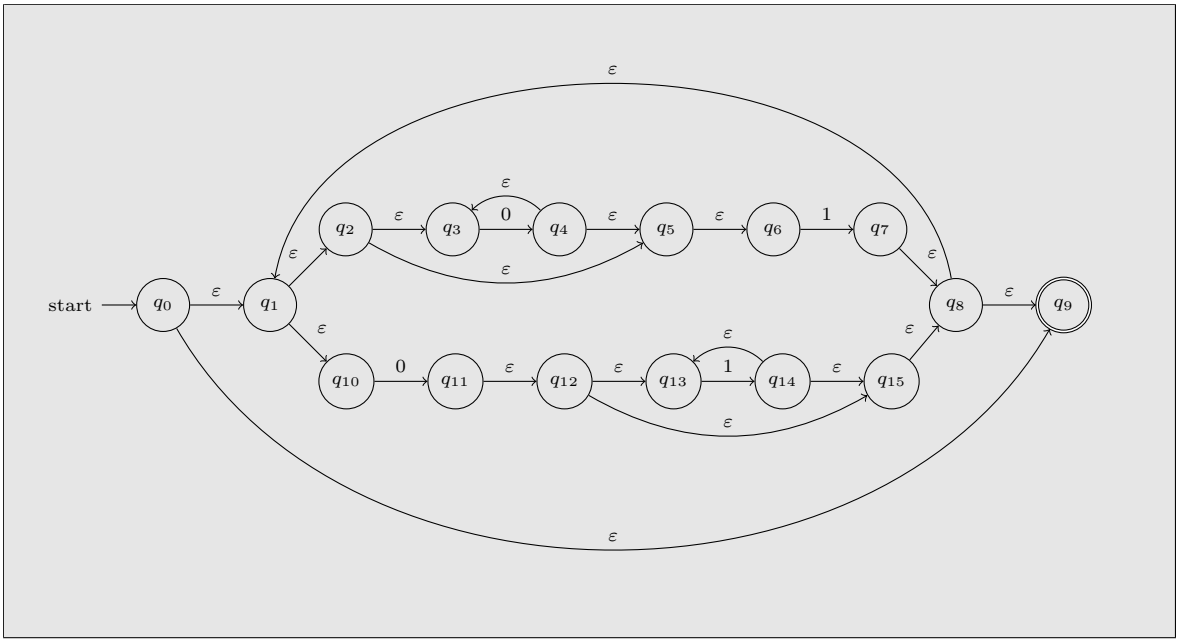
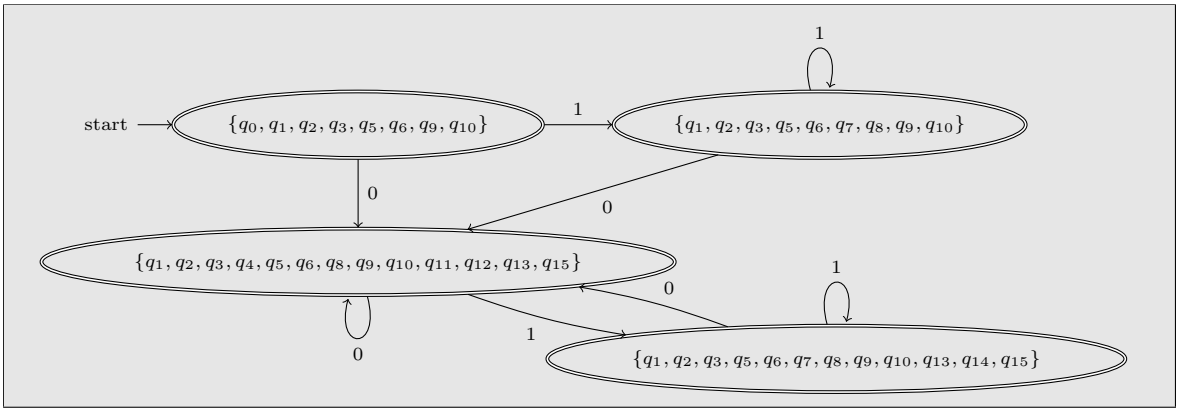
Automi e Linguaggi Formali
Esame del 12 Giugno 2018 – Parte I

Tempo a disposizione: 1 h 30 min

1. (a) Convertire l'espressione regolare $(\mathbf{0}^*\mathbf{1} + \mathbf{01}^*)^*$ in un ε-NFA usando le regole viste a lezione.



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



2. (a) Dimostrare che il linguaggio $L_1 = \left\{0^{2n}1^n : n \geq 0\right\}$ *non* è regolare.

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

- sia h la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 0^{2h}1^h$, che appartiene ad L_1 ed è di lunghezza maggiore di h;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq h$;
- poiché $|xy| \leq h$, allora xy è completamente contenuta nel prefisso 0^{2h} di w, e quindi sia x che y sono composte solo da 0. Inoltre, siccome $y \neq \varepsilon$, possiamo dire che $y = 0^p$ per qualche valore $p > 0$. Allora la parola xy^2z è nella forma $0^{2h+p}1^h$, e quindi non appartiene al linguaggio perché il numero di 0 non è uguale al doppio del numero di 1 (dovrebbero essere $h + p/2$ mentre sono solo h).

Abbiamo trovato un assurdo quindi L_1 non può essere regolare.

(b) Considerate il linguaggio $L_2 = \{0^m 1^n : m \neq 2n\}$. Questo linguaggio è regolare? Giustificare formalmente la risposta (*la giustificazione non dovrebbe richiedere più di due righe di testo*).

Si può osservare che L_2 è il complementare del linguaggio L_1 (contiene tutte e sole le parole che non appartengono a L_1). Al punto precedente abbiamo dimostrato che L_1 non è regolare, quindi nemmeno L_2 può essere regolare perché i linguaggi regolari sono chiusi per complementazione.

3. Sia L un linguaggio regolare su un alfabeto Σ. Supponete che il simbolo # appartenga all'alfabeto Σ e dimostrate che il seguente linguaggio è regolare:

$$dehash(L) = \{dehash(w) : w \in L\}$$

dove $dehash(w)$ è la stringa che si ottiene eliminando tutti i simboli # da w.

Per dimostrare che $dehash(L)$ è regolare vediamo come è possibile costruire un automa a stati finiti che riconosce $dehash(L)$ a partire dall'automa a stati finiti che riconosce L.

Sia quindi $A = (Q, \Sigma, q_0, \delta, F)$ un automa a stati finiti che riconosce il linguaggio L. Costruiamo un ε-NFA $B = (Q, \Sigma, q_0, \delta_B, F)$ che ha lo stesso insieme di stati, lo stesso stato iniziale e gli stessi stati finali di A. La funzione di transizione del nuovo automa rimpiazza ogni transizione etichettata con # di A con una ε-transizione tra la stessa coppia di stati, lasciando inalterate le transizioni etichettate con gli altri simboli di Σ.

4. Si consideri la seguente grammatica libera da contesto G:

$$S \to iS \mid iSeS \mid \epsilon$$

(a) dare una descrizione del linguaggio generato da G nella forma $L = \{w \mid w \in \{i, e\}^* \text{ tali che } \ldots\}$ e dimostrare che vale $L \supseteq L(G)$; (opzionale: spiegare anche che vale $L \subseteq L(G)$)

$L = \{w \in \{i, e\}^* \mid$ per ogni prefisso di w il numero di i è maggiore o uguale al numero di $e\}$
Dimostriamo che $L \supseteq L(G)$ per induzione *sulla lunghezza della derivazione.*
Base: lunghezza 1. In questo caso l'unica produzione è $S \Rightarrow \epsilon$. Poiché $\epsilon \in L$ la tesi è dimostrata.
Passo induttivo: lunghezza $n + 1$. Assumiamo per ipotesi induttiva che la tesi sia vera per tutte le derivazioni di lunghezza minore o uguale a n.
La derivazione di lunghezza $n + 1$ può essere fatta in due modi:

- $S \Rightarrow iS \Rightarrow^n iw' = w$. Per ipotesi induttiva $w' \in L$. Poiché aggiungo una i in più, la proprietà di bilanciamento del numero di i e di e rimane vera anche per iw' e quindi ho dimostrato che $w \in L$.
- $S \Rightarrow iSeS \Rightarrow^* iw'ew'' = w$. Per ipotesi induttiva w' e $w'' \in L$. Quindi la proprietà di bilanciamento del numero di i e di e rimane vera anche per iw' e per $iw'e$, e di conseguenza anche per $iw'ew''$. Quindi ho dimostrato che $w \in L$.

