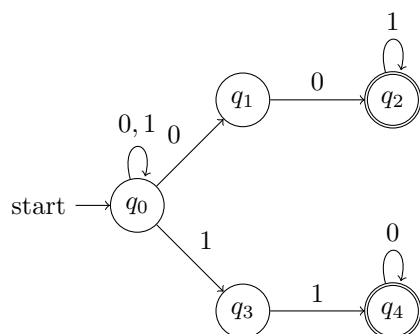


Linguaggi Regolari

1. Descrivere in italiano il linguaggio generato dall'espressione regolare $(11 + \varepsilon)(0011)^*(00 + \varepsilon)$

Soluzione: L'espressione definisce il linguaggio di tutte le parole di lunghezza pari (anche vuote) che alternano coppie di zero (**00**) e coppie di uno (**11**)

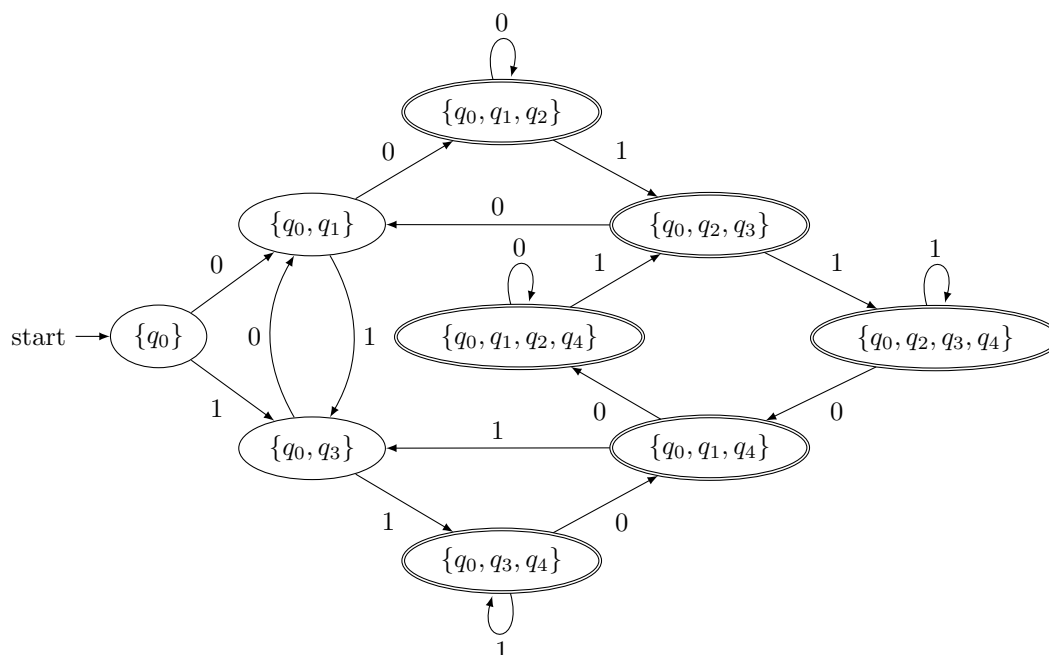
2. Trasformare il seguente NFA in DFA



Soluzione: Applicando la costruzione a sottoinsiemi si ottiene il DFA con la seguente tabella di transizione (dove gli stati non raggiungibili da $\{q_0\}$ sono omessi):

	0	1
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_3, q_4\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2, q_3\}$
$*\{q_0, q_1, q_4\}$	$\{q_0, q_1, q_2, q_4\}$	$\{q_0, q_3\}$
$*\{q_0, q_2, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2, q_3, q_4\}$
$*\{q_0, q_3, q_4\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_3, q_4\}$
$*\{q_0, q_1, q_2, q_4\}$	$\{q_0, q_1, q_2, q_4\}$	$\{q_0, q_2, q_3\}$
$*\{q_0, q_2, q_3, q_4\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_2, q_3, q_4\}$

e con il seguente diagramma di transizione:



3. Il linguaggio

$$L = \{w \in \{a, b, c\}^* \mid \text{il numero di } a \text{ è uguale al numero di } b \text{ e maggiore del numero di } c\}$$

è regolare? Motivare la risposta.

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

- sia $n > 0$ la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = a^n b^n$, 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 a^n 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^0z è nella forma $a^{n-p}b^n$, e quindi non appartiene al linguaggio perché il numero di a è minore del numero di b .

Abbiamo trovato un assurdo quindi L non può essere regolare.

Nota: In alternativa si può utilizzare xy^2z (o un qualsiasi esponente $k > 1$): in questo caso si ottiene una parola che non appartiene ad L perché il numero di a è maggiore del numero di b .

Linguaggi Regolari

1. Scrivere una espressione regolare che definisca il linguaggio

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

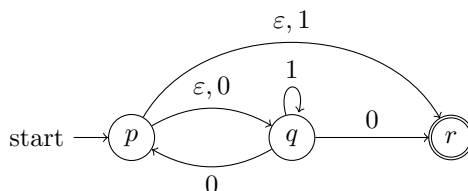
Soluzione. Per definire correttamente il linguaggio occorre tener presente che:

- le coppie di zeri non sono necessariamente vicine; parole come 010, 0110, 01...10 appartengono a L
- una parola che non contiene 0 è nel linguaggio; quindi 1, 11, ... sono in L
- anche la parola vuota è nel linguaggio

Le soluzioni elencate qui sotto utilizzano la RE 01^*0 per generare coppie di zeri separate da un numero arbitrario (anche nullo) di 1, gestendo poi le parole che non contengono 0 in modo diverso:

$$(1 + 01^*0)^* \quad (1^*01^*0)^*1^* \quad 1^*(01^*0)^*1^* \quad (1^*01^*01^*)^* + 1^*$$

