Abbiamo con dimostrato alcune equivalenze tra definizioni alternative del concetto di calcolabilità.

L'equivalenza con la Turing-calcolabilità però è stata mostrata facendo uso del linguaggio dei programmi di Post-Turing.

Sarebbe auspicabile mostrare che questi simulano rigorosamente le MdT tradizionali.

Inoltre NON abbiamo ancora preso in considerazione il linguaggio delle funzioni ricorsive generali (pur avendo dimostrato la relazione intercorrente tra le funzioni ricorsive primitive, la calcolabilità in $\mathcal{S}$ ed il linguaggio LOOP).

I due problemi precedenti sono quelli che dobbiamo adesso affrontare.

MD

EQUIVALENZA TRA PROGRAMMI DI POST-TURING E MACCHINE DI TURING.

PROPOSIZIONE

Ogni funzione parziale calcolabile da un programma di POST-TURING può essere calcolata da una MdT che usa lo stesso alfabeto.

Dimostrazione

Sia P un programma di POST-TURING formato dalle k istruzioni: $I_1, \dots, I_k$ e siano $s_0, \dots, s_n$ tutti i simboli che compaiono in P.

Vogliamo adesso costruire una MdT associando stati q_i alle istruzioni I_i e scrivendo quadruple opportune.

- In corrispondenza ad istruzioni del tipo I_i: "PRINT S_k"

  mettiamo in M le quadruple

  $q_i s_j s_k q_{i+1}$ per $j = 0, \dots, n$

- In corrispondenza all'istruzione I_i: "RIGHT"

  poniamo le quadruple

  $q_i s_j R q_{i+1}$ $j = 0, \dots, n$

- In corrispondenza all'istruzione I_i: "LEFT"

  poniamo:

  $q_i s_j L q_{i+1}$ $j = 0, \dots, n$

e infine

Se I_i è IF S_k GOTO L:

$q_i s_k s_k q_m$

per $j = 0, 1, \dots, n$ ma $j \neq k$ $q_i s_j s_j q_{i+1}$

dove m è l'indice dell'istruzione etichettata L oppure k+1 se non esiste una tale istruzione

Per dimostrare la proposizione inversa è conveniente introdurre la nozione di MdT a quintuple.

UNA MdT a QUINTUPLE è un insieme di quintuple del tipo:

$$q_i\, s_j\, s_k\, R\, q_\ell$$
$$q_i\, s_j\, s_k\, L\, q_\ell$$

(cioè abbiamo compattato l'informazione di scrivere e spostarsi)

Si mostra immediatamente che:

PROPOSIZIONE

Se una funzione parziale è calcolabile da una MdT (a quadruple) allora è calcolabile da una MdT a quintuple nello stesso alfabeto.

Dimostr.

M (a quadruple) ha: stati $q_1, \dots, q_k$; alf. $\{s_1, \dots, s_u\}$

$\overline{M}$ (a quintuple) stati $q_1, \dots, q_k, q_{k+1}, \dots, q_{2k}$

Simulazione:

$$q_i\, s_j\, R\, q_\ell \longrightarrow q_i\, s_j\, s_j\, R\, q_\ell$$
$$q_i\, s_j\, L\, q_\ell \longrightarrow q_i\, s_j\, s_j\, L\, q_\ell$$

Restano da simulare le quadruple di stampa senza moto:

$$q_i\, s_j\, s_h\, q_\ell \longrightarrow q_i\, s_j\, s_h\, R\, q_{k+\ell}$$
$$q_{k+\ell}\, s_j\, s_h\, L\, q_\ell$$

Gli stati aggiuntivi servono per far tornare indietro senza danno di quel quadrato di cui ci siamo dovuti spostare.

Possiamo infine mostrare che:

PROPOSIZIONE

Ogni funzione parziale calcolata da una MdT a quintuple è calcolata da un programma di POST-TURING che usa lo stesso alfabeto.

Dimostrazione.