(b) dimostrare che la grammatica è ambigua;

G è ambigua perché posso derivare la parola iie in due modi diversi:

- $S \Rightarrow iSeS \Rightarrow iiSeS \Rightarrow^* iie$
- $S \Rightarrow iS \Rightarrow iiSeS \Rightarrow^* iie$

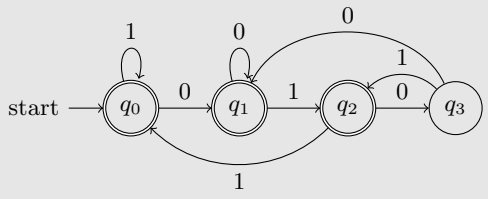
(c) osservando che questa grammatica modella l'annidamento di $if - then$ e $if - then - else$ nei programmi, fornire una grammatica non ambigua che generi lo stesso linguaggio della grammatica di partenza. Spiegare l'idea alla base della nuova grammatica.

$S \to iS \mid iS'eS \mid \epsilon$
$S' \to iS'eS' \mid \epsilon$

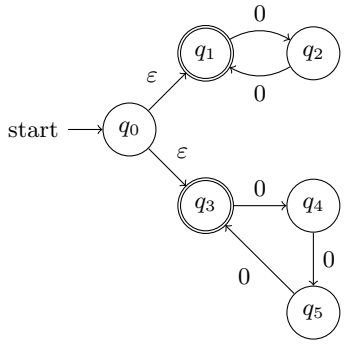
Automi e Linguaggi Formali
Esame del 5 Luglio 2018 – Parte I

Tempo a disposizione: 1 h 30 min

1. Considerate il linguaggio di tutte le parole sull'alfabeto $\{0,1\}$ che *non* terminano con 010 (incluse ε e le parole di lunghezza < 3). Definire un'espressione regolare oppure un automa a stati finiti che riconoscano questo linguaggio.



2. Considerate il seguente ε-NFA che riconosce stringhe sull'alfabeto $\Sigma = \{0,1\}$:

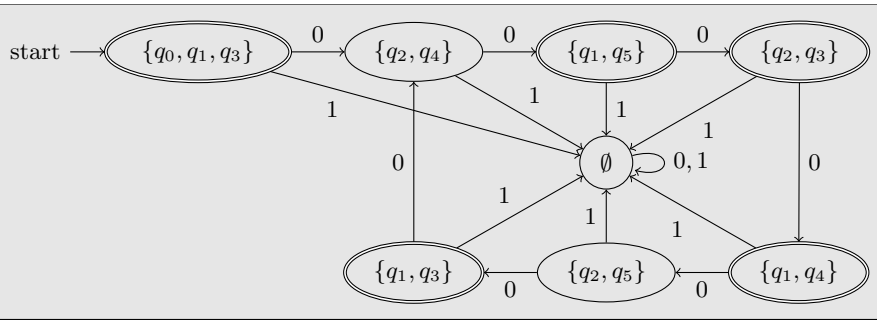


(a) Descrivete il linguaggio riconosciuto dall'automa.

L'automa riconosce il linguaggio di tutte le sequenze di zeri di lunghezza divisibile per 2 o per 3:

$$L = \{0^n \mid n \text{ è divisibile per 2 o per 3}\}$$

(b) Convertite l'automa in un DFA equivalente.



3. Considerate il linguaggio $L_1 = \{0^n 1^m 0^m : n, m > 0\}$. 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 $w = 01^h0^h$, che appartiene ad L_1 ed è di lunghezza maggiore di h;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq h$;
- poiché $|xy| \leq h$, allora xy è completamente contenuta nel prefisso 01^h di w. Ci sono due casi possibili.

- Se $x \neq \varepsilon$ allora y è composta solo da 1. Inoltre, siccome $y \neq \varepsilon$, possiamo dire che $y = 1^p$ per qualche valore $p > 0$. Allora la parola xy^2z è nella forma $01^{h+p}0^h$, e quindi non appartiene al linguaggio perché il numero di 1 è diverso dal numero di 0 nell'ultima parte della parola.
- Se $x = \varepsilon$ allora, siccome $y \neq \varepsilon$, possiamo dire che $y = 01^p$ per qualche valore $p \geq 0$. Notate in questo caso lo zero iniziale è compreso in y (perché x è vuota). Allora la parola xy^0z è nella forma $1^{h-p}0^h$, e quindi non appartiene al linguaggio perché non inizia con 0, mentre tutte le parole di L_1 devono iniziare con 0 perché $n > 0$.

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

4. Sia L un linguaggio regolare su un alfabeto Σ e dimostrate che il seguente linguaggio è regolare:

$$\mathit{suffixes}(L) = \{y \mid xy \in L \text{ per qualche stringa } x \in \Sigma^*\}$$

Intuitivamente, $\mathit{suffixes}(L)$ è il linguaggio di tutti i suffissi delle parole che stanno in L.

Per dimostrare che $\mathit{suffixes}(L)$ è regolare basta mostrare come costruire un automa a stati finiti che riconosce $\mathit{suffixes}(L)$ a partire dall'automa a stati finiti che riconosce L.

Sia quindi $A = (Q, \Sigma, q_0, \delta, F)$ un automa a stati finiti che riconosce il linguaggio L. Costruiamo un ε-NFA $B = (Q \cup \{q_0'\}, \Sigma, q_0', \delta_B, F)$ che ha uno stato in più di A. Chiamiamo q_0' questo nuovo stato e gli assegnamo il ruolo di stato iniziale di B. La funzione di transizione del nuovo automa aggiunge una ε-transizione dal nuovo stato iniziale q_0' verso ogni stato di Q che è raggiungibile dal vecchio stato iniziale. Il resto delle transizioni rimane invariato. Gli stati finali sono gli stessi di A.

5. Costruire una CFG che genera il linguaggio

$$L = \{a^n b^m c^k \mid \text{con } n = m \text{ o } m = k \text{ e } n, m, k \geq 0\}.$$

$$\begin{aligned} S &\to S_1 C \mid A S_2 \\ S_1 &\to \varepsilon \mid a S_1 b \\ S_2 &\to \varepsilon \mid b S_2 c \\ A &\to \varepsilon \mid aA \\ C &\to \varepsilon \mid cC \end{aligned}$$

Automi e Linguaggi Formali
Compitino del 9 Aprile 2018

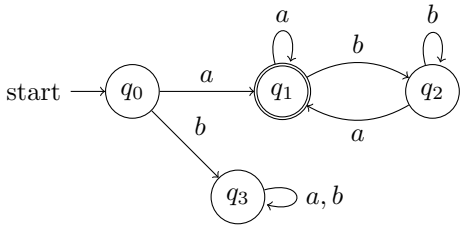
Soluzione della Parte I – Linguaggi Regolari

1. Considerare il linguaggio $L = \{\text{stringhe di } a \text{ e } b \text{ che iniziano con } a \text{ e finiscono con } a\}$

(a) Dare un automa a stati finiti *deterministico* che accetti il linguaggio L.
(b) Dare un'espressione regolare che rappresenti il linguaggio L.

Soluzione:

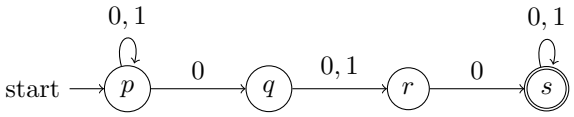
(a)



(b) Alcune soluzioni possibili:

$$\mathbf{a}(\mathbf{a}+\mathbf{b})^*\mathbf{a}+\mathbf{a} \qquad\qquad \mathbf{a}(\mathbf{b}^*\mathbf{a})^*$$

2. Dato il seguente NFA

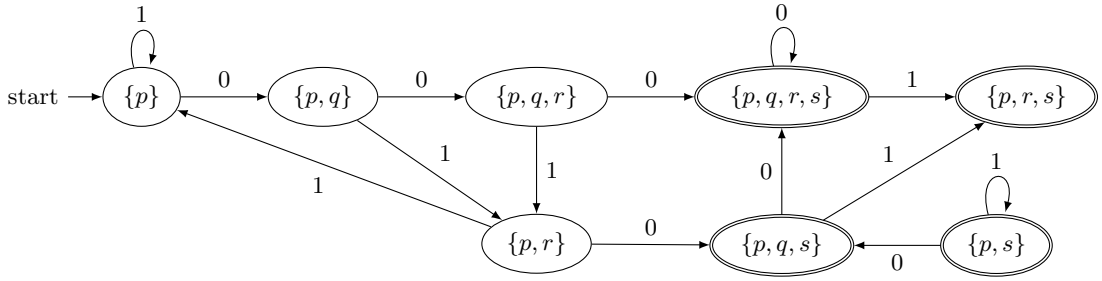


costruire un DFA equivalente

Soluzione: Applicando la costruzione a sottoinsiemi si ottiene il DFA con la seguente tabella di transizione (dove gli stati non raggiungibili dallo stato iniziale $\{p\}$ sono omessi):

	0	1
$\to \{p\}$	$\{p,q\}$	$\{p\}$
$\{p,q\}$	$\{p,q,r\}$	$\{p,r\}$
$\{p,r\}$	$\{p,q,s\}$	$\{p\}$
$\{p,q,r\}$	$\{p,q,r,s\}$	$\{p,r\}$
$*\{p,q,s\}$	$\{p,q,r,s\}$	$\{p,r,s\}$
$*\{p,r,s\}$	$\{p,q,s\}$	$\{p,s\}$
$*\{p,s\}$	$\{p,q,s\}$	$\{p,s\}$
$*\{p,q,r,s\}$	$\{p,q,r,s\}$	$\{p,r,s\}$

e con il seguente diagramma di transizione:



3. Il linguaggio

$$L = \left\{a^n b^m c^{n-m} : n > m > 0\right\}$$

è regolare? Motivare in modo formale la risposta.