2. Dato il seguente ε -NFA

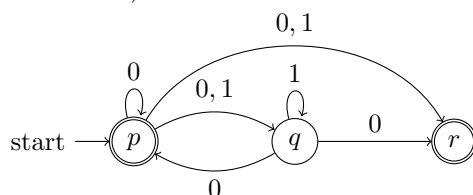


- (a) calcolare la ε -chiusura di ogni stato
- (b) costruire un DFA equivalente

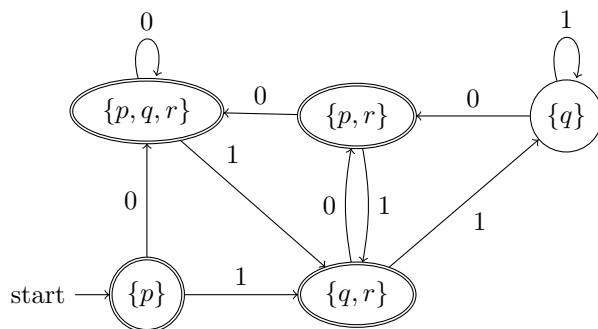
Soluzione.

$$(a) \text{ ECLOSE}(p) = \{p, q, r\} \quad \text{ECLOSE}(q) = \{q\} \quad \text{ECLOSE}(r) = \{r\}$$

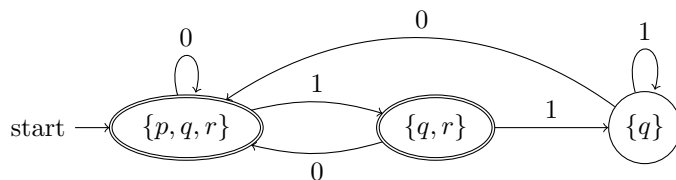
- (b) Per costruire un DFA equivalente si può procedere in due modi. Il primo modo utilizza due passaggi: prima si eliminano le ε -transizioni, ottenendo il seguente NFA (dove p diventa uno stato finale)



e poi si usa la costruzione a sottoinsiemi per ottenere l'automa deterministico finale (dove gli stati non raggiungibili sono rimossi)



Il secondo modo è quello di utilizzare la trasformazione diretta da ε -NFA a DFA descritta nel libro di testo (non vista a lezione), che porta al seguente DFA



3. Il linguaggio

$$L = \{v00v \mid v \in \{0, 1\}^*\}$$

è regolare? Motivare la risposta.

Soluzione. Il linguaggio contiene tutte le parole formate da due ripetizioni di una stringa v (di lunghezza arbitraria), separate dalla coppia 00. Intuitivamente, non può essere regolare perché per riconoscere le parole nel linguaggio occorre memorizzare tutti i simboli della prima occorrenza di v per poi controllare che si ripetano esattamente uguali dopo il separatore 00.

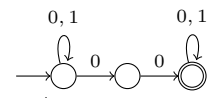
Per dimostrarlo formalmente, assumiamo per assurdo che L sia regolare:

- sia $n > 0$ la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 1^n 00 1^n$, che è di lunghezza maggiore di n ed è nella forma $v00v$ se poniamo $v = 1^n$. Quindi w appartiene ad L ;
- sia $w = xyz$ una suddivisione arbitraria di w tale che $y \neq \varepsilon$ e $|xy| \leq n$;
- poiché $|xy| \leq n$, allora xy è completamente contenuta nella prima ripetizione di $v = 1^n$, 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^0z è nella forma $1^{n-p} 00 1^n$, e non può appartenere a L perché il numero di 1 che precedono il separatore 00 è minore del numero di 1 che lo seguono. Quindi non esiste nessuna parola u che ci consenta di scrivere xy^0z come $u00u$.

Abbiamo trovato un assurdo quindi L non può essere regolare.

Note:

- (1) In alternativa si può utilizzare xy^2z (o un qualsiasi esponente $k > 1$): in questo caso si ottiene una parola che non appartiene ad L perché il numero di 1 prima del separatore è maggiore del numero di 1 dopo il separatore.
- (2) Il ragionamento non funziona quando $v = 0^n$: in questo caso se l'avversario sceglie $y = 0^p$ con p pari allora è sempre possibile trovare un modo per scrivere xy^kz come $u00u$ (spostando la posizione del separatore 00). Quindi non è possibile utilizzare una v qualsiasi nella dimostrazione: occorre scegliere esplicitamente una parola ben precisa. Quella dove la dimostrazione è più semplice è $v = 1^n$.

- (3) L'espressione regolare $(0 + 1)^* 00 (0 + 1)^*$ e l'automa  non riconoscono il linguaggio L perché accettano anche le parole $u00v$ con $u \neq v$.

Linguaggi Liberi dal Contesto

4. Data la seguente grammatica libera da contesto $G : S \rightarrow aS \mid Sb \mid a \mid b$, dimostrare, per induzione sulla lunghezza della stringa che nessuna stringa in $L(G)$ contiene "ba" come sottostringa. Descrivere $L(G)$ a parole.

Soluzione. Operiamo per induzione sulle stringhe w di $L(G)$.

- Base: $|w| = 1$, le sole stringhe di lunghezza 1 di $L(G)$ sono "a" e "b" che non contengono "ab" ovviamente.
- Passo induttivo: assumiamo che tutte le stringhe w di $L(G)$ tali che $|w| = n \geq 1$ non contengano "ab". Consideriamo una stringa w' di lunghezza $n + 1$. Visto che è in $L(G)$ esiste una derivazione di w' da S e questa derivazione deve iniziare con $S \Rightarrow aS$ o $S \Rightarrow bS$, dopo di che $S \Rightarrow aS \Rightarrow aw \Rightarrow w'$ oppure $S \Rightarrow bS \Rightarrow bw \Rightarrow w'$. Visto che w ha lunghezza n , l'ipotesi induttiva dice che w non contiene "ab" e quindi neppure aw e bw possono contenere "ab".

Quindi nessuna stringa in $L(G)$ contiene "ab".

$L(G)$ contiene le stringhe non vuote che iniziano con una sequenza anche vuota di a , seguita da una sequenza anche vuota di b .

