Atti Accademia Nazionale dei Lincei

Classe Scienze Fisiche Matematiche Naturali

Rendiconti

Corrado Böhm, Giuseppe Jacopini

Nuove tecniche di programmazione semplificanti la sintesi di macchine universali di Turing

Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti, Serie 8, Vol. **32** (1962), n.6, p. 913–922.

Accademia Nazionale dei Lincei

¡http://www.bdim.eu/item?id=RLINA_1962_8_32_6_913_0¿

Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti, Accademia Nazionale dei Lincei, 1962.

Calcolo numerico. — *Nuove tecniche di programmazione semplificanti la sintesi di macchine universali di Turing.* Nota di CORRADO BÖHM e GIUSEPPE JACOPINI, presentata (*) dal Socio M. PICONE.

Dopo Turing (1) vari Autori, tra cui Kleene (2), Davis (3), Wang (4), Hermes (5) hanno fornito definizioni differenti (ma non in modo essenziale) di macchine di Turing. Essi hanno scopi simili connessi con l'identificazione della classe delle funzioni calcolabili mediante m. di Turing del tipo considerato con la classe delle funzioni ricorsive.

Il punto di vista di Shannon (6) e parzialmente di Lee (7) e di altri (8) è invece quello di paragonare fra loro macchine di Turing di una certa famiglia, quella delle macchine universali (in grado cioè di simulare qualsiasi altra macchina Z di Turing) ed aventi determinate proprietà di minimo, per esempio il numero di elementi della matrice che rappresenta una di queste macchine. In un recente lavoro di Watanabe (9) si è fatto ad esempio scendere tale numero a 40 (10).

Nel presente lavoro siamo partiti dall'idea che ogni progresso di tecnica di programmazione sia in qualche modo trasferibile da un tipo di macchina ad un altro: in particolare che la tecnica di programmazione dei calcolatori attuali sia essenzialmente la stessa di quella delle macchine di Turing (11).

(*) Nella seduta del 12 giugno 1962.

(1) A. M. TURING, *On Computable Numbers, with an Application to the Entscheidungsproblem*, « Proc. London Math. Society », ser. 2, vol. XLII, pp. 230–265 (1936).

(2) S. C. KLEENE, *Introduction to Metamathematics*, van Nostrand (1952).

(3) M. DAVIS, *Computability and Unsolvability*, McGraw–Hill (1958).

(4) H. WANG, *A Variant to Turing's Theorie of Computing Machines*, « Journ. of the Ass. for Computing Machinery », vol. IV, pp. 63–92 (1957).

(5) H. HERMES, *Aufzahlbarkeit, Entscheidbarkeit, Berechenbarkeit*, Springer,Verlag (1961).

(6) C. E. SHANNON, *A Universal Turing Machine with Two Internal States* in *Automata Studies*, pp. 157–165; « Annals of Math. Studies », 34 (1956).

(7) C. Y. LEE, *Automata and finite automata*, « Bell Syst. Techn. Journ. », *39*, 5, Sept. pp. 1267–1295 (1960).

(8) Vedi bibliografia di (9).

(9) S. WATANABE, *5 Symbol 8–State and 5–Symbol 6–State Universal Turing Machines*, « Journ. of the Ass. for Comp. Mach. », vol. VIII, 4, pp. 476–483 (1961).

(10) Lo stesso Watanabe avrebbe creato anche una U_6^5 ma questa come Egli stesso ammette non è una macchina di Turing, in quanto la descrizione della macchina da simulare ha lunghezza infinita (in effetti ammettere che il nastro contenga sin da principio una infinità di caratteri diversi dal vuoto, distrugge il concetto usuale di calcolo riducendolo in certi casi alla lettura di una tabella).