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

- sia h la lunghezza data dal Pumping Lemma; possiamo supporre senza perdita di generalità che $h > 1$;
- consideriamo la parola $w = a^h b c^{h-1}$, che appartiene ad L ed è di lunghezza maggiore di h;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq h$;
- poiché $|xy| \leq h$, allora xy è completamente contenuta nel prefisso a^h di w, 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 $a^{h+p} b c^{h-1}$, e quindi non appartiene al linguaggio perché il numero di c non è uguale al numero di a meno il numero di b (dovrebbero essere $h + p - 1$ mentre sono solo $h - 1$).

Abbiamo trovato un assurdo quindi L non può essere regolare.

4. Sia L un linguaggio regolare su un alfabeto Σ. Dimostrare che anche il seguente linguaggio è regolare:

$$init(L) = \{w \in \Sigma^* : \text{ esiste } x \in \Sigma^* \text{ tale che } wx \in L\}$$

Soluzione: Per dimostrare che $init(L)$ è regolare vediamo come è possibile costruire un automa a stati finiti che riconosce $init(L)$ a partire dall'automa a stati finiti che riconosce L.

Sia quindi $A = (Q, \Sigma, q_0, \delta, F)$ un DFA che riconosce il linguaggio L. Costruiamo il DFA $B = (Q, \Sigma, q_0, \delta, G)$ che ha gli stessi stati, le stesse transizioni e lo stesso stato iniziale di A. Definiamo l'insieme G degli stati finali del nuovo automa come $G = \{q \in Q :$ esiste una sequenza di transizioni da q ad uno stato finale $f \in F\}$, ossia come tutti gli stati a partire dai quali possiamo raggiungere uno stato finale di A. Dobbiamo dimostrare che $L(B) = init(L)$.

- Sia $w \in init(L)$: allora per la definizione deve esistere $x \in \Sigma^*$ tale che $wx \in L$. Poiché A è un automa deterministico, esiste una sola sequenza di transizioni che parte da q_0 e accetta la parola wx in A. Possiamo spezzare questa sequenza in due parti: una prima sequenza che parte da q_0, legge w e arriva in uno stato intermedio q, e una seconda sequenza che parte da q, legge x e arriva ad uno stato finale $f \in F$. Ma allora lo stato intermedio q deve appartenere agli stati finali G di B! Quindi la parola w viene accettata dall'automa B.
- Prendiamo ora una parola $w \in L(B)$. Poiché B è un automa deterministico, esiste una sola sequenza di transizioni che parte da q_0 e accetta la parola w arrivando ad uno stato finale $q \in G$. Per la definizione di G, esiste una sequenza di transizioni che porta da q ad uno stato finale f di A. Quindi deve esistere una parola x i cui simboli etichettano le transizioni della sequenza da q a f. Ma allora possiamo creare una sequenza di transizioni da q_0 a f che riconosce la parola wx, che quindi appartiene a L. Dalla definizione di $init(L)$ segue che $w \in init(L)$.

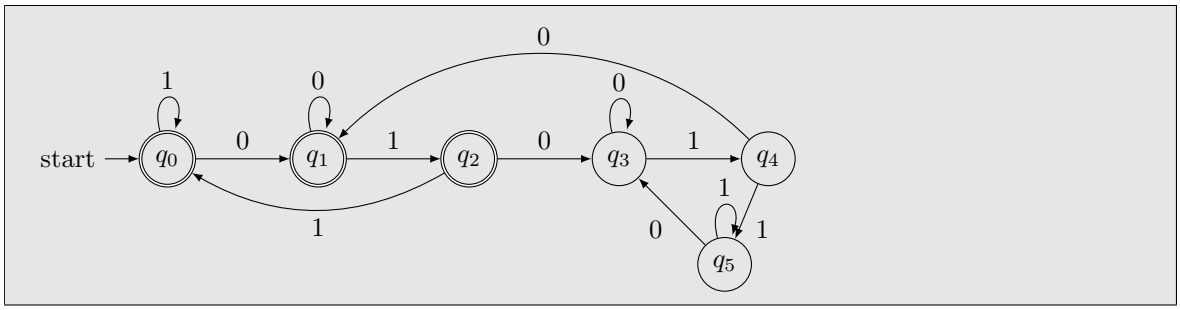
Automi e Linguaggi Formali
Esame del 19 Giugno 2019 – Parte I

Tempo a disposizione: 1 h 30 min

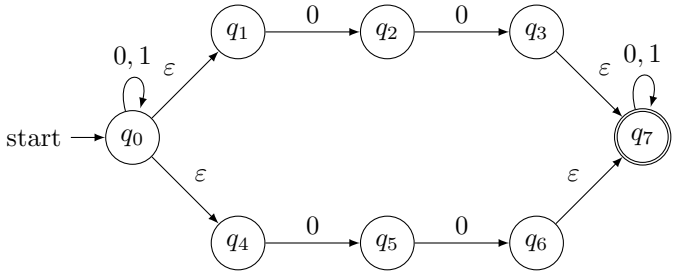
1. Definire un automa a stati finiti (di qualsiasi tipologia) che riconosca il linguaggio

$$L = \{w \in \{0,1\}^* \mid w \text{ contiene un numero pari di occorrenze della sottostringa } 010\}$$

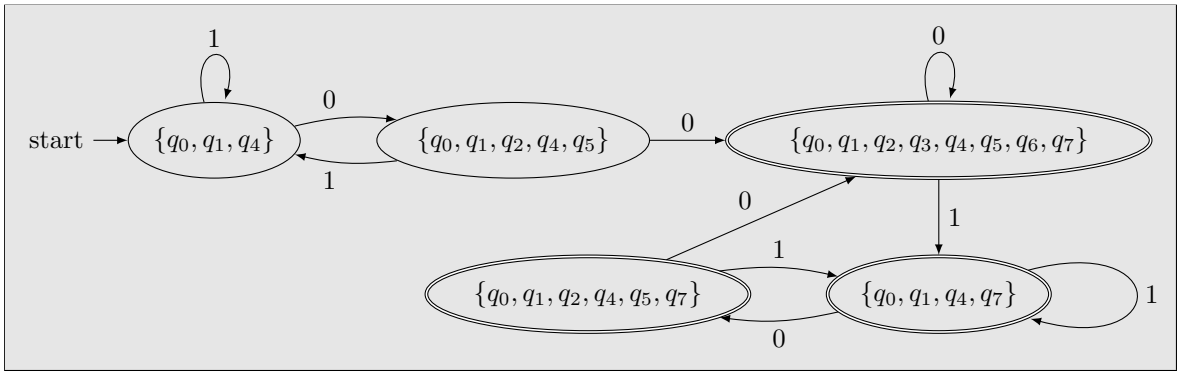
Per esempio, la parola 0100010 appartiene al linguaggio perché contiene due occorrenze di 010, la parola 01111 appartiene al linguaggio perché contiene zero occorrenze di 010, mentre la parola 0101010 non appartiene al linguaggio perché contiene tre occorrenze di 010.



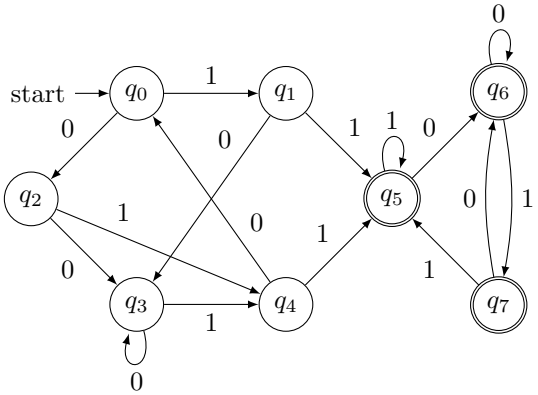
2. Dato il seguente ε-NFA



costruire un DFA equivalente. Dare solo la parte del DFA che è raggiungibile dallo stato iniziale.



