1. **(12 punti)** Una macchina di Turing con "save e restore" (SRTM) è una macchina di Turing deterministica a singolo nastro, che può salvare la configurazione corrente per poi ripristinarla in un momento successivo. Oltre alle normali operazioni di spostamento a sinistra e a destra, una SRTM può effettuare l'operazione di SAVE, che salva la configurazione corrente, e l'operazione di RESTORE che ripristina la configurazione salvata. L'operazione di SAVE sovrascrive una eventuale configurazione salvata in precedenza. Fare il RESTORE in assenza di configurazione salvata non ha effetto: si mantiene inalterata la configurazione corrente.

   (a) Dai una definizione formale della funzione di transizione di una SRTM.

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

(a) $\delta : Q \times \Gamma \longmapsto Q \times \Gamma \times \{L, R, SAVE, RESTORE\}$

$\underbrace{\sqcup\sqcup\sqcup\sqcup WW}$ SPAZIO DEDICATO SALVATAGGIO

SAVE

RESTORE

OFFSET / FLAG (-1) → IDEA OTTIMA

DUE NASTRI → SAVE (1) / RESTORE (2)

(b) – $\delta(q, a) = (r, b, L)$

LA MACCHINA SI SPOSTA DALLO STATO "q" ALLO STATO "r" SCRIVENDO "b" SUL NASTRO E POI ANDANDO VERSO L

– $\delta(q, a) = (r, b, R)$

SAME AS L

[PIÙ NASTRI = MULTINASTRO]

– $\delta(q, a) = (r, b, SAVE)$

$T_1 = [\ ]$ → L

HEAD

← R

→ DELIMITO LA PORZIONE SALVATA DA SIMBOLI SUL NASTRO DI LAVORO T_1 E SU T_2, INSERISCO TUTTI I ⊔ FORNENDO SPAZIO PER PERMETTERE ALLA COMPUTAZIONE DI ESSERE SEMPRE RIPRISTINATA

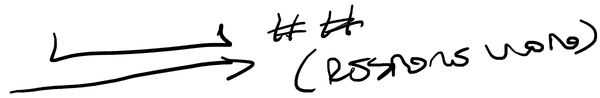
$T_2 = \underline{\qquad}$ RESTORE $\cup \{\sqcup, \sqcup, \sqcup\}$ SPAZI VUOTI MEMORIA

T_1 → COPIO L'INPUT SUL NASTRO SEGUENDO L, R
E, OGNI VOLTA CHE TROVO UNA COMPUT.,
INSERISCO NELL'INPUT PIÙ A DX UN SIMBOLO

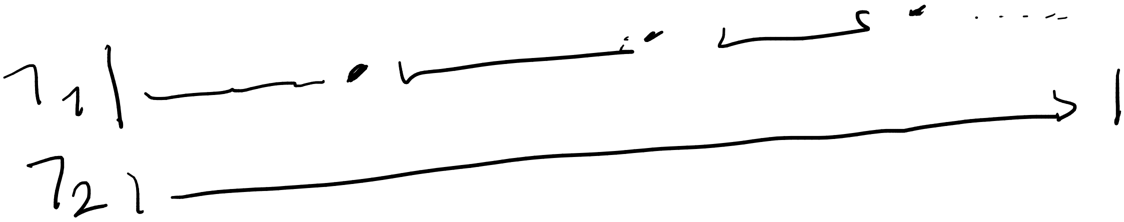
T_2 → COPIA L'INPUT SUL NASTRO 2 FINO A DOVE STA •