(11) A sostegno di questa idea, circa metà del corso di « Tecnica di programmazione » di Böhm, presso l'Università di Roma, è stato dedicato alle m. di Turing. Il presente lavoro, parziale oggetto della tesi di laurea di Jacopini, può essere considerato come un primo frutto dell'orientamento prescelto. Cogliamo l'occasione per ringraziare il prof. Wolf Gross per la sua costruttiva critica.

Il presente lavoro termina con la descrizione di una macchina universale U_7^5 la cui tabella consta di 32 quintuple. Noi consideriamo il risultato raggiunto non alla stregua di un record (che forse può essere ulteriormente abbassato) ma come una misura obbiettiva del presente stato raggiunto dalle tecniche di programmazione. In particolare è stato fornito un esempio di « compilazione » al livello delle macchine di Turing. Ci si è limitati a compilare i « salti condizionati » mediante una schiera ordinata di super–programmi ognuno dei quali ha lo scopo di compilare i salti del programma oggetto a cui esso viene applicato; tuttavia il metodo si presterebbe ad ulteriori applicazioni.

Nella prima parte della Nota il numero di tipi diversi di macchine elementari in cui ogni macchina di Turing si può decomporre viene ridotto successivamente da $m + 2$ (m è il numero di caratteri dell'alfabeto) a 3 e da 3 a 2.

Si mostra poi (1.2) l'impossibilità di un'ulteriore riduzione. Nella seconda parte i risultati precedenti vengono applicati alla costruzione di una macchina universale. Viene introdotto (2.1) un alfabeto di due simboli ed il programma normalizzato in forma « rudimentale ». I salti condizionati vengono poi eliminati formalmente e viene introdotto il programma definitivo parziale (2.2) il super programma ed il programma definitivo totale (2.3). Infine viene descritta una macchina universale U_7^5 (2.4) e ne viene spiegato il comportamento (2.5).

1. Una generica macchina di Turing M_n^m a m caratteri:

(c_0, c_1, $\cdots$ $c_p \cdots$, c_{m-1}) e n stati interni (q_1, q_2, $\cdots$ q_z, $\cdots$ q_n) è, come è noto, rappresentata da una matrice [12] $n \times m$, in cui l'elemento generico (all'incrocio della H^a riga con la K^a colonna) o è vuoto o è costituito da una terna del tipo:

$$c_{p_{HK}} \quad t_{HK} \quad q_{z_{HK}} \qquad \text{dove:} \quad p_{HK} \in \{0, 1, \cdots, m-1\},$$

$$t_{HK} \in \{s, i, d\} \qquad \text{e} \qquad z_{HK} \in \{1, 2, \cdots, n\}.$$

Hermes [5] ha dimostrato che tale macchina può essere decomposta per diramazione in un certo numero di esemplari di macchine cosiddette « elementari » appartenenti agli $m + 2$ tipi:

C_0, C_1, $\cdots$, C_{m-1}, S, D dove:

$C_i =$

q_1	$c_i \; i \; q_2$ | $c_i \; i \; q_2$	$c_i \; i \; q_2$
q_2		

(12) Tuttavia noi considereremo equivalenti tutte le macchine che, pur essendo rappresentate da differenti matrici, hanno l'effetto di far passare dalle stesse configurazioni iniziali, alle stesse configurazioni finali.

questa macchina sostituisce qualsiasi carattere si trovi nella cella osservata con c_i

$$S = \begin{array}{|c|c|c|c|c|}\hline q_1 & c_0\ s\ q_2 & c_1\ s\ q_2 & \cdots & c_{m-1}\ s\ q_2 \\ \hline q_2 & & & & \\ \hline \end{array}$$