5. Dato l'automa a pila $P = (\{q\}, \{a, b\}, \{a, Z\}, \delta, q, Z, \{q\})$ dove δ è come segue:

$$\delta(q, a, Z) = \{(q, aZ)\}, \quad \delta(q, a, a) = \{(q, aa)\}, \quad \delta(q, b, a) = \{(q, \epsilon)\}.$$

Descrivere il linguaggio riconosciuto da P . Trasformare P in un PDA P' che accetta per pila vuota lo stesso linguaggio accettato da P per stato finale.

Soluzione. Il linguaggio $L(P)$ è costituito da tutte le stringhe di a e di b t.c. ogni loro prefisso ha un numero di a pari o superiore al numero di b .

Per trasformare P in P' t.c. $N(P') = L(P)$ seguiamo la costruzione generale.

- P' ha per lo stack i simboli $\{a, Z, X_0\}$ di cui X_0 è il fondo dello stack di P' .
- P' ha anche due stati q_0 e q' in più oltre a q di P ed ha le seguenti transizioni aggiuntive:
 $\delta(q_0, \epsilon, X_0) = \{(q, ZX_0)\}$, $\delta(q, \epsilon, ?) = \{(q', \epsilon)\}$, $\delta(q', \epsilon, ?) = \{(q', \epsilon)\}$, in cui $?$ sta per ogni simbolo dello stack.

Le spiegazioni sulla necessità (in generale) di X_0 e sulle transizioni aggiuntive le trovate sul libro di testo.

6. In generale gli automi a pila possono accettare per pila vuota o per stato finale. I linguaggi riconosciuti sono gli stessi. Per gli automi a pila deterministici questo non è più vero. Spiegate le ragioni di questa differenza. Cercate di specificare quali linguaggi vengono accettati nelle due modalità di accettazione. La differenza tra le 2 classi di linguaggi vi pare importante?

Soluzione. Nel caso degli automi a pila deterministici (DPDA), le modalità di accettazione fanno grande differenza. Quelli che accettano per stato finale, accettano tutti i linguaggi regolari e anche qualche linguaggio libero da contesto, ma non tutti. I DPDA che accettano per pila vuota accettano i linguaggi accettati dai DPDA che accettano per stato finale che hanno la proprietà del prefisso, cioè quei linguaggi tali che non contengano 2 stringhe di cui una sia prefisso dell'altra.

Questo fatto implica che i DPDA che accettano per pila vuota non accettano neppure tutti i linguaggi regolari, visto che il linguaggio a^* è regolare e non ha la proprietà del prefisso. D'altra parte è facile far sì che un linguaggio abbia la proprietà del prefisso: basta aggiungere ad ogni stringa del linguaggio un simbolo speciale (unico) che marchi la fine della stringa. Per esempio la stringa w diventerebbe $w\$$. E' ovvio che in questo modo nessuna stringa del linguaggio può essere prefisso di un'altra.

È importante che i DPDA accettano solo linguaggi liberi da contesto non inerentemente ambigui, ma non tutti questi linguaggi.

Tempo a disposizione: 1 h 30 min

Gli esercizi (1, 2, 3) e (4, 5, 6) vanno consegnati su due fogli differenti

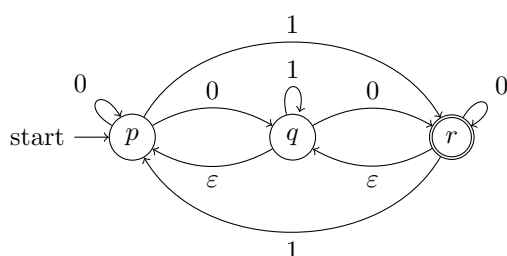
Linguaggi Regolari

1. Determinare il linguaggio definito dall'espressione regolare $0^*1(0 + 10^*1)^*$

Soluzione. L'espressione regolare riconosce il linguaggio di tutte le stringhe sull'alfabeto $\{0, 1\}$ che contengono un numero *dispari* di 1.

L'espressione $(0 + 10^*1)^*$ genera tutte le stringhe con un numero pari di 1 (vedi Es. 1 dell'Esame del 28 Giugno), mentre 0^*1 aggiunge un ulteriore 1 alla stringa, portando il numero di 1 ad essere dispari.

2. Dato il seguente ε -NFA

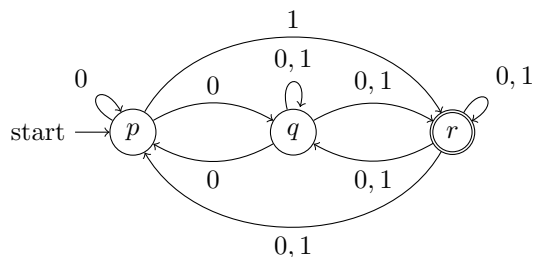


- (a) calcolare la ε -chiusura di ogni stato
- (b) costruire un DFA equivalente

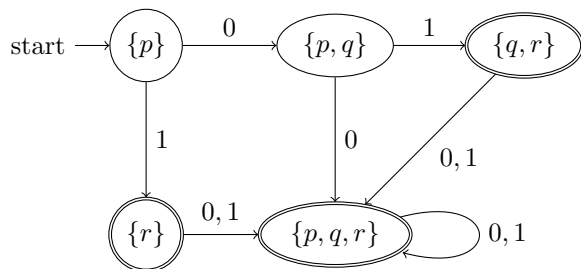
Soluzione.