Sia M la nostra MdT a quintuple con stati $q_1, \dots, q_k$ e alfabeto $\{s_1, \dots, s_n\}$

Associamo delle etichette (che individuano opportune serie di istruzioni) a ciascuno stato di M ed a ciascuna coppia di simboli iniziali delle quintuple di M.

Avremo:

- $q_i \longrightarrow [A_i]$ IF S_j GOTO B_{ij} $\quad i = 1, \dots, k$ $\quad j = 1, \dots, n$
- per quegli indici i e j tali che esistono quintuple $q_i S_j S_k R q_\ell \longrightarrow$ $[B_{ij}]$ PRINT S_k RIGHT GOTO A_ℓ
- per quegli indici i e j tali che esistono quintuple $q_i S_j S_k L q_\ell \longrightarrow$ $[B_{ij}]$ PRINT S_k LEFT GOTO A_ℓ
- infine per tutti gli indici i e j tali che non esistono quintuple che iniziano i e j poniamo $[B_{ij}]$ GOTO E.

Con l'unica accortezza di porre come prima istruzione la $[A_1]$ IF S_j GOTO B_{1j} (che corrisponde al fatto che una MdT parte dallo stato q_1) un insieme di istruzioni così costruite simula il comportamento di una MdT a quintuple.

Il programma di POST-TURING che simula M con le convenzioni che abbiamo adottato prima è quindi il seguente:

→ $[A_1]$ IF S_i GOTO B_{1i}

. . . . . .

$[A_k]$ IF S_i GOTO B_{ki}

. . . . . .

$[B_{i'j'}]$ PRINT $S_{k'}$
LEFT
GOTO $A_{e'}$

. . . . . .

$[B_{i''j''}]$ PRINT $S_{k'}$
RIGHT
GOTO $A_{e''}$

. . . . . .

$[B_{i'''j'''}]$ GOTO E

} varie e variamente mescolate.

DEFINIZIONE

Si dice che una stringa $u \in A^*$, $A = \{s_1, \dots, s_n\}$, è accettata da una macchina di Turing M il cui alfabeto è A, se M partendo dalla configurazione:

$$\begin{array}{l} s_0 u \\ \uparrow \\ q_1 \end{array}$$

prima o poi si ferma.

L'insieme di tutte le stringhe $u \in A^*$ accettate da M si chiama LINGUAGGIO ACCETTATO DA M e si dice

Caratterizzazione

Proposizione.

Un linguaggio è accettato da una macchina di Turing M se e solo se è r.e.

Dimostrazione

- L sia il linguaggio accettato da M con alfabeto A. Sia g la funzione di una variabile su A^* calcolata da M.
  Allora g è parzialmente calcolabile.
  Ma $$L = \{x \in A^* \mid g(x)\downarrow\} \qquad (*)$$
  e quindi L è r.e.
- Sia adesso L r.e. Allora esisterà una funzione parzialmente calcolabile $g(x)$ tale che è verificata la $(*)$. Usando nuovamente l'equivalenza tra funzioni calcolabili e Turing-calcolabili, abbiamo che esiste una macchina di Turing M che calcola $g(x)$ e quindi M accetta L.

Corollario

Un insieme V di numeri è ricorsivamente enumerabile se e solo se esiste una macchina di Turing M con alfabeto $\{1\}$ che accetta $1^{[x]}$ se e solo se $x \in V$.

Dim.

Basta osservare che $1^{[x]}$ è la rappresentazione in base 1 del numero x.

Passiamo direttamente a mostrare alcune proprietà delle MdT.

Prima osservazione

ESISTE UNA MACCHINA DI TURING UNIVERSALE.

