

## AUTOMI E LINGUAGGI FORMALI

### SOLUZIONE DELLA PRIMA PROVA INTERMEDIA

1. Per dimostrare che gli all- $\epsilon$ -NFA riconoscono esattamente la classe dei linguaggi regolari occorre procedere in due versi: dimostrare che ogni linguaggio regolare è riconosciuto da un all- $\epsilon$ -NFA, e che ogni linguaggio riconosciuto da un all- $\epsilon$ -NFA è regolare.

- Per dimostrare che ogni linguaggio regolare è riconosciuto da un all- $\epsilon$ -NFA si parte dal fatto che ogni linguaggio regolare ha un DFA che lo riconosce. È facile vedere che ogni DFA è anche un all- $\epsilon$ -NFA, che non ha  $\epsilon$ -transizioni e dove  $\delta(q, a) = \{q'\}$  per ogni stato  $q \in Q$  e simbolo dell'alfabeto  $a \in \Sigma$ . In un DFA c'è una sola computazione possibile, quindi lo stato in cui si trova l'automa dopo aver consumato l'input è unicamente determinato: se questo stato è finale il DFA accetta, altrimenti rifiuta. Questo è coerente con la condizione di accettazione degli all- $\epsilon$ -NFA quando c'è un solo possibile stato dove si può trovare l'automa dopo aver consumato l'input.
- Per dimostrare che ogni linguaggio riconosciuto da un all- $\epsilon$ -NFA è regolare, mostriamo come possiamo trasformare un all- $\epsilon$ -NFA in un DFA equivalente. La trasformazione è descritta dal seguente algoritmo:

```

Require: Un all- $\epsilon$ -NFA  $N = (Q_N, \Sigma, q_0, \delta_N, F_N)$ 
Ensure: Un DFA  $D = (Q_D, \Sigma, S_0, \delta_D, F_D)$  equivalente a  $N$ 

 $S_0 \leftarrow \text{ECLOSE}(q_0)$                                  $\triangleright$  Lo stato iniziale è la chiusura di  $q_0$ 
 $Q_D \leftarrow \{S_0\}$                                       $\triangleright$   $Q_D$  sarà l'insieme degli stati del DFA
 $\text{if } S_0 \subseteq F_N \text{ then}$                        $\triangleright$  Se  $S_0$  contiene solamente stati finali dell'all- $\epsilon$ -NFA ...
     $F_D \leftarrow \{S_0\}$                                   $\triangleright \dots$  allora  $S_0$  è stato finale del DFA, ...
 $\text{else}$                                                   $\triangleright \dots$  altrimenti no
     $F_D \leftarrow \emptyset$ 
 $\text{end if}$ 
 $\text{while } Q_D \text{ contiene stati senza transizioni uscenti do}$            $\triangleright$  Ciclo principale
    Scegli  $S \in Q_D$  senza transizioni uscenti
     $\text{for all } a \in \Sigma \text{ do}$                           $\triangleright$  una transizione per ogni simbolo dell'alfabeto
         $S' \leftarrow \emptyset$                                 $\triangleright$  stato di arrivo della transizione
         $\text{for all } q \in S \text{ do}$                       $\triangleright$  lo stato di partenza  $S$  è un insieme di stati di  $N$ 
             $S' \leftarrow S' \cup \delta_N(q, a)$             $\triangleright$  aggiungi gli stati  $\delta_N(q, a)$  ad  $S'$ 
         $\text{end for}$ 
         $S' \leftarrow \text{ECLOSE}(S')$                        $\triangleright$  Chiusura di  $S'$  per  $\epsilon$ -transizioni
         $Q_D \leftarrow Q_D \cup \{S'\}$                     $\triangleright$  Aggiungi lo stato  $S'$  al DFA
         $\text{if } S' \subseteq F_N \text{ then}$                   $\triangleright$  Se  $S'$  contiene solamente stati finali ...
             $F_D \leftarrow F_D \cup \{S'\}$                    $\triangleright \dots$  allora  $S'$  è finale per il DFA
         $\text{end if}$ 
         $\delta_D(S, a) \leftarrow S'$                           $\triangleright$  Aggiungi la transizione da  $S$  ad  $S'$  con input  $a$  al DFA
     $\text{end for}$ 
 $\text{end while}$ 
return  $D = (Q_D, \Sigma, S_0, \delta_D, F_D)$ 

```

L'algoritmo è del tutto analogo a quello usato per trasformare un  $\epsilon$ -NFA in un DFA, con la sola differenza della definizione degli stati finali, che in questo caso sono tutti gli insiemi di stati  $S$  che contengono solamente stati finali, per rappresentare il fatto che un all- $\epsilon$ -NFA accetta quando tutti i possibili stati in cui si può trovare dopo aver consumato l'input sono stati finali.

**Soluzione alternativa:** in alternativa all'algoritmo si può dare la definizione del DFA  $D = (Q_D, \Sigma, S_0, \delta_D, F_D)$  equivalente all'all- $\epsilon$ -NFA  $N = (Q_N, \Sigma, q_0, \delta_N, F_N)$  specificando le componenti del DFA:

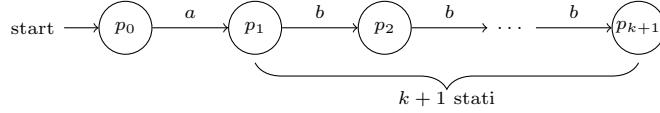
- l'insieme degli stati è l'insieme delle parti di  $Q_N$ :  $Q_D = \{S \mid S \subseteq Q_N\}$ ;
- lo stato iniziale è la  $\epsilon$ -chiusura di  $q_0$ :  $S_0 = \text{ECLOSE}(q_0)$ ;
- la funzione di transizione “simula” le transizioni di  $N$ :

$$\delta_D(S, a) = \text{ECLOSE}(\bigcup_{p \in S} \delta_N(p, a))$$

- l'insieme degli stati finali è l'insieme delle parti di  $F_N$ :  $F_D = \{S \mid S \subseteq F_N\}$ .

Anche questa definizione è analoga a quella che trasforma un  $\epsilon$ -NFA in un DFA, con la sola differenza della definizione degli stati finali.

2. (a) **Prima alternativa:** Possiamo dimostrare che  $L_1$  non è regolare modificando la dimostrazione che il linguaggio  $\{0^n1^n \mid n \geq 0\}$  non è regolare. Supponiamo che  $L_1$  sia regolare: allora deve esistere un DFA  $A$  che lo riconosce. Il DFA avrà un certo numero di stati  $k$ . Consideriamo la computazione di  $A$  con l'input  $ab^k$ :



Poiché la sequenza di stati  $p_1, p_2, \dots, p_{k+1}$  che legge  $b^k$  è composta da  $k + 1$  stati, allora esiste uno stato che si ripete: possiamo trovare  $i < j$  tali che  $p_i = p_j$ . Chiamiamo  $q$  questo stato. Cosa succede quando l'automa  $A$  legge  $c^i$  partendo da  $q$ ?

- Se termina in uno stato finale, allora l'automa accetta, sbagliando, la parola  $ab^j c^i$ .
- Se termina in uno stato non finale allora l'automa rifiuta, sbagliando, la parola  $ab^i c^i$ .

In entrambi i casi abbiamo trovato un assurdo, quindi  $L_1$  non può essere regolare.

**Seconda alternativa:** Per le proprietà di chiusura dei linguaggi regolari, sappiamo che l'intersezione di linguaggi regolari è un linguaggio regolare. Se intersechiamo  $L_1$  con un linguaggio regolare e quello che otteniamo non è un linguaggio regolare, allora possiamo concludere che  $L_1$  non può essere regolare. Consideriamo il linguaggio  $L' = L_1 \cap \{ab^*c^*\} = \{ab^m c^m \mid m \geq 0\}$ , e usiamo il Pumping Lemma per dimostrare che non è regolare. Supponiamo per assurdo che  $L'$  sia regolare:

- sia  $k$  la lunghezza data dal Pumping Lemma;
- consideriamo la parola  $w = ab^k c^k$ , che appartiene ad  $L'$  ed è di lunghezza maggiore di  $k$ ;
- sia  $w = xyz$  una suddivisione di  $w$  tale che  $y \neq \varepsilon$  e  $|xy| \leq k$ ;
- siccome  $|xy| \leq k$ , allora  $x$  e  $y$  devono cadere all'interno del prefisso  $ab^k$  della parola  $w$ . Ci sono due casi possibili, secondo la struttura di  $y$ :
  - $y$  contiene la  $a$  iniziale. In questo caso la parola  $xy^2z$  non appartiene ad  $L'$  perché contiene due  $a$ ;
  - $y$  contiene solamente  $b$ . In questo caso la parola  $xy^2z$  non appartiene ad  $L'$  perché contiene più  $b$  che  $c$ .

In entrambi i casi abbiamo trovato un assurdo quindi  $L'$  non è regolare, e possiamo concludere che neanche  $L_1$  può essere regolare.

- (b) Mostriamo che  $L_1$  si comporta come un linguaggio regolare rispetto al Pumping Lemma.

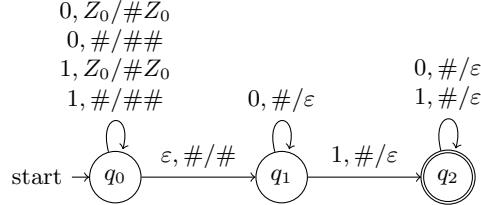
- Poniamo come lunghezza del pumping  $k = 2$ .
- Data una qualsiasi parola  $w = a^\ell b^m c^n \in L_1$  di lunghezza maggiore o uguale a 2, si possono presentare vari casi, secondo il numero di  $a$  presenti nella parola:
  - se c'è una sola  $a$ , allora  $w = ab^m c^m$ . Scegliamo la suddivisione  $x = \varepsilon$ ,  $y = a$  e  $z = b^m c^m$ . Per ogni esponente  $i \geq 0$ , la parola  $xy^i z = a^i b^m c^m$  appartiene a  $L_1$ : se  $i = 1$  allora il numero di  $b$  è uguale al numero di  $c$  come richiesto, mentre se  $i \neq 1$  il linguaggio non pone condizioni sul numero di  $b$  e  $c$ ;
  - se ci sono esattamente due  $a$ , allora  $w = aab^m c^n$ . Scegliamo la suddivisione  $x = \varepsilon$ ,  $y = aa$  e  $z = b^m c^n$ . Per ogni esponente  $i \geq 0$ , la parola  $xy^i z = a^{2i} b^m c^n$  appartiene a  $L_1$ : il numero di  $a$  è pari, quindi sempre diverso da 1, e ricadiamo nelle situazioni in cui il linguaggio non pone condizioni sul numero di  $b$  e  $c$ ;
  - se ci sono almeno tre  $a$ , allora  $w = a^\ell b^m c^n$  con  $\ell \geq 3$ . Scegliamo la suddivisione  $x = \varepsilon$ ,  $y = a$  e  $z = a^{\ell-1} b^m c^n$ . Per ogni esponente  $i \geq 0$ , la parola  $xy^i z = a^{i+\ell-1} b^m c^n$  contiene almeno due  $a$ , e quindi appartiene a  $L_1$ , perché rientra nelle situazioni in cui il linguaggio non pone condizioni sul numero di  $b$  e  $c$ ;
  - se non ci sono  $a$ , allora  $w = b^m c^n$ . Scegliamo la suddivisione che pone  $x = \varepsilon$ ,  $y$  uguale al primo carattere della parola e  $z$  uguale al resto della parola. Per ogni esponente  $i \geq 0$ , la parola  $xy^i z$  sarà nella forma  $b^p c^q$  per qualche  $p, q \geq 0$  e quindi appartenente a  $L_1$ , perché quando non ci sono  $a$  il linguaggio non pone condizioni sul numero di  $b$  e  $c$ .

In tutti i casi possibili la parola può essere pompata senza uscire dal linguaggio, quindi  $L_1$  rispetta le condizioni del Pumping Lemma.

- (c) Il Pumping Lemma stabilisce che se un linguaggio è regolare, allora deve rispettare certe condizioni. Il verso opposto dell'implicazione non è vero: possono esistere linguaggi, come  $L_1$ , che rispettano le condizioni ma non sono regolari. Di conseguenza, i punti (a) e (b) non contraddicono il lemma.

3. (a) Il PDA che riconosce  $L_2$  opera nel modo seguente:

- inizia a consumare l'input ed inserisce un carattere  $\#$  per ogni simbolo che consuma, rimanendo nello stato  $q_0$ ;
- ad un certo punto, sceglie nondeterministicamente che ha consumato la prima metà della parola, e si sposta nello stato  $q_1$ ;
- in  $q_1$ , estrae un carattere  $\#$  dalla pila per ogni 0 che consuma dall'input;
- quando legge il primo 1 nella seconda parte della parola, estrae un  $\#$  dalla pila e si sposta in  $q_2$ , che è uno stato finale;
- in  $q_2$ , continua ad estrarre un  $\#$  dalla pila per ogni carattere che consuma (0 o 1).



Per accettare una parola, il PDA deve:

- inserire nella pila un certo numero di  $\#$
- estrarre dalla pila un numero di  $\#$  minore o uguale di quanti ne ha inserito in pila
- consumare almeno un 1 durante lo svuotamento della pila

Quindi l'automa accetta solo parole  $w = uv$  dove  $u$  è la parte di parola consumata durante la fase di riempimento della pila e  $v$  è la parte di parola consumata durante lo svuotamento della pila. La parola  $v$  contiene almeno un 1 ed è di lunghezza minore o uguale a  $u$ . Se  $u$  è più corta di  $v$  il PDA svuota la pila prima di riuscire a consumare tutta la parola e si blocca. Se invece  $v$  non contiene 1 allora il PDA termina la computazione nello stato  $q_1$  che non è finale.

- (b) Per costruire una CFG che genera  $L_2$  prendiamo una qualsiasi parola  $w$  che sta in  $L_2$ . Se consideriamo l'ultima occorrenza di un 1 nella parola, possiamo riscrivere la parola come  $w = u10^k$ , con  $k \geq 0$  e  $|u| \geq k + 1$ , perché l'ultima occorrenza di 1 deve stare nella seconda metà della parola. Se spezziamo ulteriormente  $u$  in  $u = xy$  con  $|x| = k + 1$  e  $|y| \geq 0$ , allora possiamo definire la grammatica che genera  $L_2$  come segue:

$$\begin{aligned} S &\rightarrow 0S0 \mid 1S0 \mid 0T1 \mid 1T1 \\ T &\rightarrow 0T \mid 1T \mid \epsilon \end{aligned}$$

Nella grammatica, la variabile  $S$  genera stringhe del tipo  $xT10^k$  con  $x \in \{0, 1\}^*$  e  $|x| = k + 1$ , mentre  $T$  genera stringhe  $y \in \{0, 1\}^*$  con  $|y| \geq 0$ . Quindi la grammatica genera tutte e sole le stringhe del tipo  $xy10^k$  dove  $|xy| \geq k + 1$ , che corrispondono alle stringhe che stanno nel linguaggio  $L_2$ .

AUTOMI E LINGUAGGI FORMALI  
 SOLUZIONE DELLA PRIMA PROVA INTERMEDIA

1. Per risolvere l'esercizio dobbiamo dimostrare che (a) ogni linguaggio riconosciuto da una TM con reset a sinistra è Turing-riconoscibile e (b) ogni linguaggio Turing-riconoscibile è riconosciuto da una TM con reset a sinistra.

(a) Mostriamo come convertire una TM con reset a sinistra  $M$  in una TM standard  $S$  equivalente.  $S$  simula il comportamento di  $M$  nel modo seguente. Se la mossa da simulare prevede uno spostamento a destra, allora  $S$  esegue direttamente la mossa. Se la mossa prevede un *RESET*, allora  $S$  scrive il nuovo simbolo sul nastro, poi scorre il nastro a sinistra finché non trova il simbolo  $\triangleright$ , e riprende la simulazione dall'inizio del nastro. Per ogni stato  $q$  di  $M$ ,  $S$  possiede uno stato  $q_{RESET}$  che serve per simulare il reset e riprendere la simulazione dallo stato corretto.

$S =$  "Su input  $w$ :

1. scrive il simbolo  $\triangleright$  subito prima dell'input, in modo che il nastro contenga  $\triangleright w$ .
2. Se la mossa da simulare è  $\delta(q, a) = (r, b, R)$ , allora  $S$  la esegue direttamente: scrive  $b$  sul nastro, muove la testina a destra e va nello stato  $r$ .
3. Se la mossa da simulare è  $\delta(q, a) = (r, b, RESET)$ , allora  $S$  esegue le seguenti operazioni: scrive  $b$  sul nastro, poi muove la testina a sinistra e va nello stato  $r_{RESET}$ . La macchina rimane nello stato  $r_{RESET}$  e continua a muovere la testina a sinistra finché non trova il simbolo  $\triangleright$ . A quel punto la macchina sposta la testina un'ultima volta a sinistra, poi di una cella a destra per tornare sopra al simbolo di fine nastro. La computazione riprende dallo stato  $r$ .
4. Se non sei nello stato di accettazione o di rifiuto, ripeti da 2."

(b) Mostriamo come convertire una TM standard  $S$  in una TM con reset a sinistra  $M$  equivalente.  $M$  simula il comportamento di  $S$  nel modo seguente. Se la mossa da simulare prevede uno spostamento a destra, allora  $M$  può eseguire direttamente la mossa. Se la mossa da simulare prevede uno spostamento a sinistra, allora  $M$  simula la mossa come descritto dall'algoritmo seguente. L'algoritmo usa un nuovo simbolo  $\triangleleft$  per identificare la fine della porzione di nastro usata fino a quel momento, e può marcare le celle del nastro ponendo un punto al di sopra di un simbolo.

$M =$  "Su input  $w$ :

1. Scrive il simbolo  $\triangleleft$  subito dopo l'input, per marcare la fine della porzione di nastro utilizzata. Il nastro contiene  $\triangleright w \triangleleft$ .
2. Simula il comportamento di  $S$ . Se la mossa da simulare è  $\delta(q, a) = (r, b, R)$ , allora  $M$  la esegue direttamente: scrive  $b$  sul nastro, muove la testina a destra e va nello stato  $r$ . Se muovendosi a destra la macchina si sposta sulla cella che contiene  $\triangleleft$ , allora questo significa che  $S$  ha spostato la testina sulla parte di nastro vuota non usata in precedenza. Quindi  $M$  scrive un simbolo blank marcato su questa cella, sposta  $\triangleleft$  di una cella a destra, e fa un reset a sinistra. Dopo il reset si muove a destra fino al blank marcato, e prosegue con la simulazione mossa successiva.
3. Se la mossa da simulare è  $\delta(q, a) = (r, b, L)$ , allora  $S$  esegue le seguenti operazioni:
  - 3.1 scrive  $b$  sul nastro, marcandolo con un punto, poi fa un reset a sinistra
  - 3.2 Se il simbolo subito dopo  $\triangleright$  è già marcato, allora vuol dire che  $S$  ha spostato la testina sulla parte vuota di sinistra del nastro. Quindi  $M$  scrive un blank e sposta il contenuto del nastro di una cella a destra finché non trova il simbolo di fine nastro  $\triangleleft$ . Fa un reset a sinistra e prosegue con la simulazione della prossima mossa dal nuovo blank posto subito dopo l'inizio del nastro. Se il simbolo subito dopo  $\triangleright$  non è marcato, lo marca, resetta a sinistra e prosegue con i passi successivi.
  - 3.3 Si muove a destra fino al primo simbolo marcato, e poi a destra di nuovo.
  - 3.4 se la cella in cui si trova è marcata, allora è la cella da cui è partita la simulazione. Toglie la marcatura e resetta. Si muove a destra finché non trova una cella marcata. Questa cella è quella immediatamente precedente la cella di partenza, e la simulazione della mossa è terminata
  - 3.5 se la cella in cui si trova non è marcata, la marca, resetta, si muove a destra finché non trova una marcatura, cancella la marcatura e riprende da 3.3.
4. Se non sei nello stato di accettazione o di rifiuto, ripeti da 2."

2. (a) Dimostriamo separatamente i due versi del se e solo se.

- Supponiamo che  $A$  sia Turing-riconoscibile. Allora esiste una Macchina di Turing  $M$  che riconosce  $A$ . Consideriamo la funzione  $f$  tale che  $f(w) = \langle M, w \rangle$  per ogni stringa  $w \in \Sigma^*$ . Questa funzione è calcolabile ed è una funzione di riduzione da  $A$  a  $A_{TM}$ . Infatti, se  $w \in A$  allora anche  $\langle M, w \rangle \in A_{TM}$  perché la macchina  $M$  accetta le stringhe che appartengono ad  $A$ . Viceversa, se  $w \notin A$ , allora  $\langle M, w \rangle \notin A_{TM}$  perché la macchina  $M$  rifiuta le stringhe che non appartengono ad  $A$ .
- Supponiamo che  $A \leq_m A_{TM}$ . Sappiamo che  $A_{TM}$  è un linguaggio Turing-riconoscibile. Per le proprietà delle riduzioni mediante funzione, possiamo concludere che anche  $A$  è Turing-riconoscibile.

(b) Dimostriamo separatamente i due versi del se e solo se.

- Supponiamo che  $A$  sia decidibile. Allora esiste una Macchina di Turing  $M$  che decide  $A$ . Consideriamo la funzione  $f$  definita nel modo seguente:

$$f(w) = \begin{cases} 01 & \text{se } M \text{ accetta } w \\ 10 & \text{se } M \text{ rifiuta } w \end{cases}$$

$M$  è un decisore e la sua computazione termina sempre. Quindi la funzione  $f$  può essere calcolata dalla seguente macchina di Turing:

$F =$  "su input  $w$ :

1. Esegui  $M$  su input  $w$ .
2. Se  $M$  accetta, restituisci 01, se  $M$  rifiuta, restituisci 10."

$f$  è anche funzione di riduzione da  $A$  a  $0^*1^*$ . Infatti, se  $w \in A$  allora  $f(w) = 01$  che appartiene al linguaggio  $0^*1^*$ . Viceversa, se  $w \notin A$ , allora  $f(w) = 10$  che non appartiene a  $0^*1^*$ .

