FONDAMENTI DI INFORMATICA

Docente

Flavio Corradini

Studente

Giorgio Della Roscia

Università degli studi di Camerino

Informatica per la Comunicazione Digitale (L-31)

Scuola di Scienze e Tecnologie

A.A. 2022/2023

Indice

Figure

Tabelle

Algoritmi

Approfondimenti

Sezione 1

MACCHINA DI TURING

«Cosa è possibile svolgere con l'ausilio di una macchina?»

Congegno composto fisicamente da un nastro infinito, e rispettive celle su cui agisce una testina di scrittura e lettura collegata ad un blocco di controllo. Questa può spostarsi a destra e sinistra. schema macchina di Turing

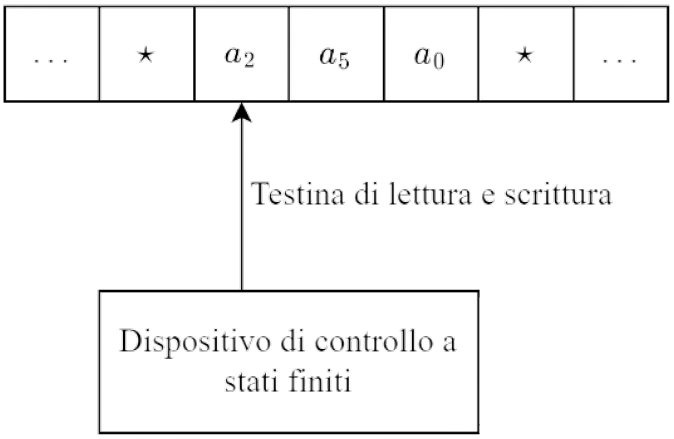


Figura 1.1: Schema macchina di Turing

Le istruzioni, o comandi, sono finite come la loro complessità.

Definizione (MdT)**.** Una macchina di Turing M è una quadrupla $(Q,\ A,\ \delta,\ q_0)$:

- Q = insieme finito e non vuoto di stati;
- A = alfabeto a cui si aggiunge il simbolo bianco $\star$;
- δ = funzione di transizione i cui elementi sono detti istruzioni;

$$Q \times (A \cup \{\star\}) \to Q \times (A \cup \{\star\}) \times \{-1,\ +1\}$$

- $q_0 \in Q$ = stato iniziale.

Dopo un'operazione, se si vuole che la MdT non vari il puntamento della testina si aggiungono due quintuple in modo da effettuare la seguente procedura:

i. spostamento al quadro vicino e riscrittura del simbolo presente in esso;

ii. ritorno alla locazione e allo stato iniziale.

1.1 Funzionamento di una MdT

Il programma prende forma di molteplici quintuple, ognuna delle quali regola un passaggio, che prendono forma di $(q,\ a,\ q',\ a',\ x)$

$$q = \text{stato attuale} \qquad a = \text{simbolo attuale}$$

$$q' = \text{stato futuro} \qquad a' = \text{simbolo futuro} \qquad x = \pm 1\text{spostamento testina}$$

Dunque una MdT è deterministica se vige incondizionata la seguente relazione:

$$(q,\ a) \xrightarrow{\delta} (q',\ a',\ x)$$

1.1.1 Matrici funzionali

Il suddetto programma può essere descritto tramite una matrice, di transizione, dove le righe indicano i possibili stati mentre le colonne i simboli dell'alfabeto vigente.

Tabella 1.1: Matrice di transizione

		A			
		a_0	a_1	$\dots$	a_m
Q	q_0	$\circledast$	$\circledast$	$\dots$	$\circledast$
	q_1	$\circledast$	$\circledast$	$\dots$	$\circledast$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	q_n	$\circledast$	$\circledast$	$\dots$	$\circledast$

I simboli del tipo $\circledast$ presenti nella tabella indicano l'istruzione da eseguire quando dallo stato q si legge il simbolo a. Quindi la terna che descrive:

i. revisione del suo stato di entrata;

ii. sovrascrittura di un simbolo nella cella indicata dalla testina;

iii. spostamendo della testina di una posizione a sinistra o a destra lungo il nastro infinito.

Una casella può essere bianca, in quel caso la MdT non ha azioni da eseguire e termina il calcolo.

1.2 Computazione di una macchina di Turing

Definizione (configurazione istantanea)**.** Si tratta di una sorta di fotogramma della MdT che la ritrae in un determinato istante. Una configurazione istantanea di una macchina di Turing del tipo

$$M = (Q,\ A,\ \delta,\ q_0)$$

è un elemento dell'insieme

$$(A \cup \{\star\})^\star \times Q \times (A \cup \{\star\}) \times (A \cup \{\star\})^\star$$

La computazione di una macchina M su una stringa di input $w \in A^\star$ avviene secondo:

i. M esamina w da sinistra a destra, identifica il primo carattere a_0 e lo stato iniziale q_0;

ii. ad un generico $i-$esimo passo M si trova in C_i e passa alla nuova C_{i+1} se esiste, altrimenti si arresta;

iii. se l'elaborazione si arresta allora si dice chiama C_n configurazione finale e si dice che M converge sull'input w, in notazione: $M \downarrow w$. Altrimenti non si arriverà mai ad una C_n, M diverge sull'input w e la dicitura sarà: $M \uparrow w$.

Ecco la rappresentazione matematica dei vari stati nelle macchine convergenti sull'input:

$$\begin{array}{c} \text{configurazione istantanea iniziale} \\ \downarrow \\ C_0 \vdash C_1 \vdash \ldots \vdash C_n \\ \uparrow \\ \text{configurazione finale} \end{array}$$

Esercizio (macchina di Turing)**.** Scrivere una MdT che calcoli il successivo di un naturale in notazione decimale.

