Argomenti trattati durante la lezione:

- Relazione tra TM ed algoritmi
- Problemi decidibili e relazioni tra classi di linguaggi

Intuitive notion of algorithms	equals	*Turing machine algorithms*

Informally speaking, an algorithm is a collection of simple instructions for carrying out some task. Commonplace in everyday life, algorithms sometimes are called procedures or recipes.

Let's phrase Hilbert's tenth problem in our terminology. Doing so helps to introduce some themes that we explore in Chapters 4 and 5. Let

$$D = \{p \mid p \text{ is a polynomial with an integral root}\}.$$

Hilbert's tenth problem asks in essence whether the set D is decidable. The answer is negative. In contrast, we can show that D is Turing-recognizable. Before doing so, let's consider a simpler problem. It is an analog of Hilbert's tenth problem for polynomials that have only a single variable, such as $4x^3 - 2x^2 + x - 7$. Let

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}.$$

Here is a TM M_1 that recognizes D_1:

M_1 = "On input $\langle p \rangle$: where p is a polynomial over the variable x.

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \ldots$. If at any point the polynomial evaluates to 0, *accept*."

If p has an integral root, M_1 eventually will find it and accept. If p does not have an integral root, M_1 will run forever. For the multivariable case, we can present a similar TM M that recognizes D. Here, M goes through all possible settings of its variables to integral values.

Both M_1 and M are recognizers but not deciders. We can convert M_1 to be a decider for D_1 because we can calculate bounds within which the roots of a single variable polynomial must lie and restrict the search to these bounds. In Problem 3.21 you are asked to show that the roots of such a polynomial must lie between the values

$$\pm k \frac{c_{\max}}{c_1},$$

where k is the number of terms in the polynomial, $c_{\max}$ is the coefficient with the largest absolute value, and c_1 is the coefficient of the highest order term. If a root is not found within these bounds, the machine *rejects*. Matijasevič's theorem shows that calculating such bounds for multivariable polynomials is impossible.

Come descrivere una Turing Machine

- Descrizione formale
  - Dichiara esplicitamente tutto quanto
  - Estremamente dettagliata
  - Da evitare a tutti i costi !!!
- Descrizione implementativa
  - Descrive a parole il movimento della testina e la scrittura sul nastro
  - Nessun dettaglio sugli stati
- Descrizione di alto livello
  - Descrizione a parole dell'algoritmo
  - Nessun dettaglio implementativo
  - Da utilizzare sempre, se non indicato altrimenti

EXAMPLE 3.23

Let A be the language consisting of all strings representing undirected graphs that are connected. Recall that a graph is ***connected*** if every node can be reached from every other node by traveling along the edges of the graph. We write

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}.$$

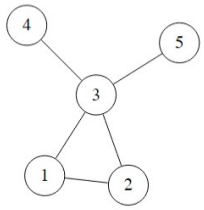
Una TM che decide A

Descrizione di alto livello:

M = "Su input $\langle G \rangle$, la codifica di un grafo G:

1. Seleziona il primo nodo di G e lo marca.
2. Ripeti la fase seguente fino a quando non vengono marcati nuovi nodi:
3. per ogni nodo in G, marcalo se è connesso con un arco ad un nodo già marcato.
4. Esamina tutti i nodi di G: se sono tutti marcati, accetta, altrimenti rifiuta."

- Codifica di G: lista dei nodi + lista degli archi

$G =$ 

$\langle G \rangle =$ `(1,2,3,4,5)((1,2),(1,3),(2,3),(3,4),(3,5))`

- M verifica che l'input sia sia una codifica di un grafo:
  - Se l'input non è nella forma corretta, rifiuta
  - Se l'input codifica un grafo, prosegue con la fase 1

Alcuni piccoli esercizi prima di passare al prossimo set di slide.

(Riferimento – File “Esercizi da 3.17 a 3.21 (Visti a lezione)”)

A3.22 Let A be the language containing only the single string s, where

$$s = \begin{cases} 0 & \text{if life never will be found on Mars.} \\ 1 & \text{if life will be found on Mars someday.} \end{cases}$$

Is A decidable? Why or why not? For the purposes of this problem, assume that the question of whether life will be found on Mars has an unambiguous YES or NO answer.