- Supponiamo che  $A \leq_m 0^*1^*$ . Sappiamo che  $0^*1^*$  è un linguaggio decidibile. Per le proprietà delle riduzioni mediante funzione, possiamo concludere che anche  $A$  è decidibile.

3. Per mostrare che  $LPATH$  è NP-completo, dimostriamo prima che  $LPATH$  è in NP, e poi che è NP-Hard.

- $LPATH$  è in NP. Il cammino da  $s$  a  $t$  di lunghezza maggiore o uguale a  $k$  è il certificato. Il seguente algoritmo è un verificatore per  $LPATH$ :

$V$  = “Su input  $\langle\langle G, s, t, k \rangle, c \rangle$ :

1. Controlla che  $c$  sia una sequenza di vertici di  $G$ ,  $v_1, \dots, v_m$  e che  $m$  sia minore o uguale al numero di vertici in  $G$  più uno. Se non lo è, rifiuta.
2. Se la sequenza è di lunghezza minore o uguale a  $k$ , rifiuta.
3. Controlla se  $s = v_1$  e  $t = v_m$ . Se una delle due è falsa, rifiuta.
4. Controlla se ci sono ripetizioni nella sequenza. Se ne trova una diversa da  $v_1 = v_m$ , rifiuta.
5. Per ogni  $i$  tra 1 e  $m - 1$ , controlla se  $(p_i, p_{i+1})$  è un arco di  $G$ . Se non lo è rifiuta.
6. Se tutti i test sono stati superati, accetta.”

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. Ognuna delle fasi è un controllo sugli  $m$  elementi del certificato, e quindi richiede un tempo polinomiale rispetto ad  $m$ . Poiché l'algoritmo rifiuta immediatamente quando  $m$  è maggiore del numero di vertici di  $G$  più uno, allora possiamo concludere che l'algoritmo richiede un tempo polinomiale rispetto al numero di vertici del grafo.

- Dimostriamo che  $LPATH$  è NP-Hard per riduzione polinomiale da  $HAMILTON$  a  $LPATH$ . La funzione di riduzione polinomiale  $f$  prende in input un grafo  $\langle G \rangle$  e produce come output la quadrupla  $\langle G, s, s, n \rangle$  dove  $s$  è un vertice arbitrario di  $G$  e  $n$  è uguale al numero di vertici di  $G$ . Dimostriamo che la riduzione polinomiale è corretta:

- Se  $\langle G \rangle \in HAMILTON$ , allora esiste un circuito Hamiltoniano in  $G$ . Dato un qualsiasi vertice  $s$  di  $G$ , possiamo costruire un cammino che parte da  $s$  e segue il circuito Hamiltoniano per tornare in  $s$ . Questo cammino attraversa tutti gli altri vertici di  $G$  prima di tornare in  $s$  e sarà quindi di lunghezza  $n$ . Di conseguenza  $\langle G, s, s, n \rangle \in LPATH$ .
- Se  $\langle G, s, s, n \rangle \in LPATH$ , allora esiste un cammino semplice nel grafo  $G$  che inizia e termina in  $s$  ed è di lunghezza maggiore o uguale a  $n$ . Un cammino semplice non ha ripetizioni, ad eccezione dei vertici iniziali e finali. Un cammino semplice da  $s$  ad  $s$  di lunghezza  $n$  deve attraversare tutti gli altri nodi una sola volta prima di tornare in  $s$ , ed è quindi un circuito Hamiltoniano per  $G$ . Cammini semplici più lunghi non possono esistere perché dovrebbero ripetere dei vertici. Quindi l'esistenza di un cammino semplice nel grafo  $G$  che inizia e termina in  $s$  ed è di lunghezza maggiore o uguale a  $n$  implica l'esistenza di un circuito Hamiltoniano in  $G$ , ed abbiamo dimostrato che  $\langle G \rangle \in HAMILTON$ .

La funzione di riduzione si limita ad aggiungere tre nuovi elementi dopo la codifica del grafo  $G$ : due vertici ed un numero, operazione che si può fare in tempo polinomiale.

- 1.** Supponiamo che  $L$  ed  $M$  siano due linguaggi regolari, e mostriamo che anche  $\text{faro}(L, M)$  è regolare. Poiché  $L$  ed  $M$  sono regolari, sappiamo che esiste un DFA  $A_L$  che riconosce  $L$  ed un DFA  $A_M$  che riconosce  $M$ . Per dimostrare che  $\text{faro}(L, M)$  è regolare esibiamo un automa a stati finiti  $A$  che riconosce  $\text{faro}(L, M)$ .

La funzione ricorsiva  $\text{faro}(x, z)$  rimescola le parole  $x$  e  $z$  in questo modo:

- se  $|x| = |z|$ , allora il risultato è una parola che alterna i simboli di  $x$  nelle posizioni dispari con i simboli di  $z$  nelle posizioni pari:  $\text{faro}(x, z) = x_1z_1x_2z_2 \dots x_nz_n$ ;
- se  $x$  è più corta di  $z$ , allora  $\text{faro}(x, z)$  alterna simboli di  $x$  con simboli di  $z$  finché possibile, e poi continua con la parte rimanente di  $z$ ;
- se  $z$  è più corta di  $x$ , allora  $\text{faro}(x, z)$  alterna simboli di  $x$  con simboli di  $z$  finché possibile, e poi continua con la parte rimanente di  $x$ .

L'automa che accetta  $\text{faro}(L, M)$  procede alternando transizioni di  $A_L$  con transizioni di  $A_M$  per ottenere l'alternanza dei simboli della parola  $x \in L$  con quelli della parola  $z \in M$ . Per poter simulare l'alternanza l'automa deve memorizzare lo stato corrente di  $A_L$ , lo stato corrente di  $A_M$  e quale automa simulare alla prossima transizione. Gli stati  $A$  saranno quindi delle triple  $(r_L, r_M, t)$  dove  $r_L$  è uno stato di  $A_L$ ,  $r_M$  è uno stato di  $A_M$  e  $t \in \{L, M\}$  un valore che rappresenta il turno della simulazione. Le transizioni sono definite come segue:

- $(r_L, r_M, L) \xrightarrow{a} (s_L, r_M, M)$  se  $\delta_L(r_L, a) = s_L$ : quando è il turno di  $A_L$  si simula la transizione di  $A_L$ , si mantiene inalterato lo stato di  $A_M$  e si cambia il turno a  $M$  per la prossima transizione;
- $(r_L, r_M, M) \xrightarrow{a} (r_L, s_M, L)$  se  $\delta_M(r_M, a) = s_M$ : quando è il turno di  $A_M$  si simula la transizione di  $A_M$ , si mantiene inalterato lo stato di  $A_L$  e si cambia il turno a  $L$  per la prossima transizione.

Dobbiamo ora stabilire quali sono gli stati finali di  $A$ . Quando la simulazione raggiunge uno stato finale di  $A_L$  possono succedere due cose: si continua ad alternare transizioni di  $A_L$  con transizioni di  $A_M$ , oppure “scommettere” che abbiamo terminato di consumare i simboli della parola  $x$ , e continuare con la parte rimanente di  $z$  fino a raggiungere uno stato finale di  $A_M$ . Viceversa, quando la simulazione raggiunge uno stato finale di  $A_M$ , sceglie se continuare con l'alternanza oppure con la parte rimanente di  $x$ . Useremo il nondeterminismo per rappresentare queste scelte:  $A$  sarà un NFA anche se  $A_L$  e  $A_M$  sono dei DFA. Oltre al nondeterminismo dobbiamo aggiungere altri due tipi di turno:  $L_{suff}$  e  $M_{suff}$  per rappresentare le computazioni sulla parte rimanente di parola in  $L$  o sulla parte rimanente di parola in  $M$ . Quindi gli stati di  $A$  saranno delle triple  $(r_L, r_M, t)$  con  $t \in \{L, M, L_{suff}, M_{suff}\}$ , e dobbiamo aggiungere le seguenti transizioni all'automa:

- $(r_L, r_M, L) \xrightarrow{a} (s_L, r_M, L_{suff})$  se  $\delta_L(r_L, a) = s_L$  e  $r_M$  è uno stato finale di  $A_M$ ;
- $(r_L, r_M, M) \xrightarrow{a} (r_L, s_M, M_{suff})$  se  $\delta_M(r_M, a) = s_M$  e  $r_L$  è uno stato finale di  $A_L$ ;
- $(r_L, r_M, L_{suff}) \xrightarrow{a} (s_L, r_M, L_{suff})$  se  $\delta_L(r_L, a) = s_L$ ;
- $(r_L, r_M, M_{suff}) \xrightarrow{a} (r_L, s_M, M_{suff})$  se  $\delta_M(r_M, a) = s_M$ ;
- $(q_L^0, q_M^0, L) \xrightarrow{a} (q_L^0, s_M, M_{suff})$  se  $\delta_M(q_M^0, a) = s_M$  e  $q_L^0$  è uno stato finale di  $A_L$ .

Si può notare che se il turno diventa uguale a  $L_{suff}$  allora  $A$  prosegue simulando solamente le transizioni di  $A_L$ , senza cambiare lo stato  $r_M$ . Viceversa, quando il turno diventa uguale a  $M_{suff}$   $A$  prosegue simulando solamente le transizioni di  $A_M$ , senza cambiare lo stato  $r_L$ . L'ultimo caso serve per gestire il caso in cui la parola vuota appartiene a  $L$  e la computazione deve iniziare immediatamente a riconoscere una parola che appartiene a  $M$ .

Gli stati finali di  $A$  sono tutte le triple  $(r_L, r_M, t)$  tali che  $r_L$  è uno stato finale di  $A_L$  e  $r_M$  è uno stato finale di  $M$ . Lo stato iniziale è la tripla  $(q_L^0, q_M^0, L)$ .

## 2. Consideriamo il linguaggio

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

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

$$xy^0z = xz = 1^q 1^{k-q-p} \# 1^{k-1} \# \dots \# 1 \# = 1^{k-p} \# 1^{k-1} \# \dots \# 1 \#$$

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.

- 3.** Per dimostrare che ogni grammatica context-free generalizzata descrive un linguaggio context-free dobbiamo dimostrare che le grammatiche generalizzate sono equivalenti alle normali grammatiche context free.

È facile vedere che le grammatiche context-free sono un caso particolare di grammatiche context-free generalizzate: una regola  $A \rightarrow u$  dove  $u$  è una stringa di variabili e terminali è una regola valida anche per le grammatiche generalizzate.

Dimostreremo che consentire espressioni regolari nelle regole non aumenta il potere espressivo delle grammatiche mostrando come possiamo costruire una grammatica context-free equivalente ad una grammatica generalizzata. La costruzione procede rimpiazzando le regole con espressioni regolari con altre regole equivalenti, finché tutte le regole sono nella forma semplice consentita nelle grammatiche context-free normali:

- (a) rimpiazza ogni regola  $A \rightarrow R + S$  con le due regole  $A \rightarrow R$  e  $A \rightarrow S$ ;
- (b) per ogni regola  $A \rightarrow R.S$ , aggiungi due nuove variabili  $A_R$  e  $A_S$  e rimpiazza la regola con le regole  $A \rightarrow A_R A_S$ ,  $A \rightarrow R$  e  $A \rightarrow S$ ;
- (c) per ogni regola  $A \rightarrow S^*$ , aggiungi una nuova variabile  $A_S$  e rimpiazza la regola con le regole  $A \rightarrow A_S A$ ,  $A \rightarrow \epsilon$  e  $A_S \rightarrow S$ ;
- (d) rimpiazza ogni regola  $A \rightarrow \emptyset$  con la regola  $A \rightarrow A$  (*in alternativa*: rimuovi le regole  $A \rightarrow \emptyset$ );
- (e) ripeti da (a) finché non rimangono solamente regole del tipo  $A \rightarrow u$  dove  $u$  è una stringa di variabili e terminali, oppure  $A \rightarrow \epsilon$ .

Ogni passaggio della costruzione modifica la grammatica in modo da essere sicuri che generi lo stesso linguaggio. Inoltre, la costruzione termina quando tutte le regole sono nella forma “standard”  $A \rightarrow u$ , senza operatori regolari.

AUTOMI E LINGUAGGI FORMALI – A.A. 2020/21  
 SOLUZIONE DELLA PRIMA PROVA INTERMEDIA

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

$$w_1^{\bullet} w_2 \dots w_n \underline{\quad} \quad \dots$$

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

$$w_1 w_2 \dots w_n \# w_1^{\bullet} w_2 \dots w_n \underline{\quad},$$

- che la testina di lettura si trovi in corrispondenza del  $w_1^{\bullet}$  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 stati 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  $a$  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  $a$  e  $\delta(st, a) = (q, b, L)$ :
    - aggiorna lo stato della simulazione  $st = q$ ;
    - scrive  $prec$ ; se  $prec = \#$  scrive  $\#\bullet$ ;
    - scrive  $b$ .
  5. Copia il resto della configurazione fino al  $\#$  escluso. Al termine della copia la testina di lettura si 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.”

2. (a) Dimostriamo che  $\text{ALWAYSDIVERGE}$  è un linguaggio indecidibile mostrando che  $\overline{A_{TM}}$  è riducibile ad  $\text{ALWAYSDIVERGE}$ . La funzione di riduzione  $f$  è calcolata dalla seguente macchina di Turing:

$F =$  "su input  $\langle M, w \rangle$ , dove  $M$  è una TM e  $w$  una stringa:

1. Costruisci la seguente macchina  $M'$ :
- $M' =$  "su input  $x$ :

  1. Se  $x \neq w$ , vai in loop.
  2. Se  $x = w$ , esegue  $M$  su input  $w$ .
  3. Se  $M$  accetta, *accetta*.
  4. Se  $M$  rifiuta, vai in loop."

2. Restituisci  $\langle M' \rangle$ ."

Dimostriamo che  $f$  è una funzione di riduzione da  $\overline{A_{TM}}$  ad  $\text{ALWAYSDIVERGE}$ .

- Se  $\langle M, w \rangle \in \overline{A_{TM}}$  allora la computazione di  $M$  su  $w$  non termina oppure termina rifiutando. Di conseguenza la macchina  $M'$  costruita dalla funzione non termina mai la computazione per qualsiasi input. Quindi  $f(\langle M, w \rangle) = \langle M' \rangle \in \text{ALWAYSDIVERGE}$ .
- Viceversa, se  $\langle M, w \rangle \notin \overline{A_{TM}}$  allora la computazione di  $M$  su  $w$  termina con accettazione. Di conseguenza la macchina  $M'$  termina la computazione sull'input  $w$  e  $f(\langle M, w \rangle) = \langle M' \rangle \notin \text{ALWAYSDIVERGE}$ .

Per concludere, siccome abbiamo dimostrato che  $\overline{A_{TM}} \leq_m \text{ALWAYSDIVERGE}$  e sappiamo che  $\overline{A_{TM}}$  è indecidibile, allora possiamo concludere che  $\text{ALWAYSDIVERGE}$  è indecidibile.

- (b)  $\text{ALWAYSDIVERGE}$  è un linguaggio coTuring-riconoscibile. Dato un ordinamento  $s_1, s_2, \dots$  di tutte le stringhe in  $\Sigma^*$ , la seguente TM riconosce il complementare  $\text{ALWAYSDIVERGE}$ :

$D =$  "su input  $\langle M \rangle$ , dove  $M$  è una TM:

1. Ripeti per  $i = 1, 2, 3, \dots$ :
2. Esegue  $M$  per  $i$  passi di computazione su ogni input  $s_1, s_2, \dots, s_i$ .
3. Se  $M$  termina la computazione su almeno uno, *accetta*. Altrimenti continua."

3. (a) WSP è in NP. La disposizione degli ospiti tra i tavoli è il certificato. Se gli ospiti sono numerati da 1 a  $n$  la possiamo rappresentare con un vettore  $T$  tale che  $T[i]$  è il tavolo assegnato all'ospite  $i$ . Il seguente algoritmo è un verificatore per WSP:

$V = \text{"Su input } \langle \langle n, k, R \rangle, T \rangle:$

1. Controlla che  $T$  sia un vettore di  $n$  elementi dove ogni elemento ha un valore compreso tra 1 e  $k$ . Se non lo è, rifiuta.
2. Per ogni coppia  $i, j$  tale che  $R[i, j] = 1$ , controlla che  $T[i] = T[j]$ . Se il controllo fallisce, rifiuta.
3. Per ogni coppia  $i, j$  tale che  $R[i, j] = -1$ , controlla che  $T[i] \neq T[j]$ . Se il controllo fallisce, rifiuta.
4. Se tutti i test sono stati superati, accetta."

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. La prima fase è un controllo sugli  $n$  elementi del vettore  $T$ , e quindi richiede un tempo polinomiale rispetto alla dimensione dell'input. La seconda e terza fase controllano gli elementi della matrice  $R$ , operazione che si può fare in tempo polinomiale rispetto alla dimensione dell'input.

- (b) Dimostriamo che WSP è NP-Hard per riduzione polinomiale da 3-COLOR a WSP. La funzione di riduzione polinomiale  $f$  prende in input un grafo  $\langle G \rangle$  e produce come output la tripla  $\langle n, 3, R \rangle$  dove  $n$  è il numero di vertici di  $G$  e la matrice  $R$  è definita in questo modo:

$$R[i, j] = \begin{cases} -1 & \text{se c'è un arco da } i \text{ a } j \text{ in } G \\ 0 & \text{altrimenti} \end{cases}$$

Dimostriamo che la riduzione polinomiale è corretta:

- Se  $\langle G \rangle \in \text{3-COLOR}$ , allora esiste un modo per colorare  $G$  con tre colori 1, 2, 3. La disposizione degli ospiti che fa sedere l'ospite  $i$  nel tavolo corrispondente al colore del vertice  $i$  nel grafo è corretta:
  - ogni invitato siede ad un solo tavolo;
  - per ogni coppia di invitati rivali  $i, j$  si ha che  $R[i, j] = -1$  e, per la definizione della funzione di riduzione, l'arco  $(i, j)$  appartiene a  $G$ . Quindi la colorazione assegna colori diversi ad  $i$  e  $j$ , ed i due ospiti siedono su tavoli diversi;
  - per la definizione della funzione di riduzione non ci sono ospiti che sono amici tra di loro.
- Se  $\langle n, 3, R \rangle \in \text{WSP}$ , allora esiste una disposizione degli  $n$  ospiti su 3 tavoli dove i rivali siedono sempre su tavoli diversi. La colorazione che assegna al vertice  $i$  il colore corrispondente al tavolo dove siede l'ospite  $i$  è corretta: se c'è un arco tra  $i$  e  $j$  allora i due ospiti sono rivali e siederanno su tavoli diversi, cioè avranno colori diversi nella colorazione. Di conseguenza abbiamo dimostrato che  $\langle G \rangle \in \text{3-COLOR}$ .

La funzione di riduzione deve contare i vertici del grafo  $G$  e costruire la matrice  $R$  di dimensione  $n \times n$ , operazioni che si possono fare in tempo polinomiale.

- 1.** Dimostra che se  $L$  ed  $M$  sono linguaggi regolari sull'alfabeto  $\{0, 1\}$ , allora anche il seguente linguaggio è regolare:

$$L \sqcap M = \{x \sqcap y \mid x \in L, y \in M \text{ e } |x| = |y|\},$$

dove  $x \sqcap y$  rappresenta l'and bit a bit di  $x$  e  $y$ . Per esempio,  $0011 \sqcap 0101 = 0001$ .

Poiché  $L$  e  $M$  sono regolari, sappiamo che esiste un DFA  $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$  che riconosce  $L$  e un DFA  $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$  che riconosce  $M$ .

Costruiamo un NFA  $A$  che riconosce il linguaggio  $L \sqcap M$ :

- L'insieme degli stati è  $Q = Q_L \times Q_M$ , che contiene tutte le coppie composte da uno stato di  $A_L$  e uno stato di  $A_M$ .
- L'alfabeto è lo stesso di  $A_L$  e di  $A_M$ ,  $\Sigma = \{0, 1\}$ .
- La funzione di transizione  $\delta$  è definita come segue:

$$\begin{aligned}\delta((r_L, r_M), 0) &= \{(\delta_L(r_L, 0), \delta_M(r_M, 0)), (\delta_L(r_L, 1), \delta_M(r_M, 0)), (\delta_L(r_L, 0), \delta_M(r_M, 1))\} \\ \delta((r_L, r_M), 1) &= \{(\delta_L(r_L, 1), \delta_M(r_M, 1))\}\end{aligned}$$

La funzione di transizione implementa le regole dell'and tra due bit: l'and di due 1 è 1, mentre è 0 se entrambi i bit sono 0 o se un bit è 0 e l'altro è 1.

- Lo stato iniziale è  $(q_L, q_M)$ .
- Gli stati finali sono  $F = F_L \times F_M$ , ossia tutte le coppie di stati finali dei due automi.

- 2.** Considera il linguaggio

$$L_2 = \{w \in \{0, 1\}^* \mid w \text{ contiene lo stesso numero di } 00 \text{ e di } 11\}.$$

Dimostra che  $L_2$  non è regolare.

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  $w = 0^k 1^k$ , 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 sequenza di 0. Inoltre, siccome  $y \neq \emptyset$ , abbiamo che  $x = 0^q$  e  $y = 0^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 0^{k-q-p} 1^k$ . Consideriamo l'esponente  $i = 0$ : la parola  $xy^0 z$  ha la forma

$$xy^0 z = xz = 0^q 0^{k-q-p} 1^k = 0^{k-p} 1^k$$

e contiene un numero di occorrenze di 00 minore delle occorrenze di 11. Di conseguenza, la parola non appartiene al linguaggio  $L_2$ , in contraddizione con l'enunciato del Pumping Lemma.

- 3.** Dimostra che se  $L$  è un linguaggio context-free, allora anche  $L^R$  è un linguaggio context-free.

Se  $L$  è un linguaggio context free allora esiste una grammatica  $G$  che lo genera. Possiamo assumere che  $G$  sia in forma normale di Chomksy. Di conseguenza le regole di  $G$  sono solamente di due tipi:  $A \rightarrow BC$ , con  $A, B, C$  simboli non terminali, oppure  $A \rightarrow b$  con  $b$  simbolo nonterminale.

