Informatica Teorica

Massimo Perego

Indice

Introduzione **2**

1 Teoria della Calcolabilità **4**
1.1 Notazione 4
1.1.1 Funzioni 4
1.1.2 Prodotto Cartesiano 6
1.1.3 Funzione di Valutazione 7
1.2 Sistemi di Calcolo 7
1.3 Potenza Computazionale 8
1.4 Relazioni di Equivalenza 9
1.4.1 Partizione indotta dalla relazione di equivalenza . . . 9
1.4.2 Classi di equivalenza e Insieme quoziente 9
1.5 Cardinalità 10
1.5.1 Isomorfismi 10
1.5.2 Cardinalità finita 11
1.5.3 Cardinalità infinita 11
1.6 Potenza Computazionale di un sistema di calcolo 16
1.6.1 Validità dell'inclusione $F(\mathcal{C}) \subseteq \mathrm{DATI}_{\perp}^{\mathrm{DATI}}$ 16
1.7 DATI $\sim \mathbb{N}$ 16
1.7.1 Funzione Coppia di Cantor 17
1.7.2 Applicazione alle strutture dati 20
1.8 PROG $\sim \mathbb{N}$ 22
1.8.1 Sistema di calcolo RAM 23
1.8.2 Aritmetizzazione di un programma 27

Introduzione

Si "contrappone" all'informatica applicata, ovvero qualsiasi applicazione dell'informatica atta a raggiunger uno scopo, dove l'informatica è solamente lo strumento per raggiungere in maniera efficace un obiettivo.
Con "*informatica teorica*" l'oggetto è l'informatica stessa, si studiano i fondamenti della disciplina in modo rigoroso e scientifico. Può essere fatto ponendosi delle questioni fondamentali: il *cosa* e il *come* dell'informatica, ovvero cosa è in grado di fare l'informatica e come è in grado di farlo.

Cosa: L'informatica è "la disciplina che studia l'informazione e la sua elaborazione automatica", quindi l'oggetto sono l'informazione e i dispositivi di calcolo per gestirla; scienza dell'informazione. Diventa lo studio come risolvere automaticamente un problema. Ma tutti i problemi sono risolvibili in maniera automatica? Cosa è in grado di fare l'informatica?

La branca dell'informatica teorica che studia cosa è risolvibile si chiama **Teoria della Calcolabilità**, studia cosa è calcolabile per via automatica. Spoiler: non tutti i problemi sono risolvibili per via automatica, e non potranno mai esserlo per limiti dell'informatica stessa. Cerchiamo una caratterizzazione generale di cosa è calcolabile e cosa no, si vogliono fornire strumenti per capire ciò che è calcolabile. La caratterizzazione deve essere fatta matematicamente, in quanto il rigore e la tecnica matematica permettono di trarre conclusioni sull'informatica.

Come: Una volta individuati i problemi calcolabili, come possiamo calcolarli? Il dominio della **Teoria della Complessità** vuole descrivere le risoluzione dei problemi tramite mezzi automatici in termini di risorse computazionali necessarie. Una "risorsa computazionale" è qualsiasi cosa che viene consumata durante l'esecuzione per risolvere il problema, come pos-

sono essere elettricità o numero di processori, generalmente i parametri più importanti considerati sono tempo e spazio di memoria. Bisognerà definire in modo preciso cosa si intende con "tempo" e "spazio". Una volta fissati i parametri bisogna definire anche cosa si intende con "risolvere efficientemente" un problema, in termini di tempo e spazio.

La teoria della calcolabilità dice quali problemi sono calcolabili, la teoria della complessità dice, all'interno dei problemi calcolabili, quali sono risolvibili efficientemente.

Capitolo 1

Teoria della Calcolabilità

1.1 Notazione

1.1.1 Funzioni

Funzione: Una funzione f dall'insieme A all'insieme B è una legge che dice come associare a ogni elemento di A un elemento di B. Si scrive

$$f : A \to B$$

E chiamiamo A dominio e B codominio. Per dire come agisce su un elemento si usa $f(a) = b$, b è l'immagine di a secondo f (di conseguenza a è la controimmagine).
Per definizione di funzione, è possibile che elementi del codominio siano raggiungibili da più elementi del dominio, ma non il contrario. Possiamo classificare le funzioni in base a questa caratteristica:

- **Iniettiva:** $f : A \to B$ è iniettiva sse $\forall a, b \in A,\ a \neq b \implies f(a) \neq f(b)$
- **Suriettiva:** $f : A \to B$ è suriettiva sse $\forall b \in B,\ \exists a \in A : f(a) = b$: un altro modo per definirla è tramite l'insieme immagine di f, definito come
  $$\mathrm{Im}_f = \{b \in B : \exists a, f(a) = b\} = \{f(a) : a \in A\}$$
  Solitamente $\mathrm{Im}_f \subseteq B$, ma f è suriettiva sse $\mathrm{Im}_f = B$;
- **Biettiva:** $f : A \to B$ è biettiva sse è sia iniettiva che suriettiva, ovvero

$$\begin{array}{ll} \forall a, b \in A, a \neq b : & f(a) \neq f(b) \\ \forall b \in B, \exists a \in A : & f(a) = b \end{array} \implies \forall b \in B, \exists! a \in A : f(a) = b$$

Inversa: Per le funzioni biettive si può naturalmente associare il concetto di "inversa": dato $f : A \to B$ biettiva, si definisce inversa la funzione $f^{-1} : B \to A$ tale che $f^{-1}(b) = a \Leftrightarrow f(a) = b$.

Composizione di funzioni: Date $f : A \to B$ e $g : B \to C$, f composto g è la funzione $g \circ f : A \to C$ definita come $g \circ f(a) = g(f(a))$. Generalmente non commutativo, $f \circ g \neq g \circ f$, ma è associativo.

Funzione identità: Dato l'insieme A, la funzione identità su A è la funzione $i_A : A \to A$ tale che $i_A(a) = a$, $\forall a \in A$.