$\delta(q, a) = (k, b, \text{RESTORE})$

→ IN CASO DI CONFIG. NUOVA ⟹ METTIAMO UN ALTRO #



→ DA NASTRO T_2, COPIA SU T_1



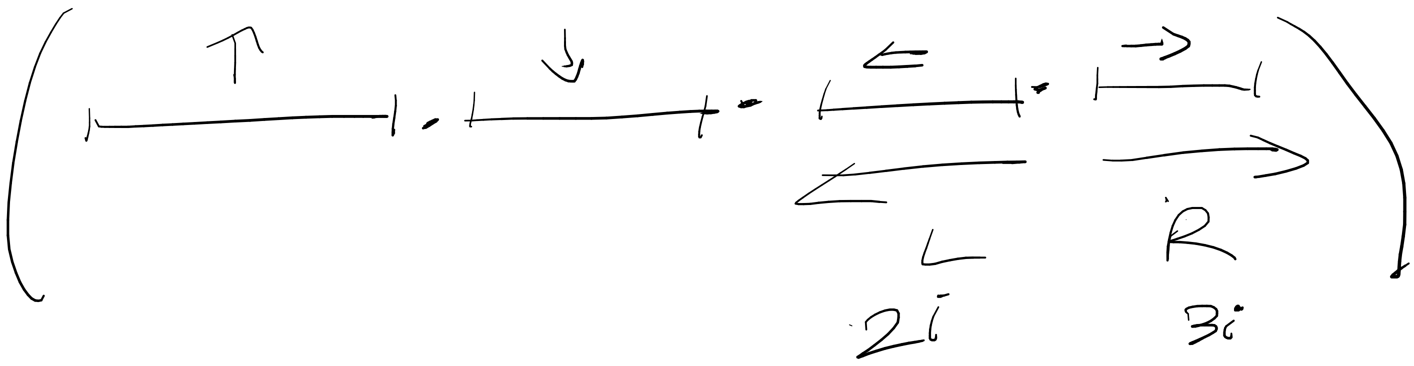
↳ SR_{TM} è TURING-RICONOSCIBILE

1. *Una* macchina di Turing bidimensionale *utilizza una griglia bidimensionale infinita di celle come nastro. Ad ogni transizione, la testina può spostarsi dalla cella corrente ad una qualsiasi delle quattro celle adiacenti. La funzione di transizione di tale macchina ha la forma*

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\uparrow, \downarrow, \rightarrow, \leftarrow\},$$

dove le frecce indicano in quale direzione si muove la testina dopo aver scritto il simbolo sulla cella corrente.

Dimostra che ogni macchina di Turing bidimensionale può essere simulata da una macchina di Turing deterministica a nastro singolo.



$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$

S → SU INPUT w

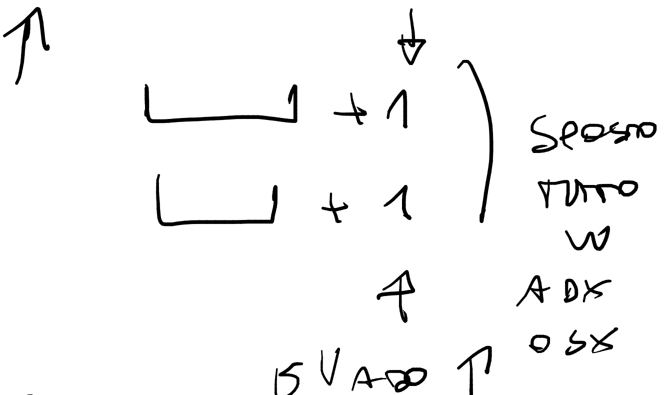
$- \delta(q, a) = (r, b, \uparrow)$

\#\# w \#\#

START

SCRIVO B SUL NASTRO →

w ←



SCRIVO w NON MARCATO, POI SCRIVO $\overline{w}$, SPOSTA TUTTI GLI INPUT A DX → SE ALLA FINE DELL'INPUT, METTO \#\#

→ \#\# $w\overline{w}$ \#\#

$- \delta(q, a) = (r, b, \downarrow)$

SCRIVO w NON MARCATO PARTENDO DA \#\# w \#\#
SCRIVO $\overline{w}$, SPOSTA TUTTI GLI INPUT A SX

→ SE ALLA FINE DELL'INPUT → \#\# $w\overline{w}$ \#\#

$- \delta(q, a) = (r, b, \leftarrow)$ (SAME ON "→")

SCRIVO w NON MARCATO PARTENDO DA \#\# w \#\#
SCRIVO • PER INDICARE LO SPOSTAMENTO A SX POSIZIONE → \#\# w • \#\# (SPOSTO TUTTO GLI INPUT E RITORNO)

(x, y)

$\downarrow$

$(\perp, \bullet)$

$(x-1, y)$

$(x-1, y-1)$

$(x, y-1)$

(x, y)

ALLA COND. BASE + INPUT DI SX

$(\uparrow, \downarrow, \leftarrow, \rightarrow)$

1. (12 punti) Data una stringa $w \in \Sigma^*$, definiamo una operazione che scambia di posizione i caratteri della stringa a due a due:

$$\mathrm{SWAP}(w) = \begin{cases} \varepsilon & \text{se } w = \varepsilon \\ a & \text{se } w = a \text{ con } a \in \Sigma \\ a_1 a_0 \mathrm{SWAP}(u) & \text{se } w = a_0 a_1 u \text{ con } a_0, a_1 \in \Sigma, u \in \Sigma^* \end{cases}$$

Per esempio, SWAP(ABCDE) = BADCE.

Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio regolare, allora anche il seguente linguaggio è regolare:

$$\mathrm{SWAP}(L) = \{\mathrm{SWAP}(w) \mid w \in L\}.$$

SE L È REGOLARE, ∃! DFA D RICONOSCITORE.

ORA

-> CASO BASE $\delta(q_0, \varepsilon) = \varepsilon$ / $\delta(q_0, a) = a \in \Sigma$

-> CASO INDUTTIVO:

$\forall q \in Q, \forall a \in \Sigma, \delta((q, a), b)) = ((q, b), a)$

$a, b \in F$

2. Lettura primo carattere di ogni coppia:
∀q ∈ Q, ∀a ∈ Σ:
δ'(q, a) = {(q, a)} ∪ (F se δ(q,a) ∈ F)

3. Lettura secondo carattere e scambio:
∀q ∈ Q, ∀a,b ∈ Σ:
δ'((q, a), b) = {δ(δ(q, a), b)}

-> EQUIVALENTE

QUINDI

→ ① $\delta(q_0, \varepsilon) = \underline{q_F}$ ↓ ε → $\mathrm{SWAP}(\varepsilon) = \varepsilon$

GOOD

→ ② $\delta(q_0, a) = \underline{q_F}$ ↓ a ⇒ $\mathrm{SWAP}(a) = a$

GOOD

→ ③ $\delta((a_0 a_0), a_1) = \delta((q_F, a_1), a_0)) \to \underline{\delta(\delta(q_i)}$

≥ 2

$\underline{a_0 a_1} \to \underline{a_1 a_0}$ → $\underline{\mathrm{SWAP}(a_0 a_1) = \mathrm{SWAP}(a_1 = a_0)}$

SWAP

$A \cdot B$ / --- → GOOD

```
M' accetta w iff M accetta a₁a₀SWAP(u)
iff a₀a₁u ∈ L
iff SWAP(a₀a₁u) = a₁a₀SWAP(u) ∈ SWAP(L) ✓
```

$\in \mathrm{SWAP}(L)$

$\left[w \in L \iff \mathrm{SWAP}(L) \text{ è REGOLARE} \right]$

(⇒) se $w \in L$, $\exists\, w \in L$ //

① $w = \varepsilon \to \mathrm{SWAP}(\varepsilon) = \varepsilon$

② $w = a \to \delta(q_0, a) \Rightarrow ((q_F, a), a) \Rightarrow \mathrm{SWAP}(a) = a$

③ $w = a_1 a_2 u$

∃ COMPUT. ACCETTANTE TALE CHE

$$\left\{ \begin{array}{l} s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \cdots \xrightarrow{w_n} s_n \\ \underbrace{s_0 \xrightarrow{w_2} s_1 \xrightarrow{w_1} s_2 \xrightarrow{w_3}}_{\text{ES. DI SWAP}} \cdots \xrightarrow{w_n} s_n \end{array} \right\}$$

SE ACCETTA LO SWAP DI W → W ∈ SWAP(L)

↓

REGOLARE

Poiché w ∈ L, esiste una computazione accettante di A su w:

```
q₀ →^a₀ s₁ →^a₁ s₂ →^u sₙ ∈ F
```

Invertendo lo scambio: A ha anche una computazione:

```
q₀ →^a₁ s₁' →^a₀ s₂' →^SWAP(u) sₙ' ∈ F
```

dove gli stati s2', s_n' sono determinati dalla struttura di SWAP.

3. (12 punti) Una CFG è detta *lineare a destra* se il corpo di ogni regola ha al massimo una variabile, e la variabile si trova all'estremità di destra. In altre parole, tutte le regole di una grammatica lineare a destra sono nella forma $A \to wB$ o $A \to w$, dove A e B sono variabili e w è una stringa di zero o più simboli terminali.

Dimostra che ogni grammatica lineare a destra genera un linguaggio regolare. *Suggerimento:* costruisci un ε-NFA che simula le derivazioni della grammatica.

NFA → $(Q, \Sigma, q_0, \delta, F)$

CFG → (V, R, Σ, S)

$$\begin{bmatrix} A \to wB \\ A \to w \end{bmatrix} \equiv \text{FNC} \begin{bmatrix} A \to BC \\ A \to a \end{bmatrix}$$ EQUIVALENTE ...

