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
$$\overset{\bullet}{w}_1 w_2 \dots w_n \underset{\bullet}{\sqcup}.$$
2. Per simulare una transizione, $S$ scorre il nastro per trovare la posizione della testina di lettura e determinare il simbolo letto da $M$. Se la funzione di transizione stabilisce che la testina di lettura deve spostarsi a destra, allora $S$ sposta il pallino nella cella immediatamente a destra, altrimenti lo lascia dov'è. Successivamente $S$ si sposta verso destra finché non trova la cella dove si trova la testina di scrittura, scrive il simbolo stabilito dalla funzione di transizione nella cella e sposta la marcatura nella cella immediatamente a destra.
3. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di $M$, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di $M$ allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

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

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

1. Scrive il simbolo # subito dopo l'input, seguito dalla configurazione iniziale, in modo che il nastro contenga
$$w_1 w_2 \dots w_n \# \overset{\bullet}{w}_1 w_2 \dots w_n \sqcup,$$
che la testina di lettura si trovi in corrispondenza del $\overset{\bullet}{w}_1$ e quella di lettura in corrispondenza del blank dopo la configurazione iniziale. Imposta lo stato corrente della simulazione $st$ allo stato iniziale di $S$ e memorizza l'ultimo simbolo letto $prec = \#$. L'informazione sui valori di $st$ e $prec$ sono codificate all'interno degli stato di $M$.
2. Finché il simbolo sotto la testina di lettura non è marcato, scrive il simbolo precedente $prec$ e muove a destra. Aggiorna il valore di $prec$ con il simbolo letto.
3. Quando si trova un simbolo marcato $\overset{\bullet}{a}$ 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 $\overset{\bullet}{a}$ e $\delta(st, a) = (q, b, L)$:
   - aggiorna lo stato della simulazione $st = q$;
   - scrive $\overset{\bullet}{prec}$; se $prec = \#$ scrive $\#\underset{\bullet}{\sqcup}$;
   - scrive $b$.
5. Copia il resto della configurazione fino al # escluso. Al termine della copia la testina di lettura di trova in corrispondenza della prima cella nella configurazione corrente, e quella di lettura sulla cella vuota dopo la configurazione.
6. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di $S$, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di $M$ allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

**2.** (a) Dimostriamo che AlwaysDiverge è un linguaggio indecidibile mostrando che $\overline{A_{TM}}$ è riducibile ad 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 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 $f(\langle M, w\rangle) = \langle M'\rangle \in$ 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 $f(\langle M, w\rangle) = \langle M'\rangle \notin$ AlwaysDiverge.

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

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

$D =$ "su input $\langle M\rangle$, dove $M$ è una TM:
1. Ripeti per $i = 1, 2, 3, \ldots$:
2. Esegue $M$ per $i$ passi di computazione su ogni input $s_1, s_2, \ldots s_i$.
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:

$V$ = "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 $\langle G\rangle \in$ 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 $\langle n, 3, R\rangle \in$ 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 $\langle G\rangle \in$ 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.