentare algoritmi interi come un numero.

Intuizione: si può associare ad ogni macchina di Turing un numero e le posso enumerare con un indice i . Stesso procedimento anche per le computazioni, configurazioni...

Tutte le funzioni definibili mediante gli schemi di ricorsione primitiva sono totali, e data una funzione ricorsiva primitiva si può scrivere un programma *FOR* che dati gli stessi argomenti produce lo stesso risultato e viceversa. Quindi si è trovato il formalismo definitivo che esprime tutte le funzioni calcolabili? NO perché esiste la *funzione di Ackermann*.

Definizione 2.8: Funzione di Ackermann

La funzione di Ackermann non è *definibile* mediante gli schemi di ricorsione primitiva I-V ma è totale.

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

Intuizione: la funzione di Ackermann è una funzione totale non ricorsiva primitiva. Otteniamo quindi che non esiste un formalismo capace di esprimere tutte le funzioni totali.

Definizione 2.9: Diagonalizzazione

Qualunque formalismo esprime solo funzioni totali ma non tutte, oppure, esprime anche funzioni parziali.

Intuizione: siamo obbligati a considerare anche le funzioni parziali che vengono indicate con φ e ψ . Non si applica la diagonalizzazione ad esse perché sia ψ_n la funzione con n -esima definizione e proviamo a diagonalizzare

$$\varphi(x) = \psi_x(x) + 1$$

diciamo che $\psi(x)$ ha indice i , quindi $\psi_i(x) = \varphi(x) = \psi_x(x) + 1$. Se $\psi_i(x)$ diverge allora $\psi_x(x)$ diverge e allora anche $\psi_x(x) + 1$ diverge. Quindi il discorso si applica ad ogni formalismo che definisca solo funzioni totali. Inoltre non si possono estendere tutte le funzioni parziali a funzioni totali perché non sempre c'è un algoritmo che calcola la funzione estesa.

Dimostrazione: sia A algoritmo finito e un alfabeto con $\#$ finito di simboli, allora

- si possono enumerare le funzioni f_n (ad esempio con la funzione di Gödel)
- si definisce $g(x) = f_x(x) + 1$, stesso x sia per indice che per parametro (g è totale)
- se cerco g nella lista delle funzioni ricorsive primitive, non la trovo perché $\forall n. g(n) \neq f_n(n)$ quindi $\forall n. g \neq f_n$, o meglio la funzione calcolata dalla g su n restituisce un valore diverso dalla funzione calcolata dalla f_n su n

Notazione: l'operatore μ , detto di *minimizzazione*, se applicato ad un insieme di numeri naturali ne restituisce il minimo.

Definizione 2.10: Funzioni ricorsive generali

La classe delle funzioni ricorsive generali è la minima classe \mathcal{R} tale che soddisfa le condizioni

- I-V per le ricorsive primitive
- VI (*minimizzazione*), se $\varphi(x_1, \dots, x_n, y) \in \mathcal{R}$ in $n+1$ variabili allora la funzione ψ in n variabili è in \mathcal{R} se è definita da

$$\psi(x_1, \dots, x_n) = \mu y [\varphi(x_1, \dots, x_n, y) = 0 \quad \text{e} \quad \forall z \leq y. \varphi(x_1, \dots, x_n, z) \downarrow]$$

Intuizione: si arricchiscono gli schemi per la definizione delle funzioni ricorsive primitive con un nuovo schema, attraverso il quale è possibile anche esprimere funzioni parziali.

Definizione 2.11: Tesi di Church-Turing

Le funzioni (*intuitivamente*) calcolabili sono tutte e sole le funzioni (parziali) T-calcolabili.

Intuizione: si possono chiamare *calcolabili* indifferentemente le funzioni esprimibili nel formalismo delle macchine di Turing o le funzioni μ -ricorsive o i programmi WHILE, ... purché la definizione rispetti le cinque condizioni intuitive poste agli algoritmi

3 Teoremi sulle funzioni

Definizione 3.1: Enumerazione effettiva

Una *enumerazione effettiva* è una funzione biunivoca che prende un algoritmo e restituisce un numero, e dipende solo dalla sintassi con cui scriviamo gli algoritmi e non dal significato che attribuiamo loro.

I seguenti risultati sono tutti invarianti rispetto all'enumerazione scelta.

Teorema 3.1: Cardinalità funzioni calcolabili, esistenza funzioni non calcolabili

- i) le funzioni calcolabili sono $\#(\mathbb{N})$; inoltre anche le funzioni calcolabili totali sono $\#(\mathbb{N})$;
- ii) esistono funzioni non calcolabili.

Dimostrazione:

- i) costruisci $\#(\mathbb{N})$ MdT M_i che svuotano il nastro dell'input, ci scrivono la stringa $|^i$ e si arrestano (sono tutte funzioni costanti). Che non siano più di $\#(\mathbb{N})$ segue dal fatto che le MdT si possono enumerare;
- ii) con una costruzione analoga a quella di Cantor si vede che $\{f : \mathbb{N} \rightarrow \mathbb{N}\}$ ha cardinalità $2^{\#(\mathbb{N})}$ e quindi dato che le funzioni calcolabili sono $\#(\mathbb{N})$, abbiamo dimostrato l'esistenza di funzioni non calcolabili.

Teorema 3.2: Padding lemma

Ogni funzione calcolabile φ_x ha $\#(\mathbb{N})$ indici. Inoltre $\forall x$ si può costruire, mediante una funzione ricorsiva primitiva, un insieme infinito A_x di indici tale che

$$\forall y \in A_x. \varphi_y = \varphi_x$$

cioè $\varphi_y(n) = m$ se e solo se $\varphi_x(n) = m$ (e ovviamente $\varphi_y(n) \uparrow$ se e solo se $\varphi_x(n) \uparrow$).

Intuizione: ci sono infinite numerabili MdT, ovvero infiniti numerabili algoritmi, che calcolano la stessa funzione. In altre parole la semantica della macchina o del programma non è cambiata, ma è cambiata la sua sintassi.

Dimostrazione: sia M_i un programma P . Prendo $P; \text{skip}$ poi $P; \text{skip}; \text{skip} \dots$ metto tanti $; \text{skip}$ quanti voglio. Si può generare un numero infinito di programmi che calcolano tutti la stessa funzione.

Teorema 3.3: Forma normale, Kleene 1

Esistono un predicato $T(i, x, y)$ e una funzione $U(y)$ calcolabili totali tali che $\forall i, x, \varphi_i(x) = U(\mu y. T(i, x, y))$. Inoltre T e U sono primitive ricorsive.

Intuizione: tra tutti gli algoritmi che calcolano una data funzione ce n'è uno privilegiato, nel senso che ha una forma speciale. Di conseguenza, ogni funzione ha una rappresentazione privilegiata.

Dimostrazione:

- sia $T(i, x, y)$ detto **predicato di Kleene**, vero se e solamente se y è la codifica di una computazione terminante di M_i con dato iniziale x . Il calcolo di T avviene tramite i seguenti passi:
 1. dato i recupera la macchina M_i
 2. decodifica y
 3. dato x si controlla se il risultato è una computazione terminante della forma $M_i(x) = c_0, \dots, c_n$questa sequenza di passi termina sempre quindi T è totale.
- se y è la codifica di una computazione terminante di $M_i(x)$ allora $c_n = (h, \triangleright z \#)$ e si definisce $U(y) = z$. I passi necessari a questo calcolo sono finiti e terminano tutti quindi U è totale.

Inoltre U e T sono ricorsivi primitivi perché le codifiche che usiamo lo sono e perché lo sono i controlli effettuati.

Dato il teorema di forma normale si possono dedurre i seguenti risultati:

Corollario: le funzioni T -calcolabili sono μ -ricorsive.

Lemma: le funzioni μ -calcolabili sono T -calcolabili.

Teorema 3.4: Equivalenza tra MdT e funzioni μ -ricorsive

Una funzione è T -calcolabile se e solamente se è μ -calcolabile.

Quest'ultimo teorema e quello di forma normale ha il seguente interessante corollario:

Corollario: ogni funzione calcolabile parziale può essere ottenuta da due funzioni primitive ricorsive e una sola applicazione dell'operatore μ .

Teorema 3.5: Enumerazione

Esiste una funzione calcolabile parziale $\varphi_z(i, x)$ tale che $\varphi_z(i, x) = \varphi_i(x)$.

Intuizione: un formalismo universale, cioè uno che esprima tutte le funzioni calcolabili, è così potente da riuscire ad esprimere l'interprete dei propri programmi.

Dimostrazione: la funzione $U(\mu y.T(i, x, y))$ usata nel teorema di forma normale è definita per composizione e μ -ricorsione a partire da funzioni primitive ricorsive, quindi è essa stessa una funzione calcolabile in due argomenti i e x . Avrà allora un indice che chiameremo z , cioè $\varphi_z(i, x) = U(\mu y.T(i, x, y))$. Applichiamo allora il teorema di forma normale, da cui $U(\mu y.T(i, x, y)) = \varphi_i(x)$. Per transitività dell'uguaglianza ottengo la tesi $\varphi_z(i, x) = \varphi_i(x)$.

Teorema 3.6: Del parametro $s - m - n$

- i) caso $m = n = 1$, esiste una funzione s calcolabile totale (iniettiva) s_1^1 con due argomenti, tale che $\forall x, y$

$$\lambda y. \varphi_i(x, y) = \varphi_{s(i, x)}(y)$$

- ii) caso generale, $\forall m, n \geq 0$ esiste una funzione calcolabile totale (iniettiva) s_n^m con $m + 1$ argomenti tale che $\forall x, y_1, \dots, y_m$

$$\varphi_{s_n^m}(x, y_1, \dots, y_m) = \lambda z_1, \dots, z_n. \varphi_x^{(m+n)}(y_1, \dots, y_m, z_1, \dots, z_n)$$

Intuizione: considerando x come parametro, la funzione può essere calcolata da un'altra macchina di indice dipendente da i e x . Intuitivamente, il programma $P_{s(i, x)}$ opera su y soltanto mentre P_i opera su x e y . Quindi x è un parametro di P_i .

Dimostrazione: per calcolare $\varphi_{s_1^1(i, x)}(y)$ trova M_i grazie al teorema di enumerazione e predisponi lo stato iniziale di $M_i(x, y)$ con x fissato in anticipo. L'algoritmo delineato, per la tesi di Church-Turing, calcola una funzione che è proprio la funzione $s = s_1^1$ che stavamo cercando. Ci manca da dimostrare che la s è iniettiva. Basta costruire s strettamente crescente e questo è possibile perché ci sono $\#(\mathbb{N})$ MdT che calcolano $s(i, x)$. Quindi s deve essere tale che se la codifica delle coppie $(i_0, x_0) < (i_1, x_1) \Rightarrow s(i_0, x_0) < s(i_1, x_1)$.

Teorema 3.7: Espressività

Un formalismo è Turing-equivalente (calcola tutte e sole le funzioni T -calcolabili, è universale) se e solamente se:

- ha un algoritmo universale (cioè vale il teorema di enumerazione);
- vale il teorema del parametro.

Teorema 3.8: Ricorsione, Kleene 2

$\forall f$ funzione calcolabile totale, esiste un indice n , tale che $\varphi_n = \varphi_{f(n)}$. L'indice n viene detto punto fisso di f .

Intuizione: posso trasformare il mio programma in maniera da ottenerne uno perfettamente equivalente: la funzione f "trasforma" programmi in programmi. Infatti f trasforma indici: dato n , ovvero il programma P_n lo trasforma in $P_{f(n)}$. Considerando il punto fisso, la trasformazione di f non cambia la funzione calcolata, cioè P_n e $P_{f(n)}$ sono equivalenti con la stessa semantica.

Dimostrazione:

1. si definisce la seguente funzione calcolabile "diagonale"

$$\psi(u, z) = \begin{cases} \varphi_{\varphi_u(u)}(z) & \text{se } \varphi_u(u) \downarrow \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

cioè da $\psi(u, z)$ si prende il primo argomento u , si individua l' u -esima macchina, la si applica ad u e si controlla se converge. Se converge abbiamo l'indice di $\varphi_{\varphi_u(u)}(z)$.