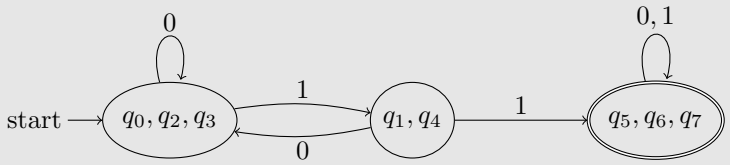
3. Minimizzare il seguente DFA usando l'algoritmo riempi-tabella.



L'esecuzione dell'algoritmo riempi-tabella porta alla seguente tabella finale:

	q_0	q_1	q_2	q_3	q_4	q_5	q_6
q_1	X						
q_2		X					
q_3		X					
q_4	X		X	X			
q_5	X	X	X	X	X		
q_6	X	X	X	X	X		
q_7	X	X	X	X	X		

Fondendo i tre blocchi stati equivalenti $q_0 \equiv q_2 \equiv q_3$, $q_1 \equiv q_4$ e $q_5 \equiv q_6 \equiv q_7$ otteniamo il DFA minimo con tre stati:



4. Sia $\Sigma = \{0, 1\}$ e considerate il linguaggio

$$M3N = \{0^m 1^n \mid m \leq 3n\}$$

(a) Completate il seguente schema di partita per il Gioco del Pumping Lemma in modo da far vincere il Giocatore 2:

Giocatore 1: sceglie il valore di $h = \mathbf{2}$

Giocatore 2: sceglie la parola $w \in M3N$ di lunghezza maggiore di h

$$w = \mathbf{00000011}$$

Giocatore 1: suddivide w in

- $x = \mathbf{0}$
- $y = \mathbf{0}$
- $z = \mathbf{000011}$

rispettando le condizioni che $|xy| \leq h$ e $y \neq \varepsilon$

Giocatore 2: sceglie una potenza $k = \mathbf{2}$

La parola $xy^k z = \mathbf{0000000111} \notin M3N$: vince il **Giocatore 2**

(b) Dimostrate che $M3N$ non è un linguaggio regolare usando il Pumping Lemma.

Supponiamo per assurdo che $M3N$ sia regolare. Di conseguenza, $M3N$ deve rispettare il Pumping Lemma.

- sia h la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 0^{3h}1^h$, che appartiene ad $M3N$ ed è di lunghezza maggiore di h;
- sia $w = xyz$ una suddivisione arbitraria di w tale che $y \neq \varepsilon$ e $|xy| \leq h$;
- poiché $|xy| \leq h$, allora xy è completamente contenuta nel prefisso 0^{3h} di w posto prima della sequenza di 1, e quindi sia x che y sono composte solo da 0. Inoltre, siccome $y \neq \varepsilon$, possiamo dire che $y = 0^p$ per qualche valore $p > 0$. Allora la parola xy^2z è nella forma $0^{3h+p}1^h$, e non appartiene al linguaggio perché il numero di 0 nel prefisso è maggiore del triplo del numero di 1 nel suffisso.

Abbiamo trovato un assurdo: $M3N$ non è un linguaggio regolare.

5. Si consideri la seguente grammatica CF, G: $S \to aA\ ,\ A \to Bb \mid b,\ B \to aA \mid \epsilon$

(a) Descrivere il linguaggio $L(G)$ in termini di un linguaggio L composto da "stringhe w con certe proprietà".

(b) Dimostrare per induzione che tutte le stringhe in $L(G)$ sono in L, cioè che $L(G) \subseteq L$.

$L(S) = \{a^n b^n | n > 0\}$. Dimostriamo per induzione sulla lunghezza della derivazione che la seguente tesi è vera:

$L(A) = \{a^n b^{n+1} | n \geq 0\}$ e $L(B) = \{a^n b^n | n \geq 0\}$

Base per $L(A)$: lunghezza 1. $A \Rightarrow b$, soddisfa la tesi

Base per $L(B)$: lunghezza 1. $B \Rightarrow \epsilon$, soddisfa la tesi.

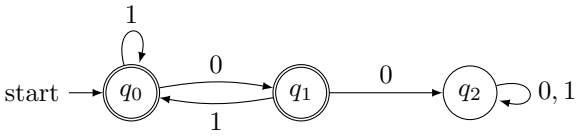
Step per $L(A)$ e per $L(B)$. Supponiamo che la tesi sia vera per derivazioni da A e da B di $n > 0$ passi. Dimostriamo che allora vale per derivazioni di n+1 passi. Consideriamo una derivazione di n+1 passi che inizia con A, allora il primo passo è $A \Rightarrow Bb$ e dopo in n passi, per ipotesi induttiva, $B \Rightarrow^n a^k b^k$ per $k \geq 0$, e quindi $A \Rightarrow^{n+1} a^k b^{k+1}$. Se prendiamo una derivazione di n+1 passi che inizia con B, il primo passo sarà $B \Rightarrow aA$ e di nuovo per ipotesi induttiva, A in n passi produce $a^k b^{k+1}$ per $k \geq 0$. Per cui l'intera derivazione diventa $B \Rightarrow^{n+1} a^{k+1} b^{k+1}$, per $k \geq 0$.

Da quanto appena dimostrato, segue che da S le derivazioni di qualsiasi lunghezza producono $S \Rightarrow aA \Rightarrow^* a^k b^k$, per $k > 0$.

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

Tempo a disposizione: 1 h 30 min

1. Considerate il DFA che riconosce il linguaggio di tutte le stringhe sull'alfabeto $\{0,1\}$ che *non contengono* la sottostringa 00:



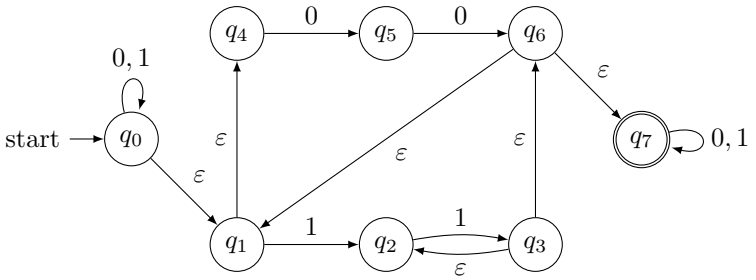
Trasformate il DFA in una Espressione Regolare usando l'algoritmo di eliminazione degli stati.

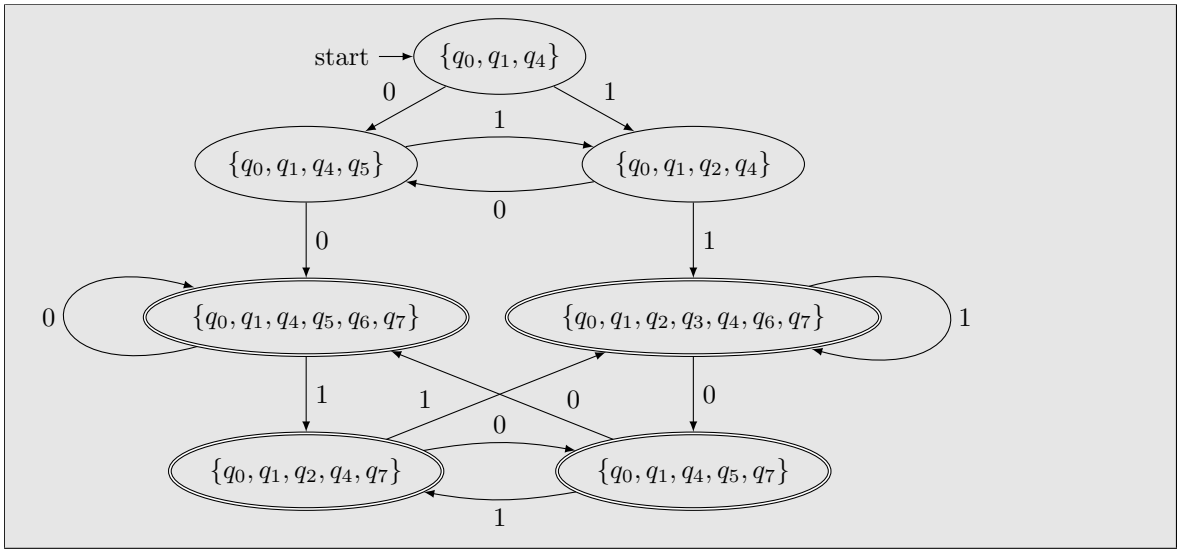
La soluzione ottenuta con l'algoritmo di eliminazione degli stati è $(1+01)^* + (1+01)^*0$, che può essere riscritta in forma più compatta come $(1+01)^*(0+\varepsilon)$

2. Usate la soluzione dell'esercizio precedente per creare una Espressione Regolare che definisca il linguaggio di tutte le stringhe sull'alfabeto $\{0,1\}$ tali che tutte le occorrenze di 11 appaiono prima di tutte le occorrenze di 00.

