Automi e Linguaggi Formali
Esame del 3 Luglio 2019 – Parte II

Tempo a disposizione: 1 h 30 min

1. Si consideri la seguente grammatica CF, G:
$S \to aBb$, $B \to aBb \mid BB \mid \epsilon$

i) Descrivere un DPDA P che riconosce per stack vuoto il linguaggio L(G). Spiegare perché il vostro P fa quanto richiesto. In particolare, assicuratevi che P sia deterministico.

ii) Descrivere il calcolo di P con input uguale a "abab"e spiegare se vi sembra corretto oppure no, spiegando la vostra posizione.

i) Il DPDA che riconosce $L(S)$ è come segue:

$$\delta(q_0, a, Z_0) = \{((q_1, aZ_0)\}, \qquad \delta(q_1, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, b, a) = \{(q_1, \epsilon)\}, \qquad \delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$$

ii) consumando il primo a andrebbe in q_1 con a sullo stack, poi consumerebbe il b facendo il pop dell'a sullo stack. Quindi sullo stack avremmo Z_0 per cui la sola mossa possibile sarebbe quella con ϵ che svuota lo stack. Visto che la stringa non sarebbe finita e che il DPDA si blocca non appena lo stack si svuota, "abab" verrebbe rifiutata il che è giusto visto che non è in $L(S)$. Infatti $L(S)$ è un linguaggio con la proprietà del prefisso.

2. Il linguaggio $L = \{a^n ww^r b^n | n \geq 0 \text{ e } w \in \{a, b\}^*\}$ è CF, ma fingete di non saperlo ed applicate ad esso il Pumping Lemma per cercare di dimostare che L non sia CF. Cercate di individuare in quale passaggio la prova fallisce e perché.

Sia m la costante del PL e consideriamo una qualsiasi z di L di lunghezza maggiore di m. Il PL ci dice che $z = uvtxy$ con $|vtx| \leq m$ e $vx \neq \epsilon$ e tale che per ogni $k \geq 0$, $uv^k tx^k y \in L$. Se in z la parte centrale ww^r fosse piccola rispetto ad m, sarebbe facile considerare che $t = ww^r$ e $v = a^q$ e $x = b^q$ per cui anche pompando v e x a piacimento si resterebbe in L. Consideriamo quindi il caso in cui ww^r sia di lunghezza maggiore o uguale di m. Anche in questo caso abbiamo una soluzione che ci fa restare in L: $t = \epsilon$, mentre v è un suffisso di w di lunghezza q e x è un prefisso di w^r della stessa lunghezza. Allora vx sarebbe un palindromo e $v^q x^q$ continuerebbe ad essere un palindromo, rimanendo in L.

3. La domanda riguarda le variabili inutili di una CFG e come eliminarle:

i) Definire le variabili inutili di una grammatica CF.

ii) Spiegare in generale come si possono eliminare tutte le variabili inutili di una CFG.

iii) Applicare la procedura descritta nel punto precedente alla seguente CFG ottenendo una nuova grammatica equivalente a quella precedente e senza variabili inutili:
$S \to AB \mid CA$, $A \to a$, $B \to BC \mid AB \mid DB$, $C \to aB \mid b$, $D \to aDA \mid a$

i) Si devono eliminare le variabili non generatrici e poi quelle non raggiungibili. Una variabile X è generatrice se $X \Rightarrow^* w$ con w composto da soli terminali. Si osservi che w può essere anche ϵ. Una variabile X è raggiungibile se $S \Rightarrow^* \alpha X \beta$, dove S è il simbolo iniziale della grammatica.

ii) Per calcolare le variabili generatrici, si parte con un insieme Q che contiene tutti i simboli terminali, poi si aggiungono a Q le variabili X tali che ci sia una produzione $X \to \alpha$ con $\alpha \in Q*$ (si osservi che α può anche essere vuota). Si continua ad aggiungere variabili a Q fino a quando è possibile. Per calcolare i simboli raggiungibili, Si parte con $P = \{S\}$ e poi si aggiungono a P i simboli nella parte destra α di produzioni $X \to \alpha$ tali che $X \in P$. Si continua fino a che si aggiungono variabili a P.

- Nella grammatica data B non è generatrice, quindi eliminando dalla grammatica le produzioni con B, otteniamo, $S \to CA$, $A \to a$, $C \to b$, $D \to aDA \mid a$

  Poi è facile vedere che D non è raggiungibile e quindi alla fine resta: $S \to CA$, $A \to a$, $C \to b$

4. La domanda riguarda le macchine di Turing (TM):

i) Descrivere come si rappresenta una TM con una sequenza di 0 e 1.

ii) Spiegare perché la rappresentazione binaria delle TM è importante per dimostrare che esistono linguaggi non RE.

iii) Dare un esempio di linguaggio non RE e dimostrare che effettivamente è non RE.

i) Una macchina di Turing viene codificata con una sequenza di 0 e 1 che rappresenta le transizioni della TM. Una transizione: $\delta(q_i, X_k) = (q_j, X_t, L)$ è rappresentata da $0^i10^k10^j10^t10$. Quindi ogni simbolo è rappresentato da una opportuna sequenza di 0. Gli 1 servono per separare gli 0. Lo stato iniziale è sempre q_1 e lo stato finale q_2. C'è un solo stato finale. Il simbolo di nastra X_1 rappresenta lo 0, X_2 rappresenta 1 e X_3 è il blank. Gli altri simboli $X_4, X_5, ...$ sono simboli utili per la TM rappresentata. Left/Right sono rappresentati con 0/00. Le diverse transizioni sono separate da coppie di 1, quindi la rappresentazione di una TM è una stringa $C_1 11 C_2 11 ... 11 C_k$, dove ciascun C_l rappresenta una transizione come quella appena descritta.