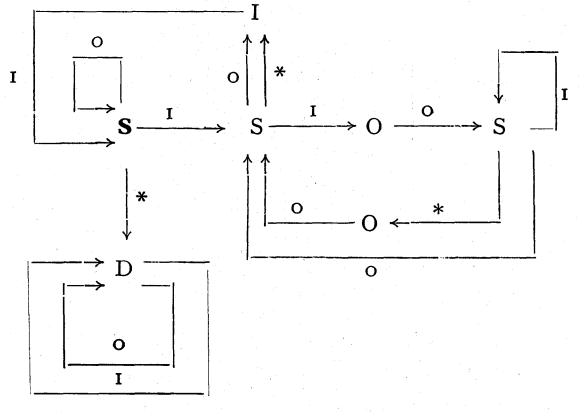
questa macchina sposta di un passo a sinistra la cella osservata

$$D = \begin{array}{|c|c|c|c|c|}\hline q_1 & c_0\ d\ q_2 & c_1\ d\ q_2 & \cdots & c_{m-1}\ d\ q_2 \\ \hline q_2 & & & & \\ \hline \end{array}$$

questa macchina sposta di un passo a destra la cella osservata. Tale decomposizione ammette una comoda rappresentazione attraverso un diagramma di flusso. (Cfr. Hermes [5], cap. II, p. 46).

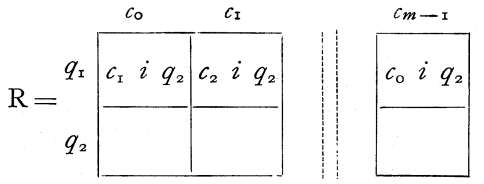
Esempio ($m = 3$ $c_0 = *$, $c_1 = 0$, $c_2 = 1$). È stato tracciato il diagramma di flusso della macchina T [13].

$$T = \begin{array}{|c|c|c|c|}\hline & * & 0 & 1 \\ \hline A & * \ s\ B & 0\ s\ B & 1\ s\ B \\ \hline B & * \ d\ E & 0\ s\ B & 1\ s\ C \\ \hline C & 1\ d\ E & 1\ s\ B & 0\ s\ D \\ \hline D & 0\ s\ C & 0\ s\ C & 1\ s\ D \\ \hline E & & 0\ d\ E & 1\ d\ E \\ \hline \end{array}$$



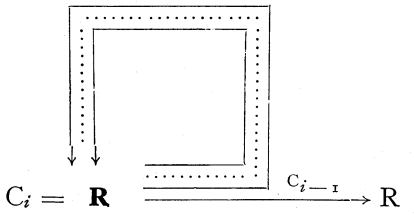
(13) Se la configurazione iniziale è per esempio $\mathfrak{C}_1 \equiv \cdots *** n\underline{*}*\cdots$ (dove n sta per la sua rappresentazione binaria nei caratteri c_1, c_2, e la sottolineatura indica la casella osservata) la configurazione finale è $T(\mathfrak{C}) \equiv \cdots ** 3\, n\underline{*}*\cdots$

1.1. Ci si può domandare se non sia possibile ridurre il numero di tipi di macchine elementari differenti, mantenendo la possibilità di descrivere per mezzo di esse qualsiasi altra macchina. Per far ciò osserviamo che ogni macchina C_i può ottenersi componendo per diramazione la macchina R con se stessa.



L'effetto di questa macchina è di sostituire il carattere (contenuto nella cella osservata) c_j con c_{j+1} (dove $c_m \equiv c_0$).

Risulta:



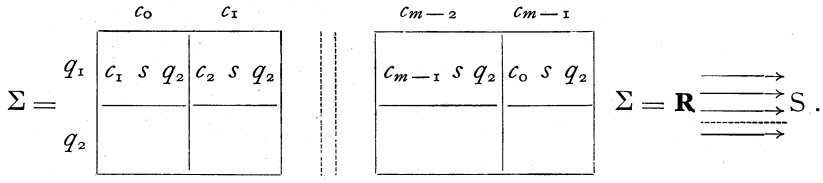
(dove sono sottintesi tutti i contrassegni delle linee tranne uno).