Sia $\Phi(x,z)$ la funzione che compare nel teorema del programma universale. Ricordiamo che $\Phi(x,z)$ è la funzione parziale di una sola variabile calcolata dal programma numero z.
Sia adesso M_V una macchina di Turing che calcola Φ.
CHIAMEREMO M_V MACCHINA DI TURING UNIVERSALE
Infatti, se $g(x)$ è una qualsiasi funzione parzialmente calcolabile di una variabile calcolata da un qualche programma numero z_0, $g(x) = \Phi(x, z_0)$ verrà calcolata da M_V a partire dalla configurazione iniziale $\underset{\uparrow q_1}{B\,x\,B\,z_0}$

IL PROBLEMA DELLA FERMATA PER LE MACCHINE DI TURING.

È ovvio che vale l'analogo del teorema della fermata, ma mostriamo il

Teorema

Esiste una macchina di Turing K tale che non esiste alcun algoritmo per determinare se K prima o poi si fermerà partendo da una configurazione data.

Dimostrazione

L'idea è di usare il corollario precedente utilizzando un opportuno insieme r.e.
Allora K abbia alfabeto $\{1\}$ e sia la Macchina di Turing corrispondente ad un insieme k ricorsivamente enumerabile ma non ricorsivo.

In base al corollario precedente

$x \in k$ se e solo se K prima o poi si fermerà a partire dalla configurazione

$$B\,1^{[x]}$$
$$\uparrow$$
$$q_1$$

Se esistesse un algoritmo per decidere il problema della fermata per questa macchina di Turing, allora saremmo in grado di decidere l'appartenenza di x a k, il che contraddice l'ipotesi che K NON È RICORSIVO.

Osservazione

Il precedente risultato dice qualcosa di più del teorema della fermata dimostrato nel contesto del linguaggio S.
Se avessimo applicato direttamente tale teorema nel caso della Turing-calcolabilità avremmo trovato che NON ESISTE ALCUN ALGORITMO PER DECIDERE IN GENERALE SE UNA GENERICA MACCHINA DI TURING SI FERMA O NO SU UN CERTO INGRESSO.

Il risultato precedente non vieta che tali algoritmi possano esistere per ogni specifica macchina di Turing.

Il teorema precedentemente dimostrato esibisce una serie di Macchine di Turing Specifiche (una per ogni insieme r.e. ma non ricorsivo) che non ammettono la risolubilità algoritmica del loro problema della fermata.

Mostriamo adesso la

Proposizione

Esiste una macchina di Turing $\tilde{K}$ con alfabeto $\{1\}$ ed uno stato q_m tale che non esiste alcun algoritmo per decidere se $\tilde{K}$ raggiungerà lo stato q_m a partire da una certa configurazione iniziale.

Dimostrazione

Sia K una macchina di Turing con alfabeto $\{1\}$ che ha un problema della fermata insolubile.
Siano $q_1, \dots, q_k$ gli stati di K.
Costruiamo una nuova macchina di Turing $\tilde{K}$ aggiungendo le seguenti quadruple a quelle di K:

$q_i B B q_{k+1}$ per ogni $i = 1, \dots, k$ tale che non esiste in K una quadrupla che inizia con $q_i B$.

$q_i 1 1 q_{k+1}$ per ogni $i = 1, \dots, k$ tale che non esiste in K una quadrupla che inizia con $q_i 1$.

Abbiamo così ricondotto il problema del raggiungimento dello stato $q_m = q_{k+1}$ per la macchina $\tilde{K}$ a quello della fermata di K, perché $\tilde{K}$ raggiunge q_{k+1} a partire da una certa configurazione iniziale se e solo se K si ferma.
Non esiste quindi nessun algoritmo per determinare se $\tilde{K}$ raggiungerà o meno q_{k+1}.

Macchine di Turing non-deterministiche.

Una macchina di Turing non deterministica è semplicemente un insieme finito di quadruple.

Cioè non imponiamo più la condizione che due quadruple non debbano cominciare con la stessa coppia di simboli.

Una conseguenza di ciò è che il calcolo di una MdT non deterministica non è rigorosamente fissato dalle regole (dalle quadruple).
Quando incontriamo due quadruple che iniziano con la stessa coppia di simboli possiamo seguire due vie.