Ecco di seguito la tabella che risolve la macchina di Turing richiesta.

Tabella 1.2: Risoluzione MdT successivo decimale

		A				
		0	1	...	9	$\star$
Q	q_0	$(q_0, 0, D)$	$(q_0, 1, D)$	...	$(q_0, 9, D)$	$(q_1, \star, S)$
	q_1	$(q_2, 1, D)$	$(q_2, 2, D)$	...	$(q_1, 0, S)$	$(q_2, 1, S)$
	q_2	/	/	/	/	/

Dove q_2 equivale allo stato di terminazione.

1.2.1 Invariazione del puntamento della testina

Dopo un'operazione, se si vuole che la MdT non vari il puntamento alla cella si aggiungono due quintuple in modo da effettuare la seguente procedura:

i. spostamento al quadro vicino e riscrittura del simbolo presente in esso;

ii. ritorno alla locazione e allo stato iniziale.

1.3 Gödelizzazione

L'insieme di tutte le possibili MdT è enumerabile, ossia che è in corrispondenza biunivoca con i numeri naturali.

$$\begin{array}{ccccccc} \mathbb{N} & 0 & 1 & \ldots & n \\ \updownarrow & \updownarrow & \updownarrow & \ldots & \updownarrow \\ \text{MdT} & \text{MdT}_0 & \text{MdT}_1 & \ldots & \text{MdT}_n \\ \downarrow & \downarrow & \downarrow & \ldots & \downarrow \\ \varphi & \varphi_0 & \varphi_1 & \ldots & \varphi_n \end{array}$$

La notazione è la seguente:

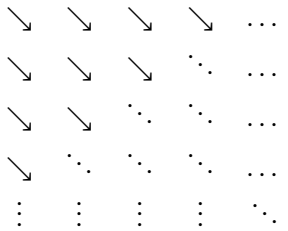
- MdT_i è l'i-esima MdT nell'enumerazione;
- φ_i è la funzione calcolata da MdT_i;
- i ed n sono gli input da dare alle MdT.

1.3.1 Codifica di naturali

Vengono associati gli elementi dell'alfabeto formale della MdT a dei numeri identificativi.

$$r(n,\ m) = \frac{(n+m)\cdot(n+m+1)}{2} + m$$

Tabella 1.3: Tecnica della coda di colomba



		m				
		0	1	2	3	...
n	0	0	1	3	6	...
	1	2	4	7	⋱	...
	2	5	8	⋱	⋱	...
	3	9	⋱	⋱	⋱	...
	⋮	⋮	⋮	⋮	⋮	⋱

Essendo $\mathbb{N}$ in corrispondenza biunivoca con $A^\star$, il rapporto tra $\mathbb{N}$ e $A^\star$ con A alfabeto è data fissando un ordinamento su quest'ultimo.

1.4 MdT universale

Definizione (MdT universale)**.** Esiste una MdT universale $\mathcal{U}$.

$$g\ :\ \mathbb{N}^2 \to \mathbb{N} \quad \forall x,\ y \in \mathbb{N} \qquad\qquad g(x,\ y) = \varphi_x(y)$$

Tale funzione è calcolabile secondo Turing.

> **Dimostrazione** (MdT universale)**.** Innanzitutto si decodifica x ottenendo M_x e dandogli poi in input y. $\mathcal{U}$ controlla innanzitutto che il suo input sia una coppia $(M,\ y)$, per qualche MdT $M = M_x$ ed un input y. Poi esegue $M_x(y)$ effettuando l'aggiornamento della quintupla.

1.5 Algoritmi

Non volendo essere interpretabili, gli algoritmi seguono alcune linee guida prestabilite:

- la lettura è latina, da sinistra a destra;
- l'ordine di esecuzione va dall'alto verso il basso;
- è possibile scrivere più comandi in una riga solo se separati da virgole o congiunzioni.

Tesi (algoritmo)**.** Il concetto di algoritmo si basa su un decalogo:

1. ha lunghezza finita;
2. il calcolatore svolge la computazione eseguendo istruzioni;
3. il calcolatore ha a disposizione una memoria;
4. il calcolo avviene per passi discreti;

5. il calcolo non è probabilistico;
6. non esiste alcun limite finito alla lunghezza dei dati di input;
7. non esiste alcun limite finito alla quantità di memoria disponibile;
8. esiste un limite finito alla complessità delle istruzioni eseguibili dal calcolatore;
9. il numero di passi della computazione è finito ma non limitato;
10. sono ammesse computazioni infinite.

Esempio (astrazione di algoritmo)**.** Dato un algoritmo come quello che segue

Algoritmo 1: Funzione d'astrazione di algoritmo

```
begin
    input(x);
    if x = 5 then
        begin
            output (x^2)
        end
end
```

è sempre possibile associarvi una funzione che lo astrae:

$$f : \mathbb{N} \to \mathbb{N} \qquad f(x) = \begin{cases} x^2, & x = 5 \\ \uparrow, & x \neq 5 \end{cases}$$

Sezione 2

PROBLEMI INCOMPUTABILI

L'insieme dei problemi risolvibili non coincide con la totalità di essi. Dunque non esiste algoritmo che ne prenda in input un secondo e ne determini la convergenza.

Problema (arresto)**.** Determinare un algoritmo che stabilisca, $\forall$ MdT M e dato un input $x \in \mathbb{N}$, se $M \downarrow x$ o meno.