Quindi $\psi(u, z)$ è calcolabile, allora si può applicare la tesi di Church-Turing e quindi sappiamo che esiste un indice i cioè $\varphi_i(u, z)$. Adesso si applica il teorema del parametro:

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

Ci sono infiniti indici per ψ allora ne prendiamo uno, quindi la funzione $s(i, u)$ dipende solo da u e fissiamo l'argomento i . Questa funzione che dipende solo da u la si chiama d , cioè abbiamo $d(u) = \lambda u. s(i, u)$. La funzione d è calcolabile totale e iniettiva, allora

$$\varphi_{s(i,u)}(z) = \varphi_i(u, z) = \psi(u, z) = \begin{cases} \varphi_{\varphi_u(u)}(z) & \text{se } \varphi_u(u) \downarrow \\ \text{indefinita} & \text{altrimenti} \end{cases} = \varphi_{d(u)}(z)$$

2. data la d la si compone con f cioè $f(d(x))$ e dato che sia f che d sono totali allora lo sarà anche la loro composizione. Essendo calcolabile avrà un indice v quindi $\varphi_v(x) = f(d(x))$
3. essendo non solo calcolabile ma anche totale allora se si prende v e la si applicasse v convergerebbe cioè $\varphi_v(v) \downarrow$ allora per il punto 1. abbiamo $\varphi_v(v) \downarrow \Rightarrow \varphi_{d(v)}(z) = \varphi_{\varphi_v(v)}(z)$. Adesso ponendo $n = d(v)$ possiamo dimostrare che n è punto fisso di f (cioè $\varphi_n = \varphi_{f(n)}$)

$$\varphi_n \stackrel{3.}{=} \varphi_{d(v)} \stackrel{1.}{=} \varphi_{\varphi_v(v)} \stackrel{2.}{=} \varphi_{f(d(v))} \stackrel{3.}{=} \varphi_{f(n)}$$

Proprietà 3.1: Teorema di ricorsione

Nelle ipotesi del teorema di ricorsione:

- i) il punto fisso è calcolabile mediante una funzione totale (iniettiva) g a partire dall'indice di f .
- ii) ci sono $\#(\mathbb{N})$ punti fissi di f .

Dimostrazione:

- i) prendo $h(x)$ calcolabile totale tale che $\forall n, \varphi_{h(x)}(n) = \varphi_x(d(n))$. Allora $g(x) = d(h(x))$.
- ii) sappiamo che ogni funzione calcolabile totale ha $\#(\mathbb{N})$ indici. Segue che se ho un punto fisso di f posso trovare $\#(\mathbb{N})$ funzioni calcolabili totali equivalenti.

4 Insieme ricorsivo e ricorsivamente enumerabili

Finora abbiamo considerato il calcolo di funzioni come unico modo di risolvere problemi. Ora passiamo a studiare i problemi di appartenenza o di non appartenenza di un elemento ad un dato insieme.

Definizione 4.1: Insieme ricorsivo

Un insieme I è *ricorsivo* (o *decidibile*) se e solo se la sua funzione caratteristica

$$\chi_I(x) = \begin{cases} 1 & \text{se } x \in I \\ 0 & \text{se } x \notin I \end{cases} \quad \text{è calcolabile totale.}$$

Intuizione: un insieme è ricorsivo se è possibile costruire un algoritmo che dato x , determina se $x \in I$ in un numero finito di passi.

Se non poniamo alcun limite al numero dei passi consentiti a una macchina, otteniamo un'altra classe di insiemi più ampia della precedente:

Definizione 4.2: Insieme ricorsivamente enumerabile

Un insieme I è *ricorsivamente enumerabile* (o *semi-decidibile*) se e solo se $\exists i. I = \text{dom}(\varphi_i)$.

Intuizione: un insieme S è ricorsivamente enumerabile se esiste un algoritmo (o una funzione ricorsiva totale) che dato x , termina se $x \in S$; oppure, equivalentemente genera in output tutti e soli gli elementi di S . È quindi il dominio di una funzione calcolabile (il più delle volte *parziale*, infatti se fosse totale $I = \mathbb{N}$)

Proprietà 4.1: Insiemi ricorsivi e ricorsivamente enumerabili

- se I è ricorsivo allora I è ricorsivamente enumerabile (non è vero il contrario);
- I, \bar{I} sono ricorsivamente enumerabili se e solo se I (e \bar{I}) sono ricorsivi.

Dimostrazione:

- sapendo che I è ricorsivo sappiamo che ha una funzione caratteristica calcolabile totale

$$\chi_I(x) = \begin{cases} 1 & \text{se } x \in I \\ 0 & \text{se } x \notin I \end{cases}$$

Dobbiamo trovare $i. I = \text{dom}(\varphi_i)$ cioè una funzione calcolabile il cui dominio è proprio I . Possiamo definire questa:

$$\varphi_i = \begin{cases} 1 & \text{se } \chi_I(x) = 1 \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

il cui dominio è I .

- devo trovare la funzione caratteristica di χ_I . Considero φ_i e $\varphi_{\bar{i}}$ le funzioni i cui domini sono rispettivamente I e \bar{I} . Per cui dato un x , una delle due funzioni deve convergere. Si esegue il seguente ciclo:

- esegui un passo $\varphi_i(x)$; se $\varphi_i(x) \downarrow$ allora $x \in I$ e ponì $\chi_I(x) = 1$
- altrimenti esegui un passo nel calcolo di $\varphi_{\bar{i}}$; se $\varphi_{\bar{i}}(x) \downarrow$ allora $x \notin I$ e ponì $\chi_I(x) = 0$

Si nota quindi che gli insiemi ricorsivi sono inclusi negli insiemi ricorsivamente enumerabili.

Teorema 4.1: Equivalenza fra caratterizzazioni

I è ricorsivamente enumerabile se e solamente se è vuoto oppure è l'immagine di una funzione calcolabile totale.

Dimostrazione:

- i) nel caso $I = \emptyset$, è banale, la funzione caratteristica da sempre dà sempre 0;
- ii) nel caso $I \neq \emptyset$, si costruisce una funzione calcolabile totale f tale che $I = \text{Imm}(f)$. Sappiamo per definizione di insieme ricorsivamente enumerabile che $I = \text{dom}(\varphi_i)$.
 - (a) per la costruzione della funzione f usiamo φ_i e troviamo un $\bar{n} \in I$ tale che $\varphi_i(\bar{n}) \downarrow$. Possiamo trovare questo \bar{n} mediante un procedimento a *coda di colomba* in cui l'indice di riga m rappresenta il numero di passi del calcolo di φ_i e l'indice di colonna n il suo argomento. Quando la macchina converge, ovvero $\varphi_i(\bar{n}) \downarrow$, abbiamo trovato che $\bar{n} \in I$.
 - (b) a questo punto si inizia un secondo procedimento a coda di colomba eseguendo $\varphi_i(n)$ per m passi, ma stavolta, se tale calcolo si arresta, poniamo $f(\langle n, m \rangle) = n$, altrimenti si pone $f(\langle n, m \rangle) = \bar{n}$ (valore che già si conosce). Si itera il procedimento incrementando la codifica $\langle n, m \rangle$, ovvero considerano $\langle n, m \rangle + 1$. Si generano così tutti gli elementi di I .

4.1 Insieme speciale K

Definizione 4.3: Insieme K

$$K = \{x \mid \varphi_x(x) \downarrow\}.$$

Intuizione: è l'insieme degli algoritmi applicati a se stessi e convergenti.

Proprietà 4.2: Insieme K

K è ricorsivamente enumerabile.

Dimostrazione: K è il dominio della funzione calcolabile parziale

$$\psi(x) = \begin{cases} x & \text{se } \varphi_x(x) \downarrow \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile perché posso prendere la MdT M_x e applicarla ad x ; se e quando si arresta, restituisci x .

Proprietà 4.3: Insieme K

K non è ricorsivo.

Intuizione: non esiste un algoritmo per decidere se $x \in K$ o no. Quindi questo problema è insolubile anche se è semidecidibile

Dimostrazione: sia χ_K la funzione caratteristica di K e per assurdo sia totale e calcolabile. Ma allora anche la seguente funzione

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{se } x \in K \\ 0 & \text{se } x \notin K \end{cases}$$

sarebbe calcolabile totale. Tuttavia si ottiene una contraddizione perché $\forall x. f(x) \neq \varphi_x(x)$; quindi non troviamo alcun indice per f che di conseguenza non è calcolabile.

Inoltre \overline{K} non è ricorsivamente enumerabile, infatti se \overline{K} fosse ricorsivamente enumerabile, sia K che \overline{K} sarebbero ricorsivi per la proprietà 4.1. Abbiamo così stabilito un piccolo frammento di gerarchia:

$$R \subsetneq RE \subsetneq \text{non}RE$$

dove R è la classe degli insiemi ricorsivi; RE quella degli insiemi ricorsivamente enumerabili e $\text{non}RE$ (o $\text{co}RE$) quella degli insiemi non ricorsivamente enumerabili.

4.2 Problema della fermata

Definizione 4.4: Problema della fermata

Dati $x, y. \varphi_y(x) \downarrow$? cioè il programma $P_y(x)$ si ferma?

Questo problema si formalizza e si studia introducendo un altro insieme:

Definizione 4.5: Insieme K_0

$$K_0 = \{(x, y) \mid \varphi_y(x) \downarrow\} \text{ ovvero } K_0 = \{(x, y) \mid \exists z. T(y, x, z)\}.$$

Proprietà 4.4: Insieme K_0

K_0 non è ricorsivo.

Dimostrazione: si ha che $x \in K$ se e solamente se $(x, x) \in K_0$ quindi se K_0 fosse ricorsivo lo sarebbe anche K ma abbiamo già dimostrato che non lo è.

5 Riducibilità

Definizione 5.1: Riducibilità

A si riduce a B secondo la riduzione f , in simboli $A \leq_f B$, tutte e sole le volte che

$$a \in A \text{ se e solamente se } f(a) \in B, \text{ ovvero, } f(A) \subseteq B \text{ e } f(\overline{A}) \subseteq \overline{B}$$

Intuizione: è una particolare funzione f che trasforma un problema (ovvero un insieme o una classe) A in un altro problema B , in modo da mantenerne inalterata la caratteristica principale.

Proprietà 5.1: Riducibilità

$$A \leq_f B \text{ se e solamente se } \overline{A} \leq_f \overline{B}.$$

Dimostrazione: si ha che $x \in \overline{A}$ se e solamente se $x \notin A$ se e solamente se $f(x) \notin B$ se e solamente se $f(x) \in \overline{B}$.

Più in generale, si definisce una **relazione di riduzioni** (\leq_F) dove F è una particolare classe di funzioni. Allora scriveremo:

$$A \leq_F B \text{ se e solamente se esiste } f \in F \text{ tale che } A \leq_f B$$

Definizione 5.2: Classificazione

Siano \mathcal{D} ed \mathcal{E} due classi di problemi con $\mathcal{D} \subseteq \mathcal{E}$. Una relazione di riduzioni \leq_F classifica \mathcal{D} ed \mathcal{E} se e solo se per ogni problema A, B, C .

- i) $A \leq_F A$ (Riflessiva)
- ii) $A \leq_F B, B \leq_F C$ implica $A \leq_F C$ (Transitiva)
- iii) $A \leq_F B, B \in \mathcal{D}$ implica $A \in \mathcal{D}$ (\mathcal{D} ideale = chiuso all'ingiù per riduzione)
- iv) $A \leq_F B, B \in \mathcal{E}$ implica $A \in \mathcal{E}$ (\mathcal{E} ideale = chiuso all'ingiù per riduzione)

Lemma: una relazione di riduzione \leq_F classifica \mathcal{D} ed \mathcal{E} , tali che $\mathcal{D} \subseteq \mathcal{E}$, se e solo se

- i) $id \in F$ (F ha identità)
- ii) $f, g \in F \Rightarrow f \circ g \in F$ (F chiusa per composizione)
- iii) $f \in F, B \in \mathcal{D} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{D}$
- iv) $f \in F, B \in \mathcal{E} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{E}$

Per la dimostrazione vedi punto per punto la definizione di classificazione.

Definizione 5.3: Problema arduo e completo

Se \leq_F classifica \mathcal{D} ed \mathcal{E} , $\forall A, B, H$ problemi

- i) $A \equiv B$ se $A \leq_F B$ e $B \leq_F A$ (A e B hanno lo stesso grado se si riducono l'uno all'altro);
- ii) H è $\leq_{F-arduo}$ per \mathcal{E} se $\forall A \in \mathcal{E}. A \leq_F H$;
- iii) H è $\leq_{F-completo}$ per \mathcal{E} se H è $\leq_{F-arduo}$ per \mathcal{E} e $H \in \mathcal{E}$.

Proprietà 5.2: Proprietà completezza

Se \leq_F classifica \mathcal{D} ed \mathcal{E} , $\mathcal{D} \subseteq \mathcal{E}$ e C è completo per \mathcal{E} , allora $C \in \mathcal{D}$ se e solamente se $\mathcal{D} = \mathcal{E}$.

Intuizione: se un problema è completo per una classe \mathcal{E} e appartiene ad una sottoclasse \mathcal{D} , allora le due classi coincidono.

Dimostrazione: sia $C \in \mathcal{D}$ e $A \in \mathcal{E}$. Per completezza $A \leq_F C$ e $A \in \mathcal{D}$ per la condizione (iii) del lemma. Quindi $\mathcal{E} \subseteq \mathcal{D}$ e la tesi.

Proprietà 5.3: Proprietà completezza

Se A è completo per \mathcal{E} , $A \leq_F B$, e $B \in \mathcal{E}$, allora B è completo per \mathcal{E} .

Intuizione: se un problema è completo e questo si riduce ad un altro problema allora anche quest'ultimo è completo.

Dimostrazione: $\forall D \in \mathcal{E}, D \leq_F A$ per completezza, ma \leq_F -classifica D ed \mathcal{E} e allora $D \leq_F A$ e $A \leq_F B$ implicano $D \leq_F B$ e quindi B è arduo e, poiché appartiene a \mathcal{E} , è completo.

