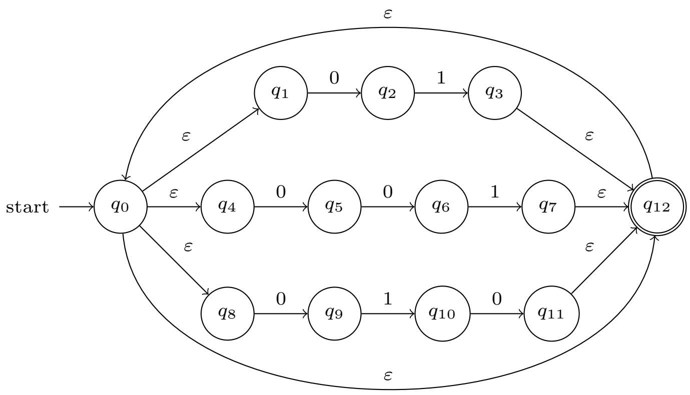
Automi e Linguaggi Formali
Esame del 20 Settembre 2018

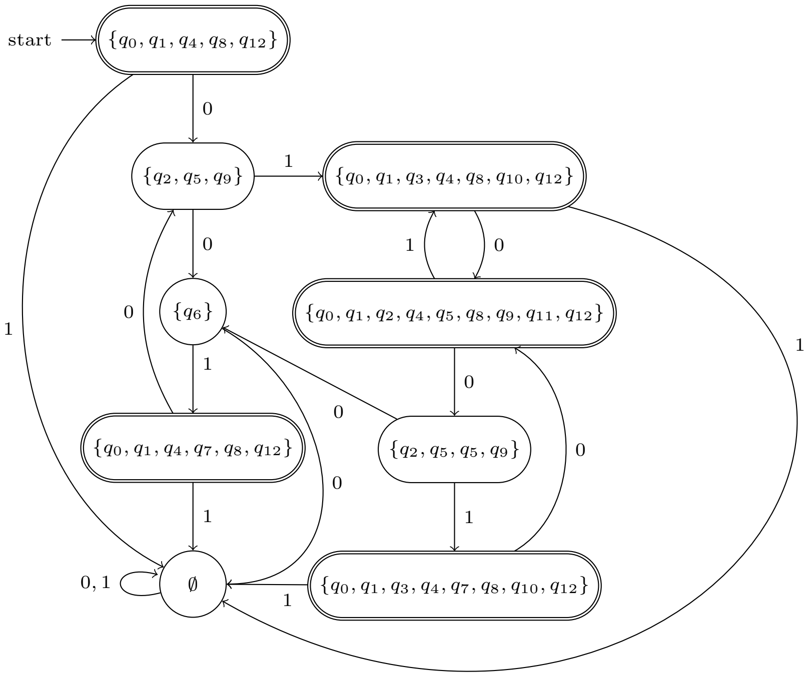
Tempo a disposizione: 2 h 30 m

1. (a) Convertire l'espressione regolare $(\mathbf{01} + \mathbf{001} + \mathbf{010})^*$ in un ε-NFA (si può usare l'algoritmo visto a lezione oppure creare direttamente l'automa). **Importante:** l'automa deve essere nondeterministico e deve sfruttare le ε-transizioni per riconoscere il linguaggio.

L'esercizio ha molte possibili soluzioni, una delle quali è la seguente:



(b) Trasformare l'ε-NFA ottenuto al punto precedente in un DFA.



2. Considerate il linguaggio $L = \{www \mid w \in \{a, b\}^*\}$. Questo linguaggio è regolare? Dimostrare formalmente la risposta.

Il linguaggio non è regolare. Supponiamo per assurdo che lo sia:

- *sia h la lunghezza data dal Pumping Lemma;*
- *consideriamo la parola $v = a^h ba^h ba^h b$, che appartiene ad L ed è di lunghezza maggiore di h;*
- *sia $v = xyz$ una suddivisione di v tale che $y \neq \varepsilon$ e $|xy| \leq h$;*
- *poiché $|xy| \leq h$, allora xy è completamente contenuta nel prefisso a^h di v, e quindi sia x che y sono composte solo da a. Inoltre, siccome $y \neq \varepsilon$, possiamo dire che $y = a^p$ per qualche valore $p > 0$. Allora la parola xy^2z è nella forma $a^{2h+p}ba^h ba^h b$, e quindi non appartiene al linguaggio perché non è possibile suddividerla in tre sottostringhe uguali tra di loro.*

Abbiamo trovato un assurdo quindi L non può essere regolare.

3. (a) Se i sta per la keyword if ed e sta per la keyword else, allora si chiede di definire una CFG capace di generare tutte le stringhe che rappresentano comandi if e if-else annidati e concatenati, come per esempio iii, iiee, iie, ieieiie e anche iieeiiee.

Una CFG che genera il linguaggio richiesto è $S \to iS \mid iSeS \mid \epsilon$. *E' abbasanza facile convincersi che è capace di generare qualsiasi sequenza di if-else annidati.*

(b) Riuscite a spiegare qual'è la regola che i compilatori dei linguaggi di programmazione usano per associare ogni else ad un if nelle stringhe del punto precedente?