> **Dimostrazione** (problema dell'arresto)**.** Presa la seguente funzione non calcolabile e posta invece, per assurdo, come se lo sia
>
> $$h(x) = \begin{cases} 1, \text{ se } \varphi_x(x) \downarrow \\ 0, \text{ se } \varphi_x(x) \uparrow \end{cases}$$
>
> Se poniamo che esista una MdT che la calcoli si ha la funzione
>
> $$k(x) = \begin{cases} \uparrow, \text{ se } h(x) = 1 \\ 0, \text{ se } h(x) = 0 \end{cases}$$
>
> Dalla calcolabilità di $h(x)$ segue quella di $k(x)$, con $\varphi_j = k$, ma allora si avrebbe
>
> $$\varphi_j(j) \downarrow \iff k(j) = 0 \iff h(j) = 0 \iff \varphi_j(j) \uparrow$$
>
> Notiamo dunque che si evince un'evidente contraddizione, quindi $h(x)$ non è calcolabile.

2.1 Insiemi

Definizione (insieme)**.** Collezione di elementi di qualsiasi tipologia.

Esistono insiemi finiti ed infiniti che possono essere definiti in modo estensionale o intensionale:

finito estensionale: $A = \{0, 4, a, \varphi, 7, C, \triangle\}$ infinito intensionale: $B = \{x \in \mathbb{N} \mid x + x = 2 \cdot x\}$

Tabella 2.1: Operazioni tra insiemi

UNIONE	INTERSEZIONE	DIFFERENZA	PRODOTTO	COMPLEMENTO
$A \cup B$	$A \cap B$	$A \setminus B$	$A \times B$	$\mathbb{N} - A$

2.1.1 Linguaggi

Definizione (alfabeto)**.** Insieme non vuoto e finito di simboli non interpretati.

Definizione (stringa)**.** Sequenza finita ottenuta tramite concatenazione di simboli dell'alfabeto.

$$u \cdot v$$
$$\uparrow$$
simbolo di concatenazione

Ecco alcune notazioni relative agli alfabeti e alle stringhe:

- λ = stringa vuota.
- A^* = insieme infinito di tutte le stringhe ottenibili da un alfabeto A.
- $\mu \cdot \nu$ = concatenazione di stringhe.

Definizione (linguaggio formale)**.** Ogni linguaggio, sia esso formale o naturale, si basa su un alfabeto. Datone uno della tipologia

$$A = \{a_1, a_2, \ldots, a_n\}$$

e sia inoltre A^* l'insieme universo delle stringhe su A. Un linguaggio formale nell'alfabeto A è un sottoinsieme di $A^\star$:

$$L \subseteq A^\star$$

2.1.1.1 Lunghezza delle stringhe

La lunghezza $l(u)$ equivale al numero di simboli di presenti in u:

$$l(u) = \begin{cases} 0, \text{ se } u = \lambda \\ 1 + l(u'), \text{ se } u = au' \quad a \in A \end{cases}$$

Esempio (lunghezza di una stringa)**.** La lunghezza della sequenza abc corrisponde a 3, cioè $l(abc) = 3$. Analogamente $l(da252fsxF) = 9$. Caso noto riguarda invece $l(\lambda) = 0$.

2.1.1.2 Ordinamento

L'ordinamento dei simboli dell'alfabeto è decidibile a piacimento.

Definizione (ordinamento)**.** Dato un alfabeto

$$A = \{a_1, a_2, \ldots, a_n\}$$

e sia $<$ un ordinamento totale

$$a_1 < \ldots < a_n$$

e siano ancora α, β delle stringhe su A. Sapendo che α è lessicograficamente minore di β si può scrivere $\alpha > \beta$. Ciò è vero se, e solo se, vale uno dei seguenti punti:

- α è prefisso di β; $\exists u \ : \ \alpha \cdot u = \beta$
- α è suffisso di β; $\exists u \ : \ u \cdot \alpha = \beta$
- α ha un prefisso γa, β ha un prefisso γb e $a < b$;

2.1.2 Insiemi decidibili

Definizione (linguaggio ricorsivo)**.** Definito un linguaggio $L \in \mathbb{N}$, esso è decidibile se, e solo se, c'è una MdT che sa decidere, $\forall x \in \mathbb{N}$, se $x \in L$ o meno.

$$x_L(w) = \begin{cases} 1, \text{ se } w \in L \\ 0, \text{ se } w \notin L \end{cases} \quad \text{funzione caratteristica di } L$$

L è decidibile se, e solo se, x_L è calcolabile. Dunque, un insieme è decidibile se, e solo se, la sua funzione caratteristica è calcolabile.

Dunque se c'è un algoritmo che riesce a risolvere il problema.

Teorema (Post)**.** Sia $L \subseteq \mathbb{N}$, allora è decidibile se, e solo se, tanto L quanto il suo complemento $\overline{L}$ sono semidecidibili.

> **Dimostrazione** (Post)**.** Sappiamo che se L è decidibile anche $\overline{L}$ lo è. Ciò implica che sia L che $\overline{L}$ sono semidecidibili. Se invece assumiamo sia L che $\overline{L}$ semidecidibili, allora delle due rispettive MdT solamente una convergerà. Dunque
>
> $$\begin{cases} M(L) \downarrow \text{, allora } x \in L \\ M(\overline{L}) \downarrow \text{, allora } x \in \overline{L} \end{cases}$$

2.1.3 Insiemi semidecidibili

Definizione (linguaggio ricorsivamente enumerabile)**.** Definito un linguaggio $L \in \mathbb{N}$, esso è semidecidibile se, e solo se, c'è una MdT che lo accetta. Converge dunque sugli elementi di L.

$$x'_L(w) = \begin{cases} 1, \text{ se } w \in L \\ \uparrow \text{, se } w \notin L \end{cases} \quad \text{funzione semicaratteristica di } L$$

L è semidecidibile se, e solo se, x'_L è calcolabile.