Un problema completo per \mathcal{E} "rappresenta la difficoltà" massima dei problemi di \mathcal{E} . Infatti è facile vedere che il grado di un problema A completo per \mathcal{E} è il grado massimo di \mathcal{E} in \leq_F . Infatti valgono le seguenti affermazioni:

- se $B \leq_F A$ allora B al più ha il grado di A , cioè è più facile o altrettanto difficile di A ;
- se $A \leq_F B$ allora B ha almeno il grado di A , cioè è di difficoltà maggiore o uguale a quella di A .

6 Classificare R e RE

Da qui in avanti $REC = \{\varphi_x \mid \forall y \in \mathbb{N}. \varphi_x(y) \downarrow\}$ sarà il nome dell'insieme delle funzioni calcolabili totali.

Definizione 6.1: Riduzione REC

A è riducibile a B ($A \leqslant_{REC} B$) se e solamente se esiste una funzione calcolabile totale $f : \mathbb{N} \rightarrow \mathbb{N}$ tale che $x \in A$ se e solamente se $f(x) \in B$.

Teorema 6.1: Relazione di riduzione REC

La relazione di riduzione \leqslant_{REC} classifica R ed RE .

Dimostrazione: si conosce già che $R \subseteq RE$ grazie alla proprietà 4.1. Si utilizza allora il lemma per dimostrare la tesi. Facciamo allora vedere che tutte le ipotesi del lemma sono soddisfatte:

- i) facile, dalla definizione di μ -ricorsiva;
- ii) ovvio perché la composizione conserva la totalità;
- iii) la funzione caratteristica di $\{x \mid f(x) \in B\}$ è $\chi_B \circ f$, che è calcolabile totale perché f e χ_B sono entrambe calcolabili totali;
- iv) analoga al punto precedente, con la semi-caratteristica di B .

A questo punto se trovassimo un problema che sia \leqslant_{REC} -completo per R , potremmo vedere quali sono decidibili e quali no. Se ne trovassimo uno \leqslant_{REC} -completo per RE sapremmo quali sono al più semi-decidibili e quali nemmeno semi-decidibili.

Osservazione: nel caso in cui $A \leqslant_{REC} B$ abbiamo alcune possibilità:

- $A \notin R \Rightarrow B \notin R$
- $A \notin RE \Rightarrow B \notin RE$
- $B \in R \Rightarrow A \in R$
- $B \in RE \Rightarrow A \in RE$

Torniamo a $K = \{x \mid \varphi_x(x) \downarrow\}$.

Proprietà 6.1: Insieme K

i) $\overline{K} \not\leqslant_{REC} K$.

ii) $K \not\leqslant_{REC} \overline{K}$.

Dimostrazione:

- i) se così non fosse usando la proprietà 4.1 \overline{K} sarebbe ricorsivamente enumerabile e anche ricorsivo così come K , poiché quest'ultimo è ricorsivamente enumerabile per la proprietà 4.2.
- ii) per la proprietà 5.1, $A \leqslant_{REC} B$ se e solamente se $\overline{A} \leqslant_{REC} \overline{B}$. Quindi se valesse $K \leqslant_{REC} \overline{K}$ avremmo anche $\overline{K} \leqslant_{REC} K$ che è appena stato dimostrato falso.

Teorema 6.2: K è RE -completo

K è \leqslant_{REC} -completo per RE .

Dimostrazione: si deve dimostrare che se $A \in RE$ allora $A \leqslant_{REC} K$. Per definizione A è il dominio di una funzione calcolabile ψ , cioè $A = \{x \mid \psi(x) \downarrow\}$. Dato che è calcolabile si può costruire una nuova funzione ψ' a due variabili di cui ignora la seconda, cioè sia $\psi'(x, y) = \psi(x)$, che è a sua volta una funzione calcolabile e quindi avrà un indice, diciamo i ; in simboli $\varphi_i = \psi'$. Per il teorema del parametro $\varphi_{s(i,x)}(y) = \varphi_i(x, y) = \psi'(x, y)$, con s calcolabile, iniettiva e totale. Ricapitolando posso riscrivere la definizione di A come segue:

$$\begin{aligned}
A &= \{x \mid \psi(x) \downarrow\} \\
&= \{x \mid \psi'(x, y) \downarrow\} \\
&= \{x \mid \varphi_i(x, y) \downarrow\} \\
&= \{x \mid \varphi_{s(i,x)}(y) \downarrow\} \text{ per il teorema del parametro} \\
&= \{x \mid \varphi_{s(i,x)}(s(i, x)) \downarrow\} \text{ ponendo } s(i, x) = y \\
&= \{x \mid s(i, x) \in K\}
\end{aligned}$$

prendiamo i tale che $\varphi_{s(i,x)}(y) = \varphi_i(x, y) = \psi'(x, y)$ da cui $f = \lambda x.s(i, x)$. Quindi $x \in A$ se e solamente se $f(x) \in K$, con $f(x) = \lambda x.s(i, x)$ che è totale, calcolabile, iniettiva.

Definizione 6.2: Insieme di indici che rappresentano le funzioni

A è un insieme di indici che rappresentano le funzioni (i.i.r.f) se e solamente se

$$\forall x, y. \text{ se } x \in A \text{ e } \varphi_x = \varphi_y \text{ allora } y \in A$$

Teorema 6.3: Pre-Rice

Sia A un insieme di indici che rappresentano le funzioni tale che $\emptyset \neq A \neq \mathbb{N}$. Allora $K \leqslant_{REC} A$ oppure $K \leqslant_{REC} \overline{A}$.

Dimostrazione:

1. prendi un indice i_0 tale che $\varphi_{i_0}(y)$ sia ovunque indefinita. Supponiamo che $i_0 \in \overline{A}$ e dimostriamo che $K \leqslant_{REC} A$.
2. poiché $A \neq \emptyset$ posso prendere un altro indice $i_1 \in A$ con associata la funzione $\varphi_{i_1}(y)$.
3. si nota che $\varphi_{i_0} \neq \varphi_{i_1}$ perché A è un i.i.r.f e un i.i.r.f contiene tutti gli indici di $\varphi_{i_1}(y)$.
4. definiamo la seguente funzione

$$\psi(x, y) = \begin{cases} \varphi_{i_1}(y) & \text{se } x \in K \\ \varphi_{i_0}(y) = \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile allora per la tesi di Church-Turing abbiamo un indice i tale che $\varphi_i(x, y) = \psi(x, y)$. Adesso si può applicare il teorema del parametro, ovvero $\varphi_{s(i,x)}(y) = \varphi_i(x, y)$. Ci sono infiniti indici per ψ allora ne prendiamo uno, quindi la funzione $s(i, x)$ dipende solo da x e fissiamo l'argomento i . Questa funzione che dipende solo da x la si chiama f cioè abbiamo $f(x) = \lambda x.s(i, x)$. Ricapitolando abbiamo:

$$\varphi_{f(x)}(y) = \varphi_{s(i,x)}(y) = \varphi_i(x, y) = \psi(x, y)$$

5. si prende in esame il caso $x \in K$ e quindi $\varphi_{f(x)}(y) = \varphi_{i_1}(y)$: notiamo che siccome $i_1 \in A$ e A è un i.i.r.f allora $f(x) \in A$.
6. il caso complementare è analogo: se $x \notin K$ e quindi $\varphi_{f(x)}(y) = \varphi_{i_0}(y)$ allora dato che $i_0 \in \overline{A}$ e A è un i.i.r.f allora $f(x) \notin A$.
7. abbiamo dimostrato che

- $x \in K \Rightarrow \varphi_{f(x)} = \varphi_{i_1} \Rightarrow f(x) \in A$
- $x \notin K \Rightarrow \varphi_{f(x)} = \varphi_{i_0} \Rightarrow f(x) \in \overline{A}$

quindi se $x \in K$ se e solo se $f(x) \in A$ cioè $K \leqslant_{REC} A$.

Teorema 6.4: Rice

Sia \mathcal{A} una classe di funzioni calcolabili. L'insieme $A = \{n \mid \varphi_n \in \mathcal{A}\}$ è ricorsivo se e solo se $\mathcal{A} = \emptyset$ oppure \mathcal{A} è la classe di tutte le funzioni calcolabili.

Dimostrazione: si noti che A è un insieme di indici mentre \mathcal{A} è una classe di funzioni (i primi sono sintassi, i secondi sono semantiche). Si esamina ogni caso:

- se $\mathcal{A} = \emptyset$ allora A è ovviamente ricorsivo (χ_A totale perché restituisce sempre 0);
- se \mathcal{A} è la classe di tutte le funzioni calcolabili allora A è anche ricorsivo (χ_A totale perché restituisce sempre 1);
- in tutti gli altri casi basta applicare il teorema Pre-Rice perché A è un insieme di indici che rappresentano le funzioni.

Un'applicazione immediata del teorema di Rice è che $K_1 = \{x \mid \text{dom}(\varphi_x) \neq \emptyset\}$ cioè l'insieme (degli indici) delle funzioni che sono definite in almeno un punto *non è ricorsivo*, sebbene sia ricorsivamente enumerabile. Inoltre $K \equiv K_0 \equiv K_1$, cioè i tre insiemi si riducono l'uno all'altro e sono *RE*-completi. Altre classi che non sono ricorsive sono:

- $FIN = \{x \mid \text{dom}(\varphi_x) \text{ finito}\}$
- $REC = \{x \mid \text{dom}(\varphi_x) \text{ e' ricorsivo}\}$
- $INF = \{x \mid \text{dom}(\varphi_x) \text{ e' infinito}\} = \mathbb{N} \setminus FIN$
- $CONST = \{x \mid \varphi_x \text{ totale e costante}\}$
- $TOT = \{x \mid \varphi_x \text{ totale}\} = \{x \mid \text{dom}(\varphi_x) = \mathbb{N}\}$
- $EXT = \{x \mid \varphi_x \text{ e' estendibile a f calcolabile totale}\}$

Parte II

Complessità

Nella prima parte si è posta l'enfasi su *cosa* si calcola. Ora si studia *come* si calcola e *quali risorse* siano necessarie per calcolare. Le risorse di cui si tiene principalmente conto sono relative al **tempo**, lo **spazio** e la **taglia**. Quest'ultimo, che esprimeremo con $|x|$, sono le risorse necessarie alla soluzione di un caso del problema in funzione dei suoi dati di ingresso. Dato un problema formalizzato dall'insieme I , cercheremo una funzione $f(|x|)$ che esprima la quantità di risorse in spazio o tempo necessaria al calcolo della soluzione di $x \in I$. È necessario studiare come si definisce e quali proprietà ha una funzione f che sia in grado di stimare la quantità minima di risorse necessarie per risolvere un certo problema I . Determinare questa funzione f (se e quando ciò è possibile) richiede di considerare *tutti* gli algoritmi che risolvono il problema I , i quali sono tanti quanti i numeri naturali. Data una certa funzione f , si può definire una **classe di complessità** a cui appartengono tutti quei problemi per cui esiste almeno un algoritmo che li risolve e che richiede risorse in misura inferiore o uguale a $f(n)$. Esistono due classi di complessità molto importanti:

- \mathcal{P} , classe dei problemi risolubili in tempo polinomiale *deterministico*
- \mathcal{NP} , classe dei problemi risolubili in tempo polinomiale *non deterministico*

I risultati della teoria di complessità che andremo enunciare sono invarianti rispetto al modello di calcolo scelto. Questo è il frammento di gerarchia delle classi di complessità che vedremo:

$$\text{LOGSPACE} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subset \text{R} \subset \text{RE}$$

7 Misure di complessità deterministiche

Definizione 7.1: Macchine di Turing a k nastri

Dato un numero naturale k , una macchina di Turing con k nastri è una quadrupla (Q, Σ, δ, q_0) con

- $\#, \triangleright \in \Sigma$ e $L, R, - \notin \Sigma$
- $SI, NO \notin Q$
- $\delta : Q \times \Sigma^k \rightarrow Q \cup \{SI, NO\} \times (\Sigma \times \{L, R, -\})^k$ è la funzione di transizione soggetta alle stesse condizioni della definizione 2.2 sull'unicità della stringa in $(\Sigma \times \{L, R, -\})^k$ in modo che δ sia una funzione, e sull'uso del carattere di inizio stringa \triangleright .

7.1 Complessità in tempo deterministico

Definizione 7.2: Tempo richiesto esatto

t è il *tempo richiesto* da una MdT M a k nastri per decidere il caso $x \in I$ se

$$M(x) \xrightarrow{t} (SI/NO, w)$$

Intuizione: t è il numero esatto dei passi necessari per decidere in tempo deterministico un particolare caso.

Definizione 7.3: Tempo richiesto approssimato

M decide I in tempo deterministico f se per ogni dato di ingresso x il tempo t richiesto da M per decidere x è minore di o è uguale a $f(|x|)$.

Intuizione: il numero esatto dei passi necessari al calcolo di $M(x)$ può rivelarsi troppo complicato, quindi ci accontenteremo allora di approssimare tale numero per eccesso.