Costruiamo la grammatica  $G^R$  che genera  $L^R$  in questo modo:

- ogni regola  $A \rightarrow BC$  viene sostituita dalla regola  $A \rightarrow CB$ ;
- le regole  $A \rightarrow b$  rimangono invariate.

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.

Mostriamo come simulare una TM bidimensionale  $B$  con una TM deterministica a nastro singolo  $S$ .  $S$  memorizza il contenuto della griglia bidimensionale sul nastro come una sequenza di stringhe separate da  $\#$ , ognuna delle quali rappresenta una riga della griglia. Due cancelletti consecutivi  $\#\#$  segnano l'inizio e la fine della rappresentazione della griglia. La posizione della testina di  $B$  viene indicata marcando la cella con  $\hat{}$ . Nelle altre righe, un pallino  $\bullet$  indica che la testina si trova su quella colonna della griglia, ma in una riga diversa. La TM a nastro singolo  $S$  funziona come segue:

$S = "su input  $w":$$

1. Sostituisce  $w$  con la configurazione iniziale  $\#\#w\#\#$  e marca con  $\hat{}$  il primo simbolo di  $w$ .
2. Scorre il nastro finché non trova la cella marcata con  $\hat{}$ .
3. Aggiorna il nastro in accordo con la funzione di transizione di  $B$ :
  - Se  $\delta(r, a) = (s, b, \rightarrow)$ , scrive  $b$  non marcato sulla cella corrente, sposta  $\hat{}$  sulla cella immediatamente a destra. Poi sposta di una cella a destra tutte le marcature con un pallino. Se in qualsiasi momento  $S$  sposta una marcatura sopra un  $\#$ ,  $S$  scrive un blank marcato al posto del  $\#$  e sposta il contenuto del nastro da questa cella fino al  $\#\#$  finale, di una cella più a destra.
  - Se  $\delta(r, a) = (s, b, \leftarrow)$ , scrive  $b$  non marcato sulla cella corrente, sposta  $\hat{}$  sulla cella immediatamente a sinistra. Poi sposta di una cella a sinistra tutte le marcature con un pallino. Se in qualsiasi momento  $S$  sposta una marcatura sopra un  $\#$ ,  $S$  scrive un blank marcato al posto del  $\#$  e sposta il contenuto del nastro da questa cella fino al  $\#\#$  iniziale, di una cella più a sinistra.
  - Se  $\delta(r, a) = (s, b, \uparrow)$ , scrive  $b$  marcato con un pallino nella cella corrente, e sposta  $\hat{}$  sulla prima cella marcata con un pallino posta a sinistra della cella corrente. Se questa cella marcata non esiste, aggiunge una nuova riga composta solo da blank all'inizio della configurazione.
  - Se  $\delta(r, a) = (s, b, \downarrow)$ , scrive  $b$  marcato con un pallino nella cella corrente, e sposta  $\hat{}$  sulla prima cella marcata con un pallino posta a destra della cella corrente. Se questa cella non esiste, aggiunge una nuova riga composta solo da blank alla fine della configurazione.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di  $B$ , allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di  $B$  allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

**2.** Dimostra che il seguente linguaggio è indecidibile:

$$A_{1010} = \{\langle M \rangle \mid M \text{ è una TM tale che } 1010 \in L(M)\}.$$

Dimostriamo che  $A_{1010}$  è un linguaggio indecidibile mostrando che  $A_{TM}$  è riducibile ad  $A_{1010}$ . La funzione di riduzione  $f$  è calcolata dalla seguente macchina di Turing:

$F$  = "su input  $\langle M, w \rangle$ , dove  $M$  è una TM e  $w$  una stringa:

1. Costruisci la seguente macchina  $M_w$ :

$M_w$  = "su input  $x$ :

1. Se  $x \neq 1010$ , rifiuta.
2. Se  $x = 1010$ , esegue  $M$  su input  $w$ .
3. Se  $M$  accetta, *accetta*.
4. Se  $M$  rifiuta, *rifiuta*."

2. Restituisci  $\langle M_w \rangle$ ."

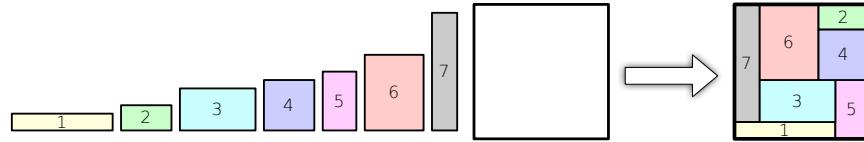
Dimostriamo che  $f$  è una funzione di riduzione da  $A_{TM}$  ad  $A_{1010}$ .

- Se  $\langle M, w \rangle \in A_{TM}$  allora la TM  $M$  accetta  $w$ . Di conseguenza la macchina  $M_w$  costruita dalla funzione accetta la parola 1010. Quindi  $f(\langle M, w \rangle) = \langle M_w \rangle \in A_{1010}$ .
- Viceversa, se  $\langle M, w \rangle \notin A_{TM}$  allora la computazione di  $M$  su  $w$  non termina o termina con rifiuto. Di conseguenza la macchina  $M_w$  rifiuta 1010 e  $f(\langle M, w \rangle) = \langle M_w \rangle \notin A_{1010}$ .

Per concludere, siccome abbiamo dimostrato che  $A_{TM} \leq_m A_{1010}$  e sappiamo che  $A_{TM}$  è indecidibile, allora possiamo concludere che  $A_{1010}$  è indecidibile.

**3.** Il problema SETPARTITIONING chiede di stabilire se un insieme di numeri interi  $S$  può essere suddiviso in due sottoinsiemi disgiunti  $S_1$  e  $S_2$  tali che la somma dei numeri in  $S_1$  è uguale alla somma dei numeri in  $S_2$ . Sappiamo che questo problema è NP-hard.

Il problema RECTANGLETILING è definito come segue: dato un rettangolo grande e diversi rettangoli più piccoli, determinare se i rettangoli più piccoli possono essere posizionati all'interno del rettangolo grande senza sovrapposizioni e senza lasciare spazi vuoti.



Un'istanza positiva di RECTANGLETILING.

Dimostra che RECTANGLETILING è NP-hard, usando SETPARTITIONING come problema di riferimento.

Dimostriamo che RECTANGLETILING è NP-Hard per riduzione polinomiale da SETPARTITIONING. La funzione di riduzione polinomiale prende in input un insieme di interi positivi  $S = \{s_1, \dots, s_n\}$  e costruisce un'istanza di RECTANGLETILING come segue:

- i rettangoli piccoli hanno altezza 1 e base uguale ai numeri in  $S$  moltiplicati per 3:  $(3s_1, 1), \dots, (3s_n, 1)$ ;
- il rettangolo grande ha altezza 2 e base  $\frac{3}{2}N$ , dove  $N = \sum_{i=1}^n s_i$  è la somma dei numeri in  $S$ .

Dimostriamo che esiste un modo per suddividere  $S$  in due insiemi  $S_1$  e  $S_2$  tali che la somma dei numeri in  $S_1$  è uguale alla somma dei numeri in  $S_2$  se e solo se esiste un tiling corretto:

- Supponiamo esista un modo per suddividere  $S$  nei due insiemi  $S_1$  e  $S_2$ . Posizioniamo i rettangoli che corrispondono ai numeri in  $S_1$  in una fila orizzontale, ed i rettangoli che corrispondono ad  $S_2$  in un'altra fila orizzontale. Le due file hanno altezza 1 e base  $\frac{3}{2}N$ , quindi formano un tiling corretto.
- Supponiamo che esista un modo per disporre i rettangoli piccoli all'interno del rettangolo grande senza sovrapposizioni né spazi vuoti. Moltiplicare le base dei rettangoli per 3 serve ad impedire che un rettangolo piccolo possa essere disposto in verticale all'interno del rettangolo grande. Quindi il tiling valido è composto da due file di rettangoli disposti in orizzontale. Mettiamo i numeri corrispondenti ai rettangoli in una fila in  $S_1$  e quelli corrispondenti all'altra fila in  $S_2$ . La somma dei numeri in  $S_1$  ed  $S_2$  è pari ad  $N/2$ , e quindi rappresenta una soluzione per SETPARTITIONING.

Per costruire l'istanza di RECTANGLETILING basta scorrere una volta l'insieme  $S$ , con un costo polinomiale.

AUTOMI E LINGUAGGI FORMALI – 5/7/2021  
PRIMA PARTE – LINGUAGGI REGOLARI E CONTEXT FREE

1. Considera la seguente funzione da  $\{0,1\}^*$  a  $\{0,1\}^*$ :

$$\text{stutter}(w) = \begin{cases} \varepsilon & \text{se } w = \varepsilon \\ aa.\text{stutter}(x) & \text{se } w = ax \text{ per qualche simbolo } a \text{ e parola } x \end{cases}$$

Dimostra che se  $L$  è un linguaggio regolare sull'alfabeto  $\{0,1\}$ , allora anche il seguente linguaggio è regolare:

$$\text{stutter}(L) = \{\text{stutter}(w) \mid w \in L\}.$$

2. Considera il linguaggio

$$L_2 = \{w0^n \mid w \in \{0,1\}^* \text{ e } n = |w|\}.$$

Dimostra che  $L_2$  non è regolare.

3. Per ogni linguaggio  $L$ , sia  $\text{suffix}(L) = \{v \mid uv \in L \text{ per qualche stringa } u\}$ . Dimostra che se  $L$  è un linguaggio context-free, allora anche  $\text{suffix}(L)$  è un linguaggio context-free.

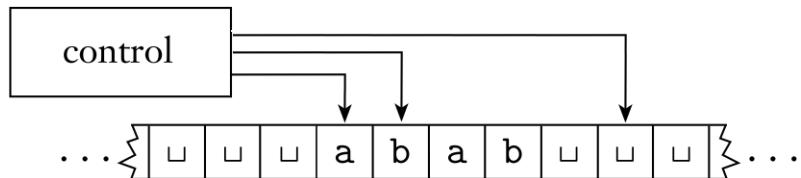
- 1.** Una macchina di Turing a testine multiple è una macchina di Turing con un solo nastro ma con varie testine. Inizialmente, tutte le testine si trovano sopra alla prima cella dell'input. La funzione di transizione viene modificata per consentire la lettura, la scrittura e lo spostamento delle testine. Formalmente,

$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R\}^k$$

dove  $k$  è il numero delle testine. L'espressione

$$\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$$

significa che, se la macchina si trova nello stato  $q_i$  e le testine da 1 a  $k$  leggono i simboli  $a_1, \dots, a_k$  allora la macchina va nello stato  $q_j$ , scrive i simboli  $b_1, \dots, b_k$  e muove le testine a destra e a sinistra come specificato.



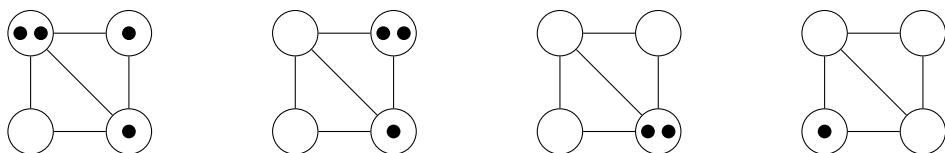
Un esempio di TM con tre testine.

Dimostra che qualsiasi macchina di Turing a testine multiple può essere simulata da una macchina di Turing deterministica a nastro singolo.

- 2.** Dimostra che il seguente linguaggio è indecidibile:

$$L_2 = \{\langle M, w \rangle \mid M \text{ accetta la stringa } ww^R\}.$$

- 3.** Pebbling è un solitario giocato su un grafo non orientato  $G$ , in cui ogni vertice ha zero o più ciottoli. Una mossa del gioco consiste nel rimuovere due ciottoli da un vertice  $v$  e aggiungere un ciottolo ad un vertice  $u$  adiacente a  $v$  (il vertice  $v$  deve avere almeno due ciottoli all'inizio della mossa). Il problema PEBBLEDESTRUCTION chiede, dato un grafo  $G = (V, E)$  ed un numero di ciottoli  $p(v)$  per ogni vertice  $v$ , di determinare se esiste una sequenza di mosse che rimuove tutti i sassolini tranne uno.



Una soluzione in 3 mosse di PEBBLEDESTRUCTION.

Dimostra che PEBBLEDESTRUCTION è NP-hard usando il problema del Circuito Hamiltoniano come problema NP-hard noto (un circuito Hamiltoniano è un ciclo che attraversa ogni vertice di  $G$  esattamente una volta).

**1. (8 punti)** Considera il linguaggio

$$L = \{0^m 1^n \mid m/n \text{ è un numero intero}\}.$$

*Dimostra che  $L$  non è regolare.*

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

Supponiamo per assurdo che  $L$  sia regolare:

- sia  $k$  la lunghezza data dal Pumping Lemma;
- consideriamo la parola  $w = 0^{k+1} 1^{k+1}$ , che è di lunghezza maggiore di  $k$  ed appartiene ad  $L$  perché  $(k+1)/(k+1) = 1$ ;
- 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 sequenza di 0. Inoltre, siccome  $y \neq \varepsilon$ , abbiamo che  $x = 0^q$  e  $y = 0^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 0^{k+1-q-p} 1^{k+1}$ . Consideriamo l'esponente  $i = 0$ : la parola  $xy^0 z$  ha la forma

$$xy^0 z = xz = 0^q 0^{k+1-q-p} 1^{k+1} = 0^{k+1-p} 1^{k+1}.$$

Si può notare che  $(k+1-p)/(k+1)$  è un numero strettamente compreso tra 0 e 1, e quindi non può essere un numero intero. Di conseguenza, la parola non appartiene al linguaggio  $L$ , in contraddizione con l'enunciato del Pumping Lemma.

**2. (8 punti)** Per ogni linguaggio  $L$ , sia  $\text{prefix}(L) = \{u \mid uv \in L \text{ per qualche stringa } v\}$ . Dimostra che se  $L$  è un linguaggio context-free, allora anche  $\text{prefix}(L)$  è un linguaggio context-free.

Se  $L$  è un linguaggio context-free, allora esiste una grammatica  $G$  in forma normale di Chomski che lo genera. Possiamo costruire una grammatica  $G'$  che genera il linguaggio  $\text{prefix}(L)$  in questo modo:

- per ogni variabile  $V$  di  $G$ ,  $G'$  contiene sia la variabile  $V$  che una nuova variabile  $V'$ . La variabile  $V'$  viene usata per generare i prefissi delle parole che sono generate da  $V$ ;
- tutte le regole di  $G$  sono anche regole di  $G'$ ;
- per ogni variabile  $V$  di  $G$ , le regole  $V' \rightarrow V$  e  $V' \rightarrow \varepsilon$  appartengono a  $G$ ;
- per ogni regola  $V \rightarrow AB$  di  $G$ , le regole  $V' \rightarrow AB'$  e  $V' \rightarrow A'$  appartengono a  $G'$ ;
- se  $S$  è la variabile iniziale di  $G$ , allora  $S'$  è la variabile iniziale di  $G'$ .

**3. (8 punti)** Una Turing machine con alfabeto binario è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è  $\Sigma = \{0, 1\}$  e l'alfabeto del nastro è  $\Gamma = \{0, 1, \_\}$ . Questo significa che la macchina può scrivere sul nastro solo i simboli 0, 1 e blank: non può usare altri simboli né marcare i simboli sul nastro.

*Dimostra che che le Turing machine con alfabeto binario riconoscono tutti e soli i linguaggi Turing-riconoscibili sull'alfabeto {0, 1}.*

Per risolvere l'esercizio dobbiamo dimostrare che (a) ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è Turing-riconoscibile e (b) ogni linguaggio Turing-riconoscibile sull'alfabeto  $\{0, 1\}$  è riconosciuto da una Turing machine con alfabeto binario.

- (a) Questo caso è semplice: una Turing machine con alfabeto binario è un caso speciale di Turing machine deterministica a nastro singolo. Quindi ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è anche Turing-riconoscibile.
- (b) Per dimostrare questo caso, consideriamo un linguaggio  $L$  Turing-riconoscibile, e sia  $M$  una Turing machine deterministica a nastro singolo che lo riconosce. Questa TM potrebbe avere un alfabeto del nastro  $\Gamma$  che contiene altri simboli oltre a 0, 1 e blank. Per esempio potrebbe contenere simboli marcati o separatori.

Per costruire una TM con alfabeto binario  $B$  che simula il comportamento di  $M$  dobbiamo come prima cosa stabilire una *codifica binaria* dei simboli nell'alfabeto del nastro  $\Gamma$  di  $M$ . Questa codifica è una funzione  $C$  che assegna ad ogni simbolo  $a \in \Gamma$  una sequenza di  $k$  cifre binarie, dove  $k$  è un valore scelto in modo tale che ad ogni simbolo corrisponda una codifica diversa. Per esempio, se  $\Gamma$  contiene 4 simboli, allora  $k = 2$ , perché con 2 bit si rappresentano 4 valori diversi. Se  $\Gamma$  contiene 8 simboli, allora  $k = 3$ , e così via.

La TM con alfabeto binario  $B$  che simula  $M$  è definita in questo modo:

$B = "su input w:$

1. Sostituisce  $w = w_1w_2\dots w_n$  con la codifica binaria  $C(w_1)C(w_2)\dots C(w_n)$ , e riporta la testina sul primo simbolo di  $C(w_1)$ .
  2. Scorre il nastro verso destra per leggere  $k$  cifre binarie: in questo modo la macchina stabilisce qual è il simbolo  $a$  presente sul nastro di  $M$ . Va a sinistra di  $k$  celle.
  3. Aggiorna il nastro in accordo con la funzione di transizione di  $M$ :
    - Se  $\delta(r, a) = (s, b, R)$ , scrive la codifica binaria di  $b$  sul nastro.
    - Se  $\delta(r, a) = (s, b, L)$ , scrive la codifica binaria di  $b$  sul nastro e sposta la testina a sinistra di  $2k$  celle.
  4. 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.”
4. (8 punti) Supponiamo che un impianto industriale costituito da  $m$  linee di produzione identiche debba eseguire  $n$  lavori distinti. Ognuno dei lavori può essere svolto da una qualsiasi delle linee di produzione, e richiede un certo tempo per essere completato. Il problema del bilanciamento del carico (LOADBALANCE) chiede di trovare un assegnamento dei lavori alle linee di produzione che permetta di completare tutti i lavori entro un tempo limite  $k$ .

Più precisamente, possiamo rappresentare l'input del problema con una tripla  $\langle m, T, k \rangle$  dove:

- $m$  è il numero di linee di produzione;
- $T[1\dots n]$  è un array di numeri interi positivi dove  $T[j]$  è il tempo di esecuzione del lavoro  $j$ ;
- $k$  è un limite superiore al tempo di completamento di tutti i lavori.

Per risolvere il problema vi si chiede di trovare un array  $A[1\dots n]$  con gli assegnamenti, dove  $A[j] = i$  significa che il lavoro  $j$  è assegnato alla linea di produzione  $i$ . Il tempo di completamento (o makespan) di  $A$  è il tempo massimo di occupazione di una qualsiasi linea di produzione:

$$\text{makespan}(A) = \max_{\substack{1 \leq i \leq m \\ A[j]=i}} \sum T[j]$$

LOAD BALANCE è il problema di trovare un assegnamento con makespan minore o uguale al limite superiore  $k$ :

$$\text{LOADBALANCE} = \{\langle m, T, k \rangle \mid \text{esiste un assegnamento } A \text{ degli } n \text{ lavori su } m \text{ linee di produzione tale che } \text{makespan}(A) \leq k\}$$

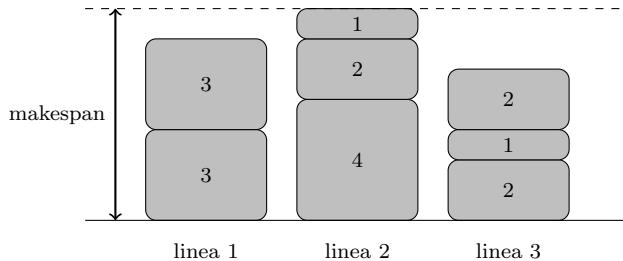


Figura 1: Esempio di assegnamento dei lavori  $T = \{1, 1, 2, 2, 3, 3, 4\}$  su 3 linee con makespan 7.

- Dimostra che LOADBALANCE è un problema NP.
- Dimostra che LOADBALANCE è NP-hard, usando SETPARTITIONING come problema NP-hard di riferimento.

- LOADBALANCE è in NP. L'array  $A$  con gli assegnamenti è il certificato. Il seguente algoritmo è un verificatore per LOADBALANCE:

$V =$  “Su input  $\langle \langle m, T, k \rangle, A \rangle$ :

1. Controlla che  $A$  sia un vettore di  $n$  elementi dove ogni elemento ha un valore compreso tra 1 e  $m$ . Se non lo è, rifiuta.
2. Calcola  $\text{makespan}(A)$ : se è minore o uguale a  $k$  accetta, altrimenti rifiuta.”

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. La prima fase è un controllo sugli  $n$  elementi del vettore  $A$ , e quindi richiede un tempo polinomiale rispetto alla dimensione dell'input. Per calcolare il makespan, la seconda fase deve calcolare il tempo di occupazione di ognuna delle  $m$  linee e poi trovare il massimo tra i tempi di occupazione, operazioni che si possono fare in tempo polinomiale rispetto alla dimensione dell'input.