Un'altra possibile definizione per l'inversa diventa:

$$f^{-1} \circ f = i_A \wedge f \circ f^{-1} = i_B$$

Funzioni Parziali: Se una funzione $f : A \to B$ è definita per $a \in A$ si indica con $f(a) \downarrow$ e da questo proviene la categorizzazione: una funzione è **totale** se definita $\forall a \in A$, **parziale** altrimenti (definita solo per qualche elemento di A).

Insieme Dominio: Chiamiamo **dominio** (o campo di esistenza) di f l'insieme

$$\mathrm{Dom}_f = \{a \in A | f(a) \downarrow\} \subseteq A$$

Quindi se $\mathrm{Dom}_f = A$ la funzione è totale, se $\mathrm{Dom}_f \subsetneq A$ allora è una funzione parziale.

Totalizzazione: Si può **totalizzare una funzione parziale** f definendo una funzione a tratti $\overline{f} : A \to B \cup \{\perp\}$ tale che

$$\overline{f}(a) = \begin{cases} f(a) & a \in \mathrm{Dom}_f(a) \\ \perp & \text{altrimenti} \end{cases}$$

Dove $\perp$ è il **simbolo di indefinito**, per tutti i valori per cui la funzione di partenza f non è definita. Da qui in poi $B_\perp$ significa $B \cup \{\perp\}$.

Insieme delle funzioni: L'insieme di tutte le funzioni che vanno da A a B si denota con

$$B^A = \{f : A \to B\}$$

La notazione viene usata in quanto la cardinalità di B^A è esattamente $|B|^{|A|}$, con A e B insiemi finiti.
Volendo includere anche tutte le funzioni parziali:

$$B_\perp^A = \{f : A \to B_\perp\}$$

Le due definizioni coincidono, $B^A = B_\perp^A$, ma quest'ultima permette di mettere in evidenza che tutte le funzioni presenti sono totali o totalizzate.

1.1.2 Prodotto Cartesiano

Chiamiamo **prodotto cartesiano** l'insieme

$$A \times B = \{(a, b) | a \in A \wedge b \in B\}$$

Rappresenta l'insieme di tutte le coppie ordinate di valori in A e B. In generale non è commutativo, a meno che $A = B$.

Può essere esteso a n-uple di valori:

$$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) | a_i \in A_i\}$$

Il prodotto di n volte lo stesso insieme verrà, per comodità, indicato come

$$A \times \cdots \times A = A^n$$

Proiettore: Operazione "opposta", il proiettore i-esimo è una funzione che estrae l'i-esimo elemento di una tupla, quindi è una funzione

$$\pi_i : A_1 \times \cdots \times A_n \to A_i \text{ t.c. } \pi_i(a_1, \ldots, a_n) = a_i$$

La proiezione sull'asse in cui sono presenti i valori dell'insieme a_i.

1.1.3 Funzione di Valutazione

Dati A, B e $B_\perp^A$ si definisce **funzione di valutazione** la funzione

$$\omega : B_\perp^A \times A \to B \text{ t.c. } \omega(f, a) = f(a)$$

Prende una funzione f e la valuta su un elemento a del dominio. Si possono fare due tipi di analisi su questa funzione:

- Fisso a e provo tutte le f, ottenendo un *benchmark* di tutte le funzioni su a
- Fisso f e provo tutte le a del dominio, ottenendo il *grafico* di f

1.2 Sistemi di Calcolo

Vogliamo modellare teoricamente un **sistema di calcolo**; quest'ultimo può essere visto come una black box che prende in input un programma P, dei dati x e calcola il risultato y di P su input x. La macchina restituisce y se è riuscita a calcolare un risultato, $\perp$ (indefinito) se è entrata in un loop.

Quindi, formalmente, possiamo definire un sistema di calcolo come una funzione

$$\mathcal{C} : \text{PROG} \times \text{DATI} \to \text{DATI}_\perp$$

Possiamo vedere un sistema di calcolo come una funzione di valutazione:

- i dati x corrispondono all'input a
- il programma P corrisponde alla funzione f

Formalmente, un programma $P \in \text{PROG}$ è una sequenza di regole che trasformano un dato input in uno di output, ovvero l'espressione di una funzione secondo una sintassi

$$P : \text{DATI} \to \text{DATI}_\perp$$

e di conseguenza $P \in \text{DATI}_\perp^{\text{DATI}}$. In questo modo abbiamo mappato l'insieme PROG sull'insieme delle funzioni, il che ci permette di definire il sistema di calcolo come la funzione

$$\mathcal{C} : \text{DATI}_\perp^{\text{DATI}} \times \text{DATI} \to \text{DATI}$$

Analoga alla funzione di valutazione. Con $\mathcal{C}(P, x)$ indichiamo la funzione calcolata da P su x dal sistema di calcolo $\mathcal{C}$, che viene detta **semantica**, ovvero il suo "significato" su input x.

Il modello solitamente considerato quando si parla di calcolatori è quello di **Von Neumann**.

1.3 Potenza Computazionale

Indicando con

$$\mathcal{C}(P, \_) : \text{DATI} \to \text{DATI}$$

la funzione che viene calcolata dal programma P (semantica di P).

La **potenza computazionale** di un calcolatore è definita come l'insieme di tutte le funzioni che quel sistema di calcolo è in grado di calcolare, ovvero

$$F(\mathcal{C}) = \{\mathcal{C}(P, \_) | P \in \text{PROG}\} \subseteq \text{DATI}_{\perp}^{\text{DATI}}$$

Ovvero, l'insieme di tutte le possibili semantiche di funzioni calcolabili con il sistema $\mathcal{C}$. Stabilire il carattere di quest'ultima inclusione equivale a stabilire *cosa può fare l'informatica*:

- se $F(\mathcal{C}) \subsetneq \text{DATI}_{\perp}^{\text{DATI}}$ allora esistono compiti **non automatizzabili**
- se $F(\mathcal{C}) = \text{DATI}_{\perp}^{\text{DATI}}$ allora l'informatica *può fare tutto*

Calcolare funzioni vuol dire risolvere problemi *in generale*, a ogni problema è possibile associare una funzione soluzione che permette di risolverlo automaticamente.