Il linguaggio deve dunque essere vuoto, il dominio o l'immagine di una funzione calcolabile.

Lemma (relazioni fra insiemi decidibili e semidecidibili)**.** Sia $L \subseteq \mathbb{N}$, allora vale:

a. L è semidecidibile;

b. L è il dominio di una funzione calcolabile;

c. L è vuoto o è l'immagine di una funzione totale e calcolabile ($f : \mathbb{N} \to \mathbb{N}$).

> **Dimostrazione** (relazioni fra insiemi decidibili e semidecidibili)**.** a e b sono eguali data la definizione di semidecidibilità. La MdT M accetta L che è il dominio della funzione χ'_L.
>
> $a \Rightarrow c$ Preso L semidecidibile e data una MdT M che converge in tutti e soli gli elementi di L, si ha che le computazioni di M sui naturali, seguendo l'ordine dato dalla tecnica della coda di colomba, enumerano L.
>
> $c \Rightarrow a$ Viene dimostrata la semidecidibilità di L costruendo una MdT M che converge su tutti e soli gli elementi di L. $\forall x \in \mathbb{N}$, M computa. Il processo si arresta solo se viene verificata la coincidenza con x.

2.1.3.1 Coda di colomba

Per dimostrare la semidecidibilità di alcuni insiemi viene adoperata una particolare tecnica. L'obiettivo è di controllare l'enumerazione dell'immagine di una funzione.

Esercizio (coda di colomba)**.** Dato il seguente insieme

$$I = \{i \in \mathbb{N} \ : \ 5 \in \Im\,(\varphi_i)\}$$

dimostrarne la semidecidibilità.

Occorre dunque scritturare un algoritmo che, letto i, ottiene la MdT corrispondente e controlla se per un qualche valore questa restituisce in output 5.

Algoritmo 2: Coda di colomba

```
begin
    INPUT(i)
    decodifica i per ottenere MdT M_i
    sia c = 0 tale che c = ⟨π_1(c), π_2(c)⟩
    sia x = π_1(c) e sia n = π_2(c)
    esegui M_i su input x per n passi
    if M_i è in una configurazione finale con output 5 then
        begin
            OUTPUT(True)
        end
    else
        begin
            sia c = c + 1 e torna all'input
        end
end
```

Il contantore c viene scomposto per tener conto dell'input, in x, e del numero di passi, in n.

2.1.3.2 Insieme K e complemento $\overline{K}$

Lemma (K semidecidibile)**.** L'insieme $K = \{x \in \mathbb{N} \ : \ M_x \downarrow x\}$ è semidecidibile.

Dimostrazione (K semidecidibile)**.** Ecco un plausibile algoritmo per una MdT M che accetta K.

Algoritmo 3: K semidecidibile

```
begin
    decodifica x per ottenere la MdT M_x
    esegui M_x sull'input x
    INPUT(x)
    if M_x termina then
        begin
            OUTPUT(True)
        end
end
```

Dal teorema di Post consegue che il complemento dell'insieme K, ossia $\overline{K}$, non è semidecidibile.

2.1.4 Insiemi non semidecidibili

Oltre agli insiemi decidibili e semidecidibili è possibile incappare in alcuni non semidecidibili. Per riconoscerli esistono delle tecniche di verifica di incomputabilità.

2.1.4.1 Diagonalizzazione

Prende il nome dall'idea di realizzazare una tabella con righe e colonne infinite. Queste stanno rispettivamente per gli oggetti dell'enumerazione e per numeri naturali crescenti.

Tabella 2.2: Tabella per la diagonalizzazione

	0	1	2	...
α_0	$\underset{\sim}{d_{11}}$	d_{12}	d_{13}	...
α_1	d_{21}	$\underset{\sim}{d_{22}}$	d_{23}	...
α_2	d_{31}	d_{32}	$\underset{\sim}{d_{33}}$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Come evidenziato è necessario focalizzarsi sulla diagonale.

2.2 Funzioni

Ogni algoritmo ha come astrazione matematica una funzione. Essa si comporterà ogni volta allo stesso modo, prendendo gli stessi input restituirà gli stessi output.

$$f \ : \ I \to O$$

Definizione (funzione)**.** Relazione che ha come vincolo l'associazione di ogni elemento del dominio ad, al più, un elemento del codominio.

La mancanza di tale connessione indica divergenza, dunque interminabilità. Possono esserci delle funzioni parziali, non è perciò detto che per ogni x del dominio corrisponde una y del codominio. Ecco di seguito alcune funzioni d'uso comune in teoria della computabilità:

- **funzione identità**, funzione totale che restituisce l'input senza modifiche;

$$id \ : \ \mathbb{N} \to \mathbb{N}, \ id(x) = x$$

- **funzione costante**, funzione totale che restituisce una costante per ogni input;

$$c_k \ : \ \mathbb{N} \to \mathbb{N}, \ c_k(x) = k$$

- **funzione indefinita**, funzione parziale che non è definita su nessun input. Perciò sia il dominio che l'immagine sono insiemi vuoti.

$$\emptyset \ : \ \mathbb{N} \to \mathbb{N}, \ \emptyset(x) \uparrow$$

2.2.1 Calcolabilità

Ad un problema potrebbe corrispondere un algoritmo che lo risolve, e a questo viene sempre associata una funzione matematica.

$$\text{problema} \overset{?}{\rightsquigarrow} \text{algoritmo} \longrightarrow \text{funzione} \qquad P \overset{?}{\rightsquigarrow} A \longrightarrow f \ : \ \mathbb{N} \to \mathbb{N}$$

Tesi (Church-Turing)**.** Una funzione si dice calcolabile, o allo stesso modo computabile, se esiste almeno una macchina di Turing che la calcola.