PUSH

POP

NFA ≃ PDA = CFG

L'automa A simula le derivazioni di G come segue:

1. **Inizializzazione:** Parte dallo stato S (variabile iniziale)
2. **Ciclo di derivazione:** Ripete finché possibile:
   - **Stato corrente = variabile A:** Sceglie non deterministicamente una regola per A
   - **Regola A → wB:** Consuma w dall'input e va nello stato B
   - **Regola A → w:** Consuma w dall'input e va in qf (accettazione)
3. **Accettazione:** Se riesce a consumare tutto l'input e raggiungere qf

$\exists$ ε-NFA N TALE CHE SIMULI $\forall r \in R$ CFG G (CNF)

2n − 1

SIMULA OGNI STATO SEGUENDO

$$\begin{bmatrix} \delta(q_0, a) = (q_1, b) \\ \delta(q_1, a_1) = (q_1, b_1) \\ \downarrow \\ \dots \\ \delta(q_n, a_n) = \{q_F\} \end{bmatrix} \rightarrow$$

OSS. BROSOLIN

δ = TRANS. STESSE

$q_F \cup \underline{\{S\}}$

↑ INIZIO

$w \in L \iff$ RIGHT-LINEAR$_{CFG}$ È REGOLARE

(⇒) $w \in L \rightarrow \exists$ COMP. ACCETTANTE A S.T.

[$q_0 \cdot q_1 \dots q_m$] STATI NFA

[$S \rightarrow S_1 B \rightarrow S_2 B \rightarrow \dots$] CFG RL

$\underline{\forall \text{ STATO } q_i \in \{1 \dots n\} \mid \forall r \in R \mid r_i \in \{1 \dots m\}}$

RAMO TUTTE

$\forall r,$

$A \rightarrow w \dots \underline{q_F}$

$\underline{q_{ACCEPT}}$

TERMINA

$\underline{q_{REJECT}}$

LOOP

RIFIUTA

2. Considera il linguaggio

$$L_2 = \{w \in \{1, \#\}^* \mid w = x_1\#x_2\#\dots\#x_k \text{ con } k \geq 0, \text{ ciascun } x_i \in 1^* \text{ e } x_i \neq x_j \text{ per ogni } i \neq j\}.$$

Dimostra che L_2 non è regolare.

$W = x_1 \# x_2 \dots \# X_k$ → FINISCE DOPO i #

PAROLA SCELTA

$\rightarrow W = X^{P+1} \#^{P}$

ASSUMO L_2 REGOLARE

$[W = x y^i z \;,\; i \geq 0 \;,\; y \neq \varepsilon]$ $\exists P, q > 0 \mid |xy| < K$

$x = X^P \#^P$

$y = X \#^Q$

$z = X^{K-P+1-Q} \#^{K-0-Q}$

$P+1 > 0 \mid P+1 < K$ → FUNZIONA!

ALTERNATIVA

$W = 1\#11\#111\#\dots 1^P$

→ PONGO IN MODO CRESCENTE

→ $\forall x_i \neq x_j$ → COPPIE DI CARATTERI DIVERSE...

Usiamo il Pumping Lemma per dimostrare che il linguaggio non è regolare.

Supponiamo per assurdo che L_2 sia regolare:

- sia k la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 1^k\#1^{k-1}\#\dots\#1\#\#$, che appartiene ad L_2 ed è di lunghezza maggiore di k;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq k$;
- poiché $|xy| \leq k$, allora x e y sono entrambe contenute nella prima sequenza di 1. Inoltre, siccome $y \neq \emptyset$, abbiamo che $x = 1^q$ e $y = 1^p$ per qualche $q \geq 0$ e $p > 0$. z contiene la parte rimanente della stringa: $z = 1^{k-q-p}\#1^{k-1}\#\dots\#1\#\#$. Consideriamo l'esponente $i = 0$: la parola xy^0z ha la forma
$$xy^0z = xz = 1^q1^{k-q-p}\#1^{k-1}\#\dots\#1\# = 1^{k-p}\#1^{k-1}\#\dots\#1\#\#$$
Siccome la parola z contiene tutte le sequenze di 1 di lunghezza decrescente da $k-1$ a 0, allora una delle sequenze sarà uguale alla sequenza 1^{k-p}, che è di lunghezza strettamente minore di k perché $p > 0$. Di conseguenza, la parola xy^0z non appartiene al linguaggio L_2, in contraddizione con l'enunciato del Pumping Lemma.

Abbiamo trovato un assurdo quindi L_2 non può essere regolare.