In particolare possiamo avere due:

- seguendo una via il calcolo non termina mai mentre seguendo un'altra via possibile il calcolo termina.
- di due vie entrambe terminanti, una sarà più breve dell'altra.

Il primo caso ci induce a definire:
UNA STRINGA $w \in A^*$ È ACCETTATA DA UNA MACCHINA DI TURING NON DETERMINISTICA SE ESISTE UNA SEQUENZA DI CONFIGURAZIONI CHE PORTANO DALLA CONFIG. INIZIALE $\overset{\uparrow}{B}w$ AD UNA CONFIGUR. FINALE q_1

Il secondo caso ci suggerisce due procedimenti di calcolo non deterministici possono svolgere un ruolo nello studio di problemi di complessità.

Esempio

MdT :

	$q_1 B R q_2$	1
	$q_2 1 R q_3$	2
	$q_2 B B q_4$	3
	$q_3 1 R q_2$	4
	$q_3 B B q_3$	5
—	$q_1 B R q_4$	6
—	$q_4 B B q_5$	7

} non-determinismo

Partiamo da : $B\,1\,1\,B$ con q_1 sotto il primo B

Si ha :

$B\,1\,1\,B$ (q_1) $\xrightarrow{(1)}$ $B\,1\,1\,B$ (q_2) $\xrightarrow{(2)}$ $B\,1\,1\,B$ (q_3) $\xrightarrow{(4)}$ $B\,1\,1\,B$ (q_2) $\xrightarrow{(3)}$ $B\,1\,1\,B$ (q_4) $\rightarrow$ ...

$\xrightarrow{(6)}$ $B\,1\,1\,B\,B$ (q_4) $\xrightarrow{(6)}$ ..... $\xrightarrow{(6)}$ $B\,1\,1\,B$ ..... B (q_4) $\rightarrow$ ...

$\xrightarrow{(7)}$ $B\,1\,1\,B$ (q_5)

la macchina si ferma.

EQUIVALENZA TRA MACCHINE DI TURING CON NASTRO INFINITO SOLO IN UNA DIREZIONE O IN ENTRAMBE LE DIREZIONI.

Partiamo da una macchina di Turing M con alfabeto $A = \{s_1, \dots, s_n\}$ e stati $q_1, \dots, q_k$.

M calcoli una funzione parziale numerica g.

La configurazione di ingresso quando M inizia a calcolare $g(x)$ sarà:

$$\begin{array}{l} B\,x \\ \uparrow \\ q_1 \end{array}$$

Costruiamo adesso una macchina di Turing $\bar{M}$ che calcola g su un nastro infinito in una sola direzione.

La configurazione iniziale di $\bar{M}$ sarà

$$\begin{array}{l} \#\,B\,x \\ \quad\uparrow \\ \quad q_1 \end{array}$$

essendo $\#$ un simbolo speciale che occupa il quadrato più a sinistra del nastro.

L'alfabeto di $\bar{M}$ sarà:

$$A \cup \{\#\} \cup \{b_j^i \mid 0 \le i, j \le n\} \qquad b_j^i = \begin{pmatrix} s_i \\ s_j \end{pmatrix}$$

b_j^i sta ad indicare che s_i è il simbolo che si trova sulla traccia superiore ed s_j quello sulla traccia inferiore.

Gli stati di $\overline{M}$ sono

$$\{q_1, q_2, q_3, q_4, q_5\} \cup \{\bar{q}_i, \tilde{q}_i \mid i = 1, 2, \ldots, k\}$$

più alcuni stati addizionali.

Suddividiamo, come siamo ormai abituati a fare, le quadruple in tre gruppi.
La parte centrale, il CORPO della nuova macchina consisterà in una opportuna simulazione delle quadruple della macchina originaria M.