Definizione 7.4: Classe di complessità in tempo deterministico

$$TIME(f) = \{I \mid \exists M \text{ che decide } I \text{ in tempo deterministico } f\}.$$

Intuizione: contiene tutti e soli i problemi risolvibili in tempo deterministico f , ovvero, affinché un problema vi appartenga occorre e basta che vi sia una macchina M che lo decide in tempo deterministico f .

Teorema 7.1: Riduzione del numero dei nastri

Data una macchina di Turing M con k nastri che decide I in tempo deterministico f , allora $\exists M'$ con 1 nastro che decide I in tempo deterministico $O(f^2)$.

Intuizione: la conseguenza di questo teorema è che l'aggiunta di nastri ad una macchina di Turing non ne modifica il tipo di funzioni calcolabili e non modifica il tempo deterministico richiesto se non polinomialmente. Tutti teoremi che avevamo sulla calcolabilità delle funzioni, insieme ricorsivo, etc... continuano a valere anche se utilizziamo una macchina a k nastri.

Dimostrazione: costruiamo M' che simula M , rappresentando ogni configurazione come segue, usando le parentesi angolate per separare i vari nastri

$$(q, \triangleright w_1 \sigma_1 u_1, \triangleright w_2 \sigma_2 u_2, \dots, \triangleright w_k \sigma_k u_k)$$

viene simulata da:

$$(q', \triangleright \langle w_1 \bar{\sigma}_1 u_1 \rangle \langle w_2 \bar{\sigma}_2 u_2 \rangle \dots \langle w_k \bar{\sigma}_k u_k \rangle)$$

i simboli $\bar{\sigma}_i$ sono sostituiti ai simboli σ_i quando il cursore è su di essi. Per cominciare la macchina M' applicata ad x dovrà generare la configurazione che simula la configurazione iniziale di M , cioè dobbiamo passare dalla configurazione iniziale di M ,

$$(q_0, \triangleright x, \triangleright, \dots, \triangleright) \quad a \quad (q, \triangleright \langle x \rangle [\langle \rangle]^{k-1})$$

per qualche q . Per fare ciò, bastano $2k + \#\Sigma$ nuovi stati e un certo numero di passi che, essendo dell'ordine di $|x|$, non influenza la complessità asintotica. Per simulare una mossa di M , la macchina M' scorre il dato di ingresso da sinistra a destra e viceversa due volte:

- la prima volta M' determina quali sono i simboli correnti di M , $\bar{\sigma}_i$
- la seconda volta M' scrive i nuovi simboli nel posto giusto

Infine quando M si ferma, anche M' si ferma, eventualmente rimuovendo tutte le parentesi $\langle \rangle$ e sostituendo i caratteri $\bar{\sigma}_i$ con σ_i . Una macchina non può toccare un numero di caselle maggiore del numero di passi che compie. Di conseguenza la lunghezza totale del nastro scritto è al più $K = k \times (f(|x|) + 2) + 1$. Allora, andare due volte avanti e indietro costa, in termini di tempo, per ogni stringa simulata $4K$. Possiamo concludere che per simulare un singolo passo di M la macchina M' richiede $O(f|x|)$ passi sul dato x . Il numero dei passi di M' sull'intera computazione è quindi in $O(f(|x|)^2)$, perché M richiede tempo $f(|x|)$ e perché M' impiega $O(f(|x|))$ per simulare ogni passo di M . Infine per costruzione M' è equivalente a M e quindi le due macchine decidono lo stesso problema; allora M' , che ha un nastro solo, decide tale problema in tempo deterministico $O(f(|x|)^2)$.

Osservazione: non si può usare più spazio che tempo!

Corollario: le macchine parallele sono polinomialmente più veloci di quelle sequenziali.

Teorema 7.2: Accelerazione lineare MdT

Se $I \in TIME(f)$ allora $\forall \epsilon \in \mathbb{R}^+$ si ha che $I \in TIME(\epsilon \times f(n) + n + 2)$.

Intuizione: dato un algoritmo che decide un problema, se ne può sempre trovare uno equivalente che è più veloce per una costante moltiplicativa ϵ (supponendo che questa sia minore di 1). Tuttavia se il problema I è deciso da un algoritmo in tempo esponenziale, non è possibile trovare un algoritmo che lo risolva in tempo deterministico polinomiale, per mezzo del solo teorema di accelerazione. Vedremo che una analoga osservazione vale anche per lo spazio.

Dimostrazione: si simula una macchina M con una macchina M' . I passi sono i seguenti:

- i) si condensa il dato di ingresso (in $n + 2$ passi con $n = |x| \leq m \times \lceil \frac{|x|}{m} \rceil + 2$): ogni sequenza di m simboli di M origina un singolo simbolo di M' cioè $\sigma_{i_1} \dots \sigma_{i_m}$ viene codificata come il singolo simbolo $[\sigma_{i_1} \dots \sigma_{i_m}]$.
- ii) gli stati di M' saranno formati da triple $[q, \sigma_{i_1} \dots \sigma_{i_m}, k]$ con $1 \leq k \leq m$ in modo da rappresentare il fatto che M si trova nello stato q e ha il cursore sul k -esimo simbolo della stringa $\sigma_{i_1} \dots \sigma_{i_m}$.
- iii) alla macchina M' bastano 6 passi per simulare la macchina M :
 - a) 4 passi per codificare in M' i simboli a sinistra e destra della testina;
 - b) 2 passi per aggiornare lo stato di M' simulando m transizioni di M .
- iv) la simulazione di M richiede in totale $6 \times \lceil \frac{f(|x|)}{m} \rceil + |x| + 2$ passi e la traccia della dimostrazione si conclude scegliendo $m = \lceil \frac{6}{\epsilon} \rceil$.

Definizione 7.5: Classe \mathcal{P}

La classe dei problemi decidibili in tempo polinomiale è

$$\mathcal{P} = \bigcup_{k \geq 1} TIME(n^k)$$

Intuizione: la classe \mathcal{P} è invariante rispetto al cambio di modelli.

7.2 Macchine di Turing I/O

Definizione 7.6: Macchine di Turing I/O

Una MdT con k nastri $M = (Q, \Sigma, \delta, q_0)$ è di tipo I/O se e solamente se la funzione di transizione δ è tale che, tutte le volte che $\delta(q, \sigma_1, \dots, \sigma_k) = (q', (\sigma'_1, D_1), \dots, (\sigma'_k, D_k))$

- $\sigma'_1 = \sigma_1$, quindi il primo nastro è di sola lettura;
- $D_k = R$ o quando $D_k = -, \sigma'_k = \sigma_k$, quindi il k -esimo nastro è a sola scrittura;
- se $\sigma_1 = \#$ allora $D_1 \in \{L, -\}$, la macchina visita al massimo una cella bianca a destra del dato di ingresso.

Intuizione: per studiare la complessità in spazio è conveniente usare questa variante di macchine di Turing, che hanno un nastro dedicato a contenere il dato di ingresso che sarà di sola lettura; uno destinato a memorizzare il risultato che sarà di sola scrittura; e $k-2$ nastri di lavoro, gli unici rilevanti ai fini della complessità.

Proprietà 7.1: Macchina di Turing I/O

Per ogni MdT con k nastri M che decide I in tempo deterministico f esiste una MdT a $k + 2$ nastri M' di tipo I/O che decide I in tempo deterministico $c \times f$, per qualche costante c .

Dimostrazione: La macchina M'

- i) copia il primo nastro di M sul proprio ultimo nastro, impiegando $|x| + 1$ passi;
- ii) opera come M senza più toccare il proprio primo nastro, impiegando $f(|x|)$ passi;
- iii) copia il risultato del calcolo di M sul proprio ultimo nastro, in al più $f(|x|)$ passi e termina.

In totale la macchina M' ha richiesto su x un numero di passi inferiore o uguale a $2 \times f(|x|) + |x| + 1$. Determinare ora la costante c è immediato.

7.3 Complessità in spazio deterministico

Per misurare lo spazio richiesto da una MdT, ne modifichiamo una in modo che tenga traccia delle caselle visitate. La modifica consiste nell'usare il simbolo $\triangleleft \in \Sigma$ per delimitare fin dove è arrivata la testina, cioè la funzione di transizione sposta implicitamente \triangleleft a destra ogni volta che la testina si sposterebbe sopra a \triangleleft .

Definizione 7.7: Spazio richiesto MdT I/O

Sia M una MdT a k nastri di tipo I/O tale che $\forall x$

$$(q_0, \triangleleft x, \triangleleft, \dots, \triangleleft) \rightarrow^* (H, w_1, w_2, \dots, w_k) \text{ con } H \in \{SI, NO\}$$

Lo spazio richiesto da M per decidere x è

$$\sum_{i=2}^{k-1} |w_i|$$

Inoltre M decide I in spazio deterministico $f(n)$ se $\forall x$ lo spazio richiesto da M per decidere x è minore o uguale a $f(|x|)$. Infine, se M decide I in spazio deterministico $f(n)$, allora

$$I \in SPACE(f(n))$$

Teorema 7.3: Compressione lineare dello spazio

Se $I \in SPACE(f(n))$ allora $\forall \epsilon \in \mathbb{R}^+. I \in SPACE(\epsilon \times f(n) + 2)$.

Intuizione: è l'analogo in spazio del teorema 7.2 di accelerazione lineare.

Definizione 7.8: PSPACE

La classe dei problemi decidibili in spazio polinomiale deterministico è

$$PSPACE = \bigcup_{k \geq 1} SPACE(n^k)$$

Definizione 7.9: LOGSPACE

La classe dei problemi decidibili in spazio logaritmico deterministico è

$$LOGSPACE = \bigcup_{k \geq 1} SPACE(k \times \log n)$$

Teorema 7.4: Gerarchia classi spazio deterministico

- i) $\text{LOGSPACE} \subsetneq \text{PSPACE}$.
- ii) $\text{LOGSPACE} \subseteq \mathcal{P}$.

Dimostrazione: del punto ii): poiché il problema I appartiene a LOGSPACE , c'è una macchina di Turing M che decide ogni sua istanza $x \in I$ in $O(\log|x|)$ spazio deterministico. Una computazione non può ripassare su una stessa configurazione, altrimenti va in ciclo, quindi una computazione ha al massimo $O(|x|^k)$ passi per qualche k .

Osservazione: lo spazio limita il tempo di calcolo!

8 Misure di complessità non deterministiche

Quando non sappiamo come risolvere un problema c'è un algoritmo che funziona sempre bene, se non per la sua complessità: l'algoritmo di **forza bruta**. Questo algoritmo origina un albero di scelte in cui i nodi interni rappresentano una computazione non ancora terminata e le foglie rappresentano una possibile soluzione del problema. Vengono esaminate tutte le possibili scelta, mossa che ha un costo esponenziale nella taglia dell'input. In alternativa c'è un altro algoritmo, chiamato **guess-and-try**, che ci permette di evitare di esaminare sistematicamente tutte le possibili scelte, "lanciando un dado" di fronte ad ogni scelta. L'algoritmo esegue una successione di scelte casuali finché non viene raggiunta una foglia. Se questa soddisfa i requisiti del problema l'algoritmo accetta la soluzione. Visto che si tratta di un procedimento stocastico, potrebbe accadere che l'algoritmo selezioni casualmente la soluzione al primo tentativo oppure potrebbe ridursi al caso precedente, andando a controllare tutto l'albero. Tuttavia possiamo calcolare la probabilità che l'algoritmo termini entro un certo numero di tentativi e, se ritenuta sufficientemente bassa, questo metodo può rivelarsi una buona alternativa pratica al metodo di forza bruta.

Definizione 8.1: Macchina di Turing non deterministica

Una MdT non deterministica, a k nastri di tipo I/O, è una quadrupla $N = (Q, \Sigma, \Delta, q_0)$, dove:

- Q, Σ, q_0 sono come nella definizione 2.2
- $\Delta \subseteq (Q \times \Sigma) \times ((Q \cup \{SI, NO\}) \times \Sigma \times \{L, R, -\})$ è la relazione di transizione

Le computazioni sono sempre una successione di passi di computazione, esattamente come nella variante deterministica. Quel che accade di veramente diverso è che la macchina esamina contemporaneamente tutte le possibili successioni di configurazioni (cioè computazioni).

Intuizione: le MdT non deterministiche generano un albero di possibili computazioni, come l'algoritmo guess-and-try, e si arrestano appena trovano una foglia che rappresenta una soluzione accettabile per il problema.

Definizione 8.2: Accettazione MdT non deterministica

La macchina non deterministica N (a k nastri) decide I tutte e sole le volte che $x \in I$ se e solamente se esiste una computazione tale che

$$(q_0, \underline{\triangleright} x, \triangleright, \dots, \triangleright) \rightarrow_N^* (SI, w_1, w_2, \dots, w_k)$$

Intuizione: per accettare una stringa in ingresso basta che esista una computazione che porta ad una configurazione il cui stato è *SI*.