- (b) Dimostriamo che LOADBALANCE è NP-Hard per riduzione polinomiale da SETPARTITIONING a LOADBALANCE. La funzione di riduzione polinomiale  $f$  prende in input un insieme di numeri interi positivi  $\langle T \rangle$  e produce come output la tripla  $\langle 2, T, k \rangle$  dove  $k$  è uguale alla metà della somma dei valori in  $T$ :

$$k = \frac{1}{2} \sum_{1 \leq i \leq n} T[i]$$

Dimostriamo che la riduzione polinomiale è corretta:

- Se  $\langle T \rangle \in \text{SETPARTITIONING}$ , allora esiste un modo per suddividere  $T$  in due sottoinsiemi  $T_1$  e  $T_2$  in modo tale che la somma dei valori contenuti in  $T_1$  è uguale alla somma dei valori contenuti in  $T_2$ . Nota che questa somma deve essere uguale alla metà della somma dei valori in  $T$ , cioè uguale a  $k$ . Quindi assegnando i lavori contenuti in  $T_1$  alla prima linea di produzione e quelli contenuti in  $T_2$  alla seconda linea di produzione otteniamo una soluzione per LOADBALANCE con makespan uguale a  $k$ , come richiesto dal problema.
- Se  $\langle 2, T, k \rangle \in \text{LOADBALANCE}$ , allora esiste un assegnamento dei lavori alle 2 linee di produzione con makespan minore o uguale a  $k$ . Siccome ci sono solo 2 linee, il makespan di questa soluzione non può essere minore della metà della somma dei valori in  $T$ , cioè di  $k$ . Quindi l'assegnamento ha makespan esattamente uguale a  $k$ , ed entrambe le linee di produzione hanno tempo di occupazione uguale a  $k$ . Quindi, inserendo i lavori assegnati alla prima linea in  $T_1$  e quelli assegnati alla seconda linea in  $T_2$  otteniamo una soluzione per SETPARTITIONING.

La funzione di riduzione deve sommare i valori in  $T$  e dividere per due, operazioni che si possono fare in tempo polinomiale.

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 15 SETTEMBRE 2021

1. (8 punti) Considera il linguaggio

$$L = \{0^m 1^n \mid m > 3n\}.$$

Dimostra che  $L$  non è regolare.

2. (8 punti) Per ogni linguaggio  $L$ , sia  $\text{substring}(L) = \{v \mid uvw \in L \text{ per qualche coppia di stringhe } u, w\}$ . Dimostra che se  $L$  è un linguaggio context-free, allora anche  $\text{substring}(L)$  è un linguaggio context-free.

3. (8 punti) Un *automa a coda* è simile ad un automa a pila con la differenza che la pila viene sostituita da una coda. Una *coda* è un nastro che permette di scrivere solo all'estremità sinistra del nastro e di leggere solo all'estremità destra. Ogni operazione di scrittura (*push*) aggiunge un simbolo all'estremità sinistra della coda e ogni operazione di lettura (*pull*) legge e rimuove un simbolo all'estremità destra. Come per un PDA, l'input è posizionato su un nastro a sola lettura separato, e la testina sul nastro di lettura può muoversi solo da sinistra a destra. Il nastro di input contiene una cella con un blank che segue l'input, in modo da poter rilevare la fine dell'input. Un automa a coda accetta l'input entrando in un particolare stato di accettazione in qualsiasi momento. Mostra che ogni linguaggio Turing-riconoscibile può essere riconosciuto da un automa deterministico a coda.

4. (8 punti) “Colorare” i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate “colori”, ai vertici del grafo in modo tale che nessuna coppia di vertici adiacenti condivida lo stesso colore. Considera la seguente variante del problema 4-COLOR. Oltre al grafo  $G$ , l'input del problema comprende anche un *colore proibito*  $f_v$  per ogni vertice  $v$  del grafo. Per esempio, il vertice 1 non può essere rosso, il vertice 2 non può essere verde, e così via. Il problema che dobbiamo risolvere è stabilire se possiamo colorare il grafo  $G$  con 4 colori in modo che nessun vertice sia colorato con il colore proibito.

$$\text{CONSTRAINED-4-COLOR} = \{\langle G, f_1, \dots, f_n \rangle \mid \text{esiste una colorazione } c_1, \dots, c_n \text{ degli } n \text{ vertici tale che } c_v \neq f_v \text{ per ogni vertice } v\}$$

- (a) Dimostra che CONSTRAINED-4-COLOR è un problema NP.  
(b) Dimostra che CONSTRAINED-4-COLOR è NP-hard, usando  $k$ -COLOR come problema NP-hard di riferimento, per un opportuno valore di  $k$ .

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 23 FEBBRAIO 2022

1. (8 punti) Considera il linguaggio

$$L = \{0^m 1^n \mid m = n^3\}.$$

Dimostra che  $L$  non è regolare.

2. (8 punti) Per ogni linguaggio  $L$  sull'alfabeto  $\Sigma$ , sia  $\text{superstring}(L) = \{xyz \mid y \in L \text{ e } x, z \in \Sigma^*\}$ . Dimostra che se  $L$  è un linguaggio context-free, allora anche  $\text{superstring}(L)$  è un linguaggio context-free.

3. (8 punti) Una *Turing machine con alfabeto ternario* è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è  $\Sigma = \{0, 1, 2\}$  e l'alfabeto del nastro è  $\Gamma = \{0, 1, 2, \_\}$ . Questo significa che la macchina può scrivere sul nastro solo i simboli 0, 1 e blank: non può usare altri simboli né marcare i simboli sul nastro.

Dimostra che ogni linguaggio Turing-riconoscibile sull'alfabeto  $\{0, 1, 2\}$  può essere riconosciuto da una Turing machine con alfabeto ternario.

4. (8 punti) “Colorare” i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate “colori”, ai vertici del grafo in modo tale che nessuna coppia di vertici adiacenti condivide lo stesso colore. Considera la seguente variante del problema 4-COLOR. Oltre al grafo  $G$ , l'input del problema comprende anche un *colore proibito*  $f_v$  per ogni vertice  $v$  del grafo. Per esempio, il vertice 1 non può essere rosso, il vertice 2 non può essere verde, e così via. Il problema che dobbiamo risolvere è stabilire se possiamo colorare il grafo  $G$  con 4 colori in modo che nessun vertice sia colorato con il colore proibito.

$$\text{CONSTRAINED-4-COLOR} = \{\langle G, f_1, \dots, f_n \rangle \mid \begin{array}{l} \text{esiste una colorazione } c_1, \dots, c_n \text{ degli } n \text{ vertici} \\ \text{tale che } c_v \neq f_v \text{ per ogni vertice } v \end{array}\}$$

- (a) Dimostra che CONSTRAINED-4-COLOR è un problema NP.  
(b) Dimostra che CONSTRAINED-4-COLOR è NP-hard, scegliendo un opportuno valore di  $k$  e usando  $k$ -COLOR come problema NP-hard di riferimento.

- 1. (12 punti)** Se  $L$  è un linguaggio e  $a$  un simbolo, allora  $a/L$ , la *derivata* di  $L$  e  $a$ , è l'insieme delle stringhe

$$a/L = \{w \mid aw \in L\}.$$

Per esempio, se  $L = \{a, aab, baa\}$ , allora  $a/L = \{\varepsilon, ab\}$ . Dimostra che se  $L$  è regolare allora anche  $a/L$  è regolare.

**Soluzione:** Se  $L$  è un linguaggio regolare, allora sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Data una parola  $aw \in L$  dove  $w = w_1 \dots w_n$ , la computazione di  $A$  su  $aw$  è una sequenza di stati  $r_0 r_1 \dots r_{n+1}$  tali che:

- $r_0 = q_0$ ;
- $\delta(r_0, a) = r_1$ ;
- $\delta(r_i, w_i) = r_{i+1}$  per ogni  $i = 1, \dots, n$ ;
- $r_{n+1} \in F$ .

Siccome  $A$  è un DFA, lo stato  $r_1$  in cui l'automa si trova è unicamente determinato, e possiamo costruire un automa  $A'$  che accetta il linguaggio  $a/L$  cambiando lo stato iniziale di  $A$  in  $r_1$ . Formalmente,  $A' = (Q, \Sigma, \delta, r_1, F)$  dove  $Q, \Sigma, \delta$  e  $F$  sono gli stessi di  $A$  e lo stato iniziale è  $r_1 = \delta(q_0, a)$ . In questo modo abbiamo che per ogni  $aw \in L$  la sequenza di stati  $r_1 \dots r_{n+1}$  descritta sopra è una computazione di  $A'$  che accetta  $w$ , ed abbiamo dimostrato che se  $aw \in L$  allora  $w \in L(A')$ .

Viceversa, se  $w \in L(A')$  allora esiste una computazione  $s_1 \dots s_{n+1}$  di  $A'$  tale che:

- $s_1 = r_1$ ;
- $\delta(s_i, w_i) = s_{i+1}$  per ogni  $i = 1, \dots, n$ ;
- $s_{n+1} \in F$ .

Di conseguenza, la sequenza di stati  $q_0 s_1 \dots s_{n+1}$  è una computazione di  $A$  su  $aw$ , ed abbiamo dimostrato che se  $w \in L(A')$  allora  $aw \in L$ . Quindi possiamo concludere che il linguaggio di  $A'$  è precisamente  $a/L$ , come richiesto.

- 2. (12 punti)** Considera il linguaggio

$$L_2 = \{w1^n \mid w \text{ è una stringa di 0 e 1 di lunghezza } n\}.$$

Dimostra che  $L_2$  non è regolare.

**Soluzione:** 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  $w = 0^k 1^k$ , 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 sequenza iniziale di 0. Inoltre, siccome  $y \neq \emptyset$ , abbiamo che  $x = 0^q$  e  $y = 0^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 0^{k-q-p} 1^k$ . Consideriamo l'esponente  $i = 2$ : la parola  $xy^2z$  ha la forma

$$xy^2z = 0^q 0^{2p} 0^{k-q-p} 1^k = 0^{k+p} 1^k$$

Poiché  $p > 0$ , la sequenza iniziale di 0 è più lunga della sequenza finale di 1, e quindi la parola iterata  $xy^2z$  non può essere scritta nella forma  $w1^n$  con  $n = |w|$  perché non contiene abbastanza 1.

Abbiamo trovato un assurdo quindi  $L_2$  non può essere regolare.

per questo esercizio scegliere l'esponente  $i = 0$  non è corretto, perché se  $p$  è pari allora la parola  $xy^0z$  appartiene al linguaggio  $L_2$ , in quanto può essere scritta come  $0^{k-p} 1^k = 0^{k-p} 1^{p/2} 1^{k-p/2} = w1^{k-p/2}$  con  $w = 0^{k-p} 1^{p/2}$  parola di lunghezza  $k - p/2$ .

**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  $A \rightarrow wB$  o  $A \rightarrow w$ , 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  $\varepsilon$ -NFA che simula le derivazioni della grammatica.

**Soluzione:** Data una grammatica lineare a destra  $G = (V, \Sigma, R, S)$ , possiamo notare che le derivazioni di  $G$  hanno una struttura particolare. Siccome le regole sono nella forma  $A \rightarrow wB$  o  $A \rightarrow w$ , dove  $A$  e  $B$  sono variabili e  $w$  è una stringa di zero o più simboli terminali, ogni derivazione di una parola della grammatica sarà formata da  $n$  applicazioni di una regola del tipo  $A \rightarrow wB$  seguita da un'applicazione finale di una regola  $A \rightarrow w$ . A partire da questa osservazione possiamo costruire un  $\varepsilon$ -NFA  $A = (Q, \Sigma, \delta, q_0, F)$  che simula le derivazioni della grammatica. Per rendere più chiara la costruzione usiamo una notazione abbreviata per la funzione di transizione, che ci fornisce un modo per far consumare un'intera stringa all'automa in un singolo passo. Possiamo simulare queste transizioni estese introducendo stati aggiuntivi per consumare la stringa un simbolo alla volta. Siano  $p$  e  $q$  stati dell' $\varepsilon$ -NFA e sia  $w = w_1 \dots w_n$  una stringa sull'alfabeto  $\Sigma$ . Per fare in modo che l'automa vada da  $p$  a  $q$  consumando l'intera stringa  $w$  introducendo nuovi stati intermedi  $r_1 \dots r_{n-1}$  e definendo la funzione di transizione come segue:

$$\begin{aligned}\delta(p, w_1) &\text{ contiene } r_1, \\ \delta(r_1, w_2) &= \{r_2\}, \\ &\dots \\ \delta(r_{n-1}, w_n) &= \{q\}.\end{aligned}$$

Useremo la notazione  $q \in \delta(p, w)$  per denotare quando l'automa può andare da  $p$  a  $q$  consumando la stringa di input  $w$ . Gli stati dell'automa sono  $Q = V \cup \{q_f\} \cup E$  dove  $V$  è l'insieme delle variabili della grammatica,  $q_f$  è l'unico stato finale dell'automa e  $E$  è l'insieme di stati necessari per realizzare le transizioni estese appena descritte. Lo stato iniziale è  $S$ , variabile iniziale della grammatica. La funzione di transizione è definita come segue:

- $B \in \delta(A, w)$  per ogni regola  $A \rightarrow wB$  della grammatica;
- $q_f \in \delta(A, w)$  per ogni regola  $A \rightarrow w$  della grammatica.

L'automa  $A$  che abbiamo costruito è in grado di riconoscere lo stesso linguaggio della grammatica  $G$  perché simula le derivazioni di  $G$  nel modo descritto di seguito. L'automa inizia la computazione dallo stato che corrisponde alla variabile iniziale di  $G$ , e ripete i seguenti passi:

- se lo stato corrente corrisponde alla variabile  $A$ , sceglie non deterministicamente una delle regole per  $A$ ;
- se la regola scelta è  $A \rightarrow wB$ , prova a consumare la stringa  $w$  dall'input. Se ci riesce va nello stato  $B$ , altrimenti la computazione si blocca su questo ramo;
- se la regola scelta è  $A \rightarrow w$ , prova a consumare la stringa  $w$  dall'input. Se ci riesce va nello stato finale  $q_f$  e accetta se ha consumato tutto l'input. La computazione si blocca su questo ramo se l'automa non riesce a consumare  $w$  o se non ha consumato tutto l'input quando arriva in  $q_f$ .

- 1. (12 punti)** Se  $L$  è un linguaggio e  $a$  un simbolo, allora  $L/a$ , il *quoziante* di  $L$  e  $a$ , è l'insieme delle stringhe

$$L/a = \{w \mid wa \in L\}.$$

Per esempio, se  $L = \{a, aab, baa\}$ , allora  $L/a = \{\varepsilon, ba\}$ . Dimostra che se  $L$  è regolare allora anche  $L/a$  è regolare.

**Soluzione:** Se  $L$  è un linguaggio regolare, allora sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Data una parola  $wa \in L$  dove  $w = w_1 \dots w_n$ , la computazione di  $A$  su  $aw$  è una sequenza di stati  $r_0 r_1 \dots r_{n+1}$  tali che:

- $r_0 = q_0$ ;
- $\delta(r_{i-1}, w_i) = r_i$  per ogni  $i = 1, \dots, n$ ;
- $\delta(r_n, a) = r_{n+1}$ ;
- $r_{n+1} \in F$ .

Data questa osservazione possiamo costruire un automa  $A'$  che accetta il linguaggio  $L/a$  cambiando gli stati finali di  $A$ . Formalmente,  $A' = (Q, \Sigma, \delta, q_0, F')$  dove  $Q, \Sigma, \delta$  e  $q_0$  sono gli stessi di  $A$  e l'insieme degli stati finali contiene tutti gli stati che raggiungono uno stato finale di  $A$  dopo aver consumato  $a$ :

$$F' = \{q \mid \delta(q, a) \in F\}.$$

In questo modo abbiamo che per ogni  $wa \in L$  la sequenza di stati  $r_0 \dots r_n$  descritta sopra è una computazione di  $A'$  che accetta  $w$  (perché  $r_n$  diventa uno stato finale di  $A'$ ), ed abbiamo dimostrato che se  $wa \in L$  allora  $w \in L(A')$ . Viceversa, se  $w \in L(A')$  allora esiste una computazione  $s_0 \dots s_n$  di  $A'$  tale che:

- $s_0 = q_0$ ;
- $\delta(s_{i-1}, w_i) = s_i$  per ogni  $i = 1, \dots, n$ ;
- $s_n \in F'$ .

Di conseguenza,  $s_{n+1} = \delta(s_n, a) \in F$  e la sequenza di stati  $s_1 \dots s_{n+1}$  è una computazione di  $A$  su  $wa$ , ed abbiamo dimostrato che se  $w \in L(A')$  allora  $wa \in L$ . Quindi possiamo concludere che il linguaggio di  $A'$  è precisamente  $L/a$ , come richiesto.

- 2. (12 punti)** Se  $w$  è una stringa di 0 e 1, allora  $\bar{w}$  è una stringa formata da  $w$  sostituendo gli 0 con 1 e viceversa; per esempio  $\overline{011} = 100$ . Considera il linguaggio

$$L_2 = \{w\bar{w} \mid w \in \{0, 1\}^*\}.$$

Dimostra che  $L_2$  non è regolare.

**Soluzione:** Prima di procedere con la soluzione ci è utile osservare che data una qualsiasi parola  $w$ , la parola  $\bar{w}$  avrà sempre un numero di 0 uguale al numero di 1 di  $w$ , ed un numero di 1 uguale al numero di 0 di  $w$ . Di conseguenza, ogni parola nella forma  $w\bar{w}$  avrà un numero di 0 uguale al numero di 1.

Ora possiamo usare 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  $w = 0^k 1^k$ , che appartiene ad  $L_2$  perché  $\overline{0^k} = 1^k$ , 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 sequenza iniziale di 0. Inoltre, siccome  $y \neq \emptyset$ , abbiamo che  $x = 0^q$  e  $y = 0^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 0^{k-q-p} 1^k$ . Consideriamo l'esponente  $i = 2$ : la parola  $xy^2z$  ha la forma

$$xy^2z = 0^q 0^{2p} 0^{k-q-p} 1^k = 0^{k+p} 1^k$$

Poiché  $p > 0$ , la parola iterata  $xy^2z$  contiene più 0 che 1 e di conseguenza non può essere scritta nella forma  $w\bar{w}$ .

Abbiamo trovato un assurdo quindi  $L_2$  non può essere regolare.

**Nota:** per questo esercizio scegliere qualsiasi esponente  $i \neq 1$  permette di arrivare all'assurdo.

3. (12 punti) Dimostra che il linguaggio  $L_2$  dell'esercizio precedente non è nemmeno un linguaggio context-free.

**Soluzione:** Possiamo usare il Pumping Lemma per linguaggi Context-Free per dimostrare che il linguaggio non è context-free. Supponiamo per assurdo che lo sia.

- Sia  $k$  la lunghezza data dal Pumping Lemma.
- Questa volta scegliere la parola è meno ovvio. Osserviamo per prima cosa che la parola  $0^k 1^k$  si può iterare, dividendola come segue, e perciò non è adatta al nostro scopo:

$$\overbrace{000 \dots 000}^{0^k} \underbrace{0}_{v} \underbrace{\varepsilon}_{x} \overbrace{1 \underbrace{111 \dots 111}_{1^k}}^{1^k}$$

Anche la parola  $0^k 1^k 1^k 0^k$  si può iterare, suddividendola in modo simile:

$$\overbrace{000 \dots 000}^{0^k} \underbrace{0}_{v} \underbrace{\varepsilon}_{x} \overbrace{1 \underbrace{111 \dots 111}_{1^k} \underbrace{000 \dots 000}_{0^k}}^{1^k 1^k 0^k}$$

Mostriamo invece che la parola  $w = 0^k 10^k 1^k 01^k$  non può essere iterata;

- sia  $w = uvxyz$  una suddivisione di  $w$  tale che  $|vy| > 0$  e  $|vxy| \leq k$ ;
- mostriamo che la sottostringa  $vxy$  deve stare a cavallo del punto centrale di  $w$ . Altrimenti, se  $vxy$  è inclusa nella prima metà della stringa, la stringa  $uv^2xy^2z$  sposta uno 0 nella prima posizione della seconda metà e quindi essa non può essere nella forma  $w\bar{w}$ . Viceversa, se  $vxy$  è inclusa nella seconda metà di  $w$ , la stringa  $uv^2xy^2z$  sposta un 1 nell'ultima posizione della prima metà e quindi essa non può essere nella forma  $w\bar{w}$ .

Ma se la sottostringa  $vxy$  è a cavallo del punto centrale di  $w$ , la stringa  $uv^0xy^0z = uxz$  ha la forma  $0^k 10^i 1^j 01^k$  dove  $i$  e  $j$  non possono essere entrambi  $k$ , e quindi non può essere nella forma  $w\bar{w}$ . Quindi  $w$  non può essere iterata.

Abbiamo trovato un assurdo quindi  $L_2$  non può essere context-free.

- 1. (12 punti)** Una *Macchina di Turing con eliminazione* è una macchina di Turing deterministica a nastro singolo che può eliminare celle dal nastro. Formalmente la funzione di transizione è definita come

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, D\}$$

dove  $L, R$  indicano i normali spostamenti a sinistra e a destra della testina, e  $D$  indica l'eliminazione della cella sotto la posizione corrente della testina. Dopo una operazione di eliminazione, la testina si muove nella cella che si trovava immediatamente a destra della cella eliminata.

Dimostra che qualsiasi macchina di Turing con eliminazione può essere simulata da una macchina di Turing deterministica a nastro singolo.

**Soluzione.** Mostriamo come convertire una macchina di Turing con Inserimento  $M$  in una TM deterministica a nastro singolo  $S$  equivalente. La simulazione usa il simbolo speciale  $\#$  per segnare le celle che vengono eliminate.

$S$  = “Su input  $w$ :

1. La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, L)$  procede come nella TM standard:  $S$  scrive  $b$  sul nastro e muove la testina di una cella a sinistra. Se lo spostamento a sinistra porta la testina sopra un  $\#$ , allora continua a spostare la testina a sinistra finché non si arriva in una cella che contiene un simbolo diverso dal  $\#$ .
2. La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, R)$  procede come nella TM standard:  $S$  scrive  $b$  sul nastro e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra un  $\#$ , allora continua a spostare la testina a destra finché non si arriva in una cella che contiene un simbolo diverso dal  $\#$ .
3. Per simulare una mossa del tipo  $\delta(q, a) = (r, b, D)$  la TM  $S$  scrive  $\#$  nella cella corrente e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra un  $\#$ , allora continua a spostare la testina a destra finché non si arriva in una cella che contiene un simbolo diverso dal  $\#$ .
4. 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.”

- 2. (12 punti)** Data una Turing Machine  $M$ , definiamo

$$\text{HALTS}(M) = \{w \mid M \text{ termina la computazione su } w\}.$$

Considera il linguaggio

$$I = \{\langle M \rangle \mid \text{HALTS}(M) \text{ è un insieme infinito}\}.$$

Dimostra che  $I$  è indecidibile.