- (a) $\text{ECLOSE}(p) = \{p\}$ $\text{ECLOSE}(q) = \{p, q\}$ $\text{ECLOSE}(r) = \{p, q, r\}$

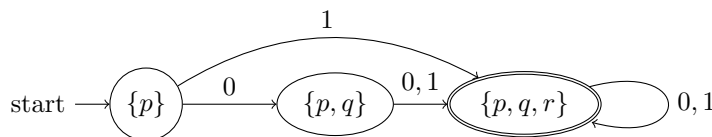
- (b) Per costruire un DFA equivalente si può procedere in due modi. Il primo modo utilizza due passaggi: prima si eliminano le ε -transizioni, ottenendo il seguente NFA (dove r rimane l'unico stato finale)



e poi si usa la costruzione a sottoinsiemi per ottenere l'automa deterministico finale (dove gli stati non raggiungibili sono rimossi)



Il secondo modo è quello di utilizzare la trasformazione diretta da ε -NFA a DFA descritta nel libro di testo (non vista a lezione), che porta al seguente DFA



3. Il linguaggio

$$L = \{w1w1w \mid w \in \{0, \dots, 9\}^*\}$$

è regolare? Motivare la risposta.

Soluzione. Il linguaggio contiene tutte le parole formate da tre ripetizioni di una stringa w (di lunghezza arbitraria), separate dal simbolo 1. Intuitivamente, non può essere regolare perché per riconoscere le parole nel linguaggio occorre memorizzare tutti i simboli della prima occorrenza di w per poi controllare che si ripetano esattamente uguali nelle altre due ripetizioni.

Per dimostrarlo formalmente, assumiamo per assurdo che L sia regolare:

- sia $n > 0$ la lunghezza data dal Pumping Lemma;
- consideriamo la parola $v = 0^n 10^n 10^n$, che è di lunghezza maggiore di n ed è nella forma $w1w1w$ se poniamo $w = 0^n$. Quindi v appartiene ad L ;
- sia $v = xyz$ una suddivisione arbitraria di v tale che $y \neq \varepsilon$ e $|xy| \leq n$;
- poiché $|xy| \leq n$, allora xy è completamente contenuta nella prima ripetizione di $w = 0^n$, 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^0z è nella forma $0^{n-p}10^n10^n$, e non può appartenere a L perché non esiste nessuna parola u che ci consenta di scrivere xy^0z come $u1u1u$.

Abbiamo trovato un assurdo quindi L non può essere regolare.

Note: L'esercizio è molto simile all'Es. 3 dell'Esame del 28 Giugno. Valgono le stesse considerazioni fatte nella soluzione pubblicata in precedenza.

Linguaggi Liberi dal Contesto

4. Data la seguente grammatica libera da contesto $G : B \rightarrow BB \mid (B) \mid \varepsilon$, rispondere alle seguenti due domande:

- (a) dimostrare, per induzione sulla lunghezza della derivazione, che $L(G)$ consiste di stringhe in $\{(\cdot, \cdot)^*\}$ in cui le parentesi siano bilanciate, cioè tali che ogni parentesi aperta ha una corrispondente parentesi chiusa che la segue, e se la coppia di parentesi venisse eliminata, si otterrebbe di nuovo una stringa bilanciata.

Soluzione: come richiesto esplicitamente dall'esercizio, usiamo un'induzione sulla lunghezza della derivazione. Quindi, consideriamo stringhe in $L(G)$ derivate da B in 1 passo, in $n > 1$ passi e in $n + 1$ passi.

- **Base:** con 1 passo, la grammatica permette di derivare solo la stringa vuota ε che non avendo parentesi è una stringa bilanciata.
- **Passo induttivo:** assumiamo come ipotesi induttiva che tutte le stringhe in $L(G)$ derivate in n passi (per qualsiasi $n \geq 1$) siano bilanciate. Da questo vogliamo dimostrare che anche le stringhe derivate in $n + 1$ passi sono bilanciate. Consideriamo quindi una qualsiasi derivazione di $n + 1$ passi: $B \Rightarrow^{(n+1)} w$. Il primo passo può essere l'applicazione della produzione $B \rightarrow (B)$ o $B \rightarrow BB$. Consideriamo i due casi separatamente:
 - $B \Rightarrow (B) \Rightarrow^n (w)$, dove $B \Rightarrow^n w$ è una derivazione di n passi e quindi ad essa si applica l'ipotesi induttiva che ci garantisce che w è bilanciata. Basta ora osservare che allora anche (w) è bilanciata, dove la seconda parentesi corrisponde alla prima e infatti eliminandole entrambe resteremmo con w che è bilanciata, come richiesto dalla definizione.
 - $B \Rightarrow BB \Rightarrow^n w'w''$, dove $B \Rightarrow^{n'} w'$ e $B \Rightarrow^{n''} w''$ con n' e $n'' > 0$ e $n' + n'' = n$, quindi, per ipotesi induttiva w' e w'' sono bilanciate, per cui la loro concatenazione, $w'w''$ è una stringa bilanciata.

- (b) Mostrare che la grammatica $G : B \rightarrow (B) \mid \varepsilon$ non genera tutte le stringhe in $\{(\cdot, \cdot)^*\}$ bilanciate.

Soluzione: basta osservare che la stringa bilanciata $()()$ non potrà in nessun caso essere generata senza la produzione $B \rightarrow BB$.

5. Dato l'automa a pila $P = (\{q, p\}, \{a, b\}, \{a, Z\}, \delta, q, Z, \{q\})$ dove δ è come segue:

$$\delta(q, a, Z) = \{(q, aZ)\}, \quad \delta(q, a, a) = \{(q, aa)\}, \quad \delta(q, b, a) = \{(q, \varepsilon)\}, \quad \delta(q, b, Z) = \{(p, \varepsilon)\},$$

rispondere alle seguenti due domande.

- (a) Descrivere il linguaggio riconosciuto da P .

Soluzione: Si tratta del linguaggio costituito da tutte quelle stringhe in $\{a, b\}^*$ tali che in ogni prefisso il numero di a è almeno pari a quello dei b . Questo linguaggio compare negli ultimi tre esami di CFL.

- (b) Trasformare P in un PDA P' che accetta per pila vuota lo stesso linguaggio accettato da P per stato finale.