3.22 The language A is one of the two languages $\{0\}$ or $\{1\}$. In either case, the language is finite and hence decidable. If you aren't able to determine which of these two languages is A, you won't be able to describe the decider for A. However, you can give two Turing machines, one of which is A's decider.

Consideriamo ora gli algoritmi <u>decidibili</u>.

- Problema dell'accettazione: testare se un DFA accetta una stringa

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ è un DFA che accetta la stringa } w\}$$

- B accetta w se e solo se $\langle B, w\rangle$ appartiene ad A_{DFA}
- Mostrare che il linguaggio è decidibile equivale a mostrare che il problema computazionale è decidibile

Idea: definire una TM che decide A_{DFA}

M = “Su input $\langle B, w\rangle$, dove B è un DFA e w una stringa:

1. Simula B su input w
2. Se la simulazione termina in uno stato finale, accetta. Se termina in uno stato non finale, rifiuta.”

Dimostrazione:

- la codifica di B è una lista dell componenti Q, Σ, δ, q_0 e F
- fare la simulazione è facile

Questo vale per ER ed epsilon-NFA (usando ad esempio la costruzione a sottoinsiemi) = aka, esiste sempre una TM che decide il linguaggio richiesto.

Anche i seguenti sono decidibili:

$$E_{DFA} = \{\langle A \rangle \mid A \text{ è un DFA e } L(A) = \emptyset\}$$

- Puoi descrivere un algoritmo per eseguire questo test?

Dimostrazione: verifica se c'è uno stato finale che può essere raggiunto a partire dallo stato iniziale.

T = "Su input $\langle A \rangle$, la codifica di un DFA A:

1. Marca lo stato iniziale di A.
2. Ripeti la fase seguente fino a quando non vengono marcati nuovi stati:
3. marca ogni stato di A che ha una transizione proveniente da uno stato già marcato.
4. Se nessuno degli stati finali è marcato, accetta; altrimenti rifiuta."

$$EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ e } B \text{ sono DFA e } L(A) = L(B)\}$$

Idea:

- costruiamo un DFA C che accetta solo le stringhe che sono accettate da A o da B, ma non da entrambi
- se $L(A) = L(B)$ allora C non accetterà nulla
- il linguaggio di C è la differenza simmetrica di A e B

MA….

~~Teorema~~: EQ_{CFG} è decidibile

$$EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ e } H \text{ sono CFG e } L(G) = L(H)\}$$

Idea:

- Usiamo la stessa tecnica di EQ_{DFA}
- Calcoliamo la differenza simmetrica di G e H per provare l'equivalenza

STOP!!!

- Le CFG non sono chiuse per complementazione ed intersezione!
- EQ_{CFG} non è decidibile!

Pur avendo:

A_{CFG} è decidibile

- Se la CFG è in forma normale di Chomsky, allora ogni derivazione di w è lunga esattamente $(2|w| - 1)$ passi
- Le TM possono convertire le grammatiche nella forma normale di Chomsky!

Dimostrazione:

S = "Su input $\langle G, w \rangle$, dove G è una CFG e w una stringa:

1. Converti G in forma normale di Chomsky
2. Elenca tutte le derivazioni di $2|w| - 1$ passi. Se $|w| = 0$, elenca tutte le derivazioni di lunghezza 1
3. Se una delle derivazioni genera w, accetta; altrimenti rifiuta."

$$E_{CFG} = \{\langle G \rangle \mid A \text{ è una CFG ed } L(G) = \emptyset\}$$

Idea: stabilisci per ogni variabile se è in grado di generare una stringa di terminali

R = "Su input $\langle G \rangle$, la codifica di una CFG G:

1. Marca tutti i simboli terminali di G.
2. Ripeti la fase seguente fino a quando non vengono marcate nuove variabili:
3. marca ogni variabile A tale che esiste una regola $A \to U_1 \dots U_k$ dove ogni simbolo $U_1 \dots U_k$ è già stato marcato.
4. Se la variabile iniziale non è marcata, accetta; altrimenti rifiuta."