**Soluzione.** La seguente macchina  $F$  calcola una riduzione mediante funzione  $A_{TM} \leq_m I$ :

$F$  = “su input  $\langle M, w \rangle$ , dove  $M$  è una TM e  $w$  una stringa:

1. Costruisci la seguente macchina  $M'$ :

$M'$  = “Su input  $x$ :

1. Esegue  $M$  su input  $w$ .
  2. Se  $M$  accetta, *accetta*.
  3. Se  $M$  rifiuta, va in loop.”
2. Ritorna  $\langle M' \rangle$ ”

Mostriamo che  $F$  calcola una funzione di riduzione  $f$  da  $A_{TM}$  a  $I$ , cioè una funzione tale che

$$\langle M, w \rangle \in A_{TM} \text{ se e solo se } M' \in I.$$

- Se  $\langle M, w \rangle \in A_{TM}$  allora la macchina  $M$  accetta  $w$ . In questo caso la macchina  $M'$  accetta tutte le parole, quindi  $\text{HALTS}(M) = \Sigma^*$  che è un insieme infinito. Di conseguenza  $M' \in I$ .
- Viceversa, se  $\langle M, w \rangle \notin A_{TM}$ , allora la macchina  $M$  su input  $w$  rifiuta oppure va in loop. In entrambi i casi la macchina  $M'$  va in loop su tutte le stringhe, quindi  $\text{HALTS}(M) = \emptyset$  che è un insieme finito. Di conseguenza  $M' \notin I$ .

- 3. (12 punti)** Una 3-colorazione di un grafo non orientato  $G$  è una funzione che assegna a ciascun vertice di  $G$  un “colore” preso dall’insieme  $\{1, 2, 3\}$ , in modo tale che per qualsiasi arco  $\{u, v\}$  i colori associati ai vertici  $u$  e  $v$  sono diversi. Una 3-colorazione è *sbilanciata* se esiste un colore che colora più di metà dei vertici del grafo.

UNBALANCED-3-COLOR è il problema di trovare una 3-colorazione sbilanciata:

$$\text{UNBALANCED-3-COLOR} = \{\langle G \rangle \mid G \text{ è un grafo che ammette una 3-colorazione sbilanciata}\}$$

- Dimostra che UNBALANCED-3-COLOR è un problema NP
- Dimostra che UNBALANCED-3-COLOR è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

### Soluzione.

- Il certificato è un vettore  $c$  dove ogni elemento  $c[i]$  è il colore assegnato al vertice  $i$ . Il seguente algoritmo è un verificatore  $V$  per UNBALANCED-3-COLOR:

$V$  = “Su input  $\langle \langle G \rangle, c \rangle$ :

- Controlla se  $c$  è un vettore di  $n$  elementi, dove  $n$  è il numero di vertici di  $G$ .
- Controlla se  $c[i] \in \{1, 2, 3\}$  per ogni  $i$ .
- Controlla se per ogni arco  $(i, j)$  di  $G$ ,  $c[i] \neq c[j]$ .
- Controlla se esiste un colore che compare in più di metà degli elementi di  $c$ .
- Se tutte le condizioni sono vere, *accetta*, altrimenti *rifiuta*.”

Ognuno dei passi dell’algoritmo si può eseguire in tempo polinomiale.

- Dimostriamo che UNBALANCED-3-COLOR è NP-Hard per riduzione polinomiale da 3-COLOR a UNBALANCED-3-COLOR. La funzione di riduzione polinomiale  $f$  prende in input un grafo  $\langle G \rangle$  e produce come output un grafo  $\langle G' \rangle$ . Se  $n$  è il numero di vertici di  $G$ ,  $G'$  è costruito aggiungendo  $n + 1$  vertici isolati a  $G$ . Un vertice è isolato se non ci sono archi che lo collegano ad altri vertici.

Dimostriamo che la riduzione è corretta:

- Se  $\langle G \rangle \in 3\text{-COLOR}$ , allora esiste un modo per colorare  $G$  con tre colori 1, 2, 3. Sia  $n$  il numero di vertici di  $G$ . Posso costruire una colorazione sbilanciata per il grafo  $G'$  come segue:
  - i colori dei vertici di  $G'$  che appartengono anche a  $G$  sono colorati come in  $G$ ;
  - i vertici isolati sono colorati tutti con il colore 1.

In questo modo il colore 1 è assegnato ad almeno metà dei vertici di  $G'$  e la colorazione è sbilanciata.

- Se  $\langle G' \rangle \in \text{UNBALANCED-3-COLOR}$ , allora esiste una colorazione sbilanciata di  $G'$ . Se elimino da  $G'$  i vertici isolati aggiunti dalla riduzione ottengo una 3-colorazione di  $G$ .

La funzione di riduzione deve contare i vertici del grafo  $G$  e aggiungere  $n + 1$  vertici al grafo, operazioni che si possono fare in tempo polinomiale.

- 1. (12 punti)** Se  $L$  è un linguaggio sull'alfabeto  $\{0, 1\}$ , la *rotazione a destra* di  $L$  è l'insieme delle stringhe

$$\text{ROR}(L) = \{aw \mid wa \in L, w \in \{0, 1\}^*, a \in \{0, 1\}\}.$$

Per esempio, se  $L = \{0, 001, 10010\}$ , allora  $\text{ROR}(L) = \{0, 100, 01001\}$ . Dimostra che se  $L$  è regolare allora anche  $\text{ROR}(L)$  è regolare.

- 2. (12 punti)** Considera l'alfabeto  $\Sigma = \{0, 1\}$ , e sia  $L_2$  l'insieme di tutte le stringhe che contengono almeno un 1 nella loro seconda metà:

$$L_2 = \{uv \mid u \in \Sigma^*, v \in \Sigma^* 1 \Sigma^* \text{ e } |u| \geq |v|\}.$$

Dimostra che  $L_2$  non è regolare.

- 3. (12 punti)** Mostra che per ogni PDA  $P$  esiste un PDA  $P_2$  con due soli simboli di stack tale che  $L(P_2) = L(P)$ . *Suggerimento:* dai una codifica binaria all'alfabeto di stack di  $P$ .

- 1. (12 punti)** Una *Macchina di Turing con stack di nastri* possiede due azioni aggiuntive che modificano il suo nastro di lavoro, oltre alle normali operazioni di scrittura di singole celle e spostamento a destra o a sinistra della testina: può salvare l'intero nastro inserendolo in uno stack (operazione di Push) e può ripristinare l'intero nastro estraendolo dallo stack (operazione di Pop). Il ripristino del nastro riporta il contenuto di ogni cella al suo contenuto quando il nastro è stato salvato. Il salvataggio e il ripristino del nastro non modificano lo stato della macchina o la posizione della sua testina. Se la macchina tenta di “ripristinare” il nastro quando lo stack è vuoto, la macchina va nello stato di rifiuto. Se ci sono più copie del nastro salvate nello stack, la macchina ripristina l'ultima copia inserita nello stack, che viene quindi rimossa dallo stack.

Mostra che qualsiasi macchina di Turing con stack di nastri può essere simulata da una macchina di Turing standard. *Suggerimento:* usa una macchina multinastro per la simulazione.

- 2. (12 punti)** Considera il linguaggio

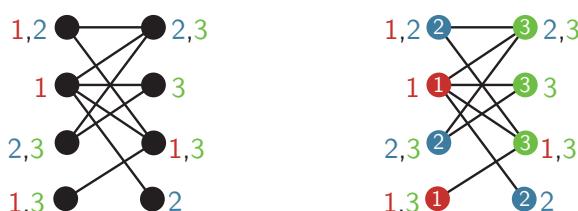
$$\text{FORTY-TWO} = \{\langle M, w \rangle \mid M \text{ termina la computazione su } w \text{ avendo solo 42 sul nastro}\}.$$

Dimostra che FORTY-TWO è indecidibile.

- 3. (12 punti)** “Colorare” i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate “colori”, ai vertici del grafo in modo tale che nessuna coppia di vertici adiacenti condivida lo stesso colore. Considera la seguente variante del problema chiamata LIST-COLORING. Oltre al grafo  $G$ , l'input del problema comprende anche una *lista di colori ammissibili*  $L(v)$  per ogni vertice  $v$  del grafo.

Il problema che dobbiamo risolvere è stabilire se possiamo colorare il grafo  $G$  in modo che ogni vertice  $v$  sia colorato con un colore preso dalla lista di colori ammissibili  $L(v)$ .

$$\text{LIST-COLORING} = \{\langle G, L \rangle \mid \text{esiste una colorazione } c_1, \dots, c_n \text{ degli } n \text{ vertici} \\ \text{tale che } c_v \in L(v) \text{ per ogni vertice } v\}$$



Esempio di istanza (a sinistra) e soluzione (a destra) di LIST-COLORING.

- (a) Dimostra che LIST-COLORING è un problema NP.
- (b) Dimostra che LIST-COLORING è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

AUTOMI E LINGUAGGI FORMALI – 8/7/2022  
SECONDO APPELLO – PRIMA PARTE

- 1. (12 punti)** Se  $L$  è un linguaggio sull'alfabeto  $\{0, 1\}$ , la *rotazione a sinistra* di  $L$  è l'insieme delle stringhe

$$\text{ROL}(L) = \{wa \mid aw \in L, w \in \{0, 1\}^*, a \in \{0, 1\}\}.$$

Per esempio, se  $L = \{0, 01, 010, 10100\}$ , allora  $\text{ROL}(L) = \{0, 10, 100, 01001\}$ . Dimostra che se  $L$  è regolare allora anche  $\text{ROL}(L)$  è regolare.

- 2. (12 punti)** Considera l'alfabeto  $\Sigma = \{0, 1\}$ , e sia  $L_2$  l'insieme di tutte le stringhe che contengono almeno un 1 nella loro prima metà:

$$L_2 = \{uv \mid u \in \Sigma^* 1 \Sigma^*, v \in \Sigma^* \text{ e } |u| \leq |v|\}.$$

Dimostra che  $L_2$  non è regolare.

- 3. (12 punti)** Mostra che per ogni PDA  $P$  esiste un PDA  $P_2$  con due soli stati tale che  $L(P_2) = L(P)$ .  
*Suggerimento:* usate la pila per tenere traccia dello stato di  $P$ .

- 1. (12 punti)** Una *macchina di Turing ad albero binario* usa un albero binario infinito come nastro, dove ogni cella nel nastro ha un figlio sinistro e un figlio destro. Ad ogni transizione, la testina si sposta dalla cella corrente al padre, al figlio sinistro oppure al figlio destro della cella corrente. Pertanto, la funzione di transizione di una tale macchina ha la forma

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{P, L, R\},$$

dove  $P$  indica lo spostamento verso il padre,  $L$  verso il figlio sinistro e  $R$  verso il figlio destro. La stringa di input viene fornita lungo il ramo sinistro dell'albero.

Mostra che qualsiasi macchina di Turing ad albero binario può essere simulata da una macchina di Turing standard.

- 2. (12 punti)** Considera il linguaggio

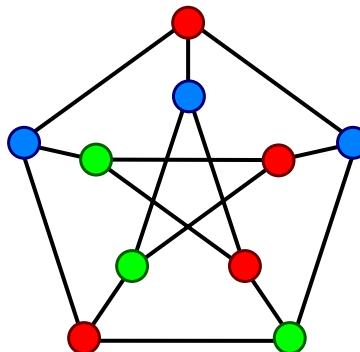
$$\text{EVEN-HALTS} = \{\langle M \rangle \mid \text{per ogni numero naturale } n \text{ pari, } M \text{ termina la computazione su } n\}.$$

Dimostra che EVEN-HALTS è indecidibile.

- 3. (12 punti)** Una 3-colorazione di un grafo non orientato  $G$  è una funzione che assegna a ciascun vertice di  $G$  un “colore” preso dall’insieme  $\{R, G, B\}$ , in modo tale che per qualsiasi arco  $\{u, v\}$  i colori associati ai vertici  $u$  e  $v$  sono diversi. Una 3-colorazione è *equa* se per ogni coppia di colori, il numero di nodi associati a ciascun colore differisce di al più 1.

EQUITABLE-3-COLOR è il problema di trovare una 3-colorazione equa:

$$\text{EQUITABLE-3-COLOR} = \{\langle G \rangle \mid G \text{ è un grafo che ammette una 3-colorazione equa}\}$$



Esempio di colorazione equa con 4 vertici rossi, 3 vertici blu e 3 vertici verdi.

- (a) Dimostra che EQUITABLE-3-COLOR è un problema NP
- (b) Dimostra che EQUITABLE-3-COLOR è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 16 SETTEMBRE 2022

- 1. (9 punti)** Considera il linguaggio

$$L = \{1^n 0^{2^n} \mid n \leq 0\}.$$

Dimostra che  $L$  non è regolare.

- 2. (9 punti)** Chiamiamo  $k$ -PDA un automa a pila dotato di  $k$  pile. In particolare, uno 0-PDA è un NFA e un 1-PDA è un PDA convenzionale. Sappiamo già che gli 1-PDA sono più potenti degli 0-PDA (nel senso che riconoscono una classe più ampia di linguaggi). Mostra che i 2-PDA sono più potenti degli 1-PDA.

- 3. (9 punti)** Una *macchina di Turing ad albero binario* usa un albero binario infinito come nastro, dove ogni cella nel nastro ha un figlio sinistro e un figlio destro. Ad ogni transizione, la testina si sposta dalla cella corrente al padre, al figlio sinistro oppure al figlio destro della cella corrente. Pertanto, la funzione di transizione di una tale macchina ha la forma

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{P, L, R\},$$

dove  $P$  indica lo spostamento verso il padre,  $L$  verso il figlio sinistro e  $R$  verso il figlio destro. La stringa di input viene fornita lungo il ramo sinistro dell'albero.

Mostra che qualsiasi macchina di Turing ad albero binario può essere simulata da una macchina di Turing standard.

- 4. (9 punti)** Un circuito Hamiltoniano in un grafo  $G$  è un ciclo che attraversa ogni vertice di  $G$  esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Considerate il seguente problema, che chiameremo HAM375: dato un grafo  $G$  con  $n$  vertici, trovare un ciclo che attraversa esattamente una volta  $n - 375$  vertici del grafo (ossia tutti i vertici di  $G$  tranne 375).

- Dimostra che HAM375 è un problema NP.
- Dimostra che HAM375 è NP-hard.

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 2 SETTEMBRE 2022

1. (9 punti) Considera il linguaggio

$$L = \{0^m 1^n \mid m > 4n\}.$$

Dimostra che  $L$  non è regolare.

2. (9 punti) Per ogni linguaggio  $L$ , sia  $\text{substring}(L) = \{v \mid uvw \in L \text{ per qualche coppia di stringhe } u, w\}$ . Dimostra che se  $L$  è un linguaggio context-free, allora anche  $\text{substring}(L)$  è un linguaggio context-free.

3. (9 punti) Un *automa a coda* è simile ad un automa a pila con la differenza che la pila viene sostituita da una coda. Una *coda* è un nastro che permette di scrivere solo all'estremità sinistra del nastro e di leggere solo all'estremità destra. Ogni operazione di scrittura (*push*) aggiunge un simbolo all'estremità sinistra della coda e ogni operazione di lettura (*pull*) legge e rimuove un simbolo all'estremità destra. Come per un PDA, l'input è posizionato su un nastro a sola lettura separato, e la testina sul nastro di lettura può muoversi solo da sinistra a destra. Il nastro di input contiene una cella con un blank che segue l'input, in modo da poter rilevare la fine dell'input. Un automa a coda accetta l'input entrando in un particolare stato di accettazione in qualsiasi momento. Mostra che ogni linguaggio Turing-riconoscibile può essere riconosciuto da un automa deterministico a coda.

4. (9 punti) Il problema SETPARTITIONING chiede di stabilire se un multi-insieme<sup>1</sup> di numeri interi  $S$  può essere suddiviso in due sottoinsiemi disgiunti  $S_1$  e  $S_2$  che formano una partizione di  $S$  e tali che la somma dei numeri in  $S_1$  è uguale alla somma dei numeri in  $S_2$ . Sappiamo che questo problema è NP-completo.

BALANCEDSETPARTITIONING è una variante di SETPARTITIONING in cui si chiede che  $S_1$  ed  $S_2$  debbano avere lo stesso numero di elementi, oltre ad avere la stessa somma. Per esempio, dato  $S = \{3, 0, 0, 2, 2, 1\}$ , una soluzione corretta per BALANCEDSETPARTITIONING è data dai due multi-insiemi  $S_1 = \{0, 2, 2\}$  e  $S_2 = \{0, 1, 3\}$ . Entrambi gli insiemi contengono 3 elementi, sommano a 5, e sono una partizione di  $S$ .

- (a) Dimostra che BALANCEDSETPARTITIONING è un problema NP.  
(b) Dimostra che BALANCEDSETPARTITIONING è NP-hard, usando SETPARTITIONING come problema NP-hard di riferimento.

---

<sup>1</sup>Un multi-insieme è un insieme dove lo stesso elemento può comparire più volte, come ad esempio  $\{3, 1, 1, 2, 2, 1\}$ .

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 2 SETTEMBRE 2022

1. (9 punti) Considera il linguaggio

$$L = \{0^m 1^n \mid m > 4n\}.$$

Dimostra che  $L$  non è regolare.

2. (9 punti) Per ogni linguaggio  $L$ , sia  $\text{substring}(L) = \{v \mid uvw \in L \text{ per qualche coppia di stringhe } u, w\}$ . Dimostra che se  $L$  è un linguaggio context-free, allora anche  $\text{substring}(L)$  è un linguaggio context-free.

3. (9 punti) Un *automa a coda* è simile ad un automa a pila con la differenza che la pila viene sostituita da una coda. Una *coda* è un nastro che permette di scrivere solo all'estremità sinistra del nastro e di leggere solo all'estremità destra. Ogni operazione di scrittura (*push*) aggiunge un simbolo all'estremità sinistra della coda e ogni operazione di lettura (*pull*) legge e rimuove un simbolo all'estremità destra. Come per un PDA, l'input è posizionato su un nastro a sola lettura separato, e la testina sul nastro di lettura può muoversi solo da sinistra a destra. Il nastro di input contiene una cella con un blank che segue l'input, in modo da poter rilevare la fine dell'input. Un automa a coda accetta l'input entrando in un particolare stato di accettazione in qualsiasi momento. Mostra che ogni linguaggio Turing-riconoscibile può essere riconosciuto da un automa deterministico a coda.

4. (9 punti) Il problema SETPARTITIONING chiede di stabilire se un multi-insieme<sup>1</sup> di numeri interi  $S$  può essere suddiviso in due sottoinsiemi disgiunti  $S_1$  e  $S_2$  che formano una partizione di  $S$  e tali che la somma dei numeri in  $S_1$  è uguale alla somma dei numeri in  $S_2$ . Sappiamo che questo problema è NP-completo.

BALANCEDSETPARTITIONING è una variante di SETPARTITIONING in cui si chiede che  $S_1$  ed  $S_2$  debbano avere lo stesso numero di elementi, oltre ad avere la stessa somma. Per esempio, dato  $S = \{3, 0, 0, 2, 2, 1\}$ , una soluzione corretta per BALANCEDSETPARTITIONING è data dai due multi-insiemi  $S_1 = \{0, 2, 2\}$  e  $S_2 = \{0, 1, 3\}$ . Entrambi gli insiemi contengono 3 elementi, sommano a 5, e sono una partizione di  $S$ .

- (a) Dimostra che BALANCEDSETPARTITIONING è un problema NP.  
(b) Dimostra che BALANCEDSETPARTITIONING è NP-hard, usando SETPARTITIONING come problema NP-hard di riferimento.

---

<sup>1</sup>Un multi-insieme è un insieme dove lo stesso elemento può comparire più volte, come ad esempio  $\{3, 1, 1, 2, 2, 1\}$ .

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 9 FEBBRAIO 2023

1. (9 punti) Considera il linguaggio

$$L = \{0^{n+m}1^m \mid m, n \geq 0\}.$$

Dimostra che  $L$  non è regolare.

2. (9 punti) Una CFG è detta *lineare a sinistra* se il corpo di ogni regola ha al massimo una variabile, e la variabile si trova all'estremità di sinistra. In altre parole, tutte le regole di una grammatica lineare a sinistra sono nella forma  $A \rightarrow Bw$  o  $A \rightarrow w$ , dove  $A$  e  $B$  sono variabili e  $w$  è una stringa di zero o più simboli terminali.

Dimostra che ogni grammatica lineare a sinistra genera un linguaggio regolare.

3. (9 punti) Un *automa a coda* è simile ad un automa a pila con la differenza che la pila viene sostituita da una coda. Una *coda* è un nastro che permette di scrivere solo all'estremità sinistra del nastro e di leggere solo all'estremità destra. Ogni operazione di scrittura (*push*) aggiunge un simbolo all'estremità sinistra della coda e ogni operazione di lettura (*pull*) legge e rimuove un simbolo all'estremità destra. Come per un PDA, l'input è posizionato su un nastro a sola lettura separato, e la testina sul nastro di lettura può muoversi solo da sinistra a destra. Il nastro di input contiene una cella con un blank che segue l'input, in modo da poter rilevare la fine dell'input. Un automa a coda accetta l'input entrando in un particolare stato di accettazione in qualsiasi momento. Mostra che ogni linguaggio Turing-riconoscibile può essere riconosciuto da un automa deterministico a coda.

4. (9 punti) Un circuito Hamiltoniano in un grafo  $G$  è un ciclo che attraversa ogni vertice di  $G$  esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Un circuito semi-Hamiltoniano in un grafo  $G$  è un ciclo che attraversa tutti i vertici di  $G$ , passando per ogni vertice *non più di due volte* (può passare una sola volta per un vertice, o due volte, ma non tre volte o più).

- (a) Dimostra che il problema del circuito semi-Hamiltoniano è un problema NP.  
(b) Dimostra che il problema del circuito semi-Hamiltoniano è NP-hard.