Soluzione: Visto che P può vuotare la pila (per rifiutare un input in caso contenga più b di a) è necessario usare la costruzione standard per ottenere P' tale che $N(P') = L(P)$. Quindi avremo bisogno di uno stato iniziale q_1 con transizione: $\delta(q_1, \varepsilon, X) = \{(q, ZX)\}$, che inserisce sotto al simbolo Z di fondo stack di P , un proprio simbolo di fondo stack X che non è noto a P che quindi non ha mosse per X e perciò non potrà mai eliminare X dalla pila. Poi avremo bisogno di uno stato q_2 che ha il compito di svuotare la pila. Ricordate che q è l'unico stato finale di P , quindi avremo la mossa: $\delta(q, \varepsilon, ?) = \{(p_2, \varepsilon)\}$, e poi $\delta(q_2, \varepsilon, ?) = \{(q_2, \varepsilon)\}$, per svuotare la pila. Con $?$ si intende un qualsiasi simbolo della pila. Si ricorda che i simboli della pila di P' sono Z , X , e a .

6. Rispondere alle seguenti due domande che riguardano gli automi a pila deterministici.

- (a) Come si dimostra che gli automi a pila deterministici riconoscono solo linguaggi non ambigui?

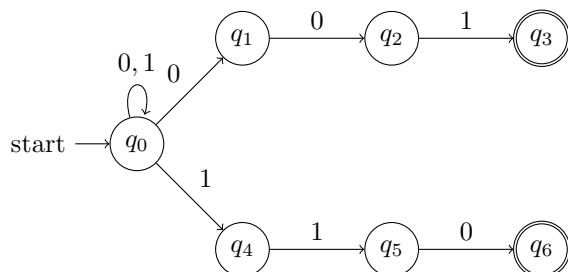
Soluzione: si osserva la costruzione generale di una grammatica CF che simula un PDA. La grammatica ha quei nonterminali $[q, X, p]$. Poiché la grammatica genera le stringhe che il PDA riconosce e la derivazione delle grammatica simula il calcolo del PDA, se il PDA è deterministico, esso ha un solo calcolo per ciascuna stringa accettata e anche la grammatica ha un'unica derivazione per ogni stringa generata. Insomma la grammatica è non ambigua.

- (b) Qual è la differenza tra un linguaggio libero da contesto ambiguo ed una grammatica libera da contesto ambigua?

Soluzione: un linguaggio CF L è ambiguo (più precisamente, inerentemente ambiguo) quando ogni CFG che genera L è ambigua. Una grammatica invece è ambigua, se esistono due alberi di derivazione diversi della CFG che hanno uguale frontiera.

Parte I – Linguaggi Regolari e Linguaggi Liberi da Contesto

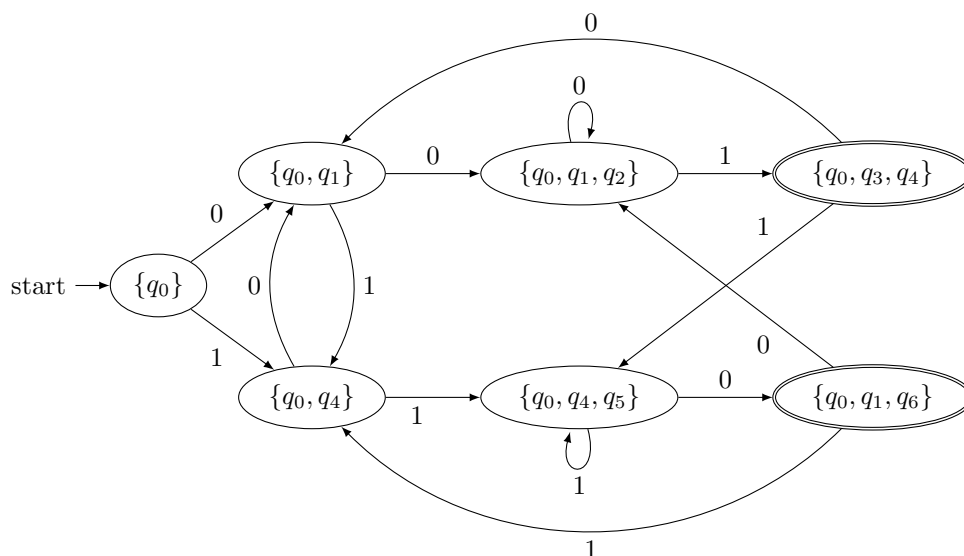
1. Dato il seguente NFA



- descrivere in italiano il linguaggio riconosciuto dall'automa
- costruire un DFA equivalente

Soluzione:

- L'automa riconosce il linguaggio di tutte le parole sull'alfabeto 0, 1 che terminano con 001 oppure con 110.
- Applicando la costruzione a sottoinsiemi si ottiene il DFA con il seguente diagramma di transizione (dove gli stati non raggiungibili da $\{q_0\}$ sono omessi):

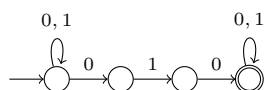


2. Il linguaggio

$$L = \{u010v \mid u, v \in \{0, 1\}^*\}$$

è regolare? Motivare la risposta.

Soluzione: Il linguaggio è regolare perché è riconosciuto dall'espressione regolare $(0 + 1)^*010(0 + 1)^*$ e dall'automa



Nota: il Pumping Lemma dice che se un linguaggio è regolare, allora rispetta certe proprietà. Quindi dà una condizione *necessaria* perché L sia regolare, ma non *sufficiente*: esistono linguaggi che rispettano il Pumping Lemma ma non sono regolari. Il fatto che lo schema di dimostrazione standard per mostrare che L non è regolare fallisca è una indicazione che L potrebbe essere regolare, ma non una dimostrazione formale. La dimostrazione corretta di regolarità di un linguaggio è quella che dà l'espressione regolare oppure l'automa che riconosce L .