La soluzione di questo esercizio si basa sull'osservazione che ogni parola in cui le occorrenze di 11 appaiono prima di tutte le occorrenze di 00 può essere suddivisa in due parti: una prima parte in cui non compare mai 00 (e può comparire 11) seguita da una seconda parte in cui non compare mai 11 (e può comparire 00). Di conseguenza, l'espressione regolare ottenuta nell'esercizio 1 può essere usata per descrivere la prima parte della stringa, mentre la sua versione "duale" in cui si scambiano 0 e 1 si può usare per la seconda parte della parola: $(1+01)^*(0+\varepsilon)(0+10)^*(1+\varepsilon)$, che si può semplificare in $(1+01)^*(0+10)^*(1+\varepsilon)$.

3. Trasformate il seguente ε-NFA in DFA:





4. Sia $\Sigma = \{0, 1\}$ e considerate il linguaggio

$$LMN = \{0^\ell 1^m 0^n \mid \ell < n\}$$

(a) Completate il seguente schema di partita per il Gioco del Pumping Lemma in modo da far vincere il Giocatore 2:

Giocatore 1: sceglie il valore di $h = \mathbf{4}$

Giocatore 2: sceglie la parola $w \in LMN$ di lunghezza maggiore di h

$$w = \mathbf{0000011000000}$$

Giocatore 1: suddivide w in

- $x = \mathbf{0}$
- $y = \mathbf{00}$
- $z = \mathbf{0011000000}$

rispettando le condizioni che $|xy| \leq h$ e $y \neq \varepsilon$

Giocatore 2: sceglie una potenza $k = \mathbf{2}$

La parola $xy^kz = \mathbf{000000011000000} \notin LMN$: vince il **Giocatore 2**

(b) Dimostrate che LMN non è un linguaggio regolare usando il Pumping Lemma.

Supponiamo per assurdo che LMN sia regolare. Di conseguenza, LMN deve rispettare il Pumping Lemma.

- sia h la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 0^h 10^{h+1}$, che appartiene ad LMN ed è di lunghezza maggiore di h;
- sia $w = xyz$ una suddivisione arbitraria di w tale che $y \neq \varepsilon$ e $|xy| \leq h$;
- poiché $|xy| \leq h$, allora xy è completamente contenuta nel prefisso 0^h di w posto prima dell'1 di separazione, e quindi sia x che y sono composte solo da 0. Inoltre, siccome $y \neq \varepsilon$, possiamo dire che $y = 0^p$ per qualche valore $p > 0$. Allora la parola xy^2z è nella forma $0^{h+1}10^{h+1}$, e non appartiene al linguaggio perché la lunghezza della prima sequenza di 0 è uguale alla lunghezza della seconda sequenza di 0.

Abbiamo trovato un assurdo: LMN non è un linguaggio regolare.

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

(a) Descrivere il linguaggio $L(G)$ in termini di un linguaggio L composto da "stringhe w con certe proprietà".

Se sostituiamo a con (e b con) è facile vedere che $L(B)$contiene tutte le stringhe con parentesi bilanciate. Formalmente $w \in L(B)$ sse $w = \varepsilon$ oppure $w = (w')w''$ con w' e $w'' \in L(B)$. $S \to aBb$ semplicemente aggiunge una a all'inizio ed una b alla fine delle stringhe generate da B. In questo modo $L(S)$ ha la proprietà del prefisso.

(b) Dimostrare per induzione che tutte le stringhe in $L(G)$ sono in L, cioè che $L(G) \subseteq L$.

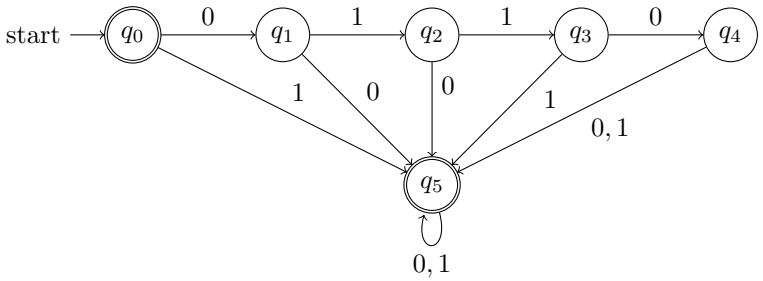
Dimostriamo innanzitutto la seguente tesi: $L(B)$ contiene solo stringhe ben parentesizzate (se sostituiamo a con (e b con)). Usiamo un'induzione sulla lunghezza della derivazione.
Base. Lunghezza 1: $B \Rightarrow \epsilon$ ed ϵ è ben parentesizzata.
Step. Supponiamo che la tesi sia vera per derivazioni di lunghezza minore o uguale a $n \geq 1$ e dimostriamo che allora vale per derivazioni di lunghezza $n+1$. Sia $B \Rightarrow^{n+1} w$, allora il primo passo della derivazione può essere o $B \to (B)$ oppure $B \to BB$. Consideriamo il primo caso: $B \Rightarrow (B) \Rightarrow^n (w')$ e quindi $B \Rightarrow^n w'$, e allora, per ipotesi induttiva, w' è per parentesizzata e quindi (w') è ben parentesizzata. Consideriamo l'altro caso: $B \Rightarrow BB \Rightarrow^n w'w''$, ma allora $B \Rightarrow^k w'$ e $B \Rightarrow^s w''$ con $k + s = n$ ed entrambe maggiori di 0. Quindi, k ed s sono minori di n per cui, per ipotesi induttiva, w' e w'' sono ben parentesizzate, per cui anche la loro concatenazione è ben parentesizzata.

Automi e Linguaggi Formali
Esame del 19 Settembre 2019 – Parte I

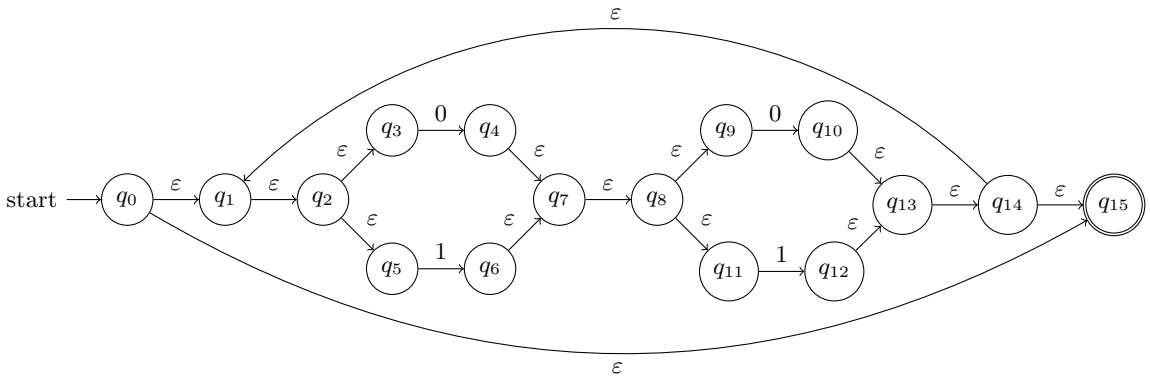
Tempo a disposizione: 1 h 30 min

1. Scrivere un automa a stati finiti che riconosca il linguaggio

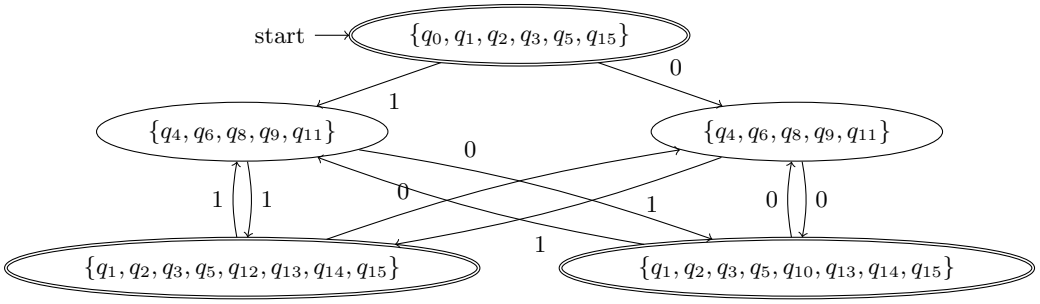
$$L = \{w \in \{0,1\}^* \mid w \neq 0110\}$$



2. Trasformare l'espressione regolare $((0+1)(0+1))^*$ in un automa *usando l'algoritmo visto a lezione.*



3. Trasformare l'ε-NFA ottenuto nell'esercizio 2 in DFA.



4. Sia $\Sigma = \{0,1\}$ e considerate i due seguenti linguaggi:

$$L_1 = \{(01)^n 0 (10)^n \mid n \geq 0\}$$
$$L_2 = \{1^n 0 1^n \mid n \geq 0\}$$

Uno dei due linguaggi è regolare, l'altro linguaggio non è regolare.