- 1. (12 punti)** Se  $L$  è un linguaggio e  $a$  un simbolo, allora  $L/a$ , il *quoziente* di  $L$  e  $a$ , è l'insieme delle stringhe

$$L/a = \{w \mid wa \in L\}.$$

Per esempio, se  $L = \{0, 001, 100\}$ , allora  $L/0 = \{\varepsilon, 10\}$ . Dimostra che se  $L$  è regolare allora anche  $L/a$  è regolare.

**Soluzione:** Se  $L$  è un linguaggio regolare, allora sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Data una parola  $wa \in L$  dove  $w = w_1 \dots w_n$ , la computazione di  $A$  su  $aw$  è una sequenza di stati  $r_0 r_1 \dots r_{n+1}$  tali che:

- $r_0 = q_0$ ;
- $\delta(r_{i-1}, w_i) = r_i$  per ogni  $i = 1, \dots, n$ ;
- $\delta(r_n, a) = r_{n+1}$ ;
- $r_{n+1} \in F$ .

Data questa osservazione possiamo costruire un automa  $A'$  che accetta il linguaggio  $L/a$  cambiando gli stati finali di  $A$ . Formalmente,  $A' = (Q, \Sigma, \delta, q_0, F')$  dove  $Q, \Sigma, \delta$  e  $q_0$  sono gli stessi di  $A$  e l'insieme degli stati finali contiene tutti gli stati che raggiungono uno stato finale di  $A$  dopo aver consumato  $a$ :

$$F' = \{q \mid \delta(q, a) \in F\}.$$

In questo modo abbiamo che per ogni  $wa \in L$  la sequenza di stati  $r_0 \dots r_n$  descritta sopra è una computazione di  $A'$  che accetta  $w$  (perché  $r_n$  diventa uno stato finale di  $A'$ ), ed abbiamo dimostrato che se  $wa \in L$  allora  $w \in L(A')$ . Viceversa, se  $w \in L(A')$  allora esiste una computazione  $s_0 \dots s_n$  di  $A'$  tale che:

- $s_0 = q_0$ ;
- $\delta(s_{i-1}, w_i) = s_i$  per ogni  $i = 1, \dots, n$ ;
- $s_n \in F'$ .

Di conseguenza,  $s_{n+1} = \delta(s_n, a) \in F$  e la sequenza di stati  $s_1 \dots s_{n+1}$  è una computazione di  $A$  su  $wa$ , ed abbiamo dimostrato che se  $w \in L(A')$  allora  $wa \in L$ . Quindi possiamo concludere che il linguaggio di  $A'$  è precisamente  $L/a$ , come richiesto.

- 2. (12 punti)** Considera il linguaggio

$$L_2 = \{1^n w \mid w \text{ è una stringa di } 0 \text{ e } 1 \text{ di lunghezza } n\}.$$

Dimostra che  $L_2$  non è regolare.

**Soluzione:** 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  $w = 1^k 0^k$ , 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 sequenza iniziale di 1. Inoltre, siccome  $y \neq \varepsilon$ , abbiamo che  $x = 1^q$  e  $y = 1^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 1^{k-q-p} 0^k$ . Consideriamo l'esponente  $i = 0$ : la parola  $xy^0 z$  ha la forma

$$xy^0 z = xz = 1^q 1^{k-q-p} 0^k = 1^{k-p} 0^k$$

Poiché  $p > 0$ , la sequenza iniziale di 1 è più corta della sequenza finale di 0, e quindi la parola iterata  $xy^0 z$  non può essere scritta nella forma  $1^n w$  con  $n = |w|$  perché non contiene abbastanza 1 nella parte iniziale.

Abbiamo trovato un assurdo quindi  $L_2$  non può essere regolare.

**Nota:** per questo esercizio scegliere un esponente  $i > 1$  non è corretto, perché se  $p$  è pari allora la parola  $xy^iz$  appartiene al linguaggio  $L_2$ , in quanto può essere scritta come  $1^{k+ip}0^k = 1^k1^{ip/2}1^{ip/2}0^k = 1^{k+ip/2}w$  con  $w = 1^{ip/2}0^k$  parola di lunghezza  $k + ip/2$ .

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  $A \rightarrow wB$  o  $A \rightarrow w$ , 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  $\varepsilon$ -NFA che simula le derivazioni della grammatica.

**Soluzione:** Data una grammatica lineare a destra  $G = (V, \Sigma, R, S)$ , possiamo notare che le derivazioni di  $G$  hanno una struttura particolare. Siccome le regole sono nella forma  $A \rightarrow wB$  o  $A \rightarrow w$ , dove  $A$  e  $B$  sono variabili e  $w$  è una stringa di zero o più simboli terminali, ogni derivazione di una parola della grammatica sarà formata da  $n$  applicazioni di una regola del tipo  $A \rightarrow wB$  seguita da un'applicazione finale di una regola  $A \rightarrow w$ . A partire da questa osservazione possiamo costruire un  $\varepsilon$ -NFA  $A = (Q, \Sigma, \delta, q_0, F)$  che simula le derivazioni della grammatica. Per rendere più chiara la costruzione usiamo una notazione abbreviata per la funzione di transizione, che ci fornisce un modo per far consumare un'intera stringa all'automa in un singolo passo. Possiamo simulare queste transizioni estese introducendo stati aggiuntivi per consumare la stringa un simbolo alla volta. Siano  $p$  e  $q$  stati dell' $\varepsilon$ -NFA e sia  $w = w_1 \dots w_n$  una stringa sull'alfabeto  $\Sigma$ . Per fare in modo che l'automa vada da  $p$  a  $q$  consumando l'intera stringa  $w$  introducendo nuovi stati intermedi  $r_1 \dots r_{n-1}$  e definendo la funzione di transizione come segue:

$$\begin{aligned}\delta(p, w_1) &\text{ contiene } r_1, \\ \delta(r_1, w_2) &= \{r_2\}, \\ &\dots \\ \delta(r_{n-1}, w_n) &= \{q\}.\end{aligned}$$

Useremo la notazione  $q \in \delta(p, w)$  per denotare quando l'automa può andare da  $p$  a  $q$  consumando la stringa di input  $w$ . Gli stati dell'automa sono  $Q = V \cup \{q_f\} \cup E$  dove  $V$  è l'insieme delle variabili della grammatica,  $q_f$  è l'unico stato finale dell'automa e  $E$  è l'insieme di stati necessari per realizzare le transizioni estese appena descritte. Lo stato iniziale è  $S$ , variabile iniziale della grammatica. La funzione di transizione è definita come segue:

- $B \in \delta(A, w)$  per ogni regola  $A \rightarrow wB$  della grammatica;
- $q_f \in \delta(A, w)$  per ogni regola  $A \rightarrow w$  della grammatica.

L'automa  $A$  che abbiamo costruito è in grado di riconoscere lo stesso linguaggio della grammatica  $G$  perché simula le derivazioni di  $G$  nel modo descritto di seguito. L'automa inizia la computazione dallo stato che corrisponde alla variabile iniziale di  $G$ , e ripete i seguenti passi:

- se lo stato corrente corrisponde alla variabile  $A$ , sceglie non deterministicamente una delle regole per  $A$ ;
- se la regola scelta è  $A \rightarrow wB$ , prova a consumare la stringa  $w$  dall'input. Se ci riesce va nello stato  $B$ , altrimenti la computazione si blocca su questo ramo;
- se la regola scelta è  $A \rightarrow w$ , prova a consumare la stringa  $w$  dall'input. Se ci riesce va nello stato finale  $q_f$  e accetta se ha consumato tutto l'input. La computazione si blocca su questo ramo se l'automa non riesce a consumare  $w$  o se non ha consumato tutto l'input quando arriva in  $q_f$ .

- 1. (12 punti)** Data una parola  $w \in \Sigma^*$ , definiamo  $\text{evens}(w)$  come la sottosequenza di  $w$  che contiene solo i simboli in posizione pari (iniziano a contare da 1). Per esempio,  $\text{evens}(INDICEPARI) = NIEAI$ . Dimostra che se  $L \subseteq \Sigma^*$  è un linguaggio regolare allora anche il linguaggio

$$\text{evens}(L) = \{\text{evens}(w) \mid w \in L\}$$

è un linguaggio regolare.

**Soluzione:** Se  $L$  è un linguaggio regolare, allora sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Costruiamo un  $\varepsilon$ -NFA  $A'$  che accetta il linguaggio  $\text{evens}(L)$ , aggiungendo un flag 0, 1 agli stati di  $A$ , dove flag 0 corrisponde a “simbolo in posizione pari” e flag 1 corrisponde a “simbolo in posizione dispari”. La funzione di transizione è fatta in modo da alternare i flag 0 e 1 dopo ogni transizione dell'automa. Se lo stato corrente ha flag 0, l'automa consuma il prossimo simbolo della stringa di input e prosegue nello stesso stato che raggiungerebbe  $A$  dopo aver consumato lo stesso simbolo. Se lo stato corrente ha flag 1, l'automa prosegue con una  $\varepsilon$ -transizione verso uno degli stati raggiungibili dall'automa  $A$  consumando un simbolo: simulando in questo modo il fatto che i simboli in posizione dispari vanno “saltati”.

Formalmente,  $A' = (Q', \Sigma, \delta', q'_0, F')$  è definito come segue.

- $Q' = Q \times \{0, 1\}$ .
- L'alfabeto  $\Sigma$  rimane lo stesso.
- $\delta'((q, 0), a) = \{(\delta(q, a), 1)\}$ . Con il flag 0 l'automa consuma un simbolo di input ed ha una sola alternativa possibile: lo stato  $\delta(q, a)$  raggiunto da  $A$  dopo aver consumato  $a$ . Il flag diventa 1.
- $\delta'((q, 1), \varepsilon) = \{(q', 0) \mid \text{esiste } a \in \Sigma \text{ tale che } \delta(q, a) = q'\}$ . Con il flag 1 l'automa procede nondeterministicamente con una  $\varepsilon$ -transizione verso uno degli stati raggiungibili da  $A$  dopo aver consumato un simbolo arbitrario.
- $q'_0 = (q_0, 1)$ . Lo stato iniziale corrisponde allo stato iniziale di  $A$  con flag 1 (simbolo in posizione dispari).
- $F' = F \times \{0, 1\}$ . L'insieme degli stati finali di  $A'$  corrisponde all'insieme degli stati finali di  $A$  con qualsiasi flag.

Per dimostrare che  $A'$  riconosce il linguaggio  $\text{evens}(L)$ , data una parola  $w = w_1 w_2 \dots w_n$ , dobbiamo considerare due casi.

- Se  $w \in L$ , allora esiste una computazione di  $A$  che accetta la parola. Di conseguenza, esiste una computazione di  $A$  che accetta la parola:

$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$

con  $s_0 = q_0$  e  $s_n \in F$ . Se  $w$  è di lunghezza pari, la computazione

$$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_2} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_4} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} (s_n, 1)$$

è una computazione accettante per  $A'$  sulla parola  $\text{evens}(w) = w_2 w_4 w_6 \dots w_n$ , perché  $(s_n, 1) \in F'$ . Se  $w$  è di lunghezza dispari, allora la computazione accettante per  $A'$  su  $\text{evens}(w)$  è

$$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_2} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_4} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_{n-1}} (s_{n-1}, 1) \xrightarrow{\varepsilon} (s_n, 1).$$

- Se  $w$  è accettata dal nuovo automa  $A'$ , allora esiste una computazione accettante che ha la forma

$$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_1} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_2} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} (s_n, 1),$$

se  $(s_n, 1)$  è uno stato finale, oppure la forma

$$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_1} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_2} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} (s_n, 1) \xrightarrow{\varepsilon} (s_{n+1}, 0),$$

quando è necessaria una  $\varepsilon$ -transizione finale per raggiungere uno stato finale. In entrambi i casi possiamo costruire una computazione accettante per  $A$  sostituendo le  $\epsilon$ -transizioni con transizioni che consumano un carattere. Nel primo caso si ottiene una computazione che ha la forma

$$s_0 \xrightarrow{u_1} s_1 \xrightarrow{w_1} s_2 \xrightarrow{u_2} s_3 \xrightarrow{w_2} s_4 \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} s_n,$$

e accetta una parola  $u = u_1 w_1 u_2 w_2 \dots u_n w_n$ . Nel secondo caso si ottiene una computazione

$$s_0 \xrightarrow{u_1} s_1 \xrightarrow{w_1} s_2 \xrightarrow{u_2} s_3 \xrightarrow{w_2} s_4 \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} s_n \xrightarrow{u_{n+1}} s_{n+1},$$

che accetta la parola  $u = u_1 w_1 u_2 w_2 \dots u_n w_n u_{n+1}$ . In entrambi i casi  $\text{evens}(u) = w$ , come richiesto.

**2. (12 punti)** Considera il linguaggio

$$L_2 = \{uwu \mid u, w \text{ sono stringhe di 0 e 1 tali che } |u| = |w|\}.$$

Dimostra che  $L_2$  non è regolare.

**Soluzione:** 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  $w = 1^k 0^k 1^k$ , che è di lunghezza maggiore di  $k$  ed appartiene ad  $L_2$  perché il primo terzo della parola è uguale all'ultimo terzo;
- 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 sequenza iniziale di 1. Inoltre, siccome  $y \neq \varepsilon$ , abbiamo che  $x = 1^q$  e  $y = 1^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 1^{k-q-p} 0^k 1^k$ . Consideriamo l'esponente  $i = 2$ : la parola  $xy^2z$  ha la forma

$$xy^2z = 1^q 1^{2p} 1^{k-q-p} 0^k 1^k = 1^{k+p} 0^k 1^k$$

Poiché  $p > 0$ , la sequenza iniziale di 1 è più lunga della sequenza finale di 1, e quindi la parola iterata  $xy^2z$  non appartiene ad  $L_2$  perché il primo terzo della parola è fatta solamente da 1 mentre l'ultimo terzo include anche un certo numero di 0.

Abbiamo trovato un assurdo quindi  $L_2$  non può essere regolare.

**3. (12 punti)** Dimostra che se  $L \subseteq \Sigma^*$  è un linguaggio context-free allora anche  $L^R$  è un linguaggio context-free, dove  $L^R = \{w^R \in \Sigma^* \mid w \in L \text{ e } w^R \text{ è la stringa } w \text{ rovesciata}\}$ .

**Soluzione:** Se  $L$  è un linguaggio context-free, allora esiste una grammatica  $G = (V, \Sigma, R, S)$  che lo genera. Per dimostrare che  $L^R$  è context-free, dobbiamo essere in grado di definire una grammatica che possa generarlo. Questa grammatica è una quadrupla  $G' = (V', \Sigma', R', S')$  definita come segue.

- L'alfabeto è lo stesso del linguaggio  $L$  originale:  $\Sigma' = \Sigma$ .
- L'insieme di variabili è lo stesso della grammatica  $G$ :  $V' = V$ .
- Il nuovo insieme di regole  $R'$  è ottenuto “rovesciando” la parte destra delle regole di  $R$ :  $R' = \{A \rightarrow u^R \mid A \rightarrow u \in R\}$ . In questo modo qualsiasi derivazione deve ora seguire le regole rovesciate.
- Si noti che mentre la parte destra delle regole deve rovesciata, l'ordine delle regole nella derivazione non deve essere invertito. Pertanto, la variabile iniziale rimane la stessa:  $S' = S$ .

- 1. (12 punti)** Dimostra che se  $L \subseteq \Sigma^*$  è un linguaggio regolare allora anche il linguaggio

$$\text{substring}(L) = \{y \in \Sigma^* \mid xyz \in L \text{ con } x, z \in \Sigma^*\}$$

è un linguaggio regolare.

**Soluzione:** Se  $L$  è un linguaggio regolare, allora sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Costruiamo un  $\varepsilon$ -NFA  $A'$  che accetta il linguaggio  $\text{substring}(L)$ . L'automa  $A'$  è costituito dagli stessi stati di  $A$  a cui viene aggiunto un nuovo stato iniziale  $q'_0$ . La funzione di transizione contiene una  $\varepsilon$ -transizione dal nuovo stato iniziale  $q'_0$  verso tutti gli stati di  $A$  che sono raggiungibili da  $q_0$ , e si comporta come  $A$  per gli altri stati. Gli stati finali dell'automa sono tutti gli stati di  $A$  a partire dai quali esiste una computazione che raggiunge uno stato finale di  $A$ .

Formalmente,  $A' = (Q', \Sigma, \delta', q'_0, F')$  è definito come segue.

- $Q' = Q \cup \{q'_0\}$ , dove  $q'_0$  è uno stato nuovo che non appartiene a  $Q$ .
- L'alfabeto  $\Sigma$  rimane lo stesso.
- $\delta'(q'_0, \varepsilon) = \{q \in Q \mid \text{esiste una computazione da } q_0 \text{ a } q \text{ in } A\}$ .
- $\delta'(q, a) = \{\delta(q, a)\}$  per ogni stato  $q \neq q'_0$  e  $a \in \Sigma$ .
- $F' = \{q \in Q \mid \text{esiste una computazione da } q \text{ ad uno stato di } F\}$ .

Per dimostrare che  $A'$  riconosce il linguaggio  $\text{substring}(L)$ , dobbiamo considerare due casi.

- Se  $w \in L$ , e data una qualsiasi suddivisione  $w = xyz$ , esiste una computazione di  $A$  che accetta la parola. Definiamo  $q_1$  come lo stato raggiunto da  $A$  dopo aver consumato  $y$ , e  $q_2$  come lo stato raggiunto da  $A$  dopo aver consumato  $xy$ . Allora possiamo costruire una computazione di  $A'$  che accetta  $y$  in questo modo:

$$q'_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{y_1} \dots \xrightarrow{y_n} y_2.$$

Siccome  $y_2 \in F'$ , la computazione è accettante per  $A'$ .

- Se  $y$  è accettata dal nuovo automa  $A'$ , allora esiste una computazione accettante che ha la forma

$$q'_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{y_1} \dots \xrightarrow{y_n} y_{n+1},$$

con  $y_{n+1} \in F'$ . Per la definizione della funzione di transizione  $\delta'$ , abbiamo che esiste una computazione da  $q_0$  a  $q_1$  di  $A$ , che consuma una certa parola  $x$ . Per la definizione dell'insieme di stati finali  $F'$ , esiste una computazione di  $A$  che raggiunge uno stato finale di  $A$  a partire da  $y_{n+1}$ , che consuma una certa parola  $z$ . Di conseguenza, la parola  $xyz$  è accettata da  $A$ .

- 2. (12 punti)** Considera il linguaggio

$$L_2 = \{wwu \mid u, w \text{ sono stringhe di 0 e 1 tali che } |u| = |w|\}.$$

Dimostra che  $L_2$  non è regolare.

**Soluzione:** 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  $w = 1^k 1^k 0^k$ , che è di lunghezza maggiore di  $k$  ed appartiene ad  $L_2$  perché il primo terzo della parola è uguale al secondo terzo;
- 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 sequenza iniziale di 1. Inoltre, siccome  $y \neq \varepsilon$ , abbiamo che  $x = 1^q$  e  $y = 1^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 1^{k-q-p}1^k0^k$ . Consideriamo l'esponente  $i = 0$ : la parola  $xy^0z$  ha la forma

$$xy^0z = xz = 1^q1^{k-q-p}1^k0^k = 1^{k-p}1^k0^k$$

Poiché  $p > 0$ , la sequenza iniziale di 1 è più corta di  $2k$ , e quindi la parola iterata  $xy^0z$  non appartiene ad  $L_2$  perché il primo terzo della parola è fatta solamente da 1 mentre il secondo terzo include anche un certo numero di 0.

Abbiamo trovato un assurdo quindi  $L_2$  non può essere regolare.

- 3. (12 punti)** Dimostra che se  $L \subseteq \Sigma^*$  è un linguaggio context-free allora anche il linguaggio

$$\text{censor}(L) = \{\#^{|w|} \mid w \in L\}$$

è un linguaggio context-free.

**Soluzione:** Se  $L$  è un linguaggio context-free, allora esiste una grammatica  $G = (V, \Sigma, R, S)$  che lo genera. Possiamo assumere che questa grammatica sia in forma normale di Chomsky. Per dimostrare che  $\text{censor}(L)$  è context-free, dobbiamo essere in grado di definire una grammatica che possa generarlo. Questa grammatica è una quadrupla  $G' = (V', \Sigma', R', S')$  definita come segue.

- L'alfabeto contiene solo #:  $\Sigma' = \{\#\}$ .
- L'insieme di variabili è lo stesso della grammatica  $G$ :  $V' = V$ .
- Il nuovo insieme di regole  $R'$  è ottenuto rimpiazzando ogni regola nella forma  $A \rightarrow b$ , con  $b$  simbolo terminale, con la regola  $A \rightarrow \#$ , e lasciando invariate le regole nella forma  $A \rightarrow BC$  e la regola  $S \rightarrow \varepsilon$  (se presente).
- La variabile iniziale rimane la stessa:  $S' = S$ .

Data una derivazione  $S \Rightarrow^* w$  della grammatica  $G$  possiamo costruire una derivazione nella nuova grammatica  $G'$  che applica le stesse regole nello stesso ordine, e che deriva una parola dove ogni simbolo terminale di  $w$  è rimpiazzato da #. Quindi,  $G'$  permette di derivare tutte le parole in  $\text{censor}(L)$ . Viceversa, data una derivazione  $S \Rightarrow^* \#^n$  della nuova grammatica  $G'$  possiamo costruire una derivazione nella grammatica  $G$  che applica le stesse regole nello stesso ordine, e che deriva una parola dove ogni # è rimpiazzato da qualche simbolo terminale in  $\Sigma$ . Quindi,  $G'$  permette di derivare solo parole che appartengono a  $\text{censor}(L)$ .

- 1. (12 punti)** Una macchina di Turing “ecologica” (ETM) è uguale a una normale macchina di Turing deterministica a singolo nastro, ma può leggere e scrivere su entrambi i lati di ogni cella del nastro: fronte e retro. La testina di una TM ecologica può spostarsi a sinistra (L), a destra (R) o passare all’altro lato del nastro (F).
- Dai una definizione formale della funzione di transizione di una TM ecologica.
  - Dimostra che le TM ecologiche riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

**Soluzione.**

- $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, F\}$
- Per dimostrare che TM ecologiche riconoscono la classe dei linguaggi Turing-riconoscibili dobbiamo dimostrare due cose: che ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM, e che ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

La prima dimostrazione è banale: le TM deterministiche a singolo nastro sono un caso particolare di ETM che usano solamente il lato di fronte del nastro e non effettuano mai la mossa F per passare dall’altro lato del nastro. Di conseguenza, ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM.