Con ciò, ammettendo come ovvio il:

LEMMA 1.11. – Se una macchina è decomponibile in certe macchine e ognuna di questo lo è in certe altre, la prima è decomponibile nelle ultime, ne segue quest'altro:

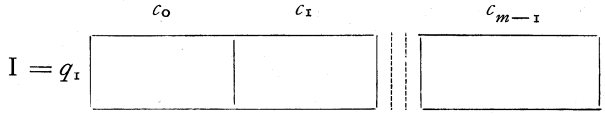
LEMMA 1.12. – Ogni macchina di Turing è decomponibile nelle tre macchine: S , D , R .

Vediamo come questo numero (3) si possa ulteriormente ridurre: introduciamo una nuova macchina:



Risulta, $R = \Sigma D$ (Intendiamo con AB, se A e B sono due macchine: **A** $\xrightarrow[c_1]{c_0}$ $\xrightarrow[c_{m-1}]{}$ B intendiamo inoltre con A^k $\underbrace{A\,A\cdots A}_{k \text{ volte}}$).

Dette inoltre: $I_1 = (D\Sigma)^m$ e $I_2 = (\Sigma D)^m$ risulta: $I_1 = I_2 = I$ (dove I è la macchina identica così definita):



Quindi essendo ovviamente $S = SI_1 = I_2 S = (\Sigma D)^{m-1} \Sigma$, si può enunciare, tenuto di nuovo conto del Lemma 1.11 e del Lemma 1.12:

TEOREMA I. – Ogni macchina è decomponibile nelle due macchine D , Σ.

1.2. Ogni tentativo di proseguire il procedimento mediante la scoperta di un'unica macchina nella quale poter decomporre la Σ e la D, è destinato a fallire poiché vale il seguente

TEOREMA 1.21. – Non esiste alcuna macchina A mediante la quale ogni altra macchina possa essere decomposta.

Possiamo enunciare i due seguenti lemmi di cui consideriamo ovvia la dimostrazione:

LEMMA 1.22. – Se $\mathcal{C}_1$ e $\mathcal{C}_2$ sono due qualsiasi configurazioni esiste una M tale che $M(\mathcal{C}_1) = \mathcal{C}_2$.

LEMMA 1.23 [14]. – Se una macchina M è decomponibile nelle $A_1 , A_2 , A_3 , \cdots , A_n$, per ogni $\mathcal{C}_1$ esiste una M* tale che:

$$M^*(\mathcal{C}_1) = M(\mathcal{C}_1) \quad \text{dove} \quad M^* = A_{i_1} , A_{i_2} \cdots A_{i_m} \qquad (i_s \in \{1 , 2 \cdots n\}).$$

Applichiamo il lemma 1.23 al caso $n = 1 , A_1 = A$. Si avranno due casi:

1) o non esiste una M* tale che $M^*(\mathcal{C}_1) = \mathcal{C}_1$: allora tenuto conto del lemma 1.22 (per $\mathcal{C}_2 = \mathcal{C}_1$) il teorema è dimostrato;

2) M* esiste; allora $M^* = A^m$ (m dipende solo da $\mathcal{C}_1$). Inoltre per qualsiasi altra configurazione $\mathcal{C} \neq \mathcal{C}_1$ supponendo falso il teorema 1.21 e facendo di nuovo ricorso al lemma 1.22 si avrebbe: $A^k(\mathcal{C}_1) = \mathcal{C}$ quindi posto: $k = r + mh \ (h \geq 0 , m > r > 0)$ $\mathcal{C} = A^{r+mh}(\mathcal{C}_1) = A^r(M^{*h}(\mathcal{C}_1)) = A^r(\mathcal{C}_1)$.

Da ciò consegue l'esistenza di al più $m - 1$ configurazioni $\mathcal{C}$ diverse da $\mathcal{C}_1$, il che è palesemente assurdo.