Un possibile approccio per risolvere l'inclusione è tramite la **cardinalità** (funzione che associa ogni insieme al numero di elementi che contiene) dei due insiemi. Potrebbe però presentare dei problemi: è efficace solo quando si parla di insiemi finiti. Ad esempio, l'insieme dei numeri naturali contiene l'insieme dei numeri pari $\mathbb{P} \subsetneq \mathbb{N}$, ma $|\mathbb{N}| = |\mathbb{P}| = \infty$.
Serve una diversa definizione di cardinalità che considera l'esistenza di infiniti *più densi di altri*.

1.4 Relazioni di Equivalenza

Dati due insiemi A, B, una relazione binaria R è un sottoinsieme $R \subseteq A \times B$ di coppie ordinate. Data $R \subseteq A^2$, due elementi sono in relazione sse $(a, b) \in R$. Indichiamo la relazione tra due elementi anche con la notazione infissa aRb.

Una classe importante di relazioni è quella delle **relazioni di equivalenza**: una relazione $R \subseteq A^2$ è una relazione di equivalenza sse rispetta le proprietà di

- riflessività: $\forall a \in A,\ (a, a) \in R$
- simmetria: $\forall a, b \in A,\ (a, b) \in R \Leftrightarrow (b, a) \in R$
- transitività: $\forall a, b, c \in A,\ (a, b) \in R \land (b, c) \in R \implies (a, c) \in R$

1.4.1 Partizione indotta dalla relazione di equivalenza

A ogni relazione di equivalenza $R \subseteq A^2$ si può associare una **partizione**, ovvero un insieme di sottoinsiemi $A_i \subseteq A$ tali che

- $\forall i \in \mathbb{N}^+,\ A_i \neq \emptyset$
- $\forall i, j \in \mathbb{N}^+$, se $i \neq j$ allora $A_i \cap A_j = \emptyset$
- $\bigcup_{i \in \mathbb{N}^+} A_i = A$

La relazione R definita su A^2 *induce* una partizione $\{A_1, A_2, \dots\}$ su A.

1.4.2 Classi di equivalenza e Insieme quoziente

Dato un elemento $a \in A$, chiamiamo **classe di equivalenza** di a l'insieme

$$[a]_R = \{b \in A | (a, b) \in R\}$$

Ovvero, tutti gli elementi in relazione con a, chiamato **rappresentante** della classe.

Si può dimostrare che

- non esistono classi di equivalenza vuote, per riflessività

- dati $a, b \in A$, allora $[a]_R \cap [b]_R = \emptyset$, oppure $[a]_R = [b]_R$, i due elementi o sono in relazione o non lo sono
- $\bigcup_{a \in A} [a]_R = A$

L'insieme delle classi di equivalenza, per definizione, è una partizione indotta da R su A, detta **insieme quoziente** di A rispetto ad R, denotato con A/R.

1.5 Cardinalità

1.5.1 Isomorfismi

Due insiemi A e B sono **isomorfi** (*equi-numerosi*) se esiste una biezione tra essi, denotato come $A \sim B$. Chiamando $\mathcal{U}$ l'insieme di tutti gli insiemi, la relazione $\sim$ è $\sim \subseteq \mathcal{U}^2$.

Dimostriamo che $\sim$ è una relazione di equivalenza:

- riflessività: $A \sim A$, la biezione è data dalla funzione identità i_A
- simmetria: $A \sim B \Leftrightarrow B \sim A$, la biezione è data dalla funzione inversa
- transitività: $A \sim B \wedge B \sim C \implies A \sim C$, la biezione è data dalla composizione delle funzioni usate per $A \sim B$ e $B \sim C$

Dato che $\sim$ è una relazione di equivalenza, permette di partizionare l'insieme $\mathcal{U}$, risultando in classi di equivalenza contenenti insiemi isomorfi, ovvero con la stessa cardinalità. Possiamo quindi definire la **cardinalità** come l'insieme quoziente di $\mathcal{U}$ rispetto alla relazione $\sim$.

Questo approccio permette il *confronto delle cardinalità di insiemi infiniti*, basta trovare una funzione biettiva tra i due insiemi per poter affermare che sono isomorfi.

1.5.2 Cardinalità finita

La prima classe di cardinalità è quella delle cardinalità finite. Definiamo la seguente famiglia di insiemi:

$$J_n = \begin{cases} \emptyset & \text{se } n = 0 \\ \{1, \dots, n\} & \text{se } n > 0 \end{cases}$$

Un insieme A ha **cardinalità finita** sse $A \sim J_n$ per qualche $n \in \mathbb{N}$; in tal caso possiamo scrivere $|A| = n$. La classe di equivalenza $[J_n]_\sim$ identifica tutti gli insiemi di $\mathcal{U}$ contenenti n elementi.

1.5.3 Cardinalità infinita

L'altra classe di cardinalità è quella delle **cardinalità infinite**, ovvero gli insiemi non in relazione con J_n. Si possono dividere in **numerabili** e **non numerabili**.

Insiemi numerabili

Un insieme A è numerabile sse $A \sim \mathbb{N}$, ovvero $A \in [\mathbb{N}]_\sim$. Vengono anche detti **listabili**, in quanto è possibile elencare tutti gli elementi dell'insieme A tramite una funzione f biettiva tra $\mathbb{N}$ e A; grazie ad f possiamo elencare gli elementi di A, formando l'insieme

$$A = \{f(0), f(1) \dots \}$$

Ed è esaustivo, in quanto elenca tutti gli elementi di A.

Questi insiemi hanno cardinalità $\aleph_0$ (*aleph*).

Insiemi non numerabili

Gli insiemi non numerabili sono insiemi a cardinalità infinita ma non listabili, sono "più fitti" di $\mathbb{N}$; ogni lista generata non può essere esaustiva.

Il più noto tra gli insiemi non numerabili è l'insieme $\mathbb{R}$ dei numeri reali.