Le funzioni non calcolabili non restituiscono alcun risultato seppur accettata la tesi precedente.

Definizione (calcolabilità)**.** Data una MdT del tipo

$$M = (Q,\ A,\ \delta,\ q_0)$$

e sia $L \subseteq A^\star$ un linguaggio formale; fissate due stringhe '$\underset{1}{\text{sì}}$' e '$\underset{0}{\text{no}}$':

- M <u>decide</u> L se $\forall w \in A^\star$ $\begin{cases} M \downarrow \text{'sì'}, \text{ se } w \in L \\ M \downarrow \text{'no'}, \text{ se } w \notin L \end{cases}$
- M <u>accetta</u> L se $\forall w \in A^\star$ $\begin{cases} M \downarrow w, \text{ se } w \in L \\ M \uparrow w, \text{ se } w \notin L \end{cases}$

Definizione (funzione calcolabile)**.** Sia $f \ : \ A^\star \to A^\star$ diciamo che una MdT della tipologia $M = (Q,\ A,\ \delta,\ q_0)$ calcola, o computa, la funzione f se, e solo se

$$\forall w \in A^\star \begin{cases} M \downarrow w \text{ con output } f(w),\ w \subset \text{ nel dominio di } f \text{ (converge)} \\ M \uparrow w \text{ con output } f(w),\ w \subsetneq \text{ nel dominio di } f \text{ (diverge)} \end{cases}$$

Esercizio (funzione calcolabile)**.** Si dica se la seguente funzione è calcolabile

$$\forall x \in \mathbb{N},\ f(x) = \begin{cases} 1, \text{ se domani piove} \\ 0, \text{ altrimenti} \end{cases}$$

La funzione f è calcolabile dato che in entrambe le alternative corrisponde ad una costante. Tuttavia non è noto a priori quale delle due venga effettivamente computata.

2.2.1.1 Teoremi di funzioni calcolabili

Teorema (punto fisso di Kleene)**.** Per ogni funzione totale e calcolabile

$$t : \mathbb{N} \to \mathbb{N} \quad \exists e \in \mathbb{N} : \varphi_e = \varphi_{t(e)}$$

e viene detto punto fisso.

Dimostrazione (punto fisso di Kleene)**.** $\forall u \in \mathbb{N}$ consideriamo la MdT $M(u)$ che, preso in input x, calcola dapprima $\varphi_{g(u)}(u)$ e poi valuta

$$\varphi_{g(u)}(u) = \begin{cases} \varphi_{\varphi_u(u)}(x), \text{ se } \varphi_u(u) \downarrow \\ \uparrow, \text{ altrimenti} \end{cases}$$

Algoritmo 4: Punto fisso di Kleene

```
begin
    M(u) :=
    begin
        input(x);
        decodifica u per ottenere la MdT M_u che calcola φ_u
        sia y il risultato di φ_u(u);
        decodifica y per ottenere M_y che calcola φ_y = φ_{φ_u(u)}
        output(φ_y(x))
    end
end
```

Sia $t \in \mathbb{N} \to \mathbb{N}$ una funzione totale e calcolabile, allora anche $t \circ g$ lo è.

$$t \circ g = \varphi_v \qquad v \in \mathbb{N} \qquad \forall x \in \mathbb{N} \qquad \varphi_{g(v)}(x) = \varphi_{\varphi_v(v)}(x) = \varphi_{t(g(v))}(x)$$

dunque il parametro e del teorema di Kleene corrisponde a $g(v)$ e soddisfa la tesi.

$$\forall x \in \mathbb{N} \qquad \varphi_e(x) = \varphi_{t(e)}(x)$$

Questo teorema appena visto ne giustifica un altro. Ma prima occorre definire un concetto.

Definizione (insieme degli indici)**.** Sia F una famiglia di funzioni calcolabili. Un insieme $S \subseteq \mathbb{N}$ si dice insieme degli indici se

$$S = \{x \in \mathbb{N} : \varphi_x \in F\}$$

Dunque un qualsiasi sottoinsieme S dei $\mathbb{N}$ è un insieme degli indici se, $\forall\ i,\ j \in \mathbb{N}$ con $i \in S$ e $\varphi_i = \varphi_j$ si ha anche che $j \in S$.

Teorema (Rice)**.** L'insieme S degli indici di una famiglia di funzioni calcolabili F è decidibile se, e solo se, rispetta una delle seguenti condizioni:

a. $S = \emptyset$;

b. $S = \mathbb{N}$.

Dimostrazione (Rice)**.** Se $S = \emptyset$ è decidibile. Se $S = \mathbb{N}$ è ancora decidibile. Supponiamo invece che S non ricada nei precedenti due casi per mostrare che è indecidibile. Se, per assurdo, S fosse decidibile, allora le ipotesi su effettuate assicurano che ci sono funzioni calcolabili dentro e fuori di F. Siano i e j i due primi indici tali che

$$\varphi_i \in F, \quad i \in S \qquad\qquad \varphi_i \notin F \quad i \notin S$$

Dato che S è ipotizzata decidibile, vengono calcolati implicitamente da una MdT i due indirizzi i e j. Di conseguenza la funzione

$$g(x) = \begin{cases} i, \text{ se } x \notin S \\ j, \text{ se } x \in S \end{cases}$$

risulta calcolabile e totale. Inoltre si ha

$$g(x) \in S \iff x \notin S \quad = \quad \varphi_{g(x)} \in F \iff \varphi_x \notin F$$

Ma siccome g è totale calcolabile, il naturale preso in prestito dal teorema di Kleene porta ad una contraddizione:

$$\varphi_e \in F \iff \varphi_e \notin F$$

Dunque S non può essere decidibile.