8.1 Complessità in tempo e spazio non deterministici

Definizione 8.3: Tempo non deterministico

La macchina non deterministica N decide I in tempo non deterministico $f(n)$ se e solo se

- N decide I .
- $\forall x \in I, \exists$ un numero di passi t tale che $N(x) \rightarrow_N^t (SI, w_1, w_2, \dots, w_k)$ e $t \leq f(|x|)$.

Intuizione: il secondo punto dice che N accetta x in tempo non deterministico $f(|x|)$ se e solo se c'è almeno una computazione che termina su uno stato di accettazione in t passi con t minore o uguale a $f(|x|)$. Una macchina N accetta $x \in I$ se esiste almeno una computazione che termina in meno di $f(|x|)$ passi; mentre per rifiutare y che non appartiene ad I (e quindi $y \in \bar{I}$), tutte le computazioni che terminano in meno di $f(|y|)$ passi devono essere di rifiuto.

Definizione 8.4: NTIME

Se una macchina di Turing non deterministica decide il problema I in tempo f allora

$$I \in \text{NTIME}(f).$$

Definizione 8.5: Classe \mathcal{NP}

La classe dei problemi decidibili in tempo non deterministico polinomiale è

$$\mathcal{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k).$$

Il motivo per cui $\mathcal{P} \subseteq \mathcal{NP}$ è che una MdT deterministica è anche non deterministica. Se fosse vero che $\mathcal{P} \supseteq \mathcal{NP}$ allora si potrebbe concludere che $\mathcal{P} = \mathcal{NP}$, cioè esisterebbe un modo per trasformare algoritmi non deterministici polinomiali in uno deterministico polinomiale.

Teorema 8.1: Simulazione non deterministica

Se I è deciso in tempo non deterministico $f(n)$ dalla macchina non deterministica N (a k nastri) allora con una perdita esponenziale, è deciso in tempo $O(c^{f(n)})$ da una macchina deterministica M (a $k+1$ nastri) con $c > 1$ dipendente solo da N . Più precisamente:

$$\text{NTIME}(f(n)) \subseteq \text{TIME}(c^{f(n)})$$

Dimostrazione: sia d il grado di non determinismo di N , cioè poniamo:

$$d = \max\{\text{Grado}(q, \sigma) \mid q \in Q, \sigma \in \Sigma\}$$

dove $\text{Grado}(q, \sigma) = \#\{(q', \sigma', D) \mid ((q, \sigma), (q', \sigma', D)) \in \Delta\}$, cioè Grado conta il numero di scelte disponibili quando la macchina si trova nello stato q e legge il simbolo σ dal nastro. La variabile d contiene il massimo grado, cioè il massimo numero di scelte che può fare N .

Per semplicità supponiamo che la macchina abbia d scelte, quindi consideriamo un albero di scelte completo con fattore di diramazione d .

- si ordina totalmente (per esempio lessicograficamente) l'insieme $\Delta(q, \sigma)$
- ogni computazione di N è una sequenza di scelte e si possono rappresentare con un numero in base d
- la macchina M deterministica ha un nastro extra rispetto a N e si comporta così:
 - i) inizializza il nastro extra con l'intero in base d che rappresenta la prima scelta da verificare-
 - ii) M legge il numero registrato sul nastro extra, lo decodifica e riproduce la successione di scelte rappresentate dal numero.

- iii) M simula quella porzione di computazione di N che verifica se la successione di scelte porta ad una soluzione accettabile.
- iv) Se N termina
 - a) in uno stato di accettazione, anche M termina e accetta; altrimenti genera la prossima successione di scelte codificandola e salvandola sul nastro extra.
 - b) in uno stato di rifiuto e sono state generate tutte le possibili successioni di scelte allora anche M termina e rifiuta.
- v) M riparte dal punto ii), stavolta esaminando la successione di scelte successiva.

Data questa costruzione M termina se e solo se N termina e M accetta se solo se N accetta, quindi M e N calcolano la stessa funzione. M deve visitare al caso pessimo un albero con fattore di diramazione d e profondo $f(n) + 1$. Di conseguenza abbiamo che M ha complessità $O(d^{f(n)+1})$ e possiamo dimostrare il teorema ponendo $c = d$.

Definizione 8.6: Spazio non deterministico

La macchina di Turing N , non deterministica a k -nastri di tipo I/O decide I in spazio non deterministico $f(n)$ se e solamente se:

- N decide I .
- $\forall x \in I, \exists w_1, \dots, w_k$ tali che $(q_0, \sqsupseteq x, \dots, \sqsupseteq) \rightarrow_N^* (SI, w_1, w_2, \dots, w_k)$ e $\sum_{2 \leq i \leq k-1} |w_i| \leq f(|x|)$.

Se N decide I in spazio non deterministico $f(n)$, allora $I \in NSPACE(f(n))$.

Definizione 8.7: NPSPACE

La classe dei problemi decidibili in spazio non deterministico polinomiale è

$$NPSPACE = \bigcup_{k \geq 1} NSPACE(n^k)$$

Teorema 8.2: Savitch

$$NPSPACE = PSPACE.$$

Intuizione: l'estensione con il non determinismo non allarga la classe dei problemi trattabili in spazio polinomiale, ma lo fa solo con il tempo.

9 Funzioni di Misura

Definizione 9.1: Funzioni di misura

La funzione $f : \mathbb{N} \rightarrow \mathbb{N}$ calcolabile totale è *appropriata* se

- i) è monotona crescente (cioè $n \geq m$ implica $f(n) \geq f(m)$).
- ii) esiste una MdT M a k nastri, tale che $\forall x \in \Sigma^*$ si arresta in tempo $O(f(|x|) + |x|)$ e in spazio $O(f|x|)$.

Sotto queste ipotesi, esiste una gerarchia di problemi.

Intuizione: una funzione di misura appropriata può essere una qualunque funzione totale e non richiede più risorse di quanto stimato dal loro stesso valore, cioè l'algoritmo che calcola $f(x)$ deve richiedere tempo $O(f(|x|) + |x|)$ e richiedere spazio $O(f|x|)$.

Teorema 9.1: Gerarchia

Se f è appropriata:

- $TIME(f(n)) \subsetneq TIME((f(2n+1))^3)$.
- $SPACE(f(n)) \subsetneq SPACE(f(n) \times \log f(n))$.

Dimostrazione: la dimostrazione del primo punto si basa sulla diagonalizzazione:

$\{x \mid \varphi_x(x) \text{ converge entro } f(|x|) \text{ passi}\} \in TIME(f^3)$ ma non appartiene a $TIME(f)$

la dimostrazione del secondo punto è analoga.

Definizione 9.2: EXP

Classe dei problemi decidibili in tempo deterministico esponenziale $EXP = \bigcup_{k \geq 1} TIME(2^{n^k})$.

Teorema 9.2: Gerarchia

$P \subsetneq EXP$.

Intuizione: questo fatto, assieme al fatto che $NTIME(f(n)) \subseteq TIME(c^{f(n)})$ permette di concludere che \mathcal{NP} è inclusa in EXP .

Dimostrazione: l'inclusione è ovvia perché 2^n cresce più velocemente di ogni polinomio; è propria perché

$$P \subseteq TIME(2^n) \subsetneq TIME((2^{(2n+1)})^3) \subseteq TIME(2^{n^2})$$

Per un quadro più completo si riportano i seguenti risultati:

- | | |
|--|--------------------------------|
| • $SPACE(f(n)) \subseteq NSPACE(f(n))$ | • $LOGSPACE \subsetneq PSPACE$ |
| • $TIME(f(n)) \subseteq NTIME(f(n))$ | • $PSPACE = NPSPACE$ |
| • $NSPACE(f(n)) \subseteq TIME(k^{\log n + f(n)})$ | • $\mathcal{NP} \subseteq EXP$ |
| • $LOGSPACE \subseteq P$ | |

La gerarchia non è superiormente limitata, quindi ci sono problemi arbitrariamente difficili, indipendentemente dagli algoritmi usati per risolverli.

Teorema 9.3: No richiesta di funzione appropriata

Per ogni funzione calcolabile totale g esiste un problema $I \in TIME(f(n))$ e $I \notin TIME(g(n))$ con $f(n) > g(n)$ quasi ovunque.

Intuizione: questo teorema non richiede che le funzioni di misura siano appropriate.

Se ignoriamo la definizione di funzione appropriata, possiamo dimostrare dei teoremi controtuitivi.

Teorema 9.4: Accelerazione, Blum

Per ogni funzione calcolabile totale h , esiste un problema I tale che, per ogni algoritmo M che decide I in tempo f esiste M' che decide I in tempo f' tale che

$$f(n) > h(f'(n)) \text{ quasi ovunque}$$

Intuizione: il teorema di accelerazione lineare diceva che possiamo accelerare linearmente la soluzione del problema. Blum ci dice che *non sempre esiste un algoritmo ottimo* per un problema.

Teorema 9.5: Della lacuna, Borodin

Esiste f calcolabile totale tale che $TIME(f(n)) = TIME(2^{f(n)})$

Intuizione: c'è un insieme di programmi che sono altrettanto veloci su una macchina lenta che su una macchina nuova. Cioè che tu abbia f risorse o 2^f risorse non fa differenza.

Definizione 9.3: Assiomi di Blum

Una funzione ϕ è una misura di complessità se restituisce un naturale a fronte di una funzione ψ e del suo dato di ingresso x e inoltre soddisfa entrambi i seguenti assiomi:

1. $\phi(\psi, x)$ è definita se e solo se $\psi(x)$ lo è;
2. per ogni ψ, x, k è decidibile se $\phi(\psi, x) = k$.

Intuizione: il primo assioma dice che ϕ misura la complessità del calcolo di $\psi(x)$; il secondo assicura che si può davvero ottenere la complessità del calcolo di $\psi(x)$. Quindi se la funzione ϕ misurasse il numero di passi, otterremmo la vecchia definizione di complessità in tempo; se misurasse le celle toccate, quella in spazio.

10 \mathcal{P} e \mathcal{NP}

Definizione 10.1: Tesi di Cook-Karp

I problemi in \mathcal{P} sono trattabili e quelli in \mathcal{NP} intrattabili. Inoltre \mathcal{P} e \mathcal{NP} resistono al cambio di modello e sono invarianti rispetto alla rappresentazione del problema.

Proprietà 10.1: Classe \mathcal{P}

- \mathcal{P} è chiusa rispetto alla composizione polinomiale sinistra, cioè se per tutti i polinomi $p, f \in \mathcal{P}$ allora $p \circ f \in \mathcal{P}$.
- \mathcal{P} è chiusa rispetto alla composizione polinomiale destra, cioè se per tutti i polinomi $p, f \in \mathcal{P}$ allora $f \circ p \in \mathcal{P}$.

Come prima l'idea è che per risolvere il problema I lo si reduce efficientemente per mezzo di una funzione f a $f(I)$ appartenente a una data classe \mathcal{D} e si risolve il problema ridotto; se la classe è chiusa rispetto a quelle riduzioni allora anche il problema I sta in \mathcal{D} . Quando si tratta di \mathcal{P} e \mathcal{NP} useremo riduzioni *logaritmiche in spazio deterministico*. Si noti che se $f \in \text{LOGSPACE}$ allora $f \in \mathcal{P}$.

Definizione 10.2: Riduzione efficiente

I si riduce efficientemente a I' ($I \leqslant_{\text{LOGSPACE}} I'$) se esiste un algoritmo $f \in \text{LOGSPACE}$ tale che $x \in I$ se e solamente se $f(x) \in I'$.

Teorema 10.1: LOGSPACE classifica \mathcal{P} e \mathcal{NP}

Siano $\mathcal{D}, \mathcal{E} \in \{P, NP, EXP, PSPACE, NPSPACE\}$ e $\mathcal{D} \subseteq \mathcal{E}$

- $\leqslant_{\text{LOGSPACE}}$ classifica LOGSPACE e \mathcal{E} .
- $\leqslant_{\text{LOGSPACE}}$ e a maggior ragione $\leqslant_{\mathcal{P}}$ classificano \mathcal{D} ed \mathcal{E} .

Intuizione: la classe di funzioni in LOGSPACE induce una relazione di riduzione che classifica LOGSPACE e \mathcal{D} , dove \mathcal{D} è una qualunque delle classi che abbiamo introdotto.

Dimostrazione: il primo enunciato è vero perché le ipotesi del lemma sulle relazioni di riduzioni sono soddisfatte:

- i) $Id \in LOGSPACE$, banale, si prende una macchina a k -nastri che prende il nastro di ingresso e lo riscrive in uscita.
- ii) $f, g \in LOGSPACE \Rightarrow f(g) \in LOGSPACE$, si fa partire M' in modo da richiedere M : dati di lavoro e usarli all'occorrenza sostituendoli sempre su una sola casella.
- iii) Se $J \in \mathcal{P}$ e $I \leqslant_{LOGSPACE} J$ allora $I \in \mathcal{P}$, se I si riscrive per un logaritmo a J e J lo si risolve in tempo logaritmico allora anche I lo si risolve in tempo logaritmico.
- iv) Se $J \in \mathcal{NP}$ e $I \leqslant_{LOGSPACE} J$ allora $I \in \mathcal{NP}$, analogo al punto precedente.