Teorema 1.5.1. *L'insieme $\mathbb{R}$ non è numerabile ($\mathbb{R} \nsim \mathbb{N}$)*

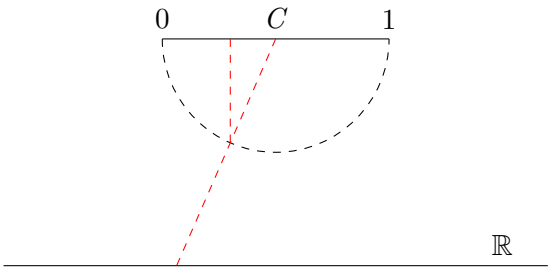
Dimostrazione. Suddividiamo la dimostrazione in 3 punti:

1. dimostriamo che $\mathbb{R} \sim (0,1)$
2. dimostriamo che $\mathbb{N} \nsim (0,1)$
3. dimostriamo che $\mathbb{R} \nsim \mathbb{N}$

Per dimostrare che $\mathbb{R} \sim (0,1)$ serve trovare una biezione tra $\mathbb{R}$ e $(0,1)$. Usiamo una rappresentazione grafica:

- disegnare una semicirconferenza di raggio $1/2$, centrata in $1/2$, quindi con diametro 1
- disegnare la perpendicolare al punto da mappare che interseca la circonferenza
- disegnare la semiretta passante per il centro C e l'intersezione precedente

L'intersezione tra asse reale (parallela al diametro) e semiretta finale è il punto mappato.



Questo approccio permette di dire che $\mathbb{R}$ è isomorfo a qualsiasi segmento di lunghezza maggiore di 0. La stessa biezione vale anche sull'intervallo chiuso $[0,1]$ (e di conseguenza qualsiasi intervallo chiuso), usando la "compattificazione" $\mathbb{R} = \mathbb{R} \cup \{\pm\infty\}$ e mappando 0 su $-\infty$ e 1 su $+\infty$.

Continuiamo dimostrando che $\mathbb{N} \nsim (0,1)$: serve dimostrare che l'intervallo $(0,1)$ non è listabile, quindi che ogni lista manca di almeno un elemento. Proviamo a "costruire" un elemento che andrà a mancare. Per assurdo, sia $\mathbb{N} \sim (0,1)$, allora possiamo listare gli elementi di $(0,1)$ come

$$\begin{array}{llllll} 0. & a_{00} & a_{01} & a_{02} & \dots \\ 0. & a_{10} & a_{11} & a_{12} & \dots \\ 0. & a_{20} & a_{21} & a_{22} & \dots \\ 0. & & \dots & & \end{array}$$

dove con a_{ij} indichiamo la cifra di posto j dell'i-esimo elemento della lista.

Costruiamo il numero $c = 0.c_0c_1\dots$ tale che

$$c_i = \begin{cases} 2 & \text{se } a_{ii} \neq 2 \\ 3 & \text{se } a_{ii} = 2 \end{cases}$$

Viene costruito "guardando" le cifre sulla diagonale principale, apparterrà sicuramente a $(0,1)$ ma differirà per almeno una posizione (quella sulla diagonale principale) da ogni numero presente all'interno della lista. Questo è assurdo sotto l'assunzione che $(0,1)$ è numerabile, quindi abbiamo provato che $\mathbb{N} \nsim (0,1)$.

Il terzo punto $\mathbb{R} \nsim \mathbb{N}$ si dimostra per transitività.

Più in generale, non si riesce a listare nessun segmento di lunghezza maggiore di 0.

□

Questa dimostrazione (punto 2 in particolare) è detta **dimostrazione per diagonalizzazione**.

L'insieme $\mathbb{R}$ viene detto **insieme continuo** e tutti gli insiemi isomorfi a $\mathbb{R}$ si dicono continui a loro volta.

Gli insiemi continui hanno cardinalità $\aleph_1$.

Insieme delle Parti

L'**insieme delle parti** di $\mathbb{N}$ (anche detto *power set*), è definito come

$$P(\mathbb{N}) = 2^{\mathbb{N}} = \{S | S \text{ è sottoinsieme di } \mathbb{N}\}$$

Teorema 1.5.2. $P(\mathbb{N}) \nsim \mathbb{N}$.

Dimostrazione. Possiamo dimostrare questo teorema tramite diagonalizzazione. Il vettore caratteristico di un sottoinsieme è un vettore che nella posizione p_i ha 1 se $i \in A$, 0 altrimenti (tipo vettore di incidenza).

Rappresentiamo $A \subseteq \mathbb{N}$ sfruttando il suo vettore caratteristico

$$\begin{array}{lcccccccc} \mathbb{N}: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & \dots \\ A: & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \dots \end{array}$$

Supponiamo, per assurdo, che $P(\mathbb{N})$ sia numerabile. Vista questa proprietà, possiamo listare tutti i vettori caratteristiche che appartengono a $P(\mathbb{N})$ come

$$\begin{array}{lcccccc} b_0 & = & b_{00} & b_{01} & b_{02} & \dots \\ b_1 & = & b_{10} & b_{11} & b_{12} & \dots \\ b_2 & = & b_{20} & b_{21} & b_{22} & \dots \end{array}$$

Vogliamo quindi costruire un vettore che appartiene a $P(\mathbb{N})$ ma non presente nella lista precedente. Definiamo

$$c = \overline{b_{00}}\,\overline{b_{11}}\,\overline{b_{22}}\dots$$

ovvero il vettore che contiene in posizione c_i il complemento di b_{ii}.

Questo vettore appartiene a $P(\mathbb{N})$, in quanto sicuramente sottoinsieme di $\mathbb{N}$, ma non è presente nella lista precedente perché diverso da ogni elemento almeno di una cifra (quella sulla diagonale principale).

Questo è assurdo per l'assunzione che $P(\mathbb{N})$ è numerabile, quindi $P(\mathbb{N}) \nsim \mathbb{N}$. □

Insieme delle funzioni

L'**insieme delle funzioni** da $\mathbb{N}$ a $\mathbb{N}$ è definito come

$$\mathbb{N}_{\perp}^{\mathbb{N}} = \{f : \mathbb{N} \to \mathbb{N}\}$$

Teorema 1.5.3. $\mathbb{N}_{\perp}^{\mathbb{N}} \nsim \mathbb{N}$.

Dimostrazione. Diagonalizzazione strikes again. Assumiamo, per assurdo, che $\mathbb{N}_{\perp}^{\mathbb{N}}$ sia numerabile. Possiamo quindi listare $\mathbb{N}_{\perp}^{\mathbb{N}}$ come $\{f_0, f_1, f_2, \dots\}$

	0	1	2	3	...	$\mathbb{N}$
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	...	...
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	...	...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	...	...
...	...	...	...	...	...	...