Andiamo quindi a vari esercizi:

1. Sia $INFINITE_{PDA} = \{\langle M \rangle \mid A \text{ è un PDA ed } L(M) \text{ è un linguaggio infinito. Si mostri che } INFINITE_{PDA} \text{ è decidibile.}$

   Soluzione

   Costruiamo una Turing Machine TM in grado di decidere il problema. Dato il PDA M, possiamo convertirlo in una CFG equivalente G' in forma normale di Chomsky e controllare se G' possiede una derivazione $A \to ua$ dove $u, v \in \Sigma$.

   Se $A \to uAv$ è una derivazione in G', allora $L(M)$ è un linguaggio infinito, altrimenti L(M) è finito. La seguente TM M decide $INFINITE_{PDA}$:

   M = "Su input $\langle M \rangle$, dove M è un PDA:

   (a) Converte M in una grammatica CFG G

   (b) Converte G in forma normale di Chomsky, diventando G'

   (c) Se G' include la derivazione $A \to uAv$ accetta, altrimenti rifiuta.

3. Uno *stato inutile* in un automa a pila non viene mai inserito in nessuna stringa di input. Consideriamo il problema di determinare se un automa pushdown ha degli stati inutili. Formulate questo problema come un linguaggio e dimostrate che è decidibile.

   Soluzione

   Sia $U_{PDA} = \{\text{P} \mid P \text{ il PDA che ha gli stati inutili}\}$. Creiamo una TM T che decide U_{PDA}. Per prima cosa, definiamo $E_{PDA} = \{\text{P} \mid P \text{ è un PDA } L(P) = \emptyset\}$. Mostriamo ora come E_{PDA} sia decidibile con un decisore D:
   $D =$ "Su input P dove P è un PDA:

   1. Convertire P in una CFG equivalente G.
   2. Eseguire E_{CFG} su input G. Output quello che E_{CFG} dà in uscita."

   Questo linguaggio è decidibile perché tutti i passi della sua costruzione richiedono un tempo finito e E_{CFG} è un decisore.
   $T =$ "Su input P dove $P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ è un PDA:

   1. Per tutti $\forall q \in Q$:
      a. Modificare P affinché $F = \{q\}$.
      b. Eseguire E_{PDA} su input P. If E_{PDA} accetta, *accept* P.
   2. $Rifiuta$ P."

   Dato che E_{PDA} è decidibile, U_{PDA} è decidibile.

4. Sia $S = \{\langle M \rangle \mid M \text{ è un DFA che accetta } w^R \text{ in qualsiasi caso accetti } w\}$. Si mostri che S è decidibile.

   Soluzione

   Per dimostrare che S è decidibile, costruiamo un decisore D per S come segue (sia C una TM che decide l'EQ_{DFA}):

   $D =$ Su input $\langle M \rangle$:

   (a) Costruisci un NFA M' tale che $\langle M' \rangle = \{w^R \mid w \in L(M)\}$
   (b) Convertire M' in un DFA M'' equivalente
   (c) Usa C per confrontare $L(M'')$ ed $L(M)$
   (d) Se $L(M'') = L(M)$, accetta, altrimenti rifiuta

   Nella TM di cui sopra, il passo 1 può essere eseguito convertendo M in M' in passi finiti. L'idea è di (i) invertire le direzioni di tutte le frecce di (ii) creare un nuovo stato q' in M' e collegare q' a ogni stato finale originale di M con ϵ-transizioni, e (iii) rendere lo stato iniziale originale di M uno stato finale di M'. È facile verificare che $\langle M' \rangle = \{w^R \mid w \in L(M)\}$ Inoltre, sia il passo B che il passo C possono essere eseguiti in passi finiti. Quindi, D si svolge in passi finiti è quindi un decisore.

A*4.23 Say that an NFA is ***ambiguous*** if it accepts some string along two different computation branches. Let $AMBIG_{\text{NFA}} = \{\langle N \rangle | \ N \text{ is an ambiguous NFA}\}$. Show that $AMBIG_{\text{NFA}}$ is decidable. (Suggestion: One elegant way to solve this problem is to construct a suitable DFA and then run E_{DFA} on it.)