Il secondo enunciato è vero perché

- i) $LOGSPACE$ e \mathcal{P} hanno identità.
- ii) $LOGSPACE$ e \mathcal{P} sono chiuse per composizione.
- iii) è vero che $f \in LOGSPACE$ oppure $\mathcal{P}, B \in \mathcal{D} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{D}$.
- iv) è vero che $f \in LOGSPACE$ oppure $\mathcal{P}, B \in \mathcal{E} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{E}$.

10.1 Problemi interessanti e riduzioni efficienti tra essi

Definizione 10.3: Problema SAT

Il problema SAT consiste nel decidere se, data un'espressione booleana esiste un assegnamento \mathcal{V} tale che $\mathcal{V} \models B$.

Intuizione: SAT appartiene a \mathcal{NP} perché basta usare una macchina non deterministica che esamina tutti i possibili assegnamenti e si arresta in tempo polinomiale.

Definizione 10.4: Problema HAM

Il problema HAM consiste nel decidere se in un grafo orientato c'è un cammino, detto hamiltoniano, che tocca tutti i nodi una e una sola volta.

Proprietà 10.2: HAM si riduce in spazio logaritmico a SAT

$HAM \leqslant_{LOGSPACE} SAT$.

Intuizione: HAM si può ridurre in spazio logaritmico a SAT, cioè SAT è un problema tanto difficile quanto lo è HAM, visto che la riduzione usata è efficiente.

Dimostrazione: dobbiamo costruire una funzione $f \in LOGSPACE$ tale che G ha un cammino hamiltoniano se e solo se $f(G)$ è soddisfacibile. Per ipotesi G ha n nodi, $f(G)$ ha n^2 variabili booleane x_{ij} , con i, j comprese tra 1 e n . Costruiamo i congiunti che rappresentano le proprietà dei cammini hamiltoniani:

1. lo stesso nodo j non può apparire in due posizioni diverse nello stesso cammino;
2. ogni nodo j deve apparire in un cammino;
3. qualche nodo deve essere l' i -esimo di un cammino;
4. due nodi non possono essere contemporaneamente l' i -esimo nello stesso cammino;
5. se (i, j) non è un arco di G , i e j non possono apparire in sequenza in un cammino (hamiltoniano o no).

Vediamo adesso che G ha un cammino hamiltoniano se $\mathcal{V} \models f(G)$. Si noti che se $\mathcal{V} \models f(G)$ allora per ogni j esiste unico i tale che $\mathcal{V}(x_{ij}) = tt$ altrimenti i punti 1. e 2. non potrebbero essere soddisfatte entrambe. Allo stesso modo per ogni i esiste un nodo j tale che $\mathcal{V}(x_{ij}) = tt$. Quindi l'espressione booleana rappresenta una permutazione dei nodi, e per il punto 5. abbiamo che tale permutazione è un cammino. Vediamo ora l'inverso: sia $H = (\Pi(1), \dots, \Pi(n))$ ha un cammino hamiltoniano. Allora è immediato verificare che

$$\mathcal{V}(x_{ij}) = \begin{cases} tt & \text{se } \pi(j) = i \\ ff & \text{se } \pi(j) \neq i \end{cases}$$

soddisfa $f(G)$. Infine per vedere che la trasformazione è logaritmica in spazio, basta notare che le uniche informazioni che dobbiamo salvare nei nastri di lavoro sono n e tre contatori i, j, k che vanno da 1 ad n tutti rappresentati in binario, quindi il costo in spazio è $O(\log n)$.

Definizione 10.5: Problema CRICCA

Il problema CRICCA consiste nel decidere se in un dato grafo non orientato $G = (N, A)$ esiste $C \subseteq N$, detto cricca (di grado k , funzione del numero dei nodi) tale che $\forall i, j \in C$, con $i \neq j$, l'arco $(i, j) \in A$.

Proprietà 10.3: SAT si riduce in spazio logaritmico a CRICCA

$SAT \leqslant \text{LOGSPACE CRICCA}$.

Intuizione: SAT si può ridurre in spazio logaritmico a CRICCA, cioè CRICCA è un problema almeno tanto difficile quanto lo è SAT.

Dimostrazione: data un'espressione booleana $B = \wedge_{1 \leq k \leq n} C_k$ si costruisce il grafo $f(B) = (N, A)$ così

- i) N è l'insieme delle occorrenze dei letterali in B .
- ii) A è l'insieme $\{(i, j) \mid i \in C_k \rightarrow (j \notin C_k \text{ e } i \neq \neg j)\}$.

B è soddisfacibile se e solamente se $f(B)$ ha una cricca di ordine pari al numero di congiunti: basta attribuire valore tt ai letterali corrispondenti ai nodi della cricca. La riduzione è logaritmica in spazio perché basta mantenere sui nastri di lavoro due indici, rappresentati in binario, che scorrono i letterali.

Definizione 10.6: Circuito Booleano

Un circuito booleano è un grafo diretto aciclico (N, A) i cui nodi $1, \dots, n$ sono detti *porte* e i cui archi sono rappresentati come coppie ordinate.

- le porte hanno 0, 1 o 2 ingressi e sono $s(i) = \{tt, ff, \neg, \vee, \wedge\} \cup X$ dove X è l'insieme delle variabili.
- gli ingressi i del circuito sono le porte di sorta $s(i) \in \{tt, ff\} \cup X$, e non hanno ingressi.
- l'uscita del circuito è la porta n senza uscite

Tutte le altre porte hanno una uscita e quando $s(i) = \neg$, la porta i ha un solo ingresso e quando $s(i) \in \{\vee, \wedge\}$ allora la porta i ha due ingressi.

Definizione 10.7: Problema CIRCUIT SAT

Il problema *CIRCUIT SAT* consiste nel decidere se esiste un assegnamento \mathcal{V} tale che $\mathcal{V}(C) = tt$.

Intuizione: un modo immediato per risolvere questo problema è di sostituire alle variabili $x, y, z \dots$ tutte le possibili combinazioni di tt, ff e di controllare una a una le combinazioni per verificare se sono una soluzione. Questa è ancora una volta una ricerca esaustiva per forza bruta. La versione non deterministica mostra che *CIRCUIT SAT* $\in \text{NP}$ e prevede due passi:

1. si sceglie in tempo polinomiale non deterministico l'assegnamento "giusto" tra i 2^n possibili, se n sono le porte.
2. polinomiale nel numero delle porte del circuito, si certifica il risultato.

Definizione 10.8: Problema CIRCUIT VALUE

Il problema *CIRCUIT VALUE* consiste nel calcolare il valore di un circuito senza variabili, ovvero in cui gli ingressi sono porte di sorta $s(i) \in \{tt, ff\}$.

Intuizione: questo problema appartiene a \mathcal{P} : in un circuito di n porte, basta calcolare livello per livello i valori di uscita. Nonostante le porte siano disposte ad albero, l'algoritmo lavora su un circuito di n porte, quindi non ci interessa il fatto che il numero di porte cresca esponenzialmente con il numero di livelli. Sul nastro della macchina teniamo i valori calcolati del livello corrente e li usiamo per calcolare il livello successivo.

10.1.1 Problemi completi per \mathcal{P} e NP

Proprietà 10.4: CIRCUIT VALUE si riduce in spazio logaritmico a CIRCUIT SAT

CIRCUIT VALUE $\leqslant_{LOGSPACE}$ *CIRCUIT SAT*.

Intuizione: ogni caso particolare di un problema si riduce al problema stesso nella sua piena generalità attraverso la funzione identità.

Proprietà 10.5: CIRCUIT SAT si riduce in spazio logaritmico a SAT

CIRCUIT SAT $\leqslant_{LOGSPACE}$ *SAT*.

Dimostrazione: dato $C = (N, A)$ con variabili in X , dobbiamo trovare una riduzione $f \in LOGSPACE$ tale che l'espressione booleana $f(C)$ sia soddisfacibile se e solamente se C lo è. Con l'algoritmo seguente possiamo costruire la funzione $f(C)$:

1. le variabili x di $f(C)$ sono quelle di C unite ad un nuovo insieme che contiene una nuova variabile per ogni porta di C
2. per ogni porta g di C costruiamo i congiunti di $f(C)$ così:
 - se g è la porta di uscita allora genera il congiunto g .
 - se $s(g) = tt$ allora genera g .
 - se $s(g) = ff$ allora genera $\neg g$.
 - se $s(g) = x \in X$ allora genera $(\neg g \vee x) \wedge (g \vee \neg x)$, ovvero g se e solamente se x .
 - se $s(g) = \neg$ e $(h, g) \in A$ allora genera $(\neg g \vee \neg h) \wedge (g \vee h)$, ovvero g se e solo se $\neg h$.
 - se $s(g) = \vee$ e $(h, g), (k, g) \in A$ allora genera $(\neg h \vee g) \wedge (\neg k \vee g) \wedge (h \vee k \vee \neg g)$, ovvero g se e solo se $(h \vee k)$.
 - se $s(g) = \wedge$ e $(h, g), (k, g) \in A$ allora genera $(\neg g \vee h) \wedge (\neg g \vee k) \wedge (\neg h \vee \neg k \vee g)$ ovvero g se e solo se $(h \wedge k)$.

La dimostrazione che la trasformazione richiede spazio logaritmico si basa sulle stesse osservazioni fatte per mostrare che HAM si riduce a SAT.

Teorema 10.2: Riduzioni in spazio logaritmico

- i) CIRCUIT VALUE $\leq_{LOGSPACE}$ SAT.
- ii) CIRCUIT VALUE $\leq_{LOGSPACE}$ CRICCA.

Definizione 10.9: Tabella di computazione di una MdT

La tabella di computazione T_M di una macchina di Turing M a 1 nastro deterministica, o semplicemente T quando non vi sia ambiguità, è una matrice quadrata il cui indice di riga i rappresenta l' i -esimo passo di computazione e il cui indice di colonna j rappresenta la j -esima posizione sul nastro. Di conseguenza, la riga i -esima rappresenta la configurazione di M dopo il passo $i - 1$ e l'elemento $T(i, j)$ contiene il simbolo contenuto nella cella j -esima del nastro dopo $i - 1$ passi di computazioni.

Se M decide il problema I in tempo polinomiale deterministico $|x|^k$, la sua tabella di computazione su x ha al massimo $|x|^k$ righe e altrettante colonne. Imponiamo le seguenti convenzioni:

1. M si arresta prima di $n^k - 2$ passi
2. il nastro è riempito con tanti $\#$ quanti ne servono per arrivare alla casella in colonna n^k
3. arricchiamo l'alfabeto di M in modo che la casella $T(i, j)$ contenga il nuovo simbolo σ_q per registrare che nella configurazione i -esima la testina è sulla j -esima casella, il simbolo letto è σ e lo stato corrente è q
4. la configurazione iniziale parte da $\triangleright \sigma_{q_0} w$ e si saltano i passi di computazione che portano la testina sul respingente per poi riportarla sul primo carattere, eccezion fatta nel caso in cui la macchina termini.
5. se $T(i, j) \in \{\sigma_{SI}, \sigma_{NO}\}$ allora "sposta" il cursore fino alla seconda colonna in al massimo $O(|x|^k)$ passi introducendo un nuovo stato di finto arresto.
6. se σ_{SI} oppure σ_{NO} appaiono nella riga $p \leq |x|^k$ e nella seconda colonna, allora tutte le righe di indice q , $p \leq q \leq |x|^k$ sono uguali alla p -esima.

Osservazione il valore di tutti gli altri $T(i, j)$ dipende solo da 3 caselle; quelle della riga precedente, nella stessa posizione o immediatamente adiacenti, ovvero $T(i - 1, j - 1), T(i - 1, j), T(i - 1, j + 1)$.

10.2 \mathcal{P} -completezza

Teorema 10.3: CIRCUIT VALUE è \mathcal{P} -completo

CIRCUIT VALUE è $\leq_{LOGSPACE}$ -completo per \mathcal{P} .

Dimostrazione: sappiamo già che CIRCUIT VALUE $\in \mathcal{P}$, quindi prendiamo un qualunque $I \in \mathcal{P}$ e facciamo vedere che c'è una riduzione $f \in LOGSPACE$ che lo trasforma in CIRCUIT VALUE; ovvero $x \in I$ se e solamente se $f(x)$ è un circuito senza variabili il cui valore è tt .

Sia M una MdT che decide I in n^k , e T la sua tabella di computazione su x e inoltre Σ' l'alfabeto di M unito ai nuovi simboli $\sigma_q \in \Sigma \times (Q \cup \{\epsilon\})$ che si usano per costruire la tabella T .

Si costruisce un circuito a partire dalla tabella di computazione di M :

1. codifichiamo ogni simbolo in una sequenza di bit $(S_1, \dots, S_m) \in \{tt, ff\}^m, m = \lceil \log \#\Sigma' \rceil$.
2. la tabella così ottenuta ha $|x|^k$ righe di stringhe di bit lunghe $m \times |x|^k$.
3. rappresentiamo ciascun elemento della tabella S_{ijl} con $1 \leq i, j \leq |x|^k$ e $1 \leq l \leq m$. Di conseguenza $\forall i. S_{i11} \dots S_{i1m}$ codifica il simbolo \triangleright e $S_{i|x|^k 1} \dots S_{i|x|^k m}$ codifica il simbolo $\#$ e il resto dipende dai 3 bit della riga sopra, come visto nella tabella di computazione.
4. ci sono m funzioni booleane con $3m$ ingressi ciascuna che calcolano il bit della stringa successiva.