Costruiamo una funzione $\varphi : \mathbb{N} \to \mathbb{N}_{\perp}$ per dimostrare l'assurdo. Un'idea potrebbe essere $\varphi(n) = f_n(n) + 1$, "spostando" la diagonale, ma non tiene in considerazione il caso $f_n(n) = \perp$ in quanto non sapremmo dare un valore a $\varphi(n) = \perp + 1$. Definiamo quindi

$$\varphi(n) = \begin{cases} 1 & \text{se } f_n(n) = \perp \\ f_n(n) + 1 & \text{se } f_n(n) \downarrow \end{cases}$$

Questa funzione appartiene a $\mathbb{N}_{\perp}^{\mathbb{N}}$, ma non è presente nella lista precedente, infatti $\forall k \in \mathbb{N}$ si ottiene

$$\varphi(k) = \begin{cases} 1 \neq f_k(k) = \perp & \text{se } f_k(k) = \perp \\ f_k(k) + 1 \neq f_k(k) & \text{se } f_k(k) \downarrow \end{cases}$$

Questo è assurdo sotto l'assunzione che $\mathbb{N}_{\perp}^{\mathbb{N}}$ è numerabile, quindi $\mathbb{N}_{\perp}^{\mathbb{N}} \nsim \mathbb{N}$. □

1.6 Potenza Computazionale di un sistema di calcolo

1.6.1 Validità dell'inclusione $F(\mathcal{C}) \subseteq \mathbf{DATI}_{\perp}^{\mathbf{DATI}}$

Dopo aver dato una più robusta definizione di cardinalità, possiamo studiare la natura dell'inclusione

$$F(\mathcal{C}) \subseteq \mathrm{DATI}_{\perp}^{\mathrm{DATI}}$$

Due intuizioni, da dimostrare, sono:

- PROG $\sim \mathbb{N}$: ogni programma può essere identificato con un numero, come la sua codifica in binario
- DATI $\sim \mathbb{N}$: anche ogni dato può essere identificato tramite la sua codifica in binario

Da questo possiamo dire che

$$F(\mathcal{C}) \sim \mathrm{PROG} \sim \mathbb{N} \nsim \mathbb{N}_{\perp}^{\mathbb{N}} \sim \mathrm{DATI}_{\perp}^{\mathrm{DATI}}$$

Questo dimostra che **esistono funzioni non calcolabili**, ci sono troppe funzioni e troppi pochi programmi.

Dobbiamo dimostrare le due assunzioni PROG $\sim \mathbb{N}$ e DATI $\sim \mathbb{N}$. Si può fare tramite tecniche di aritmetizzazione (o godelizzazione) di strutture, tecniche che rappresentano delle strutture tramite un numero.

1.7 DATI $\sim \mathbb{N}$

Serve trovare una legge che

1. Associ biunivocamente dati a numeri e viceversa
2. Consenta di operare direttamente sui numeri per operare sui corrispondenti dati, ovvero abbia delle primitive che permettano di lavorare sul numero che "riflettano" il risultato sul dato, senza passare dal dato stesso
3. Consenta di dire, senza perdita di generalità, che i programmi lavorano su numeri

1.7.1 Funzione Coppia di Cantor

La **funzione coppia di Cantor** è la funzione

$$\langle , \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}^+$$

E sfrutta le due "sotto-funzioni"

$$\begin{aligned} \sin : &\quad \mathbb{N}^+ \to \mathbb{N} \\ \mathrm{des} : &\quad \mathbb{N}^+ \to \mathbb{N} \end{aligned}$$

Tali che

$$\langle x, y \rangle = n \implies \begin{aligned} \sin(n) &= x \\ \mathrm{des}(n) &= y \end{aligned}$$

Si può rappresentare graficamente come

$x \setminus y$	0	1	2	3	...
0	1	3	6	10	...
1	2	5	9	...	
2	4	8	...		
3	7	...			

$x \backslash y$	0	1	2	3
0	1	3	6	10
1	2	5	9	
2	4	8		
3	7			

Il valore $\langle x, y \rangle$ rappresenta l'incrocio tra la x-esima riga e la y-esima colonna. Per costruirla:

1. $x = 0$
2. si parte dalla cella $(x, 0)$ e si enumerano le celle della diagonale identificata da $(x, 0)$ e $(0, x)$
3. si incrementa x di 1 e si ripete dal punto precedente

La funzione deve essere:

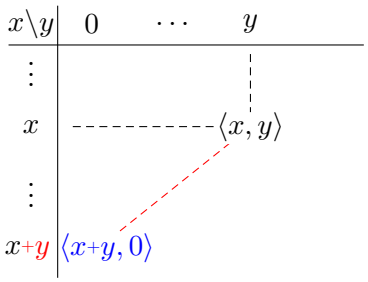
- iniettiva: non ci possono essere celle con lo stesso numero
- suriettiva: ogni numero in $\mathbb{N}^+$ deve comparire

Entrambe le proprietà sono soddisfatte, in quanto la numerazione avviene in maniera incrementale, quindi ogni numero prima o poi compare in una cella e di conseguenza ho una coppia che lo genera.

Forma analitica

Per la definizione di $\langle x, y\rangle$ si può notare che

$$\langle x, y\rangle = \langle x + y, 0\rangle + y$$



Intuitivamente, a partire da $\langle x + y, 0\rangle$ mi basta "salire" seguendo la diagonale fino a $\langle x, y$, ovvero y posti, e per definizione della funzione, y valori più in alto.