3. Data la seguente grammatica libera da contesto

$$G : S \rightarrow BB$$

$$B \rightarrow 0B0 \mid 1B1 \mid 00 \mid 11$$

rispondere alle seguenti domande:

- Dare una definizione del linguaggio $L(G)$ del tipo seguente: $L(G)$ è l'insieme delle stringhe in $\{0,1\}^*$ che soddisfano la seguente proprietà.....
- Dimostrare induttivamente che la vostra definizione di $L(G)$ è corretta.
- Descrivere un automa a pila che riconosca $L(G)$ e spiegare perché secondo voi funziona.
- Considerate ora la seguente grammatica

$$G' : S \rightarrow BB$$

$$B \rightarrow 0B1 \mid 1B0 \mid 01 \mid 10$$

Definite $L(G')$ in modo simile a quanto fatto in (a) per $L(G)$.

Soluzione:

- $L(G)$ è l'insieme delle stringhe w in $\{0,1\}^*$ tali che $w = w_1w_2$ dove w_1 e w_2 sono palindromi di lunghezza almeno 2 e pari
- E' facile dimostrare che B genera palindromi di lunghezza almeno 2 e pari. Sono necessarie 2 dimostrazioni. La prima serve a dimostrare che se $w \in L(B)$, allora è palindromo di lunghezza almeno 2 e pari. La prova è per induzione sulla lunghezza della derivazione. Per lunghezza=1, le stringhe generabili in 1 passo sono 00 e 11 che soddisfano l'ipotesi. Per il passo induttivo, una derivazione di lunghezza $n+1$ inizia con $B \Rightarrow 0B0 \Rightarrow^n 0w'0$ (il caso $1B1$ è simile). Per ipotesi induttiva w' è palindromo di lunghezza almeno 2 e pari e quindi anche $0w'0$ lo è. La seconda dimostrazione serve a dimostrare che ogni stringa w che sia palindromo di lunghezza almeno 2 e pari venga generata da B , procede per induzione sulla lunghezza di w . Nella base, w ha lunghezza 2 e, visto che i soli palindromi di questa lunghezza sono 00 e 11, ovviamente B genera tali stringhe. Per stringhe di lunghezza $2 * n$ con $n > 1$, tali stringhe sono necessariamente $0w'0$ o $1w'1$ con w' pari e di lunghezza pari. Ovviamente G ha le produzioni per produrre $0B0$ e $1B1$ e l'ipotesi induttiva garantisce che B è capace di generare w' . Per concludere la dimostrazione, basta osservare che $S \rightarrow BB$ fa sì che le stringhe generabili da S abbiano la forma w_1w_2 desiderata.
- È possibile usare la costruzione vista in classe per produrre un automa che accetta per stack vuoto il linguaggio generato da una qualsiasi grammatica libera da contesto. Oppure è possibile definire un automa ad hoc. Vediamo questa seconda strada. Sappiamo già come fare un automa che accetta i palindromi. Dobbiamo solo fare in modo che vengano accettati solo palindromi di lunghezza almeno 2 e pari. Poi dovremo collegare tra loro 2 automi di questo tipo. L'automa finale è il seguente:

$\delta(q_0, 0, Z) = \{(q_1, 0Z)\}$	$\delta(q_0, 1, Z) = \{(q_1, 1Z)\}$	$\delta(q_1, 0, ?) = \{(q_1, 0?)\}$
$\delta(q_1, 1, ?) = \{(q_1, 1?)\}$	$\delta(q_1, \epsilon, ?) = \{(q_2, ?)\}$	$\delta(q_2, 0, 0) = \{(q_2, \epsilon)\}$
$\delta(q_2, 1, 1) = \{(q_2, \epsilon)\}$	$\delta(q_2, \epsilon, Z) = \{(q'_0, Z)\}$	$\delta(q'_0, 0, Z) = \{(q'_1, 0Z)\}$
$\delta(q'_0, 1, Z) = \{(q'_1, 1Z)\}$	$\delta(q'_1, 0, ?) = \{(q'_1, 0?)\}$	$\delta(q'_1, 1, ?) = \{(q'_1, 1?)\}$
$\delta(q'_1, \epsilon, ?) = \{(q'_2, ?)\}$	$\delta(q'_2, 0, 0) = \{(q'_2, \epsilon)\}$	$\delta(q'_2, 1, 1) = \{(q'_2, \epsilon)\}$

lo stato finale è q'_2 . Aiuta la comprensione osservare il fatto che si tratta di 2 automi praticamente uguali in cascata. Ciascun automa consiste di tre stati, q_0, q_1 e q_2 , (nel secondo automa gli stati sono q'_0, q'_1 e q'_2). q_0 garantisce che il palindromo sia di lunghezza almeno 2, poi q_1 carica l'input nello stack e q_2 matcha la seconda parte dell'input con lo stack.

- Data una stringa w in $\{0,1\}^*$, con \hat{w} indichiamo la stringa ottenuta da w scambiando ciascuno 0 con un 1 e ciascun 1 con uno 0. Insomma \hat{w} è il complemento di w . $L(G)$ contiene stringhe w_1, w_2 tali che sia w_1 che w_2 hanno la forma $q\hat{q}^R$, dove R indica l'inversa di una stringa e sono di lunghezza almeno 2 e pari.

Tempo a disposizione: 2 h 15 min

Gli esercizi della Parte I e della Parte II vanno consegnati su due fogli differenti

Parte I – Linguaggi Regolari e Linguaggi Liberi da Contesto

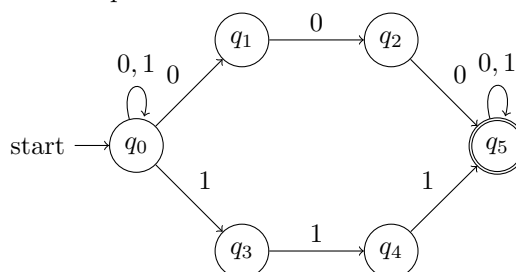
1. Data la seguente espressione regolare

$$(0 + 1)^*(111 + 000)(0 + 1)^*$$

- descrivere in italiano il linguaggio generato dall'espressione
- costruire un automa a stati finiti equivalente all'espressione

Soluzione:

- L'espressione regolare genera il linguaggio di tutte le stringhe sull'alfabeto $\{0, 1\}$ che contengono una sequenza di tre 0 consecutivi oppure una sequenza di tre 1 consecutivi.
- La descrizione a parole del linguaggio riconosciuto dall'espressione regolare ci permette di costruire un automa equivalente composto di 6 stati.