La parte iniziale servirà a preparare i dati iniziali di ingresso in una forma opportuna per la loro elaborazione da parte della macchina $\overline{M}$ a due tracce.

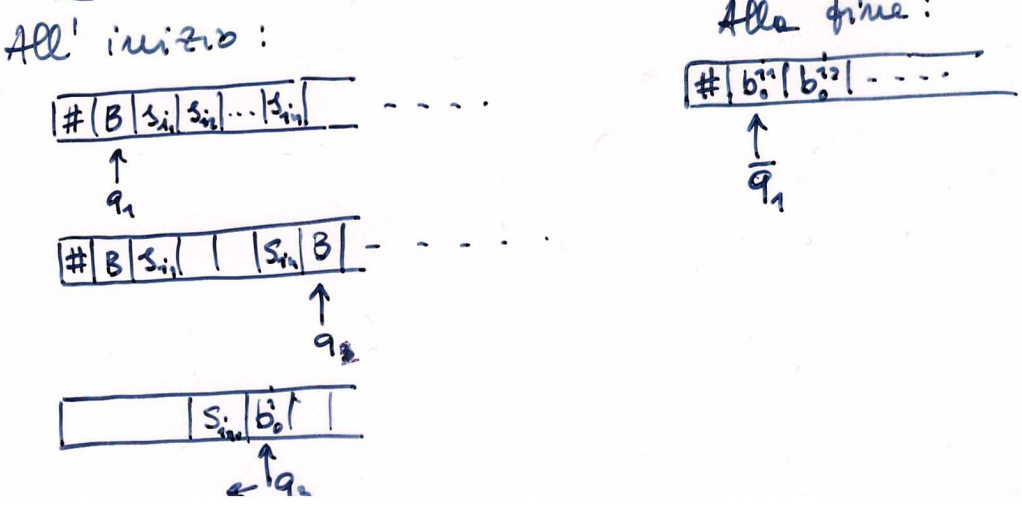
La parte finale di quadruple, infine, ha la funzione di ripresentare i valori di uscita divisi tra la traccia inferiore e quella superiore in una forma "ragionevole", cioè in una stringa dell'alfabeto di partenza.

La preparazione dei dati in modo da renderli leggibili da $\bar{M}$ consiste essenzialmente nel copiare i dati di ingresso nella traccia superiore mettendo dei Blank sui quadrati corrispondenti nella traccia inferiore.

La macchina di Turing $\bar{M}$ allora deve esaminare la stringa iniziale e sostituire ad ogni simbolo S_i il simbolo b_0^i.

Questo viene realizzato dalle quadruple:

$$q_1\ B\ R\ q_2$$
$$q_2\ S_i\ R\ q_2 \qquad i = 1, 2, \ldots, n.$$
$$q_2\ B\ L\ q_3$$
$$q_3\ S_i\ b_0^i\ q_3 \qquad i = 0, 1, \ldots, n.$$
$$q_3\ b_0^i\ L\ q_3 \qquad i = 0, 1, \ldots, n.$$
$$q_3\ \#\ R\ \bar{q}_1$$



Gruppo di quadruple che simulano direttamente le quadruple di $\mathcal{M}$.

	Quadruple of $\mathcal{M}$	Quadruple of $\bar{\mathcal{M}}$	
(a)	$q_i\ s_j\ s_k\ q_l$	$\bar{q}_i\ b_m^j\ b_m^k\ \bar{q}_l$	$m = 0, 1, \ldots, n$
		$\tilde{q}_i\ b_j^m\ b_k^m\ \tilde{q}_l$	$m = 0, 1, \ldots, n$
(b)	$q_i\ s_j\ R\ q_l$	$\bar{q}_i\ b_m^j\ R\ \bar{q}_l$	$m = 0, 1, \ldots, n$
		$\tilde{q}_i\ b_j^m\ L\ \tilde{q}_l$	$m = 0, 1, \ldots, n$
(c)	$q_i\ s_j\ L\ q_l$	$\bar{q}_i\ b_m^j\ L\ \bar{q}_l$	$m = 0, 1, \ldots, n$
		$\tilde{q}_i\ b_j^m\ R\ \tilde{q}_l$	$m = 0, 1, \ldots, n$
(d)	———	$\bar{q}_i\ B\ b_0^0\ \bar{q}_i$	$i = 1, 2, \ldots, K$
		$\tilde{q}_i\ B\ b_0^0\ \tilde{q}_i$	$i = 1, 2, \ldots, K$
(e)	———	$\bar{q}_i\ \#\ R\ \tilde{q}_i$	$i = 1, 2, \ldots, K$
		$\tilde{q}_i\ \#\ R\ \bar{q}_i$	$i = 1, 2, \ldots, K$