2. Abbiamo finora dimostrato la possibilità di descrivere una qualsiasi M_n^m con un diagramma di flusso nel quale compaiano soltanto due tipi di macchine elementari al posto delle $m + 2$ tradizionali; abbiamo anche dimostrato di non poterci spingere oltre. È nostra intenzione costruire una macchina universale che utilizzi tale diagramma (o meglio una sua codificazione) come descrizione (che chiameremo anche « programma ») della macchina simulata, nella plausibile convinzione che ciò comporti una qualche semplificazione nella macchina universale stessa.

2.1. Per far ciò ci limiteremo a supporre che si operi su un alfabeto con soli due caratteri $c_0 = 0$ e $c_1 = 1$ (e questo non rappresenta una perdita di generalità com'è dimostrato da Shannon [6]). In questo alfabeto le macchine Σ e D si particolarizzano nelle σ e d dove:

$\sigma =$

	0	1
A	1 *s* B	0 *s* B
B		

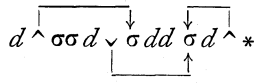
$d =$

	0	1
A	0 *d* B	1 *d* B
B		

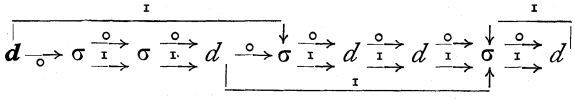
(14) Basta infatti osservare che l'ordine con cui le singole A_i intervengono nel calcolo è univocamente determinato dalla configurazione iniziale $\mathcal{C}_1$.

Il diagramma di flusso ha carattere bidimensionale. Ci proponiamo di scrivere un programma *linearizzato rudimentale* dove valgano convenzioni di cui sarà tenuto conto nella costruzione della macchina universale.

Le macchine elementari saranno scritte di seguito su una riga. Esistono alcune linee orientate o frecce senza contrassegno (che chiameremo in seguito *salti*) che partono dopo alcune macchine del tipo d ed arrivano su macchine del tipo σ. Non più di una freccia può partire dopo una d, mentre più frecce possono arrivare alla medesima σ. L'estremo destro del programma è caratterizzato dalla presenza di un $*$. Esempio:



Ogni programma è equivalente ad un diagramma di flusso ottenuto contrassegnando con 1 le frecce esistenti ed aggiungendo frecce con opportuni contrassegni fra macchine consecutive, in modo che da ogni macchina (eccettuata quella antecedente al $*$) partono due frecce con entrambi i contrassegni ed eliminando il $*$ e scrivendo in grassetto la prima macchina a sinistra. L'esempio precedente ha perciò il significato:



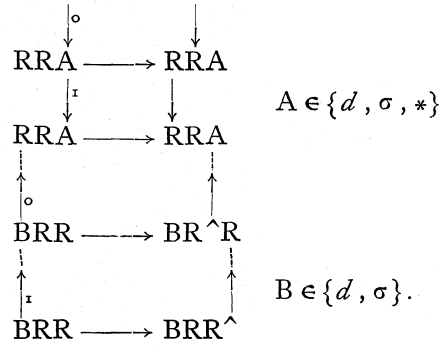
Mostriamo ora come ogni diagramma di flusso possa venire tradotto in un programma linearizzato. Si può operare, ad esempio, così:

1) Se da alcune macchine si dipartono meno di due frecce contrassegnate si introduca nel diagramma un $*$ e verso di esso si facciano convergere tutte le linee mancanti (opportunamente contrassegnate).

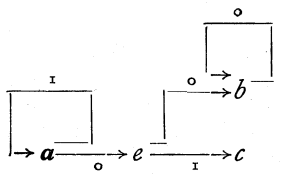
2) Si scrivano le macchine componenti il diagramma su una riga (spostando solidalmente le linee orientate) in un ordine arbitrario con la restrizione che la macchina in grassetto occupi l'estremo sinistro e il $*$ l'estremo destro.