4.23 The following procedure decides $AMBIG_{\text{NFA}}$. Given an NFA N, we design a DFA D that simulates N and accepts a string iff it is accepted by N along two different computational branches. Then we use a decider for E_{DFA} to determine whether D accepts any strings.

Our strategy for constructing D is similar to the NFA-to-DFA conversion in the proof of Theorem 1.39. We simulate N by keeping a pebble on each active state. We begin by putting a red pebble on the start state and on each state reachable from the start state along ε transitions. We move, add, and remove pebbles in accordance with N's transitions, preserving the color of the pebbles. Whenever two or more pebbles are moved to the same state, we replace its pebbles with a blue pebble. After reading the input, we accept if a blue pebble is on an accept state of N or if two different accept states of N have red pebbles on them.

The DFA D has a state corresponding to each possible position of pebbles. For each state of N, three possibilities occur: It can contain a red pebble, a blue pebble, or no pebble. Thus, if N has n states, D will have 3^n states. Its start state, accept states, and transition function are defined to carry out the simulation.

A*4.25 Let $BAL_{\text{DFA}} = \{\langle M \rangle | \ M \text{ is a DFA that accepts some string containing an equal number of 0s and 1s}\}$. Show that BAL_{DFA} is decidable. (Hint: Theorems about CFLs are helpful here.)

4.25 The language of all strings with an equal number of 0s and 1s is a context-free language, generated by the grammar $S \to 1S0S \mid 0S1S \mid \varepsilon$. Let P be the PDA that recognizes this language. Build a TM M for BAL_{DFA}, which operates as follows. On input $\langle B \rangle$, where B is a DFA, use B and P to construct a new PDA R that recognizes the intersection of the languages of B and P. Then test whether R's language is empty. If its language is empty, *reject*; otherwise, *accept*.

Alcune delle prossime soluzioni: https://imada.sdu.dk/u/hammer/compluxability.html

Variable A in CFG G usable if it appears in derivation of some string $w \in L(G)$. Problem: given G and A, check if A is usable. Formulate as language, show it is decidable.

Language formulation omitted here. To decide:

1. Variable must be reachable
    1. S is reachable
    2. If A is reachable and $A \to s_1 B s_2$, then B is reachable
2. Variable must derive something terminal (it must be generating)
    1. All variables deriving terminals are generating
    2. If $A \to BCD$ and BCD are all generating, A is generating

Both the sets (reachable and generating variables) can be computed by iterating over grammar a finite number of times. A must be in intersection.

4.13 `cfg decide`

$$C_{\text{CFG}} = \{\langle G, k\rangle \mid G \text{ is a CFG and } |L(G)| = k, k \geq 0 \vee k = \infty\}$$

Show: it is decidable.

First: convert to Chomsky normal form.

In case we are to check if infinite language: then there will, for long enough strings, be some repetition. This comes from cyclic behavior in expansions. Check if any variable can derive itself (potentially via some intermediaries)—this can be done using a finite number of cycle-detection schemes.

In case it is some finite k, we can first make sure the language is actually finite. In addition, the longest string we can achieve without has bounded length (refer to pumping lemma for rough upper bound). As such length is finite, we can enumerate and test all such strings (thm. 4.7 says that A_{CFG} is decidable). Can potentially lower the number of derivations to try, but this is not important.

Backup – esercizi TM.

1. **(12 punti)** Una *R2-L3 Turing Machine* è una macchina di Turing deterministica a nastro semi-infinito che può effettuare solo due mosse: spostarsi a destra di due celle (R2), oppure spostarsi a sinistra di tre celle (L3). Se in uno spostamento a sinistra la macchina tenta di spostare la testina a sinistra dell'inizio del nastro, allora lo spostamento termina con la testina nella prima cella del nastro.

    (a) Dai una definizione formale della funzione di transizione di una R2-L3 Turing Machine.

    (b) Dimostra che le R2-L3 Turing Machine riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

(b) Per dimostrare che le R2-L3 Turing Machine riconoscono la classe dei linguaggi Turing-riconoscibili dobbiamo dimostrare due cose: che ogni linguaggio Turing-riconoscibile è riconosciuto da una R2-L3 Turing Machine, e che ogni linguaggio riconosciuto da una R2-L3 Turing Machine è Turing-riconoscibile.