Lo stato iniziale q_0 si occupa di riconoscere il prefisso $(0 + 1)^*$ e poi sceglie nondeterministicamente se riconoscere la sequenza 000 passando per gli stati q_1, q_2 e q_5 , oppure la sequenza 111 passando per gli stati q_3, q_4 e q_5 . Lo stato finale q_5 riconosce il suffisso $(0 + 1)^*$ dell'espressione regolare.

2. Il linguaggio

$$L = \{0^n 1^m 0^{n \cdot m} \mid n \cdot m > 0\}$$

è regolare? Motivare la risposta.

Soluzione. Intuitivamente, il linguaggio non può essere regolare perché per riconoscere le parole occorre contare il numero di 0 nella prima sequenza 0^n , il numero di 1 nella seconda sequenza 1^m e poi moltiplicare i due valori per controllare se la sequenza finale di 0 è della lunghezza corretta. Visto che n ed m sono illimitati, questo richiederebbe una quantità infinita di memoria che un automa a stati finiti non possiede.

Per dimostrarlo formalmente, assumiamo per assurdo che L sia regolare:

- sia $n > 0$ la lunghezza data dal Pumping Lemma;
- consideriamo la parola $v = 0^n 110^{2n}$, che è di lunghezza maggiore di n ed è nella forma $0^n 1^m 0^{n \cdot m}$ se poniamo $m = 2$. Quindi v appartiene ad L ;
- sia $v = xyz$ una suddivisione arbitraria di v tale che $y \neq \varepsilon$ e $|xy| \leq n$;
- poiché $|xy| \leq n$, allora xy è completamente contenuta nella prima sequenza di zeri, 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^0z è nella forma $0^{n-p} 110^{2n}$, e non può appartenere a L perché $2(n-p) < 2n$.

Abbiamo trovato un assurdo quindi L non può essere regolare.

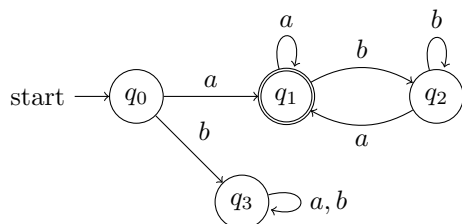
Soluzione della Parte I – Linguaggi Regolari

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)

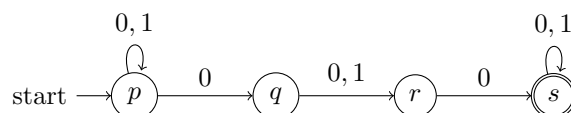


(b) Alcune soluzioni possibili:

$$a(a + b)^*a + a$$

$$a(b^*a)^*$$

2. Dato il seguente NFA

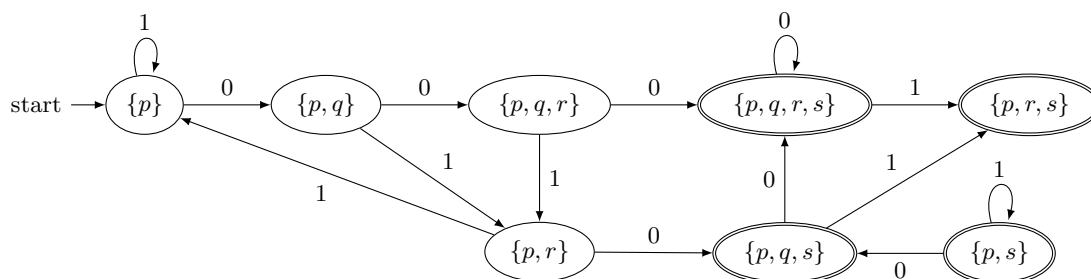


costruire un DFA equivalente

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 omissi):

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

e con il seguente diagramma di transizione:



3. Il linguaggio

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

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:

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

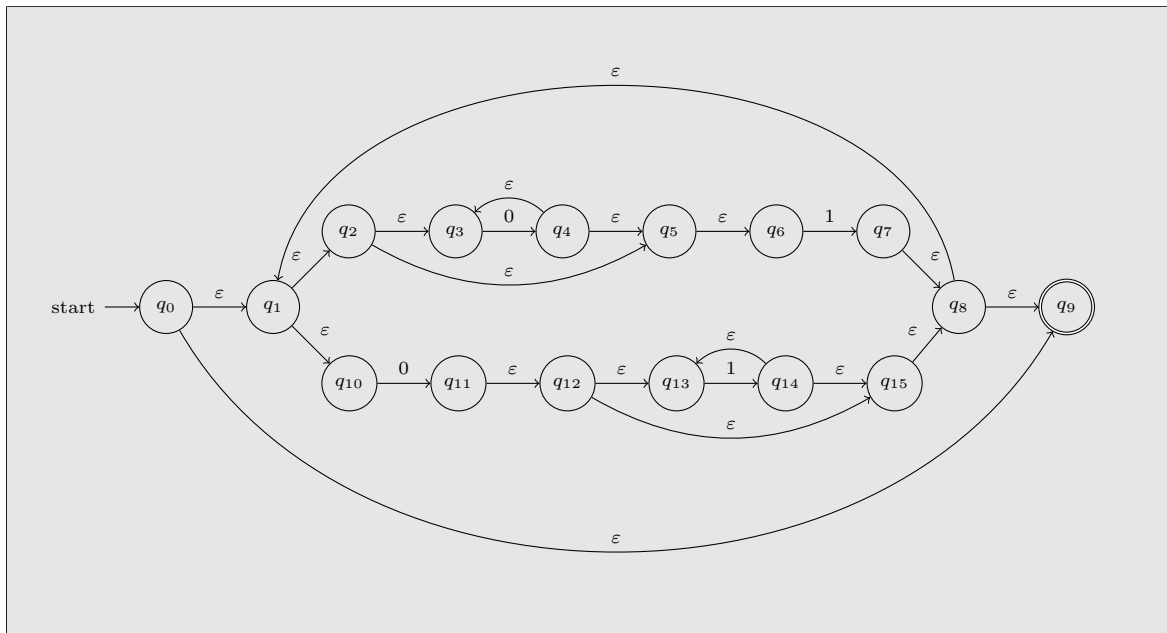
Soluzione: Per dimostrare che $\text{init}(L)$ è regolare vediamo come è possibile costruire un automa a stati finiti che riconosce $\text{init}(L)$ a partire dall'automata a stati finiti che riconosce L .