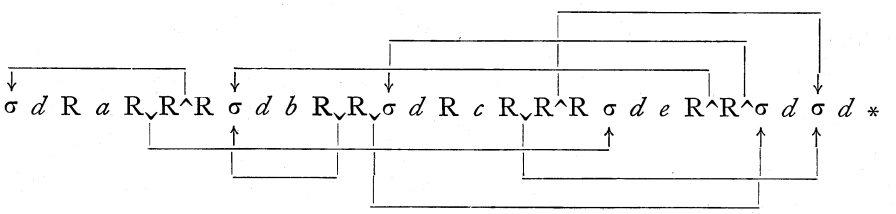
3) Ad ognuna delle macchine scritte si anteponga e si posponga RR ed al $*$ si anteponga R R, dove R sta per σd.

4) Si eseguano le seguenti sostituzioni



Esempio:

Il diagramma  dove $a, b, c, e \in \{\sigma, d\}$ diventa



È utile osservare che la macchina R nell'alfabeto (0, 1) è così definita

R =	0	1
A	1 *i* B	0 *i* B
B		

cioè ha l'effetto di scambiare il carattere osservato $\begin{pmatrix} 0 \to 1 \\ 1 \to 0 \end{pmatrix}$.

La forma di I_2 nell'alfabeto $\{0, 1\}$ è infatti $\sigma\, d\, \sigma\, d$ (la forma di I_1: $d\sigma\, d\sigma$).

Supponiamo ora che il programma contenga $n + 1$ salti; numeriamoli da 0 ad n in ordine qualsivoglia e facciamo seguire al segno di inizio del salto r–esimo il gruppo di funzioni identiche $I_1^r\, I_2$ ($I_1^0\, I_2 = I_2$). (Questa convenzione che può essere in molti modi modificata e nella maggior parte dei casi concreti ignorata, ha il solo scopo di permetterci di risolvere in modo generale il problema che più avanti ci porremo).

Aggiungiamo alla sinistra e alla destra (comunque prima del *) del programma una certa quantità di funzioni identiche (indifferentemente I_1 o I_2) del cui numero e della cui funzione diremo in seguito. Sostituiamo in ogni sua occorrenza $d \overset{\wedge}{\rightarrow}$ o $d \overset{\wedge}{\leftarrow}$ con $\hat{d}$ e rovesciamo l'intera parola (scriviamola cioè da destra verso sinistra). Infine codifichiamo i simboli $\sigma, d, \hat{d}, *$ nel modo seguente

$$\sigma \to 1 \quad , \quad d \to 10 \quad , \quad \hat{d} \to 00 \quad , \quad * \to *$$

(1, 10, 00, * saranno in seguito chiamate istruzioni). In tal modo abbiamo ottenuto dal programma in forma rudimentale una parola nei caratteri {0, 1} più un solo * iniziale; ma questa parola non è che una descrizione parziale del programma di partenza in quanto si è persa ogni traccia del punto di arrivo di ciascun salto (la chiameremo perciò: « programma definitivo parziale »).

2.3. Vogliamo ora risolvere il seguente problema: supponiamo di avere un programma definitivo parziale in cui l'r–esimo salto (ci riferiamo alla

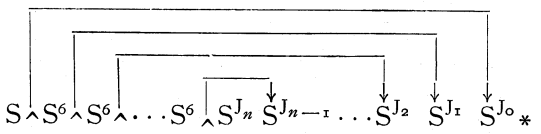
numerazione precedentemente introdotta) codificato con o o occupi le posizioni i_r e i_{r+1} a partire dal $*$ a sinistra e il punto di arrivo del salto (che non è individuabile dal programma parziale ma che è evidentemente conosciuto dal programmatore) sia un 1 (infatti $\sigma \to 1$) che occupa la posizione $i_r - t_r$ ($t_r \gtrless 0$); interpretiamo il programma parziale $\mathfrak{D}_0$ come il contenuto del nastro di una macchina di Turing \$ avente le seguenti caratteristiche:

1) se inizialmente la macchina \$ osservava la i_r–esima cella al termine del calcolo dovrà fermarsi sulla $(i_r - t_r)$–esima;

2) al principio e alla fine del calcolo il contenuto del nastro deve rimanere il medesimo (potrà però essere diverso durante l'esecuzione);

3) durante l'esecuzione del calcolo la macchina \$ non dovrà uscire dalla parte di nastro occupata dal programma parziale (non dovrà neppure portarsi sulla cella occupata dal $*$).