(a) Dire quale dei due linguaggi è regolare e quale non è regolare.
(b) Per il linguaggio regolare, dare un automa a stati finiti o un'espressione regolare che lo rappresenta.
(c) Per il linguaggio non regolare, dimostrare la sua non regolarità usando il Pumping Lemma.

(a) Il linguaggio L_1 è regolare, mentre il linguaggio L_2 non è regolare.

(b) *Il linguaggio L_1 è generato dall'espressione regolare* $(01)^*0$

(c) *Supponiamo per assurdo che L sia regolare. Di conseguenza, deve rispettare il Pumping Lemma.*

- *sia $n > 0$ la lunghezza data dal Pumping Lemma;*
- *consideriamo la parola $w = 1^n01^n$, che appartiene ad L ed è di lunghezza maggiore di n;*
- *sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq n$;*
- *poiché $|xy| \leq n$, allora xy è completamente contenuta nel prefisso 1^n di w, e quindi sia x che y sono composte solo da 1. Inoltre, siccome $y \neq \varepsilon$, possiamo dire che $y = 1^p$ per qualche valore $p > 0$. Allora la parola xy^2z è nella forma $1^{n+p}01^n$, e quindi non appartiene al linguaggio perché il numero di 1 nella prima parte della parola è maggiore del numero di 1 nella seconda parte della parola.*

Abbiamo trovato un assurdo quindi L non può essere regolare.

5. Costruire una CFG G che genera il linguaggio $L = \{a^n b^m c^k \mid$ *con* $n = m$ *o* $m = k$ *e* n, m *e* $k \geq 0\}$. Dimostrare che per la grammatica G che proponete, vale $L(G) \subseteq L$.

La grammatica G è come segue:

$$S \to AC \mid BD$$

$$A \to aAb \mid \epsilon$$

$$C \to cC \mid \epsilon$$

$$B \to aB \mid \epsilon$$

$$D \to bDc \mid \epsilon$$

Per dimostrare che $L(G) \subseteq L$ basta osservare che C genera c^m con $m \geq 0$ e che A genera $a^n b^n$ per $n \geq 0$ e quindi $S \Rightarrow AC \Rightarrow^* a^n b^n c^m$ con n ed m maggiori o uguali a 0. La stringa vuota si ottiene quando $n = m = 0$. Un ragionamento simile su B e D dimostra che $L(G)$ genera anche $a^n b^m c^m$ per n ed m maggiori o uguali a 0.

Automi e Linguaggi Formali
Soluzione della prima prova intermedia

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

- Per dimostrare che ogni linguaggio regolare è riconosciuto da un all-ε-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-ε-NFA, che non ha ε-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-ε-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-ε-NFA è regolare, mostriamo come possiamo trasformare un all-ε-NFA in un DFA equivalente. La trasformazione è descritta dal seguente algoritmo:

  Require: Un all-ε-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
  $S_0 \leftarrow \textsc{Eclose}(q_0)$ ▷ Lo stato iniziale è la chiusura di q_0
  $Q_D \leftarrow \{S_0\}$ ▷ Q_D sarà l'insieme degli stati del DFA
  if $S_0 \subseteq F_N$ **then** ▷ Se S_0 contiene solamente stati finali dell'all-ε-NFA ...
  $F_D \leftarrow \{S_0\}$ ▷ ... allora S_0 è stato finale del DFA, ...
  else
  $F_D \leftarrow \emptyset$ ▷ ... altrimenti no
  end if
  while Q_D contiene stati senza transizioni uscenti **do** ▷ Ciclo principale
  Scegli $S \in Q_D$ senza transizioni uscenti
  for all $a \in \Sigma$ **do** ▷ una transizione per ogni simbolo dell'alfabeto
  $S' \leftarrow \emptyset$ ▷ stato di arrivo della transizione
  for all $q \in S$ **do** ▷ lo stato di partenza S è un insieme di stati di N
  $S' \leftarrow S' \cup \delta_N(q, a)$ ▷ aggiungi gli stati $\delta_N(q, a)$ ad S'
  end for
  $S' \leftarrow \textsc{Eclose}(S')$ ▷ Chiusura di S' per ϵ-transizioni
  $Q_D \leftarrow Q_D \cup \{S'\}$ ▷ Aggiungi lo stato S' al DFA
  if $S' \subseteq F_N$ **then** ▷ Se S' contiene solamente stati finali ...
  $F_D \leftarrow F_D \cup \{S'\}$ ▷ ... allora S' è finale per il DFA
  end if
  $\delta_D(S, a) \leftarrow S'$ ▷ Aggiungi la transizione da S ad S' con input a al DFA
  end for
  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 ε-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-ε-NFA accetta quando tutti i possibili stati in cui si può trovare dopo aver consumato l'input sono stati finali.

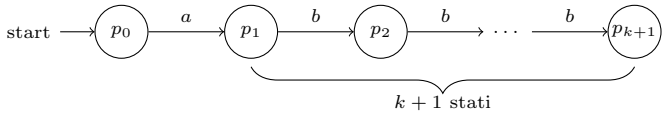
  Soluzione alternativa: in alternativa all'algoritmo si può dare la definizione del DFA $D = (Q_D, \Sigma, S_0, \delta_D, F_D)$ equivalente all'all-ε-NFA $N = (Q_N, \Sigma, q_0, \delta_N, F_N)$ specificando le componenti del DFA:
  - l'insieme degli stati è l'insieme delle parti di Q_N: $Q_D = \{S \mid S \subseteq Q_N\}$;
  - lo stato iniziale è la ε-chiusura di q_0: $S_0 = \textsc{Eclose}(q_0)$;
  - la funzione di transizione "simula" le transzioni di N:

$$\delta_D(S, a) = \textsc{Eclose}(\bigcup_{p \in S} \delta_N(p, a))$$

  - 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 ε-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 $\{0^n 1^n \mid n \geq 0\}$ 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 $p_1, p_2, \ldots, p_{k+1}$ 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 $L' = L_1 \cap \{a^* b^* c^*\} = \{ab^m c^m \mid m \geq 0\}$, 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.

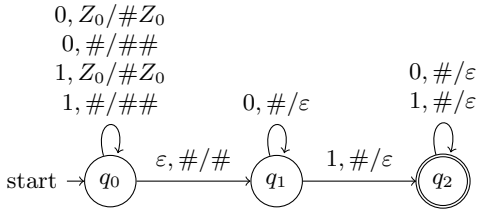
- Poniamo come lunghezza del pumping $k = 2$.
- Data una qualsiasi parola $w = a^\ell b^m c^n \in L_1$ di lunghezza maggiore o uguale a 2, si possono presentare vari casi, secondo il numero di a presenti nella parola:
  - se c'è una sola a, allora $w = ab^m c^m$. Scegliamo la suddivisione $x = \varepsilon$, $y = a$ 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:

$$S \to 0S0 \mid 1S0 \mid 0T1 \mid 1T1$$
$$T \to 0T \mid 1T \mid \varepsilon$$

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 – a.a. 2020/2021
Prima prova intermedia – Soluzione

1. Supponiamo che L ed M siano due linguaggi regolari, e mostriamo che anche $\mathit{faro}(L, M)$ 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 $\mathit{faro}(L, M)$ è regolare esibiamo un automa a stati finiti A che riconosce $\mathit{faro}(L, M)$.

La funzione ricorsiva $\mathit{faro}(x, z)$ 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 coni i simboli di z nelle posizioni pari: $\mathit{faro}(x, z) = x_1 z_1 x_2 z_2 \dots x_n z_n$;
- se x è più corta di z, allora $\mathit{faro}(x, z)$ 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 $\mathit{faro}(x, z)$ alterna simboli di x con simboli di z finché possibile, e poi continua con la parte rimanente di x.