Per dimostrare che ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile, mostriamo come convertire una macchina di Turing ecologica  $M$  in una TM deterministica a due nastri  $S$  equivalente. Il primo nastro di  $S$  rappresenta il lato di fronte del nastro di  $M$ , il secondo nastro rappresenta il lato di retro.

$S =$  “Su input  $w$ :

- $S$  usa lo stato per memorizzare lo stato di  $M$  ed il lato dove si trova la testina di  $M$ . All’inizio il lato corrente è “fronte” e lo stato di  $M$  è quello iniziale.
- Se il lato corrente è “fronte”: leggi il simbolo sotto la testina del primo nastro per stabilire la mossa da fare. Se il lato corrente è “retro”, leggi il simbolo sotto la testina del secondo nastro per stabilire la mossa da fare.
- La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, L)$  scrive  $b$  sul primo nastro se il lato corrente è “fronte”, e scrive  $b$  sul secondo nastro se il nastro corrente è “retro”. Poi sposta entrambe le testine di una cella a sinistra.
- La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, R)$  scrive  $b$  sul primo nastro se il lato corrente è “fronte”, e scrive  $b$  sul secondo nastro se il nastro corrente è “retro”. Poi sposta entrambe le testine di una cella a destra.
- Per simulare una mossa del tipo  $\delta(q, a) = (r, b, F)$  la TM  $S$  scrive  $b$  sul primo nastro se il lato corrente è “fronte”, poi cambia il valore del lato corrente in “retro”. Se il lato corrente è “retro”, scrive  $b$  sul secondo nastro, poi cambia il valore del lato corrente in “fronte”. Sposta entrambe le testine di una cella a destra e poi una cella a sinistra, in modo da ritornare nella cella di partenza.
- $r$  diventa il nuovo stato corrente della simulazione. Se  $r$  è lo stato di accettazione di  $M$ , allora  $S$  termina con accettazione. Se  $r$  è lo stato di rifiuto di  $M$ , allora  $S$  termina con rifiuto. Negli altri casi continua la simulazione dal punto 2.”

Per concludere, siccome sappiamo che le TM multinastro riconoscono i linguaggi Turing-riconoscibili, allora abbiamo dimostrato che ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

**2. (12 punti)** Considera il seguente problema: dato un DFA  $D$  e un'espressione regolare  $R$ , il linguaggio riconosciuto da  $D$  è uguale al linguaggio generato da  $R$ ?

- (a) Formula questo problema come un linguaggio  $EQ_{DFA, REX}$ .
- (b) Dimostra che  $EQ_{DFA, REX}$  è decidibile.

**Soluzione.**

- (a)  $EQ_{DFA, REX} = \{\langle D, R \rangle \mid D \text{ è un DFA, } R \text{ è una espressione regolare e } L(D) = L(R)\}$
- (b) La seguente macchina  $N$  usa la Turing machine  $M$  che decide  $EQ_{DFA}$  per decidere  $EQ_{DFA, REX}$ :

$N$  = “su input  $\langle D, R \rangle$ , dove  $D$  è un DFA e  $R$  una espressione regolare:

1. Converti  $R$  in un DFA equivalente  $D_R$
2. Esegui  $M$  su input  $\langle D, D_R \rangle$ , e ritorna lo stesso risultato di  $M$ .”

Mostriamo che  $N$  è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Sappiamo che esiste un algoritmo per convertire ogni espressione regolare in un  $\epsilon$ -NFA, ed un algoritmo per convertire ogni  $\epsilon$ -NFA in un DFA. Il primo step di  $N$  si implementa eseguendo i due algoritmi in sequenza, e termina sempre perché entrambi gli algoritmi di conversione terminano. Il secondo step termina sempre perché sappiamo che  $EQ_{DFA}$  è un linguaggio decidibile. Quindi  $N$  termina sempre la computazione.

Vediamo ora che  $N$  dà la risposta corretta:

- Se  $\langle D, R \rangle \in EQ_{DFA, REX}$  allora  $L(D) = L(R)$ , e di conseguenza  $L(D) = L(D_R)$  perché  $D_R$  è un DFA equivalente ad  $R$ . Quindi  $\langle D, D_R \rangle \in EQ_{DFA}$ , e l'esecuzione di  $M$  terminerà con accettazione.  $N$  ritorna lo stesso risultato di  $M$ , quindi accetta.
- Viceversa, se  $\langle D, R \rangle \notin EQ_{DFA, REX}$  allora  $L(D) \neq L(R)$ , e di conseguenza  $L(D) \neq L(D_R)$  perché  $D_R$  è un DFA equivalente ad  $R$ . Quindi  $\langle D, D_R \rangle \notin EQ_{DFA}$ , e l'esecuzione di  $M$  terminerà con rifiuto.  $N$  ritorna lo stesso risultato di  $M$ , quindi rifiuta.

**3. (12 point)** Una macchina di Turing  $M$  accetta una stringa unaria se esiste una stringa  $x \in \{1\}^*$  tale che  $M$  accetta  $x$ . Considera il problema di determinare se una TM  $M$  accetta una stringa unaria.

- (a) Formula questo problema come un linguaggio  $UA$ .
- (b) Dimostra che il linguaggio  $UA$  è indecidibile.

**Soluzione.**

- (a)  $UA = \{\langle M \rangle \mid \text{esiste } x \in \{1\}^* \text{ tale che } M \text{ accetta } x\}$
- (b) La seguente macchina  $F$  calcola una riduzione  $A_{TM} \leq_m UA$ :

$F$  = “su input  $\langle M, w \rangle$ , dove  $M$  è una TM e  $w$  una stringa:

1. Costruisci la seguente macchina  $M'$ :

$M'$  = “Su input  $x$ :

1. Esegue  $M$  su input  $w$ .
2. Se  $M$  accetta, accetta.
3. Se  $M$  rifiuta, rifiuta.”

2. Ritorna  $\langle M' \rangle$ .

Mostriamo che  $F$  calcola una funzione di riduzione da  $A_{TM}$  a  $UA$ , cioè una funzione tale che

$$\langle M, w \rangle \in A_{TM} \text{ se e solo se } \langle M' \rangle \in UA.$$

- Se  $\langle M, w \rangle \in A_{TM}$  allora la macchina  $M$  accetta  $w$ . In questo caso la macchina  $M'$  accetta tutte le parole, quindi esiste almeno una parola unaria accettata da  $M'$ , e di conseguenza  $\langle M' \rangle \in UA$ .
- Viceversa, se  $\langle M, w \rangle \notin A_{TM}$  allora la macchina  $M$  su input  $w$  rifiuta o va in loop. Di conseguenza, la macchina  $M'$  rifiuta o va in loop su tutte le stringhe, quindi  $M'$  ha linguaggio vuoto e non esiste una stringa unaria che sia accettata da  $M'$ . Di conseguenza  $\langle M' \rangle \notin UA$ .

- 1. (12 punti)** Una macchina di Turing salva-nastro è simile a una normale macchina di Turing deterministica a nastro singolo semi-infinito, ma può spostare la testina al centro della parte non vuota del nastro. In particolare, se le prime  $s$  celle del nastro non sono vuote, allora la testina può spostarsi nella cella numero  $\lfloor s/2 \rfloor$ . A ogni passo, la testina della TM salva-nastro può spostarsi a sinistra di una cella (L), a destra di una cella (R) o al centro della parte non vuota del nastro (J).
- Dai una definizione formale della funzione di transizione di una TM salva-nastro.
  - Dimostra che le TM salva-nastro riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

**Soluzione.**

- $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, J\}$
- Per dimostrare che TM salva-nastro riconoscono la classe dei linguaggi Turing-riconoscibili dobbiamo dimostrare due cose: che ogni linguaggio Turing-riconoscibile è riconosciuto da una TM salva-nastro, e che ogni linguaggio riconosciuto da una TM salva-nastro è Turing-riconoscibile.

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

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

$S =$  “Su input  $w$ :

1. Inizialmente  $S$  mette il suo nastro in un formato che gli consente di implementare l'operazione di salto al centro della parte non vuota del nastro, usando il simbolo speciale  $\#$  per marcare l'inizio del nastro. Se  $w$  è l'input della TM, la configurazione iniziale del nastro è  $\#w$ .
2. La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, L)$  procede come nella TM standard:  $S$  scrive  $b$  sul nastro e muove la testina di una cella a sinistra. Se lo spostamento a sinistra porta la testina sopra il  $\#$  che marca l'inizio del nastro,  $S$  si muove immediatamente di una cella a destra, lasciando inalterato il  $\#$ . La simulazione continua con la testina in corrispondenza del simbolo subito dopo il  $\#$ .
3. La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, R)$  procede come nella TM standard:  $S$  scrive  $b$  sul nastro e muove la testina di una cella a destra.
4. Per simulare una mossa del tipo  $\delta(q, a) = (r, b, J)$  la TM  $S$  scrive  $b$  nella cella corrente, e poi sposta la testina a sinistra fino a ritornare in corrispondenza del  $\#$  che marca l'inizio del nastro. A questo punto si sposta a destra: se il simbolo dopo il  $\#$  è un blank, la simulazione continua con la testina in corrispondenza del blank. Se il simbolo dopo il  $\#$  è diverso dal blank, la TM lo marca con un pallino, poi si sposta a destra fino ad arrivare al primo blank. Dopodiché marca l'ultima cella non vuota prima del blank e procede a zig-zag, marcando via via una cella all'inizio e una alla fine della porzione di nastro non vuota. Quando la prossima cella da marcare è una cella che è già stata marcata, allora la TM si sposta a sinistra, e marca la cella con un simbolo diverso, come una barra. Poi scorre il nastro per togliere tutti i pallini, e riprende la simulazione con la testina in corrispondenza della cella marcata con la barra.
5. 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.”

**2. (12 punti)** Considera il problema di determinare se i linguaggi di due DFA sono l'uno il complemento dell'altro.

- Formula questo problema come un linguaggio  $COMPLEMENT_{DFA}$ .
- Dimostra che  $COMPLEMENT_{DFA}$  è decidibile.

**Soluzione.**

- $COMPLEMENT_{DFA} = \{\langle A, B \rangle \mid A \text{ e } B \text{ sono DFA e } L(A) = \overline{L(B)}\}$
- La seguente macchina  $N$  usa la Turing machine  $M$  che decide  $EQ_{DFA}$  per decidere  $COMPLEMENT_{DFA}$ :

$N$  = “su input  $\langle A, B \rangle$ , dove  $A$  e  $B$  sono DFA:

- Costruisci l'automa  $\overline{B}$  che riconosce il complementare del linguaggio di  $B$
- Esegui  $M$  su input  $\langle A, \overline{B} \rangle$ , e ritorna lo stesso risultato di  $M$ .”

Mostriamo che  $N$  è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Sappiamo che esiste un algoritmo per costruire il complementare di un DFA (basta invertire stati finali e stati non finali nella definizione dell'automa). Di conseguenza, il primo step di  $N$  termina sempre. Il secondo step termina sempre perché sappiamo che  $EQ_{DFA}$  è un linguaggio decidibile. Quindi  $N$  termina sempre la computazione.

Vediamo ora che  $N$  dà la risposta corretta:

- Se  $\langle A, B \rangle \in COMPLEMENT_{DFA}$  allora  $L(A) = \overline{L(B)}$ , e di conseguenza  $L(A) = L(\overline{B})$  perché  $\overline{B}$  è il complementare di  $B$ . Quindi  $\langle A, \overline{B} \rangle \in EQ_{DFA}$ , e l'esecuzione di  $M$  terminerà con accettazione.  $N$  ritorna lo stesso risultato di  $M$ , quindi accetta.
- Viceversa, se  $\langle A, B \rangle \notin COMPLEMENT_{DFA}$  allora  $L(A) \neq \overline{L(B)}$ , e di conseguenza  $L(A) \neq L(\overline{B})$  perché  $\overline{B}$  è il complementare di  $B$ . Quindi  $\langle A, \overline{B} \rangle \notin EQ_{DFA}$ , e l'esecuzione di  $M$  terminerà con rifiuto.  $N$  ritorna lo stesso risultato di  $M$ , quindi rifiuta.

**3. (12 punti)** Considera il seguente problema: data una TM  $M$  a nastro semi-infinito, determinare se esiste un input  $w$  su cui  $M$  sposta la testina alla destra della cella del nastro numero 2023.

- Formula questo problema come un linguaggio  $2023_{TM}$ .
- Dimostra che il linguaggio  $2023_{TM}$  è indecidibile.

**Soluzione.**

- $2023_{TM} = \{\langle M \rangle \mid M \text{ è una TM ed esiste input } w \text{ su cui } M \text{ sposta la testina a destra della cella numero 2023}\}$ .
- Il linguaggio  $2023_{TM}$  è decidibile.** Questa non è una scelta voluta, ma la conseguenza di un errore nella definizione dell'esercizio. Di conseguenza, per il punto (b) sono stati valutati solo i criteri “sintattici” nella definizione della riduzione e la chiarezza espositiva, stralciando i criteri che valutano la correttezza della riduzione e della dimostrazione. Il punteggio totale dell'esercizio 3 rimane 12, in modo da mantenere 36 punti totali per il compito.

- 1. (12 punti)** Diciamo che una stringa  $x$  è un *prefisso* della stringa  $y$  se esiste una stringa  $z$  tale che  $xz = y$ , e che è un *prefisso proprio* di  $y$  se vale anche  $x \neq y$ . Dimostra che se  $L \subseteq \Sigma^*$  è un linguaggio regolare allora anche il linguaggio

$$NOPREFIX(L) = \{w \in L \mid \text{nessun prefisso proprio di } w \text{ appartiene ad } L\}$$

è un linguaggio regolare.

**Soluzione:** Se  $L$  è un linguaggio regolare, allora sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Costruiamo un DFA  $A'$  che accetta il linguaggio  $NOPREFIX(L)$ , aggiungendo uno stato “pozzo” agli stati di  $A$ , ossia uno stato non finale  $q_s$  tale che la funzione di transizione obbliga l'automa a rimanere per sempre in  $q_s$  una volta che lo si raggiunge. Lo stato iniziale e gli stati finali rimangono invariati. La funzione di transizione di  $A'$  si comporta come quella di  $A$  per gli stati non finali, mentre va verso lo stato pozzo per qualsiasi simbolo dell'alfabeto a partire dagli stati finali. In questo modo le computazioni accettanti di  $A'$  sono sempre sequenze di stati dove solo l'ultimo stato è finale, mentre tutti quelli intermedi sono non finali. Di conseguenza le parole che  $A'$  accetta sono accettate anche da  $A$ , mentre tutti i prefissi propri sono parole rifiutate da  $A$ , come richiesto dalla definizione del linguaggio  $NOPREFIX(L)$ .

Formalmente,  $A' = (Q', \Sigma, \delta', q'_0, F')$  è definito come segue.

- $Q' = Q \cup \{q_s\}$ , con  $q_s \notin Q$ .
- L'alfabeto  $\Sigma$  rimane lo stesso.
- $\delta'(q, a) = \begin{cases} \delta(q, a) & \text{se } q \notin F \\ q_s & \text{altrimenti} \end{cases}$
- $q'_0 = q_0$ . Lo stato iniziale non cambia.
- $F' = F$ . Gli stati finali rimangono invariati.

Per dimostrare che  $A'$  riconosce il linguaggio  $NOPREFIX(L)$ , dobbiamo considerare due casi.

- Se  $w \in NOPREFIX(L)$ , allora sappiamo che  $w \in L$ , mentre nessun prefisso proprio di  $w$  appartiene ad  $L$ . Di conseguenza esiste una computazione di  $A$  che accetta la parola:

$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$

con  $s_0 = q_0$  e  $s_n \in F$ . Siccome tutti i prefissi propri di  $w$  sono rifiutati da  $A$ , allora gli stati  $s_0, \dots, s_{n-1}$  sono tutti non finali. Per la definizione di  $A'$ , la computazione che abbiamo considerato è anche una computazione accettante per  $A'$ , e di conseguenza,  $w \in L(A')$ .

- Viceversa, se  $w$  è accettata dal nuovo automa  $A'$ , allora esiste una computazione accettante che ha la forma

$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$

con  $s_0 = q_0$ ,  $s_n \in F$  e dove tutti gli stati intermedi  $s_0, \dots, s_{n-1}$  sono non finali. Di conseguenza, la computazione è una computazione accettante anche per  $A$ , quindi  $w \in L$ . Siccome tutti gli stati intermedi della computazione sono non finali, allora  $A$  rifiuta tutti i prefissi propri di  $w$ , e quindi  $w \in NOPREFIX(L)$ .

**2. (12 punti)** Considera il linguaggio

$$L_2 = \{uvvu \mid u, v \in \{0, 1\}^*\}.$$

Dimostra che  $L_2$  non è regolare.

**Soluzione:** 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  $w = 0^k 110^k$ , che è di lunghezza maggiore di  $k$  ed appartiene ad  $L_2$  perché la possiamo scrivere come  $uvvu$  ponendo  $u = 0^k$  e  $v = 1$ ;
- 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 sequenza iniziale di 0. Inoltre, siccome  $y \neq \varepsilon$ , abbiamo che  $x = 0^q$  e  $y = 0^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 0^{k-q-p} 110^k$ . Consideriamo l'esponente  $i = 2$ : la parola  $xy^2z$  ha la forma

$$xy^2z = 0^q 0^{2p} 0^{k-q-p} 110^k = 0^{k+p} 110^k$$

La parola iterata  $xy^2z$  non appartiene ad  $L_2$  perché non si può scrivere nella forma  $uvvu$ . Visto che la parte iniziale deve essere uguale a quella finale, si deve porre  $u = 0^k$ , ma in questo caso la parte centrale della parola è  $0^p 11$  che non si può dividere in due metà uguali. Viceversa, se si pone  $v = 1$  per avere la parte centrale della parola composta da due metà uguali, allora si ottiene una sequenza iniziale di 0 che è più lunga della sequenza finale di 0.

Abbiamo trovato un assurdo quindi  $L_2$  non può essere regolare.

**3. (12 punti)** Una grammatica context-free è *lineare* se ogni regola in  $R$  è nella forma  $A \rightarrow aBc$  o  $A \rightarrow a$  per qualche  $a, c \in \Sigma \cup \{\varepsilon\}$  e  $A, B \in V$ . I linguaggi generati dalle grammatiche lineari sono detti *linguaggi lineari*. Dimostra che i linguaggi regolari sono un sottoinsieme proprio dei linguaggi lineari.

**Soluzione.** Per risolvere l'esercizio dobbiamo dimostrare che ogni linguaggio regolare è anche un linguaggio lineare (i linguaggi regolari sono un sottoinsieme dei linguaggi lineari), e che esistono linguaggi lineari che non sono regolari (l'inclusione è propria).

- Dato un linguaggio regolare  $L$ , sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Inoltre, sappiamo che ogni DFA può essere convertito in una grammatica context-free dove le regole sono del tipo  $R_i \rightarrow aR_j$  per ogni transizione  $\delta(q_i, a) = q_j$  del DFA, e del tipo  $R_i \rightarrow \varepsilon$  per ogni stato finale del  $q_i$  del DFA. Entrambi i tipi di regola rispettano le condizioni di linearità, quindi la grammatica equivalente al DFA è lineare, e questo implica che  $L$  è un linguaggio lineare.

- Consideriamo il linguaggio non regolare  $L = \{0^n 1^n \mid n \geq 0\}$ . La seguente grammatica lineare genera  $L$ :

$$S \rightarrow 0S1 \mid \varepsilon$$

Quindi, esiste un linguaggio lineare che non è regolare.

**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.

$S = \text{"Su input } w\text{"}$

1. Inizialmente  $S$  mette il suo nastro in un formato che gli consente di implementare l'operazione di salto oltre la parte non vuota del nastro, usando il simbolo speciale  $\#$  per marcare l'inizio e la fine della porzione usata del nastro. Se  $w$  è l'input della TM, la configurazione iniziale del nastro è  $\#w\#$ .
2. La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, L)$  procede come nella TM standard:  $S$  scrive  $b$  sul nastro e muove la testina di una cella a sinistra. Se lo spostamento a sinistra porta la testina sopra il  $\#$  che marca l'inizio del nastro,  $S$  si muove immediatamente di una cella a destra, lasciando inalterato il  $\#$ . La simulazione continua con la testina in corrispondenza del simbolo subito dopo il  $\#$ .
3. La simulazione delle mosse del tipo  $\delta(q, a) = (r, b, R)$  procede come nella TM standard:  $S$  scrive  $b$  sul nastro e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra il  $\#$  che marca la fine del nastro,  $S$  scrive un blank al posto del  $\#$ , e scrive un  $\#$  nella cella immediatamente più a destra. La simulazione continua con la testina in corrispondenza del blank.
4. Per simulare una mossa del tipo  $\delta(q, a) = (r, b, J)$  la TM  $S$  scrive  $b$  nella cella corrente, e poi sposta la testina a destra fino ad arrivare in corrispondenza del  $\#$  che marca la fine del nastro. A questo punto si sposta a sinistra finché non trova un simbolo diverso dal blank. Marca con un pallino il primo simbolo non blank che trova, poi si sposta di una cella a destra e la marca con il pallino. Continua procedendo a zig-zag, marcando via via una cella all'inizio e una alla fine della sequenza di pallini. Quando la prossima cella da marcare è il  $\#$  all'inizio del nastro, la TM non la marca e inizia a spostarsi a destra, scorrendo il nastro per togliere tutti i pallini, e riprendere la simulazione con la testina in corrispondenza dell'ultima cella marcata. In questa ultima fase, se una delle celle marcate è il  $\#$  alla fine del nastro, allora la TM lo sostituisce con un blank e scrive un  $\#$  immediatamente a destra dell'ultima cella marcata prima di continuare con la simulazione.
5. 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.”