Il programma che descrive questa macchina \$ può essere rappresentato in forma rudimentale nel modo (per esempio) seguente:



dove:

$$\begin{cases} S^k = (\sigma\, d\, \sigma)^{k+1}\, d \\ S^{-k} = \sigma\, d\, \sigma\, d^{k+1} \\ S^0 = \sigma\, d\, \sigma\, d \end{cases} \qquad \begin{cases} J_0 = t_0 - 1 \\ J_r = t_r - 6 - t_{r-1}. \end{cases}$$

(La condizione 3 sarà sempre verificata se sarà sufficiente il numero di funzioni identiche poste ai due estremi del programma parziale).

Questo programma verrà chiamato super–programma del programma parziale $\mathfrak{D}_0$. Se un programma parziale è privo di salti ha super–programma vuoto.

Dal super–programma in forma rudimentale si può passare alla forma definitiva $\mathfrak{D}_1$.

$\mathfrak{D}_1$ ammetterà in generale un super–super–programma $\mathfrak{D}_2$ con al più $n - 1$ salti, e così via, fino a $\mathfrak{D}_{n+1}$ privo di salti. Chiamiamo la parola:

$$\mathfrak{S} = \mathfrak{D}_{n+1}\, \mathfrak{D}_n \cdots \mathfrak{D}_1\, \mathfrak{D}_0$$

programma definitivo totale.

Da quanto precede risulta che $\mathfrak{D}_{n+1}\, \mathfrak{D}_n \cdots \mathfrak{D}_1$ è il super–programma definitivo totale cioè il programma definitivo totale della macchina \$.

2.4. Introduciamo un alfabeto a 5 caratteri $\{0, 1, *, 0', 1'\}$; verranno chiamati marcati i 3 seguenti caratteri:

$$0', 1', * \text{ e smarcati i 3 seguenti } 0, 1\, *.$$

Denoteremo con $\sim$ il generico carattere smarcato e con $\sim'$ il corrispondente $\begin{pmatrix} 0 \longleftrightarrow 0' \\ 1 \longleftrightarrow 1' \\ * \longleftrightarrow * \end{pmatrix}$ marcato.

Inoltre se $\mathfrak{L}$ è una parola nei caratteri $\sim$, $\mathfrak{L}'$ sarà la parola formata dai caratteri corrispondenti $\sim'$.

Sia Z_n^2 una macchina di Turing e $\mathfrak{S}$ il suo programma definitivo totale; supponiamo che la Z_n^2 operando su una configurazione iniziale $\mathfrak{C}_1 = (\cdots 0\,0\,0\,P_1 \simeq Q_1\,0\,0\,0\cdots)$ si arresti su una $\mathfrak{C}_2 = Z_n^2(\mathfrak{C}_1) = (\cdots 0\,0\,0\,P_2\,\underline{0}\,Q_2\,0\,0\,0\cdots)$ (P_1, Q_1, P_2, Q_2 sono parole in $\{0, 1\}$; $\sim \neq *$; il segno $\underline{\ }$ indica il carattere osservato dalla Z_n^2). Se si fa l'ipotesi [15] che la parola P_2 non vada ad occupare le celle che inizialmente erano occupate dagli 0 a sinistra della parola P_1 (la quale peraltro poteva iniziare con un numero qualsivoglia di 0) e che neppure durante l'esecuzione del calcolo queste celle siano usate, risulta che esiste una macchina U_7^5 tale che:

$$U_7^5(\cdots 0\,0\,0\,\mathfrak{S}'\,\underline{*}\,P_1 \sim' Q_1\,0\,0\,0\cdots) = (\cdots 0\,0\,0\,\mathfrak{S}'\,*\,P_2\,\underline{0}'\,Q_2\,0\,0\,0).$$