Per la prima dimostrazione, mostriamo come convertire una macchina di Turing deterministica a nastro semi-infinito M in una R2-L3 Turing Machine S equivalente.

S = "Su input w:

1. Per simulare una mossa del tipo $\delta(q, a) = (r, b, L)$, S scrive b sul nastro e muove la testina di due celle a destra, e subito dopo di tre celle a sinistra. Se lo spostamento a sinistra porta la testina oltre l'inizio del nastro, allora vuol dire che la simulazione era partita dalla prima cella del nastro. In questo caso la simulazione riprende con la testina nella prima cella del nastro, come per le TM standard. Negli altri casi, la simulazione continua con la testina in corrispondenza della cella immediatamente a sinistra di quella di partenza.
2. Per simulare una mossa del tipo $\delta(q, a) = (r, b, R)$, S scrive b sul nastro e muove la testina di due celle a destra, di nuovo di due celle a destra, e subito dopo di tre celle a sinistra. La simulazione continua con la testina in corrispondenza della cella immediatamente a destra di quella di partenza.
3. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M, allora S termina con accettazione. Se in qualsiasi momento la simulazione raggiunge lo stato di rifiuto di M, allora S termina con rifiuto. Negli altri casi continua la simulazione dal punto 2."

Per dimostrare che ogni linguaggio riconosciuto da una R2-L3 Turing Machine è Turing-riconoscibile, mostriamo come convertire una R2-L3 Turing Machine S in una TM deterministica a nastro semi-infinito M equivalente.

M = "Su input w:

1. Per simulare una mossa del tipo $\delta(q, a) = (r, b, L3)$, M scrive b sul nastro e muove la testina di tre celle a sinistra. Se lo spostamento a sinistra porta la testina oltre l'inizio del nastro, allora lo sposta meno a sinistra si ferma con la testina nella prima cella del nastro, come per le R2-L3 Turing Machine.
2. Per simulare una mossa del tipo $\delta(q, a) = (r, b, R2)$, M scrive b sul nastro e muove la testina di due celle a destra.
3. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di S, allora M termina con accettazione. Se in qualsiasi momento la simulazione raggiunge lo stato di rifiuto di S, allora M termina con rifiuto. Negli altri casi continua la simulazione dal punto 2."

1. (12 punti) Una macchina di Turing spreca-nastro è simile a una normale macchina di Turing deterministica a nastro singolo semi-infinito, ma può spostare la testina nella parte non ancora utilizzata del nastro. In particolare, se tutte le celle dopo la cella numero s del nastro sono vuote, e la cella s è non vuota, allora la testina può spostarsi nella cella numero $2s$. A ogni passo, la testina della TM spreca-nastro può spostarsi a sinistra di una cella (L), a destra di una cella (R) o dopo la parte non vuota del nastro (J).

(a) Dai una definizione formale della funzione di transizione di una TM spreca-nastro.

(b) Dimostra che le TM spreca-nastro riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

Soluzione.

(a) $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, J\}$

(b) Per dimostrare che TM spreca-nastro riconoscono la classe dei linguaggi Turing-riconoscibili dobbiamo dimostrare due cose: che ogni linguaggio Turing-riconoscibile è riconosciuto da una TM spreca-nastro, e che ogni linguaggio riconosciuto da una TM spreca-nastro è Turing-riconoscibile.

La prima dimostrazione è banale: le TM deterministiche a singolo nastro sono un caso particolare di TM spreca-nastro che non effettuano mai la mossa J per saltare oltre la parte non vuota del nastro. Di conseguenza, ogni linguaggio Turing-riconoscibile è riconosciuto da una TM spreca-nastro.

Per dimostrare che ogni linguaggio riconosciuto da una TM spreca-nastro è Turing-riconoscibile, mostriamo come convertire una macchina di Turing spreca-nastro M in una TM deterministica a nastro singolo S equivalente.

1. Una *tag-Turing machine* è una macchina di Turing con un singolo nastro e due testine: una testina può solo leggere, l'altra può solo scrivere. All'inizio della computazione la testina di lettura si trova sopra il primo simbolo dell'input e la testina di scrittura si trova sopra la cella vuota posta immediatamente dopo la stringa di input. Ad ogni transizione, la testina di lettura può spostarsi di una cella a destra o rimanere ferma, mentre la testina di scrittura deve scrivere un simbolo nella cella corrente e spostarsi di una cella a destra. Nessuna delle due testine può spostarsi a sinistra.