Sezione 3

GRAMMATICHE ED AUTOMI

Ogni linguaggio ha una sua sintassi da rispettare. I metodi per definirlo sono distinguibili in:

- *approccio generativo*
  stringhe costruite secondo le regole prefissate
- *approccio riconoscitivo*
  sfruttando macchine astratte, dette automi riconoscitori, che accertano l'appartenenza ad un linguaggio di una stringa

3.1 Gerarchia di Chomsky

Le grammatiche sono organizzate in sottoclassi ripetute sempre più ristrette:

→ Tipo 0, grammatiche a struttura di frase

→ Tipo 1, grammatiche dipendenti dal contesto

→ Tipo 2, grammatiche libere dal contesto

→ Tipo 3, grammatiche regolari

Di per sé, una grammatica rispecchia una quadrupla del tipo

$$G = (N,\ T,\ P,\ S)$$

dove

- N è un insieme finito e non vuoto di simboli non terminali
- T è un insieme finito e non vuoto di simboli terminali
- P è un insieme finito e non vuoto di produzioni
- $S \in N$ e si chiama simbolo iniziale

3.1.1 Grammatiche a struttura di frase

Definizione (grammatica di tipo 0)**.** Le grammatiche a struttura di frase hanno produzioni P della forma

$$\alpha A \beta \to \alpha \gamma \beta$$

con $\alpha,\ \beta$ e γ stringhe di terminali e non.

Le grammatiche di tipo zero non accettano stringhe vuote nella parte sinistra delle produzioni.

Definizione (grammatica monotòna)**.** Si tratta di una grammatica a struttura di frase tale che ogni produzione è della forma $\alpha \to \beta,\ l(\alpha) \leq l(\beta)$.

I linguaggi generati da grammatiche monotòne coincidono con quelli dipendenti dal contesto.

3.1.2 Grammatiche dipendenti dal contesto

Definizione (grammatica di tipo 1)**.** Le grammatiche dipendenti dal contesto hanno produzioni P della forma

$$\alpha A \beta \to \alpha \gamma \beta \ :\ \gamma \neq \lambda$$

con $\alpha,\ \beta$ e γ stringhe di terminali e non.

3.1.3 Grammatiche libere dal contesto

Definizione (grammatica di tipo 2)**.** Le grammatiche libere dal contesto hanno produzioni P della forma $A \to y : y$ =stringa di terminali e non.

3.1.4 Grammatiche regolari

Definizione (grammatica di tipo 3)**.** Le grammatiche regolari hanno produzioni P della forma

$$A \to a \qquad\qquad A \to Aa \,\|\, A \to aA$$

$$S \to \varepsilon \iff S \text{ non compare a destra nelle produzioni}$$

3.2 Alberi di derivazione

Definizione (albero di derivazione)**.** Sia $G = (N,\ T,\ P,\ S)$ una grammatica libera del contesto e $w \in L(G)$ una stringa. L'albero di $w \in G$ è finito e:

- a ogni nodo si assegna come etichetta un simbolo in $N \cup T$;
- alla radice si assegna il simbolo iniziale S, ai nodi che non sono foglie si associano simboli non terminali di N e alle foglie simboli terminali di T;
- se un nodo, di etichetta A, ha come figli nell'ordine stabilito nell'albero i nodi (terminali o no) di etichette $A_1,\ \ldots,\ A_n$, allora $A \to A_1\ \ldots\ A_n$ sia una produzione di P;
- la stringa delle etichette delle foglie è w.

La profondità di un albero di derivazione è la distanza fra la base e il nodo foglia più lontano.

3.3 Automi di riconoscimento

Si cerca di ricondurre una determinata stringa al linguaggio a cui appartiene. Le classi di automi vanno di pari passo con la tipologia di linguaggio interessato:

Tabella 3.1: Classi di automi

LINGUAGGIO	AUTOMA
tipo 0	di turing
tipo 1	lineare limitato
tipo 2	a pila
tipo 3	a stati finiti

Le caratteristiche grafiche di realizzazione sono:

- una freccia in input per lo stato iniziale
- stati standard con singola circonferenza
- stati finali con doppia circonferenza
- archi di connessione fra stati

3.3.1 Automi a stati finiti deterministici e non

Definizione (ASFD)**.** Un automa a stati finiti deterministico $M = (Q,\ A,\ \delta,\ q_0,\ F)$ ha:

- Q = insieme finito e non vuoto degli stati;
- A = alfabeto;
- δ = funzione parziale che associa al più uno stato $q' \in Q$ ad ogni coppia $(q,\ a) \in Q \times A$;
- q_0 = stato iniziale;
- $F \subseteq Q$ = insieme degli stati finali.

Lemma (equivalenza fra ASFD e ASFND)**.** Per ogni automa a stati finiti non deterministico N esiste un automa a stati finiti deterministico D equivalente: $L(D) = L(N)$.