5. dato che per ogni funzione booleana esiste un circuito booleano che la calcola, costruiamo il circuito C raggruppando le F_1, \dots, F_m funzioni in una unica funzione che restituisce m valori a fronte degli stessi $3m$ ingressi.

A questo punto definiamo la riduzione f da I a CIRCUIT VALUE: trasformiamo la tabella di computazione T in un circuito C_I che è composto da tanti circuiti \bar{C} per ogni $T(i, j)$. In realtà ne bastano $(|x|^k - 1) \times (|x|^k - 2)$ perché la prima riga, e la prima e ultima colonna di T sono fissate. Le uscite $\bar{C}_{i-1,j-1}, \bar{C}_{i-1,j}, \bar{C}_{i-1,j+1}$ sono gli ingressi di \bar{C}_{ij} . Gli ingressi di C_I sono quelli di $T(1, j), T(i, 1), T(i, |x|^k)$ che sono determinati da x o sono fissi. I simboli σ_{SI}, σ_{NO} vengono codificati con stringhe di soli tt o ff . Poiché abbiamo ipotizzato che la tabella T contiene sempre in posizione $T(|x|^k, 2)$, l'uscita di C_I è una delle uscite di $\bar{C}_{|x|^k, 2}$.

Ora che abbiamo costruito C_I verifichiamo che sia corretto cioè che $C_I = f(x) = tt$ se e solo se $x \in I$. La dimostrazione procede per induzione sul numero i di passi della computazione:

Caso base $i = 1$: gli ingressi sono le uscite e quindi ok.

Passo induttivo $i-1 \Rightarrow i$: assumo che venga correttamente calcolato il passo $i-1$, cioè $\forall j. \bar{C}_{i-1,j}$ ha come uscite $R_1 \dots R_m$ se e solo se $T(i-1, j) = \rho' \in \Sigma'$ è codificato come proprio $R_1 \dots R_m$. Il che verificato per costruzione applicando l'ipotesi induttiva la dimostrazione è fatta.

Infine vediamo che lo spazio impiegato dalla riduzione è logaritmico. Per calcolare f dobbiamo costruire e connettere tra loro:

1. le porte di ingresso;
2. gli elementi della prima e dell'ultima colonna che sono $2 \times |x|^k$ circuiti costanti;
3. $(|x|^k - 1) \times (|x|^k - 2)$ copie del circuito \bar{C} . A ciascuna copia associamo gli appropriati indici, per metterla in relazione con la casella di computazione corrispondente; tali indici sono tutti minori di x^k e averli rappresentati in binario ci consente di manipolarli in spazio $O(\log|x|)$.

Definizione 10.10: Problema MONOTONE CIRCUIT VALUE

Il problema MONOTONE CIRCUIT VALUE è una variant e di CIRCUIT VALUE che non utilizza la negazione.

Intuizione: i circuiti senza \neg sono meno espressivi di quelli con, ma sono altrettanto difficili da valutare.

Teorema 10.4: MONOTONE CIRCUIT VALUE è \mathcal{P} -completo

MONOTONE CIRCUIT VALUE è \mathcal{P} -completo.

Dimostrazione: mostriamo che CIRCUIT VALUE $\leqslant_{LOGSPACE}$ MONOTONE CIRCUIT VALUE cioè trasformiamo un circuito qualunque in uno monotono equivalente. Ciò viene fatto applicando le regole di De Morgan partendo dalle porte di uscita a scendere fino alle porte di ingresso. Le porte di ingresso tt rimpiazzano se necessario quelle ff e viceversa, e se ne aggiungono di nuove se servissero. Tutto questo si fa ovviamente in $LOGSPACE$: basta visitare una sola volta le porte, rappresentate come coppie (i, j) dove i e j sono indici di "livello e colonna", rappresentati in binario.

10.3 \mathcal{NP} -completezza

Teorema 10.5: Cook

SAT è \mathcal{NP} -completo.

Dimostrazione: sappiamo già che $SAT \in \mathcal{NP}$ perché abbiamo visto una procedura che realizza una ricerca esaustiva che lo decide in tempo polinomiale non deterministico.

Poiché $CIRCUIT\ SAT \leqslant LOGSPACE\ SAT$, basta dimostrare che $CIRCUIT\ SAT$ è \mathcal{NP} -completo, cioè che $\forall I \in \mathcal{NP}$ si ha che $I \leqslant LOGSPACE\ SAT$.

Sia allora $I \in \mathcal{NP}$. Costruiamo $f \in LOGSPACE$ tale che $x \in I$ se e solamente se $f(x)$ è soddisfacibile. Per ipotesi c'è una MdT non deterministica N che decide I in tempo n^k . Per semplicità fissiamo il grado di non determinismo sia esattamente uguale a 2, cioè che ad ogni passo vi siano sempre 2 scelte possibili. Codifichiamo la prima scelta con il bit ff e la seconda con il bit tt . Una computazione è allora una successione di bit della forma:

$$B = b_0 b_1 \dots b_{|x|^k - 1} \text{ con } b_i \in tt, ff$$

La definizione di tabella di computazione che abbiamo dato, non contempla il non determinismo, ma possiamo costruirla se fissiamo una successione di scelte B : possiamo costruire T in funzione di N , dell'input x e la successione di scelte B .

Come fatto per $CIRCUIT\ VALUE$, la prima riga, la prima colonna e l'ultima colonna sono fissate. Inoltre $T(i, j)$ dipende, oltre che da $T(i - 1, j - 1), T(i - 1, j)$ e da $T(i - 1, j + 1)$ anche da b_{i-1} , cioè dalla scelta fatta al passo precedente. Quindi il circuito C_N ha stavolta $3m + 1$ ingressi e m uscite. Possiamo costruire in $LOGSPACE$ un circuito $f(x)$ che ha come porte di ingresso le scelte non deterministiche codificate nel vettore B e i valori della riga $T(1, j)$ ottenuti dal dato di ingresso x .

Infine $f(x)$ è soddisfacibile se e solo se $x \in I$, come abbiamo visto nel teorema 10.3 di \mathcal{P} -completezza.

I seguenti problemi sono $\leqslant LOGSPACE$ -completi per \mathcal{NP} :

- HAM.
- CRICCA.
- Problema del Commesso Viaggiatore.
- Programmazione Intera.

ESERCIZI RIDUZIONE

TEO. DI RICORSIONE
e DEL PARAMETRO

FUNZIONE $f(x) = \lambda$ CHE ASSUME LO STESSO VALORE
INDIPENDENTEMENTE DA x

- DIMOSTRARE CHE $\text{CONST} = \{x \mid \varphi_x \text{ TOTALE e COSTANTE}\}$ NON È RICORSIVO

SI DEVE DIMOSTRARE $\text{L} \leq_f \text{CONST}$ CON $\text{L} = \{x \mid \varphi_x(n) \downarrow\}$

ALLORA SI DEFINISCE LA SEGUENTE FUNZIONE PARZIALE: $\psi(x, y) = \begin{cases} 1 & \text{SE } \exists z > y \text{ TALE CHE } \varphi_z(n) \downarrow \text{ IN MENO DI } 2 \text{ PASSI} \\ \text{INDEFINITA} & \text{ALTRIMENTI} \end{cases}$

POTEVI SCRIVERCI
ANCHE $\lambda x. k$

$\psi(x, y)$ È CALCOLABILE, PERCHÉ DATO x RECUPERO LA MDT x E LI SI APPLICA COME ARGOMENTO x ; SE CONVERGE E IL NUMERO DI PASSI È $> y$ RESTITUISCI \perp . ALTRIMENTI RESTITUISCI INDEFINITO.

DATO CHE È CALCOLABILE PER CHURCH-TURING ESISTE UN INDICE u TALE CHE $\varphi_u(x, y) = \psi(x, y)$.

ADesso si applica il teorema del parametro $\psi_{S(u, x)}(y) = \varphi_u(x, y)$.

L'INDICE u È COSTANTE PER CUI $S(u, x) = f(x)$ ALLORA $\varphi_{f(x)}(y)$

A QUESTO PUNTO ABBIANO LA $f(x)$ E DOBBIANO VERIFICARE CHE $x \in \text{L} \Leftrightarrow f(x) \in \text{CONST}$

PRENDE IL T^h RAMO E ABBANDONI
IL A PRESCINDERE DA X E Y
PER CUI AD OGNI Y SI ASSOCIA 1

$x \in \text{L} \Rightarrow \varphi_{f(x)}(y) = \psi(x, y) = \lambda y. 1 \Rightarrow f(x) \in \text{INDICE DI UNA FUNZIONE TOTALE e COSTANTE} \text{ CIOÈ } \varphi_{f(x)} \text{ È UNA FUNZIONE}$

CONSTANTEMENTE UGUALE A 1 ED È TOTALE PERCHÉ È DEFINITA $\forall y$ ALLORA $f(x) \in \text{CONST}$

$x \notin \text{L} \Rightarrow \varphi_{f(x)}(y) = \lambda y. \text{INDEFINITA} \Rightarrow \text{LA FUNZIONE È INDEFINITA } \forall y \text{ E QUINDI } f(x) \in \text{CONST}$

- DIMOSTRARE CHE $\text{INF} = \{x \mid \text{dom}(\varphi_x) \text{ È INFINTO}\}$ SI RIDUCE A CONST

$\text{INF} = \{x \mid \text{dom}(\varphi_x) \text{ È INFINTO}\}$, STIAMO LAVORANDO SUI NUMERI NATURALI CHE SONO UN INSIEME INFINTO CHE PARTE DA 0 → N

SE RIUSCO A DIRE CHE $\forall y$ IN PIÙ, PER UN QUALSIASI y , TROVO SEMPRE ELEMENTO DI φ_x ALLORA $\text{dom}(\varphi_x)$ È INFINTO

SI DEVE DIMOSTRARE CHE $\text{INF} \leq_{\text{REC}} \text{CONST}$ CON $\text{CONST} = \{x \mid \varphi_x \text{ TOTALE e COSTANTE}\}$

SI DEFINISCE LA SEGUENTE FUNZIONE PARZIALE $\psi(x, y) = \begin{cases} 1 & \text{SE } \exists z > y \text{ TALE CHE } \varphi_x(z) \downarrow \\ \text{INDEFINITA} & \text{ALTRIMENTI} \end{cases}$

$\psi(x, y)$ È CALCOLABILE, PERCHÉ DATO x RECUPERO LA MDT x E LA SI APPLICA COME ARGOMENTO $y+1, y+2, \dots$; SE \exists ALCUNO UNO CHE \downarrow RESTITUISCI \perp . ALTRIMENTI RESTITUISCI INDEFINITO.

DATO CHE È CALCOLABILE PER CHURCH-TURING ESISTE UN INDICE u TALE CHE $\varphi_u(x, y) = \psi(x, y)$.

ADesso si applica il teorema del parametro $\psi_{S(u, x)}(y) = \varphi_u(x, y)$.

L'INDICE u È COSTANTE PER CUI $S(u, x) = f(x)$ ALLORA $\varphi_{f(x)}(y)$

A QUESTO PUNTO ABBIANO LA $f(x)$ E DOBBIANO VERIFICARE CHE $x \in \text{INF} \Leftrightarrow f(x) \in \text{CONST}$

$x \in \text{INF} \Rightarrow \varphi_{f(x)}(y) = \psi(x, y) = \lambda y. 1 \Rightarrow f(x) \in \text{INDICE DI UNA FUNZIONE COSTANTE} \text{ QUINDI } f(x) \in \text{CONST}$

$x \notin \text{INF} \Rightarrow \varphi_{f(x)}(y) = \lambda y. \text{INDEFINITA} \Rightarrow \varphi_{f(x)} \text{ NON È TOTALE PERCHÉ } \exists y \text{ TALE PER CUI LA FUNZIONE È INDEFINITA QUINDI } f(x) \in \text{CONST}$

- DIMOSTRARE CHE $\text{TOT} = \{x \mid \varphi_x \text{ TOTALE}\}$ SI RIDUCE A INF

STIAMO LAVORANDO SUI NUMERI NATURALI SE TROVO L'INSIEME CON Y, TUTTI I NUMERI CHE SONO PRIMA DI Y DEVONO STARE DENTRO IL DOMINIO DI φ_x

SI DEVE DIMOSTRARE $\text{TOT} \leq_{\text{REC}} \text{INF}$ CON $\text{INF} = \{x \mid \text{dom}(\varphi_x) \text{ È INFINTO}\}$

SI DEFINISCE LA SEGUENTE FUNZIONE PARZIALE $\psi(x, y) = \begin{cases} 1 & \text{SE } \forall z < y \varphi_x(z) \downarrow \\ \text{INDEFINITA} & \text{ALTRIMENTI} \end{cases}$

$\psi(x, y)$ È CALCOLABILE, PERCHÉ DATO x RECUPERO M_x E LI SI APPLICA COME ARGOMENTO TUTTI I NATURALI $\leq y$. SE CONVERGE SEMPRE, RESTITUISCI \perp . SE IN NUMERO FINITO

ALTRIMENTI RESTITUISCI INDEFINITO. C'È BISOGNO DEL BUON ORDINAMENTO CIOÈ $\forall y [0, y] \subset \mathbb{N}$

DATO CHE È CALCOLABILE PER CHURCH-TURING ESISTE UN INDICE u TALE CHE $\varphi_u(x, y) = \psi(x, y)$.