ii) Che le TM siano stringhe binarie, permette di avere TM che ricevono TM come input. Seguendo questa idea è possibile definire una tabella infinita che ha stringhe binarie crescenti $w_1, w_2, ...$ nelle righe e nelle colonne. Si tratta delle stesse stringhe, ma nelle righe sono interpretate come TM e nelle colonne come input. In ogni entrata M[i,j] della tabella metteremo 1 se la TM w_i accetta la stringa w_j e 0 se non l'accetta. Su questa tabella useremo la tecnica detta della diagonalizzazione per dimostrare che esistono linguaggi che non sono RE, cioè che non sono riconosciuti da alcuna TM.

iii) Il linguaggio $L_d = \{w_i \mid w_i \textit{ non accetta } w_i \textit{ come input}\}$. Dimostriamo che L_d non è RE. Ragioniamo per assurdo e assumiamo che invece sia RE. Quindi esiste una TM, diciamo w_j, che accetta L_d. Adesso osserviamo come si comporta w_j con se stessa come input. Ovviamente può accettare w_j oppure no. Supponiamo che w_j accetti se stessa. Allora, visto che la TM w_j accetta L_d, significa che $w_j \in L_d$, ma se $w_j \in L_d$ allora, per definizione di L_d, w_j non accetta w_j. Quindi abbiamo una contraddizione. Supponiamo ora che w_j non accetti w_j, ma allora w_j dovrebbe essere in L_d, ma non lo è visto che w_j accetta L_d. Quindi abbiamo di nuovo una contraddizione. Il che dimostra che non esiste TM che accetta L_d.

5. 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-completo.

Considerate la seguente variante del problema, chiamata 3WayPartitioning: stabilire se un insieme di numeri interi S può essere suddiviso in tre sottoinsiemi disgiunti S_1, S_2 e S_3 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 che è uguale alla somma dei numeri in S_3.

(a) Dimostrare che il problema 3WayPartitioning è in NP fornendo un certificato per il *Sì* che si può verificare in tempo polinomiale.

(b) Dimostrare che il problema 3WayPartitioning è NP-hard, mostrando come si può risolvere il problema SetPartitioning usando il problema 3WayPartitioning come sottoprocedura.

(a) Il certificato è dato da tre insiemi di insiemi di numeri interi S_1, S_2, S_3. Per verificarlo occorre controllare che rispetti le seguenti condizioni:

- i tre insiemi devono essere una partizione dell'insieme S;
- la somma degli elementi in S_1 deve essere uguale alla somma degli elementi in S_2 che deve essere uguale alla somma dei numeri in S_3.

Entrambe le condizioni si possono verificare in tempo polinomiale.

(b) Dimostrare che 3WayPartitioning è NP-hard, usando SetPartitioning come problema di riferimento richiede diversi passaggi:

1. Descrivere un algoritmo per risolvere SetPartitioning usando 3WayPartitioning come subroutine. Questo algoritmo avrà la seguente forma: data un'istanza di SetPartitioning, trasformala in un'istanza di 3WayPartitioning, quindi chiama l'algoritmo magico black-box per 3WayPartitioning.
2. Dimostrare che la riduzione è corretta. Ciò richiede sempre due passaggi separati, che di solito hanno la seguente forma:
   - Dimostrare che l'algoritmo trasforma istanze "buone" di SetPartitioning in istanze "buone" di 3WayPartitioning.

- Dimostrare che se la trasformazione produce un'istanza "buona" di 3WayPartitioning, allora era partita da un'istanza "buona" di SetPartitioning.

3. Mostrare che la riduzione funziona in tempo polinomiale, a meno della chiamata (o delle chiamate) all'algoritmo magico black-box per 3WayPartitioning. (Questo di solito è banale.)

Una istanza di SetPartitioning è data da un insieme S di numeri interi da suddividere in due. Una istanza di 3WayPartitioning è data anch'essa da un insieme di numeri interi S'. Quindi la riduzione deve trasformare un insieme di numeri S input di SetPartitioning in un altro insieme di numeri S' che diventerà l'input per la black-box che risolve 3WayPartitioning.

Se $t = \sum_{x \in S} x$ è la somma dei numeri che appartengono ad S, la riduzione che crea S' aggiungendo un nuovo elemento a S di valore $t/2$ (metà della somma dei numeri che appartengono ad S). Dimostriamo che è corretta:

$\Rightarrow$ sia S un'istanza buona di SetPartitioning. Allora è possibile partizionare S in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2. In particolare, sia S_1 che S_2 sommano a $t/2$. Se aggiungiamo il nuovo valore $t/2$ ad S_1 otteniamo una soluzione per 3WayPartitioning, in cui i tre insiemi sono S_1, S_2 e $S_3 = \{t/2\}$, e abbiamo dimostrato che S' è una istanza buona di 3WayPartitioning.

$\Leftarrow$ sia S' un'istanza buona di 3WayPartitioning. Allora è possibile partizionare S' in tre sottoinsiemi S_1, S_2 ed S_3 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 che è uguale alla somma dei numeri in S_3. Per come è definito S', abbiamo che ognuno dei tre insiemi somma a $t/2$. Di conseguenza uno dei tre insiemi, che possiamo chiamare S_3, contiene solamente il nuovo elemento $t/2$, mentre gli altri due, che chiamiamo S_1 e S_2, formano una partizione degli elementi dell'insieme S di partenza. Di conseguenza, abbiamo trovato una soluzione per SetPartitioning con input S.

Per completare la dimostrazione basta osservare che per costruire S' dobbiamo aggiungere un nuovo elemento ad S, operazione che si riesce a fare in tempo polinomiale.