L'automa che accetta $\mathit{faro}(L, M)$ 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 $\delta_M(r_M, a) = s_M$.

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.

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

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

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 $w = 1^k\#1^{k-1}\#\ldots\#1\#\#$, 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: $z = 1^{k-q-p}\#1^{k-1}\#\ldots\#1\#\#$. Consideriamo l'esponente $i = 0$: la parola xy^0z ha la forma

  $$xy^0z = xz = 1^q1^{k-q-p}\#1^{k-1}\#\ldots\#1\# = 1^{k-p}\#1^{k-1}\#\ldots\#1\#\#$$

  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 $A \to u$ 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 $A \to R + S$ con le due regole $A \to R$ e $A \to S$;

(b) per ogni regola $A \to R.S$, aggiungi due nuove variabili A_R e A_S e rimpiazza la regola con le regole $A \to A_R A_S$, $A \to R$ e $A \to S$;

(c) per ogni regola $A \to S^*$, aggiungi una nuova variabile A_S e rimpiazza la regola con le regole $A \to A_S A$, $A \to \varepsilon$ e $A_S \to S$;

(d) rimpiazza ogni regola $A \to \emptyset$ con la regola $A \to A$;

(e) ripeti da (a) finché non rimangono solamente regole del tipo $A \to u$ dove u è una stringa di variabili e terminali, oppure $A \to \varepsilon$.

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" $A \to u$, senza operatori regolari.

Automi e Linguaggi Formali – 17/6/2021
Conferma del voto delle prove intermedie – secondo turno

1. Definire un automa a stati finiti (di qualsiasi tipologia) che riconosca il linguaggio

$$L_1 = \{w \in \{0,1\}^* \mid w \text{ non contiene la sottostringa } 0110\}$$

2. Definire una grammatica context-free che generi il linguaggio

$$L_2 = \{0^n 1^m 2^{m+n} \mid n, m \leq 0\}$$

3. Fornisci una descrizione a livello implementativo di una TM deterministica a nastro singolo che decide il linguaggio

$$L_3 = \{u\#w_1\#\ldots\#w_n \mid u, w_i \in \{0,1\}^* \text{ ed esiste } w_j \text{ tale che } u = w_j\}$$

Una descrizione a livello implementativo descrive a parole il movimento della testina e la scrittura sul nastro, senza dare il dettaglio degli stati.

4. Una 5-colorazione di un grafo non orientato G è una funzione che assegna a ciascun vertice di G un "colore" preso dall'insieme $\{0,1,2,3,4\}$, in modo tale che per qualsiasi arco $\{u,v\}$ i colori associati ai vertici u e v sono diversi. Una 5-colorazione è *accurata* se i colori assegnati ai vertici adiacenti sono distinti e con differenza maggiore di 1 *modulo 5*.

Fornisci un verificatore polinomiale per il seguente problema:

$$\textsc{Careful5Color} = \{\langle G\rangle \mid G \text{ è un grafo che ammette una 5-colorazione accurata}\}$$

Automi e Linguaggi Formali – 17/6/2021
Conferma del voto delle prove intermedie – primo turno

1. Definire un automa a stati finiti (di qualsiasi tipologia) che riconosca il linguaggio

$$L_1 = \{w \in \{0,1\}^* \mid \text{ogni occorrenza di } 00 \text{ compare prima di ogni occorrenza di } 11\}$$

Per esempio, la parola 01000110 appartiene al linguaggio perché 00 compare prima di 11 nella stringa, la parola 001001 appartiene al linguaggio perché non contiene occorrenze di 11, mentre la parola 001100 non appartiene al linguaggio perché l'ultima occorrenza di 00 compare dopo 11.

2. Definire una grammatica context-free che generi il linguaggio

$$L_2 = \{w \in \{0,1\}^* \mid \text{il numero di } 0 \text{ è il doppio del numero di } 1\}$$

3. Fornisci una descrizione a livello implementativo di una TM deterministica a nastro singolo che decide il linguaggio

$$L_3 = \{ww \mid w \in \{0,1\}^*\}$$

Una descrizione a livello implementativo descrive a parole il movimento della testina e la scrittura sul nastro, senza dare il dettaglio degli stati.

4. Fornisci un verificatore polinomiale per il seguente problema:

$$\begin{aligned}\textsc{DoubleHamCircuit} = \{\langle G\rangle \mid\ & G \text{ è un grafo non orientato che contiene un ciclo che} \\ & \text{visita ogni vertice } \textit{esattamente due volte} \text{ e} \\ & \text{attraversa ogni arco } \textit{esattamente una volta}\}\end{aligned}$$

Automi e Linguaggi Formali – 17/6/2021
Prima parte – Linguaggi Regolari e Context Free

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 δ è definita come segue:

  $$\delta((r_L, r_M), 0) = \{(\delta_L(r_L, 0), \delta_M(r_M, 0)), (\delta_L(r_L, 1), \delta_M(r_M, 0)), (\delta_L(r_L, 0), \delta_M(r_M, 1))\}$$
  $$\delta((r_L, r_M), 1) = \{(\delta_L(r_L, 1), \delta_M(r_M, 1))\}$$

  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^0z ha la forma

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

  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: $A \to BC$, con A, B, C simboli non terminali, oppure $A \to b$ con b simbolo nonterminale.

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

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

Automi e Linguaggi Formali
Soluzione dell'esame scritto del 1 Settembre 2021

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^0z ha la forma

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

  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 $V' \to V$ e $V' \to \varepsilon$ appartengono a G;
- per ogni regola $V \to AB$ di G, le regole $V' \to AB'$ e $V' \to A'$ 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 $\Gamma = \{0,1,\sqcup\}$. 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 machine 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 Γ 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 Γ 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 Γ contiene 4 simboli, allora $k = 2$, perché con 2 bit si rappresentano 4 valori diversi. Se Γ 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 $w = w_1 w_2 \dots w_n$ 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:*

$$\text{makespan}(A) = \max_{1 \le i \le m} \sum_{A[j]=i} T[j]$$

Load Balance *è il problema di trovare un assegnamento con makespan minore o uguale al limite superiore k:*

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

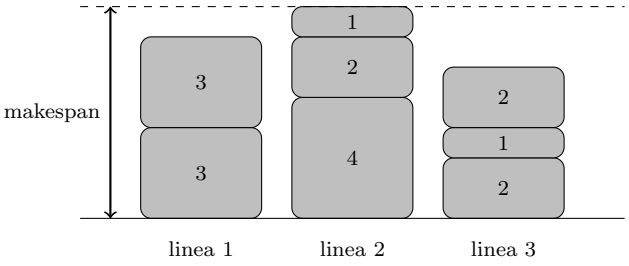


Figura 1: Esempio di assegnamento dei lavori $T = \{1, 1, 2, 2, 2, 3, 3, 4\}$ su 3 linee con makespan 7.

(a) Dimostra che LoadBalance *è un problema NP.*
(b) Dimostra che LoadBalance *è NP-hard, usando* SetPartitioning *come problema NP-hard di riferimento.*

(a) 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 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 $\langle T \rangle \in$ 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 $\langle 2, T, k \rangle \in$ 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 – 20/4/2022
Prima prova intermedia – Primo Turno – Soluzione

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, Σ, δ 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.

Nota: 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 $A \to wB$ o $A \to w$, 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 ε-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 $A \to wB$ o $A \to w$, 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 $A \to wB$ seguita da un'applicazione finale di una regola $A \to w$. A partire da questa osservazione possiamo costruire un ε-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'ε-NFA e sia $w = w_1 \dots w_n$ una stringa sull'alfabeto Σ. 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 $A \to wB$ della grammatica;
- $q_f \in \delta(A, w)$ per ogni regola $A \to w$ 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:

(a) se lo stato corrente corrente corrisponde alla variabile A, sceglie non deterministicamente una delle regole per A;
(b) se la regola scelta è $A \to wB$, prova a consumare la stringa w dall'input. Se ci riesce va nello stato B, altrimenti la computazione si blocca su questo ramo;
(c) se la regola scelta è $A \to w$, 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.

Automi e Linguaggi Formali – 20/4/2022
Prima prova intermedia – Secondo Turno – Soluzione

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 $L = \{a, aab, baa\}$, 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, Σ, δ 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 $\overline{w}$ è una stringa formata da w sostituendo gli 0 con 1 e viceversa; per esempio $\overline{011} = 100$. Considera il linguaggio

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

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 $\overline{w}$ 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 $w\overline{w}$ 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 $w\overline{w}$.