Dimostrazione (equivalenza fra ASFD e ASFND)**.** Dato un ASFND $N = (Q_N,\ A_N,\ \delta_N,\ q_0,\ F_N)$ si costruisce il corrispondente ASFD $D = \left(Q_D,\ A_D,\ \delta_D,\ q_0',\ F_D\right)$:

1. gli stati Q_D rappresentano i possibili sottoinsiemi di Q_N;
2. l'alfabeto è lo stesso, quindi $A_D = A_N$;
3. la funzione di transizione è

$$\delta_D\left([q_0,\ \ldots,\ q_i],a\right) = \left[\delta_N\left(q_0,\ a\right) \cup \delta_N\left(q_1,\ a\right) \cup \ldots \cup \delta_N\left(q_i,\ a\right)\right],\ \forall a \in A_D$$

4. $q_{0'}$ corrisponde al sottoinsieme $\{q_0\}$;
5. F_D rappresentano tutti i sottoinsiemi di Q_N che contengono un elemento $\in F_N$, ovvero i sottoinsiemi che contengono almeno un F_N.

Ecco dunque che D comprende tutte le parole riconosciute da N, e viceversa.

Sezione 4

LINGUAGGI FORMALI

Come accade per le grammatiche, si hanno diversi livelli racchiusi in delle sottoclassi.

Tabella 4.1: Classi di linguaggi

TIPO	LINGUAGGIO	CARATTERISTICHE
0	a struttura di frase	semidecidibile
1	dipendenti dal contesto	$\{a^n b^n c^n \ : \ n > 1\}$
2	liberi dal contesto	$\{a^n b^n \ : \ n > 1\}$
3	regolari	$\{a^n b^m \ : \ n, \ m > 1\}$
/	finiti	$\{aa, \ bc, \ ab\}$

4.1 Linguaggi regolari

Linguaggi generabili da una grammatica di tipo 3 e riconoscibili da un automa a stati finiti deterministico.

Teorema (pumping lemma per linguaggi regolari)**.** Sia $L \in T$ un linguaggio regolare.

$$\exists p \in \mathbb{N} \ : \ \forall z \in L \text{ se } l(z) \geq p \text{ allora } \exists u, \ v, \ w \in T^\star \mid z = uvw$$

e valgono le seguenti proprietà:

a. $l(v) \geq 1$;

b. $l(uv) \leq p$;

c. $\forall i \in \mathbb{N}, \ uv^i w \in L$.

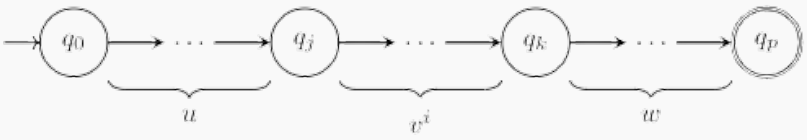
Dimostrazione (pumping lemma per linguaggi regolari)**.** Sia $L \in T$ un linguaggio regolare e $M = (Q, A, \delta, q_0, F)$ l'automa a stati finiti deterministico che lo riconosce. Fissato inoltre p come il numero degli stati di M, $p = \mid Q \mid$. Data poi una stringa

$$z \ : \ l(z) \geq p$$

che ci indica che l'automa contiene un ciclo allo stato q_s. Dati infini due indici j e k, con $j \neq k$, delle due occorrenze di q_s si ha che z si divide nella seguente concatenazione:

$$z = uvw$$

dove u è la porzione precedente al primo q_s e w quella dopo il secondo.



Le tre condizioni del teorema sono soddisfatte:

1. dato che $j \neq k$, allora $l(v) \geq 1$
2. sapendo che $k \leq p + 1$ e che $l(z) = p$, inoltre anche $l(uv) = k - 1$. Dunque $l(uv) = k - 1 \leq p$
3. ponendo $i = 0$ si ha la stringa $uv^0w = uw$ che viene accettata dall'automa ignorando il ciclo che ha generato v. Per $i > 1$ invece, la stringa che si ottiene viene accettata dall'automa ripetendo il ciclo altrettante volte.

4.2 Linguaggi liberi dal contesto

Generabili da grammatiche di tipo 2.

Teorema (pumping lemma per linguaggi liberi dal contesto)**.** Sia $L \in T$ un linguaggio libero dal contesto

$$\exists p \in L \ : \ \forall z \in L, \text{se } l(z) \geq p \text{ allora } \exists u, \ v, \ w, \ x, \ y \in T^\star \mid z = uvwxy$$

Valgono le seguenti proprietà:

(a) $l(vx) \geq 1$; (b) $l(vwx) \leq p$; (c) $\forall i \in \mathbb{N}, uv^iwx^iy \in L$.

Dimostrazione (pumping lemma per linguaggi liberi dal contesto)**.** Considerati tutti gli alberi di derivazione di una grammatica G con un simbolo non terminale come etichetta di radice e profondità $\leq n+1$, dove n sta a conteggiare il numero di simboli non terminali in essa; e sia inoltre k la lunghezza massima delle stringhe generate dai nodi foglia. Ponendo $p = k + 1$, allora z può solo essere generata da un albero di derivazione di profondità $> n + 1$, esistono due nodi N_1, N_2 tali che:

- hanno la stessa etichetta;
- il secondo discende dal primo;
- il cammino da N_1 alla foglia è di lunghezza $n + 1$.
- dimostrazione punto (c)
  Si intende ora sostituire i sottoalberi di radice dei due nodi interessati ed inoltre generare una stringa della forma $uv^iwx^iy \in L \qquad i \geq 0$.
- dimostrazione punto (b)
  Il sottoalbero di radice N_1 ha profondità $\leq n + 1$, quindi la stringa che viene generata, ossia vwx, ha lunghezza $\leq p$.
- dimostrazione punto (a)
  Se $l(vx) = 0$, allora $z = uwy$. Continua ad esser generata anche se viene diminuito il numero di nodi dell'albero, ciò però contraddice la scelta dell'albero stesso.