ADDESSO SI APPLICA IL TEOREMA DEL PARAMETRO $\varphi_{S(u,x)}(y) = \varphi_u(x,y)$.

L'INDICE U È COSTANTE PER CUI $S(u,x) = f(u)$ ALLORA $\varphi_{f(u)}(y)$

A QUESTO PUNTO ABBIANO LA $f(u)$ E DOBBIANO VERIFICARE CHE $x \in \text{TOT} \Leftrightarrow f(u) \in \text{INF}$

UNA FUNZIONE COSTANTEMENTE = 1 HA IL DOMINIO IN TUTTA N

$x \in \text{TOT} \Rightarrow \varphi_{f(u)}(y) = \psi(x,y) = \lambda y. 1 \Rightarrow f(u)$ È INDICE DI UNA FUNZIONE COSTANTE E QUINDI $f(u)$ HA DOMINIO INFINTO

$x \in \text{TOT} \Rightarrow \varphi_{f(u)}(y) = \lambda y. \text{INDEFINITO} \Rightarrow f(u)$ NON È TOTALE PERCHÉ $\exists y$ PER CUI LA FUNZIONE NON È DEFINITA

- DIMOSTRARE CHE $\text{TOT} = \{x \mid \varphi_x \text{ TOTALE}\}$ SI RIDUCE A CONST

SI DEVE DIMOSTRARE $\text{TOT} \leq_{\text{REC}} \text{CONST}$ CON $\text{CONST} = \{x \mid \varphi_x \text{ È TOTALE E COSTANTE}\}$

PER ESSERE COSTANTE UNA FUNZIONE NON DEVE SOLO CONVERGIRE PER OGNI z, y MA DEVE AVERE ANCHE LO STESSO VALORE

SI DEFINISCE LA SEGUENTE FUNZIONE PARZIALE $\psi(x,y) = \begin{cases} 1 & \text{SE } \forall z \leq y \quad \varphi_x(z) \downarrow \text{ E } \varphi_x(z) = \varphi_x(z+1) \\ \text{INDEFINITA} & \text{ALTRIMENTI} \end{cases}$

$\psi(x,y)$ È CALCOLABILE, PERCHÉ DATO X RECUPERO M_x E LI SI APPLICA COME ARGOMENTO TUTTI I NATURALI $\leq y$, TALI PER CUI DEVE CONVERGIRE E $\varphi_x(z) = \varphi_x(z+1)$; INDEFINITA ALTRIMENTI.

DATO CHE È CALCOLABILE PER CHURCH-TURING ESISTE UN INDICE U TALE CHE $\varphi_u(x,y) = \psi(x,y)$.

ADDESSO SI APPLICA IL TEOREMA DEL PARAMETRO $\varphi_{S(u,x)}(y) = \varphi_u(x,y)$.

L'INDICE U È COSTANTE PER CUI $S(u,x) = f(u)$ ALLORA $\varphi_{f(u)}(y)$

A QUESTO PUNTO ABBIANO LA $f(u)$ E DOBBIANO VERIFICARE CHE $x \in \text{TOT} \Leftrightarrow f(u) \in \text{CONST}$

$x \in \text{TOT} \Rightarrow \varphi_{f(u)}(y) = \psi(x,y) = \lambda y. 1 \Rightarrow f(u)$ È INDICE DI UNA FUNZIONE COSTANTE QUINDI $f(u) \in \text{CONST}$

$x \in \text{TOT} \Rightarrow \varphi_{f(u)}(y) = \lambda y. \text{INDEFINITO} \Rightarrow \varphi_{f(u)}$ NON È TOTALE PERCHÉ $\exists y$ TALE PER CUI LA FUNZIONE È INDEFINITA QUINDI $f(u) \notin \text{CONST}$

- DIMOSTRARE CHE $K = \{x \mid \varphi_x(w) \downarrow\}$ SI RIDUCE A TOT

SI DEVE DIMOSTRARE $K \leq_{\text{REC}} \text{TOT}$ CON $\text{TOT} = \{x \mid \varphi_x \text{ TOTALE}\}$

SI DEFINISCE LA SEGUENTE FUNZIONE PARZIALE $\psi(x,y) = \begin{cases} 1 & x \in K \\ \text{INDEFINITA} & x \notin K \end{cases}$

CHE È CALCOLABILE, DATO X RECUPERO LA MACCHINA M_x , LE PASSO COME ARGOMENTO X, SE CONVERGE RESTITUISCI 1; INDEFINITO ALTRIMENTI.

DATO CHE È CALCOLABILE PER CHURCH-TURING AVRÀ UN INDICE U TALE CHE $\varphi_u(x,y) = \psi(x,y)$

ADDESSO SI APPLICA IL TEOREMA DEL PARAMETRO $\varphi_{S(u,x)}(y) = \varphi_u(x,y)$

L'INDICE U È COSTANTE PER CUI $S(u,x) = f(u)$ ALLORA $\varphi_{f(u)}(y) = \varphi_{S(u,x)}(y)$

A QUESTO PUNTO ABBIANO LA $f(u)$ E DOBBIANO VERIFICARE CHE $x \in K \Leftrightarrow f(u) \in \text{TOT}$

$x \in K \Rightarrow \varphi_{f(u)}(y) = \psi(x,y) = \lambda y. 1 \Rightarrow f(u)$ È L'INDICE DI UNA FUNZIONE COSTANTEMENTE UGUALE A 1, QUINDI $\varphi_{f(u)}$ È TOTALE PERCHÉ È DEFINITA Y

$x \notin K \Rightarrow \varphi_{f(u)}(y) = \lambda y. \text{INDEFINITA} \Rightarrow \varphi_{f(u)}$ NON È TOTALE PERCHÉ $\exists y$ TALE PER CUI $\varphi_{f(u)}$ NON È DEFINITA

ALTRI ESERCIZI

- CARATTERIZZARE $I = \{ i \mid \text{dom}(\varphi_i) = \{3\}\}$

PER VEDERE SE UN INSIEME NON È RICORSIVO CI SONO DUE STRADE: PRE-RICE o $K \leq_{\text{REC}} I$

SI UTILIZZA PRE-RICE: "SIA I UN INSIEME DI INDICI :iff TALE CHE $\emptyset \neq I \neq \mathbb{N}$, ALLORA $K \leq_{\text{REC}} I \circ K \leq_{\text{REC}} \bar{I}$ "

SI DEVE DIMOSTRARE: $I \neq \emptyset$, PERCHÉ POSSIAMO DEFINIRE UNA FUNZIONE CALCOLABILE CHE SI PERMA SOLO SE PASSIAMO 3 IN INPUT

$$\Rightarrow f(x) = \begin{cases} 1 & \text{SE } x=3 \\ \text{INDEF ALTRIMENTI} & \end{cases} \quad \begin{array}{l} \leftarrow \text{if } x=3 \text{ then true} \\ \text{else while(true)} \end{array}$$

$I \neq \mathbb{N}$, PERCHÉ POSSIAMO DEFINIRE UNA FUNZIONE CALCOLABILE CHE NON HA COME DOMINIO ESATTAMENTE 3

$$\Rightarrow f(x) = \frac{1}{x-3} \quad \text{dom } f = \mathbb{N} \setminus \{3\} \quad \text{OPPURE} \quad f(x) = \begin{cases} 1 & \forall x \\ \text{INDEF ALTRIMENTI} & \end{cases}$$

$I = \text{iff cioè } \forall x, y \in I \text{ se } x \in I \text{ e } \varphi_x = \varphi_y \text{ ALLORA } y \in I$.

NOI SAPPIAMO CHE $\text{dom}(\varphi_n) = \{3\} \subset \forall z \varphi_n(z) = \varphi_3(z)$.

L'UNICO Z PER CUI φ_n CONVERGE È 3 QUINDI L'UQAGLIANZA È VERA ANCHE PER φ_3 , DUNQUE ANCHE $y \in I$

SI UTILIZZA $K \leq_{\text{REC}} I$. QUINDI $x \in K \Leftrightarrow f(x) \in I$ e $x \in K \Leftrightarrow f(x) \notin I$

SI DEFINISCE LA SEGUENTE FUNZIONE PARZIALE $\psi(x,y) = \begin{cases} 1 & \text{SE } \varphi_x(n) \downarrow \text{ e } y=3 \\ \text{INDEF ALTRIMENTI} & \end{cases}$

CHE È CALCOLABILE, DATO X RECUPERO LA MACCHINA M_x , LE PASSO COME ARGOMENTO X, SE CONVERGE E Y=3, RESTITUISCI 1
INDEFINITO ALTRIMENTI.

DATO CHE È CALCOLABILE PER CHURCH-TURING AVRA' UN INDICE U TALE CHE $\varphi_u(x,y) = \psi(x,y)$

ADESSO SI APPLICA IL TEOREMA DEL PARAMETRO $\varphi_{S(u,x)}(y) = \varphi_u(x,y)$

L'INDICE U È COSTANTE PER CUI $S(u,x) = f(x)$ ALLORA $\varphi_{f(x)}(y) = \varphi_{S(u,x)}(y)$

A QUESTO PUNTO ABBIAMO LA $f(x)$ E DOBBIAMO VERIFICARE CHE $x \in K \Leftrightarrow f(x) \in I$

$$x \in K \Rightarrow \varphi_{f(x)}(y) = \begin{cases} 1 & y=3 \\ \text{INDEF ALTRIMENTI} & \end{cases} \quad \text{QUINDI } \text{dom}(\varphi_{f(x)}) = 3 \text{ E ALLORA } f(x) \in I$$

$x \in K \Rightarrow \varphi_{f(x)}(y) = \lambda y. \text{INDEFINITA} \Rightarrow \text{LA FUNZIONE È OVUNQUE INDEFINITA, QUINDI } f(x) \notin I$

CONTROLLIAMO SE I È RICORSIVAMENTE ENUMERABILE.

PER FARLO SI UTILIZZA LA RIDUZIONE $\bar{K} \leq_{\text{REC}} I$ CHE EQUIVALE AD $K \leq_{\text{REC}} \bar{I}$

SI DEFINISCE LA SEGUENTE FUNZIONE PARZIALE $\psi(x,y) = \begin{cases} 1 & \text{SE } \varphi_x(n) \downarrow \text{ o } y=3 \\ \text{INDEF ALTRIMENTI} & \end{cases}$

CHE È CALCOLABILE, DATO X RECUPERO LA MACCHINA M_x , LE PASSO COME ARGOMENTO X, SE CONVERGE O Y=3, RESTITUISCI 1
INDEFINITO ALTRIMENTI.

DATO CHE È CALCOLABILE PER CHURCH-TURING AVRA' UN INDICE U TALE CHE $\varphi_u(x,y) = \psi(x,y)$

ADESSO SI APPLICA IL TEOREMA DEL PARAMETRO $\varphi_{S(u,x)}(y) = \varphi_u(x,y)$

L'INDICE ω È COSTANTE PER CUI $s(\omega, x) = f(x)$ ALLORA $\varphi_{f(\omega)}(y) = \varphi_{s(\omega, y)}(y)$

A QUESTO PUNTO ABBIANO LA $f(\omega)$ E DOBBIANO VERIFICARE CHE $x \in L \Leftrightarrow f(x) \in I$

$$x \in L \Rightarrow \varphi_{f(\omega)}(y) = \varphi_{s(\omega, y)}(y) = \lambda y. 1 \Rightarrow \text{dom}(\varphi_{f(\omega)}) = \mathbb{N} \text{ E QUINDI } f(\omega) \in I$$

$$x \notin L \Rightarrow \varphi_{f(\omega)}(y) = \begin{cases} 1 & \text{SE } y=3 \\ \text{INDEF ALTRIMENTI} & \end{cases} \Rightarrow \text{dom}(\varphi_{f(\omega)}) = \{3\} \text{ E QUINDI } f(\omega) \notin I$$

- ESISTE UNA FUNZIONE f TALE CHE $\forall x$ IL DOMINIO DI $\varphi_{f(\omega)}$ È L'INSIEME DEI NUMERI NATURALI?

SÌ PERCHÉ ESISTE LA FUNZIONE CHE MI RESTITUISCE UN INDICE TALE PER CUI $\varphi_{f(\omega)}$ È UNA FUNZIONE COSTANTE
E A QUESTO PUNTO IL DOMINIO È TUTTO \mathbb{N} $\Rightarrow f(\omega) = \lambda x. i$