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^k1^k si può iterare, dividendola come segue, e perciò non è adatta al nostro scopo:

$$\overbrace{\underbrace{000\ldots000}_{u}\underbrace{0}_{v}}^{0^k}\underbrace{\varepsilon}_{x}\overbrace{\underbrace{1}_{y}\underbrace{111\ldots111}_{z}}^{1^k}$$

  Anche la parola $0^k1^k1^k0^k$ si può iterare, suddividendola in modo simile:

$$\overbrace{\underbrace{000\ldots000}_{u}\underbrace{0}_{v}}^{0^k}\underbrace{\varepsilon}_{x}\overbrace{\underbrace{1}_{y}\underbrace{111\ldots111000\ldots000}_{z}}^{1^k1^k0^k}$$

  Mostriamo invece che la parola $w = 0^k10^k1^k01^k$ 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 $w\overline{w}$. 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 $w\overline{w}$.

  Ma se la sottostringa vxy è a cavallo del punto centrale di w, la stringa $uv^0xy^0z = uxz$ ha la forma $0^k10^i1^j01^k$ dove i e j non possono essere entrambi k, e quindi non può essere nella forma $w\overline{w}$. Quindi w non può essere iterata.

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

Automi e Linguaggi Formali – 28/4/2023
Prima prova intermedia – Soluzione

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

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

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 ε-NFA A' che accetta il linguaggio $evens(L)$, 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 ε-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 Σ 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 ε-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 $evens(L)$, 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 $evens(w) = w_2 w_4 w_6 \dots w_n$, perché $(s_n, 1) \in F'$. Se w è di lunghezza dispari, allora la computazione accettante per A' su $evens(w)$ 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 ε-transizione finale per raggiungere uno stato finale. In entrambi i casi possiamo costruire una computazione accettante per A sostituendo le ϵ-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 $u = u_1w_1u_2w_2 \dots u_nw_n$. 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 $u = u_1w_1u_2w_2 \dots u_nw_nu_{n+1}$. In entrambi i casi $evens(u) = w$, 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 $w = 1^k0^k1^k$, 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: $z = 1^{k-q-p}0^k1^k$. Consideriamo l'esponente $i = 2$: la parola xy^2z ha la forma

  $$xy^2z = 1^q1^{2p}1^{k-q-p}0^k1^k = 1^{k+p}0^k1^k$$

  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: $R' = \{A \to u^R \mid A \to u \in R\}$. 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: $S' = S$.

Automi e Linguaggi Formali – 28/4/2023
Prima prova intermedia – Soluzione

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

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

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 ε-NFA A' che accetta il linguaggio $\mathit{substring}(L)$. L'automa A' è costituto dagli stessi stati di A a cui viene aggiunto un nuovo stato iniziale q_0'. La funzione di transizione contiene una ϵ-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 Σ 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 $\mathit{substring}(L)$, 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 definzione della funzione di transizione δ', 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

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

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 $\mathit{censor}(L)$ 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 $A \to b$, con b simbolo terminale, con la regola $A \to \#$, e lasciando invariate le regole nella forma $A \to BC$ e la regola $S \to \varepsilon$ (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 $\mathit{censor}(L)$. 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 Σ. Quindi, G' permette di derivare solo parole che appartengono a $\mathit{censor}(L)$.

Automi e Linguaggi Formali – 20/4/2023
Simulazione Prima prova intermedia – Soluzione

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, Σ, δ 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

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

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 $w = 1^k 0^k$, 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^0z ha la forma

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

  Poiché $p > 0$, la sequenza iniziale di 1 è più corta della sequenza finale di 0, e quindi la parola iterata xy^0z 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 $1^{k+ip}0^k = 1^k 1^{ip/2} 1^{ip/2} 0^k = 1^{k+ip/2} w$ 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 $A \to wB$ o $A \to w$, 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 ε-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 $A \to wB$ o $A \to w$, 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 $A \to wB$ seguita da un'applicazione finale di una regola $A \to w$. A partire da questa osservazione possiamo costruire un ε-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'ε-NFA e sia $w = w_1 \dots w_n$ una stringa sull'alfabeto Σ. 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 $A \to wB$ della grammatica;
- $q_f \in \delta(A, w)$ per ogni regola $A \to w$ 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:

(a) se lo stato corrente corrente corrisponde alla variabile A, sceglie non deterministicamente una delle regole per A;
(b) se la regola scelta è $A \to wB$, prova a consumare la stringa w dall'input. Se ci riesce va nello stato B, altrimenti la computazione si blocca su questo ramo;
(c) se la regola scelta è $A \to w$, 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.

Automi e Linguaggi Formali – 26/6/2023
Primo appello – Prima Parte: Linguaggi regolari e linguaggi contex-free

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 Σ 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 $s_0, \ldots, s_{n-1}$ 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 $w \in \mathit{NOPREFIX}(L)$.

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 $w = 0^k110^k$, 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

  $$xy^2z = 0^q0^{2p}0^{k-q-p}110^k = 0^{k+p}110^k$$

  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^p11 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 $A \to aBc$ o $A \to a$ 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 $R_i \to aR_j$ per ogni transizione $\delta(q_i, a) = q_j$ del DFA, e del tipo $R_i \to \varepsilon$ 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:

$$S \to 0S1 \mid \varepsilon$$

  Quindi, esiste un linguaggio lineare che non è regolare.

Automi e Linguaggi Formali – 10/7/2023
Secondo appello – Prima Parte: Linguaggi regolari e linguaggi context-free

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

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

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 $\mathit{flip}(L)$ come segue.

- $Q' = Q$. L'insieme degli stati rimane lo stesso.
- L'alfabeto Σ 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 $\mathit{flip}(L)$, dobbiamo considerare due casi.

- Se $w \in \mathit{flip}(L)$, allora sappiamo che il flip di w appartiene ad L. Chiamiamo $\overline{w}$ il flip di w. Siccome A riconosce L, allora esiste una computazione di A che accetta $\overline{w}$:
$$s_0 \xrightarrow{\overline{w}_1} s_1 \xrightarrow{\overline{w}_2} \ldots \xrightarrow{\overline{w}_n} s_n$$
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
$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \ldots \xrightarrow{w_n} s_n$$
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 $w \in \mathit{flip}(L)$.

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 $w = 0^k10^k$, 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: $z = 0^{k-q-p}10^k$. Consideriamo l'esponente $i = 2$: la parola xy^2z ha la forma

  $$xy^2z = 0^q0^{2p}0^{k-q-p}10^k = 0^{k+p}10^k$$

  La parola iterata xy^2z non appartiene ad $\overline{L_2}$ perché non è palindroma: se la rovesciamo diventa la parola 0^k10^{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:

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

dove $\mathit{dehash}(w)$ 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 $\mathit{dehash}(L)$ 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 Σ 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 $A \to \#$ con la regola $A \to \varepsilon$, e lasciando invariate le regole nella forma $A \to BC$, le regole nella forma $A \to b$ quando $b \neq \#$, e la regola $S \to \varepsilon$ (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 ε. Quindi, G' permette di derivare tutte le parole in $dehash(L)$.

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 $A \to \varepsilon$, la derivazione per G applicherà la regola $A \to \#$ inserendo un $\#$ in qualche punto della parola w. Al termine della derivazione si ottiene una parola w' tale che $dehash(w') = w$. Quindi, G' permette di derivare solo parole che appartengono a $dehash(L)$.

Automi e Linguaggi Formali – 10/4/2024
Simulazione Prima prova intermedia – Soluzione

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 Σ 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

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

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 $w = 1^k 0^k$, 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^0z ha la forma

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

  Poiché $p > 0$, la sequenza iniziale di 1 è più corta della sequenza finale di 0, e quindi la parola iterata xy^0z 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 $1^{k+ip}0^k = 1^k 1^{ip/2} 1^{ip/2} 0^k = 1^{k+ip/2} w$ 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 $A \to wB$ o $A \to w$, 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 ε-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 $A \to wB$ o $A \to w$, 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 $A \to wB$ seguita da un'applicazione finale di una regola $A \to w$. A partire da questa osservazione possiamo costruire un ε-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'ε-NFA e sia $w = w_1 \dots w_n$ una stringa sull'alfabeto Σ. 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 $A \to wB$ della grammatica;
- $q_f \in \delta(A, w)$ per ogni regola $A \to w$ 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:

(a) se lo stato corrente corrente corrisponde alla variabile A, sceglie non deterministicamente una delle regole per A;
(b) se la regola scelta è $A \to wB$, prova a consumare la stringa w dall'input. Se ci riesce va nello stato B, altrimenti la computazione si blocca su questo ramo;
(c) se la regola scelta è $A \to w$, 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.