Dimostra che che le tag-Turing machine riconoscono la classe dei linguaggi Turing-riconoscibili.

1. Per risolvere l'esercizio dobbiamo dimostrare che *(a)* ogni linguaggio riconosciuto da una tag-Turing machine è Turing-riconoscibile e *(b)* ogni linguaggio Turing-riconoscibile è riconosciuto da una tag-Turing machine.

(a) Mostriamo come convertire una tag-Turing machine M in una TM deterministica a nastro singolo S equivalente. S simula il comportamento di M tenendo traccia delle posizioni delle due testine marcando la cella dove si trova la testina di lettura con un pallino sopra il simbolo, e marcando la cella la cella dove si trova la testina di scrittura con un pallino sotto il simbolo. Per simulare il comportamento di M la TM S scorre il nastro e aggiorna le posizioni delle testine ed il contenuto delle celle come indicato dalla funzione di transizione di M.

S = "Su input $w = w_1 w_2 \dots w_n$:

1. Scrivi un pallino sopra il primo simbolo di w e un pallino sotto la prima cella vuota dopo l'input, in modo che il nastro contenga

$$\dot{w}_1 w_2 \dots w_n \dot{\sqcup}$$

2. Per simulare una transizione, S scorre il nastro per trovare la posizione della testina di lettura e determinare il simbolo letto da M. Se la funzione di transizione stabilisce che la testina di lettura deve spostarsi a destra, allora S sposta il pallino nella cella immediatamente a destra, altrimenti lo lascia dov'è. Successivamente S si sposta verso destra finché non trova la cella dove si trova la testina di scrittura, scrive il simbolo stabilito dalla funzione di transizione nella cella e sposta la marcatura nella cella immediatamente a destra.
3. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

(b) Mostriamo come convertire una TM deterministica a nastro singolo S in una tag-Turing machine M equivalente. M simula il comportamento di S memorizzando sul nastro una sequenza di configurazioni di S separate dal simbolo #. All'interno di ogni configurazione un pallino marca il simbolo sotto la testina di S. Per simulare il comportamento di S la tag-Turing machine M scorre la configurazione corrente e scrivendo man mano la prossima configurazione sul nastro.

M = "Su input $w = w_1 w_2 \dots w_n$:

1. Scrive il simbolo # subito dopo l'input, seguito dalla configurazione iniziale, in modo che il nastro contenga

$$w_1 w_2 \dots w_n \# \dot{w}_1 w_2 \dots w_n \dot{\sqcup},$$

che la testina di lettura si trovi in corrispondenza del $\dot{w}_1$ e quella di lettura in corrispondenza del blank dopo la configurazione iniziale. Imposta lo stato corrente della simulazione st allo stato iniziale di S e memorizza l'ultimo simbolo letto $prec = \#$. L'informazione sui valori di st e $prec$ sono codificate all'interno degli stato di M.
2. Finché il simbolo sotto la testina di lettura non è marcato, scrive il simbolo precedente $prec$ e muove a destra. Aggiorna il valore di $prec$ con il simbolo letto.
3. Quando si trova un simbolo marcato $\dot{a}$ e $\delta(st, a) = (q, b, R)$:
   - aggiorna lo stato della simulazione $st = q$;
   - scrive $prec$ seguito da b, poi muove la testina di lettura a destra;
   - scrive il simbolo sotto la testina marcandolo con un pallino.
4. Quando si trova un simbolo marcato $\dot{a}$ e $\delta(st, a) = (q, b, L)$:
   - aggiorna lo stato della simulazione $st = q$;
   - scrive $\dot{prec}$; se $prec = \#$ scrive $\#\dot{\sqcup}$;
   - scrive b.
5. Copia il resto della configurazione fino al # escluso. Al termine della copia la testina di lettura di trova in corrispondenza della prima cella nella configurazione corrente, e quella di lettura sulla cella vuota dopo la configurazione.
6. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di S, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."