4.3 Linguaggio while

Esiste un linguaggio di programmazione imperativo adattato a rappresentare l'insieme di tutte le funzioni calcolabili. Viene scritto in pseudocodice, utilizzando quindi una sintassi alternativa e meno rigida dei comuni, e successivi, linguaggi di programmazione.

4.3.1 Sintassi

Come ogni linguaggio dispone di una sintassi da rispettare, in questo caso si seguono le regole di produzione di una grammatica libera dal contesto.

Definizione (programma)**.** Sequenza ordinata, eventualmente vuota, di comandi contenuti in un blocco.

Nota bene (notazione punti e virgola)**.** L'ultimo comando dentro un qualunque blocco, prima dell'etichetta **end**, non va seguito da punto e virgola.

Algoritmo 5: Blocco del ciclo while

```
begin
    while condizione do
        begin
            seqcommand;
            seqcommand;
            seqcommand;
            command
        end
end
```

Oltre che per l'intero programma, i blocchi sono disposti anche per delimitare cicli e condizioni. Il nome attribuito alle variabili può contenere cifre ma non come primo carattere.

4.3.1.1 Macro

Operazioni di aumento e decremento unitario vengono effettuate tramite funzioni apposite.

Algoritmo 6**:** Incrementazione e decrementazione di variabile

```
1. begin
2.     x := s(x)      ▷ successivo
3.     x := pd(x)     ▷ precedente
4. end
```

Di seguito sono elencate macro e sintassi specifiche del linguaggio while.

- assegnamento di costante: $x := n$, con n valore numerico
- assegnamento di variabile: $x := y$
- somma: $x := x + y,\ x := z + y$
- differenza: $x := x - y,\ x := z - y$
- differenza troncata: $x := x \dot{-} y,\ x := z \dot{-} y$
- prodotto: $x := x \cdot y,\ x := z \cdot y$
- divisione intera: $x := x/y,\ x := z/y$
- modulo: $x := x\%y,\ x := z\%y$
- operatori di confronto: $x \geq y,\ x \leq y,\ x = y,\ x > y,\ x < y$
- costanti booleane: `True`, `False`
- operatori booleani: $x_\epsilon \wedge y_\epsilon,\ x_\epsilon \vee y_\epsilon,\ \neg x_\epsilon$
- assegnamento di espressione booleana: $x := T_\epsilon$
- while con condizione booleana: `while` T_ϵ `do` command
- if-then con condizione booleana: `if` T_ϵ `then` command
- if-then else con condizione booleana: `if` T_ϵ `then` command `else` command
- input di un valore su una variabile: `input(var)`
- output del valore di una variabile: `output(var)`

Ciò considerando ξ come un'espressione booleana e T_ξ la rispettiva valutazione:

$$T_\xi = \begin{cases} 1, \text{ se } \xi \text{ è } \texttt{tt} \\ 0, \text{ se } \xi \text{ è } \texttt{ff} \end{cases}$$

Esercizio (divisione manuale)**.** Scrivere del pseudocodice che presi due numeri restituisca il risultato della loro divisione intera.

Non avendo l'operatore della divisione occorre ingegnarsi con sottrazioni troncate multiple.

Algoritmo 7: Divisione manuale

```
begin
    INPUT(x);
    INPUT(y);
    z := 0;
    while x ≥ y do
        begin
            x := x ∸ y;
            z := s(z)
        end
    OUTPUT(z)
end
```

4.3.2 Funzioni computabili

Le tre funzioni d'uso comune viste nella teoria della computabilità possono essere riproposte anche utilizzando il linguaggio WHILE.

Algoritmo 8**:** Funzione identità

```
1. begin
2.     begin
3.     end
4. end
```

In questo caso il programma sarà vuoto, proprio per riproporre la restituzione dello schema iniziale.

Algoritmo 9**:** Funzione costante

```
1. begin
2.     X := 0
3. end
```

Viene semplicemente assegnato un valore ad una variabile e terminata l'esecuzione. In questo modo la variabile non potrà cambiare stato e fungerà da costante. È inoltre noto come un singolo comando non necessiti del punto e virgola finale.

Algoritmo 10**:** Funzione indefinita

```
1. begin
2.     X := 0;
3.     Y := s(X);
4.     while X ≠ Y do
5.         begin
6.         end
7. end
```

Si assegnano a due variabili dei valori differenti: in questo caso l'uno il successivo dell'altro. Poi si va a creare un ciclo che, nel caso in cui i due elementi siano diversi non fa nulla.

Esercizio (emulazione del **repeat until**)**.** Definire una macro del tipo **repeat** *C* **until** *E*. Dove *C* è un comando ed *E* è un'espressione booleana. Informalmente, il comando esegue *C* e poi verifica *E*. A questo punto se *E* è falsa ripete *C*, altrimenti esce dal ciclo.

Non potendo adoperare il blocco **repeat until** si va a simularlo replicandolo con gli strumenti disponibili nel linguaggio.

Algoritmo 11: Emulazione del **repeat until**

```
begin
    C;
    while E = False do
        begin
            C
        end
end
```

Bibliografia

[BC23] Matteo Belenchia and Flavio Corradini. Teoria della computabilità. Serie. Università di Camerino, 2023.

[CB23] Flavio Corradini and Matteo Belenchia. Esercizi di Fondamenti di Informatica. Università di Camerino, 2023.