Automi e Linguaggi Formali – 26/6/2023
Primo appello – Seconda Parte: Macchine di Turing, decidibilità ed indecidibilità

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.

$S =$ "Su input w:

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 $\mathit{PERSISTENT}_{\text{CFG}}$.

(b) Dimostra che $\mathit{PERSISTENT}_{\text{CFG}}$ è decidibile.

Soluzione.

(a) $\mathit{PERSISTENT}_{\text{CFG}} = \{\langle G, A\rangle \mid G \text{ è una CFG, } A \text{ è una variabile persistente}\}$

(b) La seguente macchina N usa la Turing machine M che decide E_{CFG} per decidere $\mathit{PERSISTENT}_{\text{CFG}}$:

$N =$ "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 $\langle G, A\rangle \in \mathit{PERSISTENT}_{\text{CFG}}$ 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 $\langle G'\rangle \in E_{\text{CFG}}$, e l'esecuzione di M terminerà con accettazione. N ritorna lo stesso risultato di M, quindi accetta.
- Viceversa, se $\langle G, A\rangle \in \mathit{PERSISTENT}_{\text{CFG}}$ 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 $\langle G'\rangle \notin E_{\text{CFG}}$, 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 Σ 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 *ossessionata 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 ossessionata dall'ordinamento.

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

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

Soluzione.

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

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

$F =$ "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

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

- 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 $\langle M'\rangle \in SO_{\mathrm{TM}}$.
- 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 $\langle M'\rangle \notin SO_{\mathrm{TM}}$.