**2. (12 punti)** Una variabile  $A$  in una grammatica context-free  $G$  è *persistente* se compare in ogni derivazione di ogni stringa  $w$  in  $L(G)$ . Data una grammatica context-free  $G$  e una variabile  $A$ , considera il problema di verificare se  $A$  è persistente.

- (a) Formula questo problema come un linguaggio  $PERSISTENT_{CFG}$ .
- (b) Dimostra che  $PERSISTENT_{CFG}$  è decidibile.

### Soluzione.

- (a)  $PERSISTENT_{CFG} = \{\langle G, A \rangle \mid G \text{ è una CFG, } A \text{ è una variabile persistente}\}$
- (b) La seguente macchina  $N$  usa la Turing machine  $M$  che decide  $E_{CFG}$  per decidere  $PERSISTENT_{CFG}$ :

$N$  = “su input  $\langle G, A \rangle$ , dove  $G$  è una CFG e  $A$  una variabile:

1. Verifica che  $A$  appartenga alle variabili di  $G$ . In caso negativo, rifiuta.
2. Costruisci una CFG  $G'$  eliminando tutte le regole dove compare  $A$  dalla grammatica  $G$ .
3. Esegui  $M$  su input  $\langle G' \rangle$ , e ritorna lo stesso risultato di  $M$ .”

Mostriamo che  $N$  è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Verificare che una variabile appartenga alle variabili di  $G$  è una operazione che si può implementare scorrendo la codifica di  $G$  per controllare se  $A$  compare nella codifica. Il secondo passo si può implementare copiando la codifica di  $G$  senza riportare le regole dove compare  $A$ . Di conseguenza, il primo ed il secondo step terminano sempre. Anche il terzo step termina sempre perché sappiamo che  $E_{CFG}$  è un linguaggio decidibile. Quindi  $N$  termina sempre la computazione. Vediamo ora che  $N$  dà la risposta corretta:

- Se  $\langle G, A \rangle \in PERSISTENT_{CFG}$  allora  $A$  è una variabile persistente, quindi compare in ogni derivazione di ogni stringa  $w \in L(G)$ . Se la eliminiamo dalla grammatica, eliminando tutte le regole dove compare  $A$ , allora otteniamo una grammatica  $G'$  dove non esistono derivazioni che permettano di derivare una stringa di soli simboli terminali, e di conseguenza  $G'$  ha linguaggio vuoto. Quindi  $\langle G' \rangle \in E_{CFG}$ , e l'esecuzione di  $M$  terminerà con accettazione.  $N$  ritorna lo stesso risultato di  $M$ , quindi accetta.
- Viceversa, se  $\langle G, A \rangle \in PERSISTENT_{CFG}$  allora  $A$  non è una variabile persistente, quindi esiste almeno una derivazione di una parola  $w \in L(G)$  dove  $A$  non compare. Se eliminiamo  $A$  dalla grammatica, eliminando tutte le regole dove compare, allora otteniamo una grammatica  $G'$  che può derivare  $w$ , e di conseguenza  $G'$  ha linguaggio vuoto. Quindi  $\langle G' \rangle \notin E_{CFG}$ , e l'esecuzione di  $M$  terminerà con rifiuto.  $N$  ritorna lo stesso risultato di  $M$ , quindi rifiuta.

**3. (12 punti)** Considera le stringhe sull'alfabeto  $\Sigma = \{1, 2, \dots, 9\}$ . Una stringa  $w$  di lunghezza  $n$  su  $\Sigma$  si dice *ordinata* se  $w = w_1 w_2 \dots w_n$  e tutti i caratteri  $w_1, w_2, \dots, w_n \in \Sigma$  sono tali che  $w_1 \leq w_2 \leq \dots \leq w_n$ . Ad esempio, la stringa 1112778 è ordinata, ma le stringhe 5531 e 44427 non lo sono (la stringa vuota viene considerata ordinata). Diciamo che una Turing machine è *osessionata dall'ordinamento* se ogni stringa che accetta è ordinata (ma non è necessario che accetti tutte queste stringhe). Considera il problema di determinare se una TM con alfabeto  $\Sigma = \{1, 2, \dots, 9\}$  è osessionata dall'ordinamento.

- (a) Formula questo problema come un linguaggio  $SO_{TM}$ .
- (b) Dimostra che il linguaggio  $SO_{TM}$  è indecidibile.

### Soluzione.

- (a)  $SO_{TM} = \{\langle M \rangle \mid M \text{ è una TM con alfabeto } \Sigma = \{1, 2, \dots, 9\} \text{ che accetta solo parole ordinate}\}$

- (b) La seguente macchina  $F$  calcola una riduzione  $\overline{A_{TM}} \leq_m UA$ :

$F$  = “su input  $\langle M, w \rangle$ , dove  $M$  è una TM e  $w$  una stringa:

1. Costruisci la seguente macchina  $M'$ :

$M'$  = “Su input  $x$ :

1. Se  $x = 111$ , accetta.
  2. Se  $x = 211$ , esegue  $M$  su input  $w$  e ritorna lo stesso risultato di  $M$ .
  3. In tutti gli altri casi, rifiuta.
2. Ritorna  $\langle M' \rangle$ .

Mostriamo che  $F$  calcola una funzione di riduzione da  $\overline{A_{TM}}$  a  $SO_{TM}$ , cioè una funzione tale che

$$\langle M, w \rangle \in \overline{A_{TM}} \text{ se e solo se } \langle M' \rangle \in SO_{TM}.$$

- Se  $\langle M, w \rangle \in \overline{A_{TM}}$  allora la macchina  $M$  rifiuta o va in loop su  $w$ . In questo caso la macchina  $M'$  accetta la parola ordinata 111 e rifiuta tutte le altre, quindi è ossessionata dall'ordinamento e di conseguenza  $\langle M' \rangle \in SO_{TM}$ .
- Viceversa, se  $\langle M, w \rangle \notin \overline{A_{TM}}$  allora la macchina  $M$  accetta  $w$ . Di conseguenza, la macchina  $M'$  accetta sia la parola ordinata 111 che la parola non ordinata 211, quindi non è ossessionata dall'ordinamento. Di conseguenza  $\langle M' \rangle \notin SO_{TM}$ .

1. (12 punti) Data una stringa  $w$  di 0 e 1, il *flip* di  $w$  si ottiene cambiando tutti gli 0 in  $w$  con 1 e tutti gli 1 in  $w$  con 0. Dato un linguaggio  $L$ , il flip di  $L$  è il linguaggio

$$\text{flip}(L) = \{w \in \{0, 1\}^* \mid \text{il flip di } w \text{ appartiene ad } L\}.$$

Dimostra che la classe dei linguaggi regolari è chiusa rispetto all'operazione di flip.

**Soluzione:** Se  $L$  è un linguaggio regolare, allora sappiamo che esiste un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $L$ . Costruiamo un DFA  $A' = (Q', \Sigma, \delta', q'_0, F')$  che accetta il linguaggio  $\text{flip}(L)$  come segue.

- $Q' = Q$ . L'insieme degli stati rimane lo stesso.
- L'alfabeto  $\Sigma$  rimane lo stesso.
- Per ogni stato  $q \in Q$ ,  $\delta'(q, 0) = \delta(q, 1)$  e  $\delta'(q, 1) = \delta(q, 0)$ . La funzione di transizione scambia gli 0 con 1 e gli 1 con 0.
- $q'_0 = q_0$ . Lo stato iniziale non cambia.
- $F' = F$ . Gli stati finali rimangono invariati.

Per dimostrare che  $A'$  riconosce il linguaggio  $\text{flip}(L)$ , dobbiamo considerare due casi.

- Se  $w \in \text{flip}(L)$ , allora sappiamo che il flip di  $w$  appartiene ad  $L$ . Chiamiamo  $\bar{w}$  il flip di  $w$ . Siccome  $A$  riconosce  $L$ , allora esiste una computazione di  $A$  che accetta  $\bar{w}$ :

$$s_0 \xrightarrow{\bar{w}_1} s_1 \xrightarrow{\bar{w}_2} \dots \xrightarrow{\bar{w}_n} s_n$$

con  $s_0 = q_0$  e  $s_n \in F$ . Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per  $A'$  sulla parola  $w$ . Di conseguenza, abbiamo dimostrato che  $w \in L(A')$ .

- Viceversa, se  $w$  è accettata dal nuovo automa  $A'$ , allora esiste una computazione accettante che ha la forma

$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$

con  $s_0 = q_0$ ,  $s_n \in F'$ . Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per  $A$  sul flip di  $w$ . Di conseguenza, il flip di  $w$  appartiene ad  $L$  e abbiamo dimostrato che  $w \in \text{flip}(L)$ .

**2. (12 punti)** Considera il linguaggio

$$L_2 = \{w \in \{0, 1\}^* \mid w \text{ non è palindroma}\}.$$

Una parola è *palindroma* se rimane uguale letta da sinistra a destra e da destra a sinistra. Dimostra che  $L_2$  non è regolare.

**Soluzione:** Consideriamo il complementare del linguaggio  $L_2$ , ossia il linguaggio

$$\overline{L_2} = \{w \in \{0, 1\}^* \mid w \text{ è palindroma}\}.$$

Usiamo il Pumping Lemma per dimostrare che il linguaggio  $\overline{L_2}$  non è regolare. Supponiamo per assurdo che lo sia:

- sia  $k$  la lunghezza data dal Pumping Lemma;
- consideriamo la parola  $w = 0^k 1 0^k$ , che è di lunghezza maggiore di  $k$  ed è palindroma;
- 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 sequenza iniziale di 0. Inoltre, siccome  $y \neq \varepsilon$ , abbiamo che  $x = 0^q$  e  $y = 0^p$  per qualche  $q \geq 0$  e  $p > 0$ .  $z$  contiene la parte rimanente della stringa:  $z = 0^{k-q-p} 1 0^k$ . Consideriamo l'esponente  $i = 2$ : la parola  $xy^2z$  ha la forma

$$xy^2z = 0^q 0^{2p} 0^{k-q-p} 1 0^k = 0^{k+p} 1 0^k$$

La parola iterata  $xy^2z$  non appartiene ad  $\overline{L_2}$  perché non è palindroma: se la rovesciamo diventa la parola  $0^k 1 0^{k+p}$  che è una parola diversa perché  $p > 0$ .

Abbiamo trovato un assurdo quindi  $\overline{L_2}$  non può essere regolare.

Siccome i linguaggi regolari sono chiusi per complementazione, nemmeno  $L_2$  può essere regolare.

**3. (12 punti)** Dimostra che se  $L \subseteq \Sigma^*$  è un linguaggio context-free allora anche il seguente linguaggio è context-free:

$$\text{dehash}(L) = \{\text{dehash}(w) \mid w \in L\},$$

dove  $\text{dehash}(w)$  è la stringa che si ottiene cancellando ogni  $\#$  da  $w$ .

**Soluzione:** Se  $L$  è un linguaggio context-free, allora esiste una grammatica  $G = (V, \Sigma, R, S)$  che lo genera. Possiamo assumere che questa grammatica sia in forma normale di Chomsky. Per dimostrare che  $\text{dehash}(L)$  è context-free, dobbiamo essere in grado di definire una grammatica che possa generarlo. Questa grammatica è una quadrupla  $G' = (V', \Sigma', R', S')$  definita come segue.

- L'alfabeto tutti i simboli di  $\Sigma$  tranne  $\#$ :  $\Sigma' = \Sigma \setminus \{\#\}$ .
- L'insieme di variabili è lo stesso della grammatica  $G$ :  $V' = V$ .

- Il nuovo insieme di regole  $R'$  è ottenuto rimpiazzando ogni regola nella forma  $A \rightarrow \#$  con la regola  $A \rightarrow \varepsilon$ , e lasciando invariate le regole nella forma  $A \rightarrow BC$ , le regole nella forma  $A \rightarrow b$  quando  $b \neq \#$ , e la regola  $S \rightarrow \varepsilon$  (se presente).
- La variabile iniziale rimane la stessa:  $S' = S$ .

Data una derivazione  $S \Rightarrow^* w$  della grammatica  $G$  possiamo costruire una derivazione nella nuova grammatica  $G'$  che applica le stesse regole nello stesso ordine, e che deriva una parola dove ogni  $\#$  è rimpiazzato dalla parola vuota  $\varepsilon$ . Quindi,  $G'$  permette di derivare tutte le parole in  $\text{dehash}(L)$ .

Viceversa, data una derivazione  $S \Rightarrow^* w$  della nuova grammatica  $G'$  possiamo costruire una derivazione nella grammatica  $G$  che applica le stesse regole nello stesso ordine. Di conseguenza, in ogni punto in cui la derivazione per  $G'$  applica la regola modificata  $A \rightarrow \varepsilon$ , la derivazione per  $G$  applicherà la regola  $A \rightarrow \#$  inserendo un  $\#$  in qualche punto della parola  $w$ . Al termine della derivazione si ottiene una parola  $w'$  tale che  $\text{dehash}(w') = w$ . Quindi,  $G'$  permette di derivare solo parole che appartengono a  $\text{dehash}(L)$ .

**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.

### Soluzione.

- (a)  $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L3, R2\}$
- (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.”

**2. (12 punti)** Dati due DFA, considera il problema di determinare se esiste una stringa accettata da entrambi.

- (a) Formula questo problema come un linguaggio  $AGREE_{\text{DFA}}$ .
- (b) Dimostra che  $AGREE_{\text{DFA}}$  è decidibile.

### Soluzione.

- (a)  $AGREE_{\text{DFA}} = \{\langle A, B \rangle \mid A, B \text{ sono DFA, ed esiste una parola } w \text{ tale che } w \in L(A) \text{ e } w \in L(B)\}$
- (b) La seguente macchina  $N$  usa la Turing machine  $M$  che decide  $E_{\text{DFA}}$  per decidere  $AGREE_{\text{DFA}}$ :

$N$  = “su input  $\langle A, B \rangle$ , dove  $A, B$  sono DFA:

1. Costruisci il DFA  $C$  che accetta l’intersezione dei linguaggi di  $A$  e  $B$
2. Esegui  $M$  su input  $\langle C \rangle$ . Se  $M$  accetta, rifiuta, se  $M$  rifiuta, accetta.”

Mostriamo che  $N$  è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Sappiamo che esiste un algoritmo per costruire l’intersezione di due DFA. Il primo step di  $N$  si implementa eseguendo questo algoritmo, e termina sempre perché la costruzione dell’intersezione termina. Il secondo step termina sempre perché sappiamo che  $E_{\text{DFA}}$  è un linguaggio decidibile. Quindi  $N$  termina sempre la computazione.

Vediamo ora che  $N$  dà la risposta corretta:

- Se  $\langle A, B \rangle \in AGREE_{\text{DFA}}$  allora esiste una parola che viene accettata sia da  $A$  che da  $B$ , e quindi il linguaggio  $L(A) \cap L(B)$  non può essere vuoto. Quindi  $\langle C \rangle \in E_{\text{DFA}}$ , e l’esecuzione di  $M$  terminerà con rifiuto.  $N$  ritorna l’opposto di  $M$ , quindi accetta.
- Viceversa, se  $\langle A, B \rangle \notin AGREE_{\text{DFA}}$  allora non esiste una parola che sia accettata sia da  $A$  che da  $B$ , e quindi il linguaggio  $L(A) \cap L(B)$  è vuoto. Quindi  $\langle C \rangle \notin E_{\text{DFA}}$ , e l’esecuzione di  $M$  terminerà con accettazione.  $N$  ritorna l’opposto di  $M$ , quindi rifiuta.

**3. (12 punti)** Data una Turing Machine  $M$ , considera il problema di determinare se esiste un input tale che  $M$  scrive “*xyzzy*” su cinque celle adiacenti del nastro. Puoi assumere che l’alfabeto di input di  $M$  non contenga i simboli  $x, y, z$ .

- (a) Formula questo problema come un linguaggio  $MAGIC_{\text{TM}}$ .
- (b) Dimostra che il linguaggio  $MAGIC_{\text{TM}}$  è indecidibile.

### Soluzione.

- (a)  $MAGIC_{\text{TM}} = \{\langle M \rangle \mid M \text{ è una TM che scrive } xyzzy \text{ sul nastro per qualche input } w\}$
- (b) La seguente macchina  $F$  calcola una riduzione  $A_{\text{TM}} \leq_m MAGIC_{\text{TM}}$ :

$F$  = “su input  $\langle M, w \rangle$ , dove  $M$  è una TM e  $w$  una stringa:

1. Verifica che i simboli  $x, y, z$  non compaiano in  $w$ , né nell’alfabeto di input o nell’alfabeto del nastro di  $M$ . Se vi compaiono, sostituiscegli con tre nuovi simboli  $X, Y, Z$  nella parola  $w$  e nella codifica di  $M$ .
2. Costruisci la seguente macchina  $M'$ :

$M'$  = “Su input  $x$ :

1. Simula l’esecuzione di  $M$  su input  $w$ , senza usare i simboli  $x, y, z$ .
2. Se  $M$  accetta, scrivi *xyzzy* sul nastro, altrimenti rifiuta senza modificare il nastro.

3. Ritorna  $\langle M' \rangle$ .

Mostriamo che  $F$  calcola una funzione di riduzione da  $A_{\text{TM}}$  a  $MAGIC_{\text{TM}}$ , cioè una funzione tale che

$$\langle M, w \rangle \in A_{\text{TM}} \text{ se e solo se } \langle M' \rangle \in MAGIC_{\text{TM}}.$$

- Se  $\langle M, w \rangle \in A_{\text{TM}}$  allora la macchina  $M$  accetta  $w$ . In questo caso la macchina  $M'$  scrive *xyzzy* sul nastro per tutti gli input. Di conseguenza  $\langle M' \rangle \in MAGIC_{\text{TM}}$ .
- Viceversa, se  $\langle M, w \rangle \notin A_{\text{TM}}$  allora la macchina  $M$  rifiuta o va in loop su  $w$ . Per tutti gli input, la macchina  $M'$  simula l’esecuzione di  $M$  su  $w$  senza usare i simboli  $x, y, z$  (perché sono stati tolti dalla definizione di  $M$  e di  $w$  se vi comparivano), e rifiuta o va in loop senza scrivere mai *xyzzy* sul nastro. Di conseguenza  $\langle M' \rangle \notin MAGIC_{\text{TM}}$ .

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 24 AGOSTO 2023

- 1. (9 punti)** Considera il linguaggio

$$L = \{0^m 1^n \mid 2m > 3n + 1\}.$$

Dimostra che  $L$  non è regolare.

- 2. (9 punti)** Considera la seguente funzione da  $\{0, 1\}^*$  a  $\{0, 1\}^*$ :

$$\text{stutter}(w) = \begin{cases} \varepsilon & \text{se } w = \varepsilon \\ aa.\text{stutter}(x) & \text{se } w = ax \text{ per qualche simbolo } a \text{ e parola } x \end{cases}$$

Dimostra che se  $L$  è un linguaggio context-free sull'alfabeto  $\{0, 1\}$ , allora anche il seguente linguaggio è context-free:

$$\text{stutter}(L) = \{\text{stutter}(w) \mid w \in L\}.$$

- 3. (9 punti)** Dimostra che un linguaggio è decidibile se e solo se esiste un enumeratore che lo enumera seguendo l'ordinamento standard delle stringhe.

- 4. (9 punti)** Considera il seguente problema: data una TM  $M$  a nastro semi-infinito, determinare se esiste un input  $w$  su cui  $M$  sposta la testina a sinistra partendo dalla cella numero 2023 (ossia se in qualche momento durante la computazione la testina si muove dalla cella 2023 alla cella 2022).

- (a) Formula questo problema come un linguaggio  $2023_{TM}$ .
- (b) Dimostra che il linguaggio  $2023_{TM}$  è indecidibile.

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 15 SETTEMBRE 2023

- 1. (9 punti)** Considera il linguaggio

$$L = \{0^m 1^n \mid 3m \leq 2n\}.$$

Dimostra che  $L$  non è regolare.

- 2. (9 punti)** Dimostra che se  $L$  è un linguaggio context-free, allora anche il seguente linguaggio è context-free:

$$\text{delete}_{\#}(L) = \{xy \mid x\#y \in L\}.$$

- 3. (9 punti)** Una *Turing machine con alfabeto ternario* è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è  $\Sigma = \{0, 1, 2\}$  e l'alfabeto del nastro è  $\Gamma = \{0, 1, 2, \_\}$ . Questo significa che la macchina può scrivere sul nastro solo i simboli 0, 1, 2 e blank: non può usare altri simboli né marcare i simboli sul nastro.

Dimostra che ogni linguaggio Turing-riconoscibile sull'alfabeto  $\{0, 1, 2\}$  può essere riconosciuto da una Turing machine con alfabeto ternario.

- 4. (9 punti)** Una Turing Machine *somma correttamente* se, dati in input due numeri binari separati da  $\#$ , termina la computazione con la loro somma (in binario) sul nastro. (Non importa cosa fa sugli altri input.) Considera il problema di determinare se una TM somma correttamente.

- Formula questo problema come un linguaggio  $SUM_{TM}$ .
- Dimostra che il linguaggio  $SUM_{TM}$  è indecidibile.

AUTOMI E LINGUAGGI FORMALI  
ESAME SCRITTO DEL 5 FEBBRAIO 2024

- 1. (9 punti)** Considera il linguaggio

$$L = \{a^{n^2}b^{n^2} \mid n > 0\}.$$

Dimostra che  $L$  non è regolare.

- 2. (9 punti)** Dimostra che se  $L$  è un linguaggio context-free, allora anche il seguente linguaggio è context-free:

$$\text{insert}_{\#}(L) = \{x\#y \mid xy \in L\}.$$

- 3. (9 punti)** Dimostra che un linguaggio è decidibile se e solo se esiste un enumeratore che lo enumera seguendo l'ordinamento standard delle stringhe.

- 4. (9 punti)** Una Turing Machine *moltiplica correttamente* se, dati in input due numeri binari separati da  $\#$ , termina la computazione con la loro moltiplicazione (in binario) sul nastro. (Non importa cosa fa sugli altri input.) Considera il problema di determinare se una TM moltiplica correttamente.

(a) Formula questo problema come un linguaggio  $MUL_{TM}$ .

(b) Dimostra che il linguaggio  $MUL_{TM}$  è indecidibile.