Il calcolo della funzione coppia si può quindi ridurre al calcolo di $\langle x + y, 0$. Chiamando $x + y = z$, si può notare come ogni cella

$$\langle z, 0\rangle = z + \langle z - 1, 0\rangle$$

E di conseguenza

$$\begin{aligned}\langle z, 0\rangle &= z + \langle z - 1, 0\rangle \\ &= z + (z - 1) + \langle z - 2, 0\rangle \\ &= z + (z - 1) + \cdots + 1 + \langle 0, 0\rangle = \\ &= \sum_{i=1}^{z} i + 1 = \frac{z(z + 1)}{2} + 1\end{aligned}$$

Mettendo insieme le due proprietà viste possiamo ottenere la formula analitica per la funzione coppia:

$$\langle x, y\rangle = \langle x + y, 0\rangle + y = \frac{(x + 1)(x + y + 1)}{2} + y + 1$$

Forma analitica di sin **e** des

Vogliamo fare la stessa cosa per sin e des, in modo da poter computare l'inversa della funzione coppia, dato n. Grazie alle osservazioni precedenti

sappiamo che

$$\begin{aligned} \gamma = x + y \quad &\Longrightarrow \quad x = \gamma + y \\ n = y + \langle \gamma, 0 \rangle \quad &\Longrightarrow \quad y = n - \langle \gamma, 0 \rangle \end{aligned}$$

Trovando il valore di γ possiamo trovare x e y.

Notiamo come γ sia il più grande valore che, quando calcolato sulla prima colonna ($\langle \gamma, 0 \rangle$) non supera n, ovvero

$$\gamma = \max\{z \in \mathbb{N} | \langle z, 0 \rangle \leq n\}$$

Intuitivamente, si tratta dell'inizio della diagonale che contiene n, è "l'inverso" dell'osservazione fatta in precedenza per la quale $\langle x, y \rangle = \langle x + y, 0 \rangle + y$.

Risolviamo quindi la disequazione

$$\begin{aligned} \langle z, 0 \rangle \leq n \implies & \frac{z(z+1)}{2} + 1 \leq n \\ \implies & z^2 + z - 2n + 2 \leq 0 \\ \implies & z_{1,2} = \frac{-1 \pm \sqrt{1 + 8n - 8}}{2} \\ \implies & \frac{-1 - \sqrt{8n-7}}{2} \leq z \leq \frac{-1 + \sqrt{8n-7}}{2} \end{aligned}$$

Come valore di γ scegliamo

$$\gamma = \left\lfloor \frac{-1 + \sqrt{8n-7}}{2} \right\rfloor$$

E con γ noto possiamo definire le funzioni sin e des come

$$\begin{aligned} \mathrm{des}(n) &= y = n - \langle \gamma, 0 \rangle = n - \tfrac{\gamma(\gamma+1)}{2} - 1 \\ \mathrm{sin}(n) &= x = \gamma - y \end{aligned}$$

Teorema 1.7.1. $\mathbb{N} \times \mathbb{N} \sim \mathbb{N}^+$

Dimostrazione. La funzione di Cantor è una funzione biettiva tra l'insieme $\mathbb{N} \times \mathbb{N}$ e l'insieme $\mathbb{N}^+$, quindi i due insiemi sono isomorfi.

□

Possiamo estendere il risultato all'interno dell'insieme $\mathbb{N}$, ovvero:
Teorema 1.7.2. $\mathbb{N} \times \mathbb{N} \sim \mathbb{N}$

Dimostrazione. Definiamo la funzione

$$[,] : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

tale che

$$[x, y] = \langle x, y \rangle - 1$$

Questa funzione è anch'essa biettiva, quindi i due insiemi sono isomorfi.

□

Grazie a questo è possibile dimostrare anche che $\mathbb{Q} \sim \mathbb{N}$, infatti i numeri razionali si possono rappresentare come coppie (num, den) e, in generale, tutte le tuple sono isomorfe e $\mathbb{N}$, basta iterare in qualche modo la funzione coppia di Cantor.

1.7.2 Applicazione alle strutture dati

I risultati ottenuti fin'ora rendono intuibile come ogni dato possa essere trasformato in un numero, soggetto a trasformazioni matematiche. La dimostrazione *formale* non verrà fatta, anche se verranno fatti esempi di alcune strutture dati che possono essere trasformate in un numero tramite la funzione coppia di Cantor. Ogni struttura dati può essere manipolata e trasformata in una coppia (x, y).

Le **liste** sono le strutture dati più utilizzate nei programmi. In generale non ne è nota la grandezza, di conseguenza è necessario trovare un modo, soprattutto durante l'applicazione di sin e des, per capire quando abbiamo esaurito gli elementi della lista.

Estendiamo la funzione coppia a una lista di interi $x_1, \ldots, x_n$:

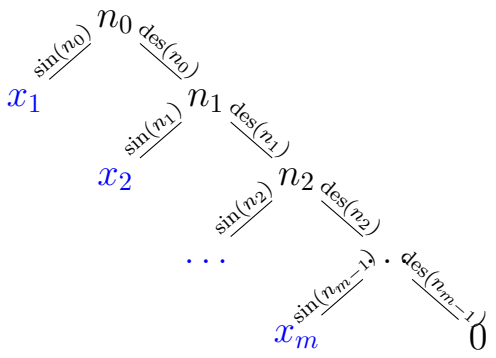
$$\langle x_1, \ldots, x_n \rangle \to \langle x_1, \langle x_2 \langle \ldots \langle x_n, 0 \rangle \ldots \rangle \rangle \rangle$$

Lo 0 rappresenta il fine lista e non è necessario nel caso in cui il numero di elementi è noto.

La decodifica è il processo inverso, partendo dal numero finale si applicano le funzioni sin e des ottenendo a ogni iterazione:

- da des la somma parziale, su cui riapplicare la funzione per ottenere il valore successivo
- da sin il valore presente all'interno della lista

Termina quando il risultato di des è zero, ovvero l'elemento di fine lista che abbiamo inserito (x_n nel caso di array).



Se è presente uno 0 all'interno della lista non è un problema in quanto solo des viene controllato e lo 0 come valore sarà risultato di sin.

Quindi è possibile codificare liste e, di conseguenza, **qualsiasi tipo di dato**, basta convertirlo in una lista di numeri. Per esempio:

- una matrice può essere vista come array di array
- un grafo può essere rappresentato tramite la sua matrice di adiacenza
- i testi sono liste di caratteri
- i suoni si possono campionare per ottenere una lista di valori
- le immagini sono una "lista" di pixel, ognuno dei quali ha un colore come valore

Abbiamo visto come i dati possano essere sostituiti da delle codifiche numeriche; di conseguenza possiamo sostituire tutte le funzioni

$$f : \mathrm{DATI} \to \mathrm{DATI} \quad \text{con funzioni} \quad f' : \mathbb{N} \to \mathbb{N}_\perp$$

In altre parole, l'universo dei problemi per i quali cerchiamo una soluzione automatica è rappresentabile da $\mathbb{N}_{\perp}^{\mathbb{N}}$ e di conseguenza DATI $\sim \mathbb{N}$.