La rappresentazione della U_7^5 è la seguente matrice:

$U_7^5 =$	0	1	0′	1′	*
A			0 *s* A	1 *s* A	* *s* B
B	0′ *s* F	1′ *s* E	0 *d* C	1′ *d* D	* *d* G
C	0 *d* C	1 *d* C	0 *d* B	1 *d* B	* *d* C
D	0 *d* D	1 *d* D	1 *s* B	0 *s* B	* *d* D
E	0 *s* E	1 *s* E	0′ *s* A	1 *s* B	* *s* E
F	0 *s* F	1 *s* F	0 *s* B	1 *s* B	* *s* F
G	0′ *d* G	1′ *d* G		1′ *i* B	* *d* G

2.5. Gli atti essenziali di questa macchina sono:

1) La U_7^5 (stato A) va ad esaminare (B) il carattere a sinistra

$$\begin{cases} 0' \longrightarrow 2) \\ 1' \longrightarrow 3) \\ * \longrightarrow 6). \end{cases}$$

2) La U_7^5 osserva uno 0′ (d o $\hat{d}$) del programma, lo smarca, va a destra a cercare (C) il primo 0′ o 1′ per eseguire l'istruzione d (sposta cioè a destra la marcatura) (C, B), quindi torna al programma (negli stati E o F a seconda del carattere marcato per ultimo) $\longrightarrow 4)$.

(15) In Kleene [2] si dimostra che tale ipotesi non lede la generalità.

3) La U_7^5 osserva un 1, non lo smarca, ma va (D) direttamente ad eseguire l'istruzione σ (D , B , A) quindi ritorna (negli stati E o F come sopra) $\longrightarrow$4).

4) La U_7^5 ritorna al programma (F o E) a seconda che il carattere marcato era 0′ o 1′; nel primo caso smarca il primo 0′ o 1′ che incontra e va ancora a sinistra a eseguire la prossima istruzione $\longrightarrow$2). Nel secondo caso se incontra un 1′ si comporta allo stesso modo, se incontra invece uno 0′ $\longrightarrow$5).

5) Va a sinistra senza alterare lo 0′, smarcando però tutti gli altri caratteri marcati che incontra (A), quindi passa ad esaminare il super–programma $\longrightarrow$ 1).

6) Va a destra marcando tutti i caratteri smarcati (G) per rientrare dal super–programma al programma oggetto, oppure per eseguire lo STOP a seconda del primo (0′ o 1′) che incontra

$$\begin{cases} 1' \longrightarrow 2) \\ 0' \longrightarrow \text{STOP} \end{cases} \text{(16)}.$$

(16) Si è imposto per semplicità che la macchina simulata Z_n^2 dovesse arrestarsi alla fine del calcolo su una cella occupata del carattere 0 ; questa limitazione alla universalità della U_7^5 (peraltro non essenziale) può però essere eliminata se si introduce alla fine del programma parziale un salto al super–programma che in questo caso dovrà avere l'effetto di portare la marcatura nel programma parziale su uno 0 anziché su un 1 in modo che il successivo rientro provochi lo STOP. Pertanto se la configurazione finale della Z_n^2 è

$$\cdots 000\, P_2\, \underline{1}\, Q_2 \cdots$$

e il suo programma

$$\mathcal{S} = \mathcal{S}_1 \circ \mathcal{S}_2$$

la configurazione finale sarà

$$\cdots 000\, \mathcal{S}_1'\, \underline{0'}\, \mathcal{S}_2 * P_2\, 1'\, Q_2\, 000 \cdots$$