La regola è che ogni "else" viene assocoato al prim "if" libero alla sua sinistra. Quindi la stringa iie è interpretata come i (ie).

(c) La vostra grammatica del punto (a) è ambigua o no? Argomentate la risposta. In caso pensiate che sia ambigua, è possibile trovare un'altra CFG che generi lo stesso linguaggio e che non sia ambigua? Argomentate la risposta e se pensate che sia possibile, allora date una tale CFG non ambigua

La grammatica è ambigua. Ci sono 2 derivazioni left-most per generare iie:

$S \Rightarrow^{lm} iS \Rightarrow^{lm} iiSeS$ *e dopo i 2 S scompaiono in* ϵ

e la seconda derivazione è:

$S \Rightarrow^{lm} iSeS \Rightarrow^{lm} iiSeS$ *e di nuono i due S diventano* ϵ.

La grammatica può essere resa non ambigua come segue: $S \to iS \mid iS'eS \mid \epsilon$ *e* $S' \to iS'eS' | \epsilon$.

4. Descrivere un PDA che accetta per pila vuota ed è capace di riconoscere il linguaggio $L = \{(ab)^n(ca)^n \mid n \geq 1\}$. Il vostro è un automa deterministico o nondeterministico? Spiegare la risposta.

Il PDA ha 2 stati q1 e q2 e le seguenti transizioni:

$\delta(q1, a, Z0) = \{((q1, aZ0)\}, \delta(q1, b, a) = \{((q1, ba)\}, \delta(q1, a, b) = \{((q1, ab)\}, \delta(q1, c, b) = \{((q2, \epsilon)\}$,

$\delta(q2, c, b) = \{((q2, \epsilon)\}, \delta(q2, a, a) = \{((q2, \epsilon)\}, \delta(q2, \epsilon, Z0) = \{((q2, \epsilon)\}$.

Il PDA è chiaramente deterministico. L'ultima transizione, svuota la pila e, se l'input è consumato, accetta.

5. Dare la definizione del linguaggio L_U e spiegare in dettaglio come si dimostra che L_U è un linguaggio RE e non ricorsivo.

$L_U = \{(M, w) \mid w \in L(M)\}$. *Esiste una macchian di Turing capace di accettare* L_U. *Questa macchina di Turing ha diversi nastri e prende in input una qualsiasi coppia* (M, w), *scrive* M *su uno dei nastri, poi scrive* w *su un altro e* $q0 = 0$ *sul terzo e simula il calcolo di* M *su* w *cercando le mosse sul primo nastro. Insomma sfrutta l'idea della macchina universale, che, avendo una qualsiasi macchina di Turing su un nastro, è capace di simulare il suo calcolo. Da questo argomento segue che* L_U *è RE. Per dimostrare che* L_U *non è ricorsivo, usiamo il fatto che, se lo fosse, allora anche* $\hat{L_U}$ *lo sarebbe e, visto che potremmo ridurre* L_d *a* $\hat{L_U}$, *avremmo un assurdo. Per capire la riduzione è sufficiente capire che* $\hat{L_U} = \{(M, w) \mid w \notin L(M)\}$. L_d *ha come istanza una qualsiasi macchina di Turing* M. *Da una tale istanza di* L_d, *la corrispondente istanza di* $\hat{L_U}$ *è semplicemente* (M, M). *Ovviamente se* $\hat{L_U}$ *fosse ricorsivo, sarebbe possibile determinare se* $M \notin L(M)$ *e quindi sarebbe possibile decidere* L_d. *Assurdo.*

6. 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 il seguente problema, che chiameremo SubsetSum: dato un insieme di numeri interi S ed un valore obiettivo t, stabilire se esiste un sottoinsieme $S' \subseteq S$ tale che la somma dei numeri in S' è uguale a t.

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

Il certificato per SubsetSum *è dato dal sottoinsieme* S'. *Occorre verificare che ogni elemento di* S' *appartenga anche ad* S *e che la somma dei numeri contenuti in* S' *sia uguale a* t.

(b) Mostrare come si può risolvere il problema SetPartitioning usando il problema SubsetSum come sottoprocedura.

Dato un qualsiasi insieme di numeri interi S *che è un'istanza di* SetPartitioning, *costruiamo in tempo polinomiale un'istanza di* SubsetSum. *L'insieme di numeri interi di input rimane* S, *mentre il valore obiettivo* t *è uguale alla metà della somma degli elementi contenuti in* S.

In questo modo, se S' è una soluzione di SubsetSum*, allora la somma dei valori contenuti nell'insieme $S - S'$ sarà pari alla metà della somma degli elementi di S. Quindi ho diviso S in sue sottoinsiemi $S_1 = S'$ e $S_2 = S - S'$ tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2.*
Viceversa, se S_1 e S_2 sono una soluzione di SetPartitioning*, allora la somma dei numeri contenuti in S_1 sarà uguale alla somma dei numeri in S_2, e quindi uguale alla metà della somma dei numeri contenuti in S. Quindi sia S_1 che S_2 sono soluzione di* SubsetSum *per il valore obiettivo t specificato sopra.*