1.8 PROG $\sim \mathbb{N}$

Adesso lavoriamo sulla parte della relazione che afferma

$$F(\mathcal{C}) \sim \text{PROG} \sim \mathbb{N}$$

Ovvero, la potenza computazionale (l'insieme dei programmi che un sistema di calcolo $\mathcal{C}$ riesce a calcolare, $F(\mathcal{C})$) è isomorfa all'insieme di tutti i programmi, a loro volta isomorfi a $\mathbb{N}$.

Vogliamo arrivare a ricavare un numero dato un programmo e viceversa. Per farlo servirà vedere l'insieme PROG come l'insieme dei programmi scritti in un certo linguaggio di programmazione.

I sistemi analizzati saranno:

- sistema di calcolo RAM
- sistema di calcolo WHILE

Il sistema RAM può apparentemente sembrare "troppo semplice", quindi il sistema WHILE verrà usato per avere un confronto tra le potenze computazionali. Un sistema più sofisticato porta a poter risolvere più problemi?

Ci sono due possibili soluzioni:

- $F(\text{RAM} \neq F(\text{WHILE})$: la computabilità *dipende dal sistema usato*
- $F(\text{RAM}) = F(\text{WHILE})$: la computabilità è *intrinseca nei problemi* e, di conseguenza, tutti i sistemi sono equivalenti (Tesi di Church-Turing)

Il secondo caso è più promettente e, in quel caso, l'obiettivo diventerebbe trovare una *caratterizzazione teorica*, ovvero un "confine" per i problemi calcolabili.

1.8.1 Sistema di calcolo RAM

Il sistema di calcolo RAM è un sistema semplice che permette di definire rigorosamente:

- PROG $\sim \mathbb{N}$
- la **semantica** dei programmi eseguibili, ovvero $\mathcal{C}(P, \_)$, con $\mathcal{C} =$ RAM, ottenendo RAM$(P, _)$
- la **potenza computazionale**, ovvero calcolare $F(\mathcal{C})$ con $\mathcal{C} =$ RAM, ottenendo F(RAM)

Struttura

Una macchina RAM è formata da un processore e da una memoria teoricamente infinita, divisa in **celle/registri** contenenti numeri naturali (dati aritmetizzati).

Indichiamo i **registri** con R_k, con $k \geq 0$. Tra questi

- R_0 contiene l'output
- R_1 contiene l'input

Inoltre è presente un registro L, anche detto **program counter** PC che indica l'indirizzo dell'istruzione successiva.

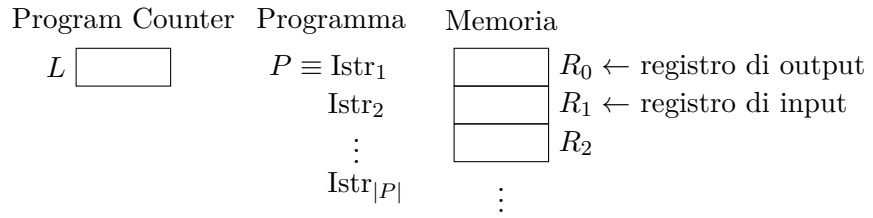
Dato un **programma** P, indichiamo con $|P|$ il numero di istruzioni che il programma contiene.

Le **istruzioni** nel linguaggio RAM sono:

- **incremento**: $R_k \leftarrow R_k + 1$
- **decremento**: $R_k \leftarrow R_k - 1$
- **salto condizionato**: `if` $R_k = 0$ `then goto` m, con $m \in \{1, \ldots, |P|\}$

L'istruzione di decremento è tale che

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



Esecuzione di un programma RAM

L'esecuzione di un programma su una macchina RAM segue i passi:

1. **Inizializzazione**:
   - viene caricato il programma $P \equiv \text{Istr}_1, \ldots \text{Istr}_n$ in memoria
   - il PC viene posto a 1 per indicare di eseguire la prima istruzione del programma
   - viene caricato l'input in R_1
   - ogni altro registro è azzerato
2. **Esecuzione**: le istruzioni vengono eseguite una dopo l'altra, a ogni iterazione passa da L a $L+1$ (escluse operazioni di salto). Essendo il linguaggio RAM *non strutturato* richiede un PC per sapere l'operazione da eseguire al passo successivo.
3. **Terminazione**: per convenzione, si usa $L = 0$ per indicare che l'esecuzione del programma è terminata o andata in loop. Nel caso in cui il programma termini, è detto **segnale di halt** e arresta la macchina
4. **Output**: il contenuto di R_0, in caso di halt, contiene il risultato dell'esecuzione del programma P. Si indica con $\varphi_P(n)$ il contenuto del registro R_0 in caso di halt, oppure $\perp$ in caso di loop

$$\varphi_P(n) = \begin{cases} cont(R_0) & \text{se halt} \\ \perp & \text{se loop} \end{cases}$$

Con $\varphi_P : \mathbb{N} \to \mathbb{N}_\perp$ indichiamo la semantica del programma P.

Con $\mathcal{C}(P, \_)$ indicavamo la semantica di P nel sistema di calcolo $\mathcal{C}$, quindi con $\text{RAM}(P, \_) = \varphi_P$ indichiamo la semantica di P nel sistema di calcolo RAM.

Semantica Operazionale

Per dare una definizione formale della semantica di un programma RAM va specificato il significato di ogni istruzione (**semantica operazionale**), esplicitando l'effetto che quell'istruzione ha sui registri della macchina.