Sostituiscono il B con un "doppio" blank. → (d)

fanno passare da una traccia all'altra. → (e)

Osservazioni

- gli stati $\bar{q}_i$ corrispondono ad azioni sulla traccia superiore, quelli $\tilde{q}_i$ ad azioni sulla traccia inferiore.

Gruppo finale

Dopo che la macchina $\overline{M}$ si è fermata ci troveremo in una certa situazione ma, in generale, non in quella in cui l'occhio della macchina guarda il quadrato Blank più a sinistra.

La prima cosa da fare è quindi far spostare la macchina a sinistra.
Per fare ciò è sufficiente passare ad uno Stato (nuovo) che dia l'ordine di spostarsi a sinistra finché non raggiunge un blank o il termine del nastro (cioè #)
Tutto questo senza toccare niente del calcolo precedente.

Possiamo allora aggiungere delle quadruple

$$\overline{q}_i \; b^j_m \; b^j_m \; q_4$$
$$\tilde{q}_i \; b^m_j \; b^m_j \; q_4$$

in tutti i casi in cui non esistono in M quadruple che iniziano con $q_i S_j$.
(cosa che corrisponde ai casi che a noi interessano perché la macchina M si è fermata).

Le quadruple precedenti hanno il solo scopo di portare ad un nuovo stato che non è stato utilizzato prima.
Ciò fatto possiamo fare intervenire le due quadruple successive che faranno il lavoro richiesto:

$$q_4\, b_j^i\, L\, q_4$$
$$q_4\, \#\, B\, q_5$$

Sotto l'effetto di queste stringhe la configurazione del nastro sarà del tipo:

$$\begin{array}{l} B\ b_{j_1}^{i_1}\ b_{j_2}^{i_2} \cdots b_{j_k}^{i_k} \\ \uparrow \\ q_5 \end{array}$$

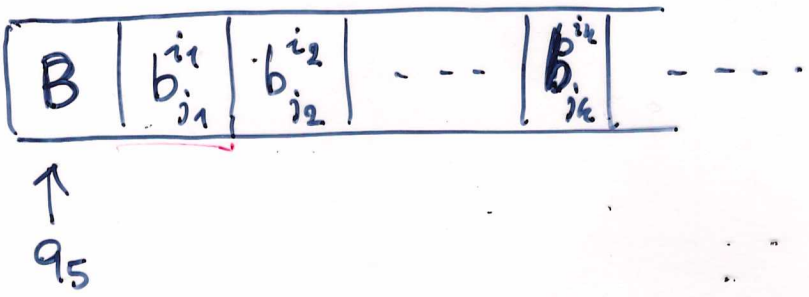
Ciò che rimane adesso da fare è di trasformare la stringa precedente nella stringa

$$S_{i_k} S_{i_{k-1}} \cdots S_{i_1} \,|\, S_{i_1} S_{i_2} \cdots S_{i_k}$$

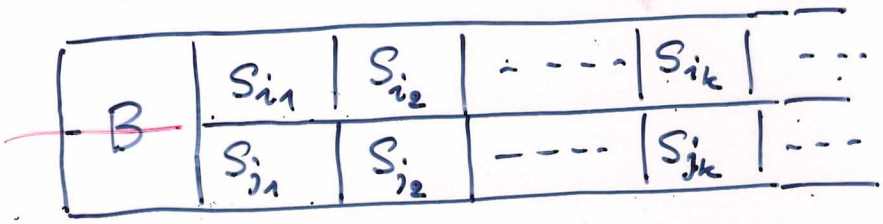
I passi elementari da compiere per ottenere questo risultato sono:

- partire dal simbolo $b_{j_1}^{i_1}$ più a sinistra, riscriverlo come $S_{j_1} S_{i_1}$ (ponendo $S_0 = \#$)
- passare al simbolo successivo $b_{j_2}^{i_2}$ e riscriverlo come $S_{j_2} S_{j_1} S_{i_1} S_{i_2}$ attorno ai due simboli di prima.

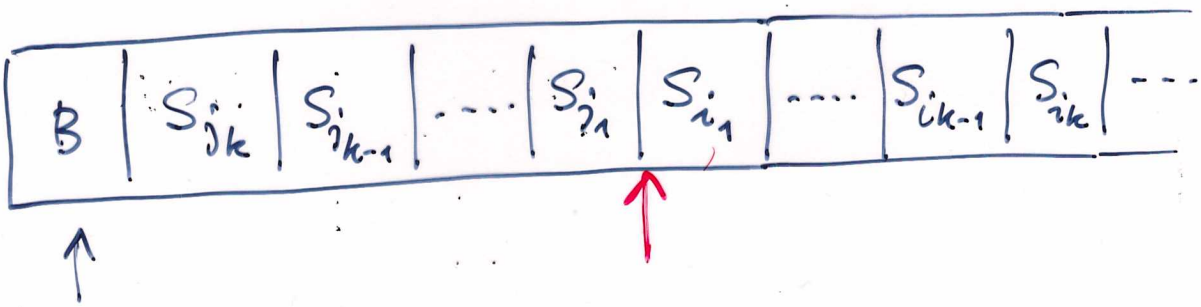
Alla fine i # verranno sostituiti da Blank.



questo lo interpretiamo come:



e vorremmo quindi poterlo leggere come:



$[D]$ RIGHT TO NEXT BLANK
MOVE BLOCK RIGHT
RIGHT

$[C]$ RIGHT
IF b_j^i GOTO A_j^i $(0 \leq i, j \leq n)$
IF B GOTO F
GOTO C

$[A_j^i]$ PRINT s_i $(0 < i \leq n, 0 \leq j \leq n)$
GOTO B_j

$[A_j^0]$ PRINT # $(0 \leq j \leq n)$
GOTO B_j

$[B_j]$ LEFT TO NEXT BLANK $(0 < j \leq n)$
PRINT s_j
GOTO D

$[B_0]$ LEFT TO NEXT BLANK
PRINT #
GOTO D

$[F]$ LEFT
IF s_j GOTO F $(0 < j \leq n)$
IF # GOTO G
IF B GOTO E

$[G]$ PRINT B
GOTO F

```
[D]     RIGHT TO NEXT BLANK
        MOVE BLOCK RIGHT
        RIGHT
[C]     RIGHT
        IF b_j^i GOTO A_j^i      (0 ≤ i, j ≤ n)
        IF B GOTO F
        GOTO C
[A_j^i] PRINT s_i                (0 < i ≤ n, 0 ≤ j ≤ n)
        GOTO B_j
[A_j^0] PRINT #                  (0 ≤ j ≤ n)
        GOTO B_j
[B_j]   LEFT TO NEXT BLANK       (0 < j ≤ n)
        PRINT s_j
        GOTO D
[B_0]   LEFT TO NEXT BLANK
        PRINT #
        GOTO D
[F]     LEFT
        IF s_j GOTO F            (0 < j ≤ n)
        IF # GOTO G
        IF B GOTO E
[G]     PRINT B
        GOTO F
```