Ogni istruzione fa passare la macchina da uno stato all'altro e la **semantica operazionale** di un'istruzione è la **coppia** formata dagli **stati** della macchina **prima e dopo l'istruzione**.

$$\text{STATO}_1 \rightarrow \boxed{\text{Istr}_i} \rightarrow \text{STATO}_2$$

$$(\text{STATO}_1, \text{STATO}_2) = \text{semantica operazionale di Istr}_i$$

Uno stato deve descrivere completamente la situazione della macchina in un certo istante. Il programma rimane uguale, quindi l'informazione da salvare è la situazione globale dei registri R_k e il registro L.

La **computazione** del programma P è una sequenza di stati $\mathcal{S}_i$, ognuno generato dall'esecuzione di un'istruzione del programma; P induce una sequenza di stati $\mathcal{S}_i$, se questa è formata da un numero infinito di stati, allora il programma è andato in loop; in caso contrario, nel registro R_0 si trova il risultato y della computazione di P.

$$\varphi_P : \mathbb{N} \rightarrow \mathbb{N}_\perp \quad \text{t.c.} \quad \varphi_P(n) = \begin{cases} y & \text{se } \exists \mathcal{S}_{fin} \\ \perp & \text{altrimenti} \end{cases}$$

Per definire come passare da uno stato all'altro, definiamo formalmente:

- **Stato**: istantanea di tutte le componenti della macchina, è una funzione
$$\mathcal{S} : \{L, R_i\} \rightarrow \mathbb{N}$$
tale che $\mathcal{S}(R_k)$ restituisce il contenuto del registro R_k quando la macchina si trova nello stato $\mathcal{S}$. Gli stati possibili di una macchina appartengono all'insieme
$$\text{STATI} = \{f : \{L, R_i\} \rightarrow \mathbb{N}\} = \mathbb{N}^{\{L, R_i\}}$$

- **Stato Finale**: uno stato finale $\mathcal{S}_{fin}$ è un qualsiasi stato $\mathcal{S}$ tale che $\mathcal{S}(L) = 0$

- **Dati**: già dimostrato come DATI $\sim \mathbb{N}$
- **Inizializzazione**: serve una funzione che, preso l'input, restituisca lo stato iniziale della macchina:

  $$\text{in} : \mathbb{N} \to \text{STATI} \;\; \text{t.c.} \;\; \text{in}(n) = \mathcal{S}_{init}$$

  Lo stato iniziale $\mathcal{S}_{init}$ è tale che

  $$\mathcal{S}_{init}(R) = \begin{cases} 1 & \text{se } R = L \\ n & \text{se } R = R_1 \\ 0 & \text{altrimenti} \end{cases}$$

- **Programmi**: PROG è definito come l'insieme dei programmi RAM

Manca da definire la *parte dinamica* del programma, ovvero l'esecuzione. Per farlo, definiamo la **funzione di stato prossimo**:

$$\delta : \text{STATI} \times \text{PROG} \to \text{STATI}_{\perp}$$

tale che

$$\delta(\mathcal{S}, P) = \mathcal{S}'$$

dove $\mathcal{S}$ rappresenta lo stato attuale e $\mathcal{S}'$ rappresenta lo stato prossimo dopo l'esecuzione di un'istruzione di P.

La funzione $\delta(\mathcal{S}, P) = \mathcal{S}'$ è tale che

- se $\mathcal{S}(L) = 0$ ho halt, ovvero deve terminare la computazione. Poniamo lo stato come indefinito, ovvero $\mathcal{S}' = \perp$
- Se $\mathcal{S}(L) > |P|$ vuol dire che P non contiene istruzioni che bloccano esplicitamente l'esecuzione del programma. Lo stato $\mathcal{S}'$ è tale che

  $$\mathcal{S}'(R) = \begin{cases} 0 & \text{se } R = L \\ \mathcal{S}(R_i) \text{ se } R = R_i \forall i \end{cases}$$

- Se $1 \leq \mathcal{S}(L) \leq |P|$ considero l'istruzione $\mathcal{S}(L)$-esima:
  - se ho un incremento/decremento sul registro R_k definisco $\mathcal{S}'$ tale che

    $$\begin{cases} \mathcal{S}'(L) & = \mathcal{S}(L) + 1 \\ \mathcal{S}'(R_k) & = \mathcal{S}(R_k) \pm 1 \\ \mathcal{S}'(R_i) & = \mathcal{S}(R_i) \;\; \text{per} \;\; i \neq k \end{cases}$$

– Se ho un `goto` sul registro R_k che salta all'indirizzo m, definisco $\mathcal{S}'$ tale che

$$\mathcal{S}'(L) = \begin{cases} m & \text{se } \mathcal{S}(R_k) = 0 \\ \mathcal{S}(L) + 1 & \text{altrimenti} \end{cases}$$

$$\mathcal{S}'(R_i) = \mathcal{S}(R_i)\ \forall i$$

L'esecuzione di un programma $P \in \text{PROG}$ su input $n \in \mathbb{N}$ genera una sequenza di stati

$$\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_i, \mathcal{S}_{i+1}, \ldots$$

tali che

$$\begin{array}{lll} \mathcal{S}_0 & = & \text{in}(n) \\ \forall i \quad \mathcal{S}_{i+1} & = & \delta(\mathcal{S}_i, P) \end{array}$$

La sequenza è infinita quando P va in loop, mentre se termina raggiunge uno stato $\mathcal{S}_m$ tale che $\mathcal{S}_m(L) = 0$, ovvero ha ricevuto il segnale di halt.

La semantica di P è

$$\varphi_P(n) = \begin{cases} y & \text{se } P \text{ termina in } \mathcal{S}_m, \text{ con } \mathcal{S}_m(L) = 0 \text{ e } \mathcal{S}_m(R_0) = y \\ \perp & \text{se P va in loop} \end{cases}$$

La potenza computazionale del sistema RAM è

$$F(\text{RAM}) = \left\{ f \in \mathbb{N}_\perp^{\mathbb{N}} | \exists P \in \text{PROG} | \varphi_P = f \right\} = \{\varphi_P | P \in \text{PROG}\} \subsetneq \mathbb{N}_\perp^{\mathbb{N}}$$

L'insieme è formato da tutte le funzioni $f : \mathbb{N} \to \mathbb{N}_\perp$ che hanno un programma che le calcola in un sistema RAM.

1.8.2 Aritmetizzazione di un programma