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 $G = \{q \in Q : \text{esiste una sequenza di transizioni da } q \text{ ad uno stato finale } f \in F\}$, ossia come tutti gli stati a partire dai quali possiamo raggiungere uno stato finale di A . Dobbiamo dimostrare che $L(B) = \text{init}(L)$.

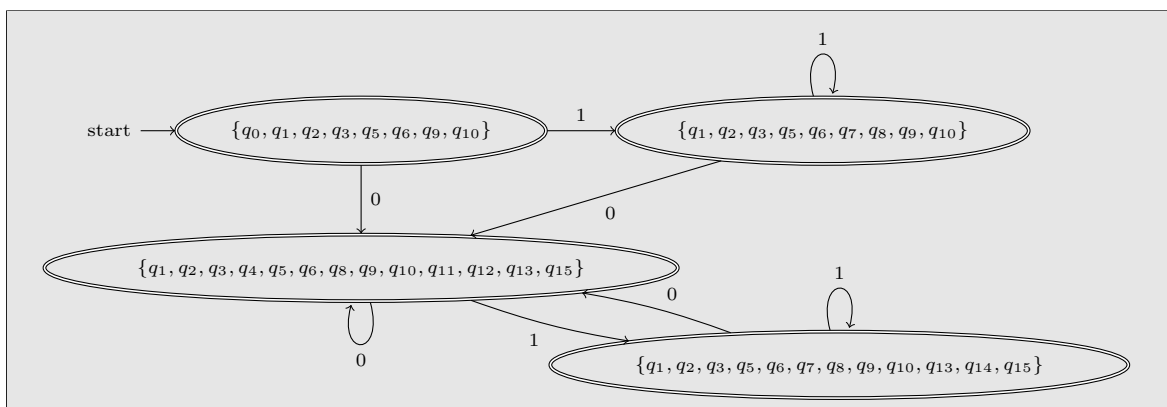
- Sia $w \in \text{init}(L)$: allora per la definizione deve esistere $x \in \Sigma^*$ tale che $wx \in L$. Poiché A è un automa deterministico, esiste una sola sequenza di transizioni che parte da q_0 e accetta la parola wx in A . Possiamo spezzare questa sequenza in due parti: una prima sequenza che 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'automata 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 $\text{init}(L)$ segue che $w \in \text{init}(L)$.

Tempo a disposizione: 1 h 30 min

1. (a) Convertire l'espressione regolare $(0^*1 + 01^*)^*$ 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 $L_1 = \{0^{2^n}1^n : n \geq 0\}$ 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 $w = 0^{2^h}1^h$, 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^{2^h} 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 $0^{2^h+p}1^h$, 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:

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

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

Per dimostrare che $\text{dehash}(L)$ è regolare vediamo come è possibile costruire un automa a stati finiti che riconosce $\text{dehash}(L)$ 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 :

$$S \rightarrow iS \mid iSeS \mid \epsilon$$

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

$L = \{w \in \{i, e\}^* \mid \text{per ogni prefisso di } w \text{ il numero di } i \text{ è maggiore o uguale al numero di } e\}$
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:

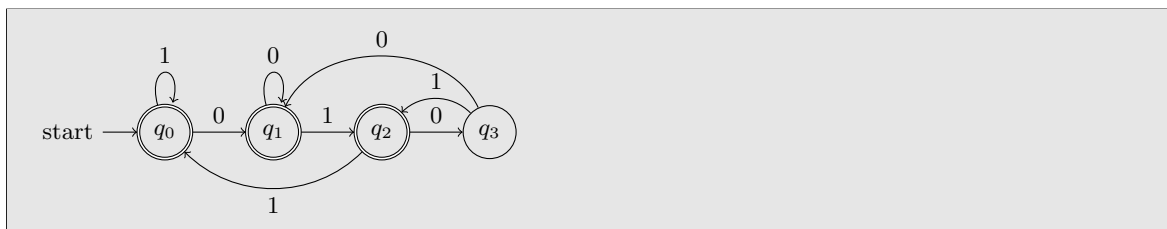
- $S \Rightarrow iSeS \Rightarrow iiSeS \Rightarrow^* iie$
- $S \Rightarrow iS \Rightarrow iiSeS \Rightarrow^* iie$

- (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.

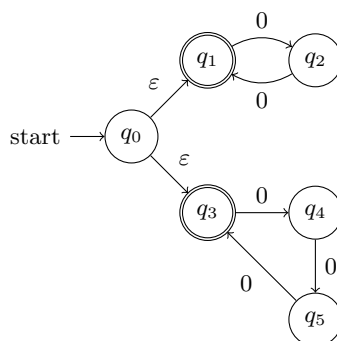
$$\begin{aligned} S &\rightarrow iS \mid iS'eS \mid \epsilon \\ S' &\rightarrow iS'eS' \mid \epsilon \end{aligned}$$

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 $\Sigma = \{0,1\}$:

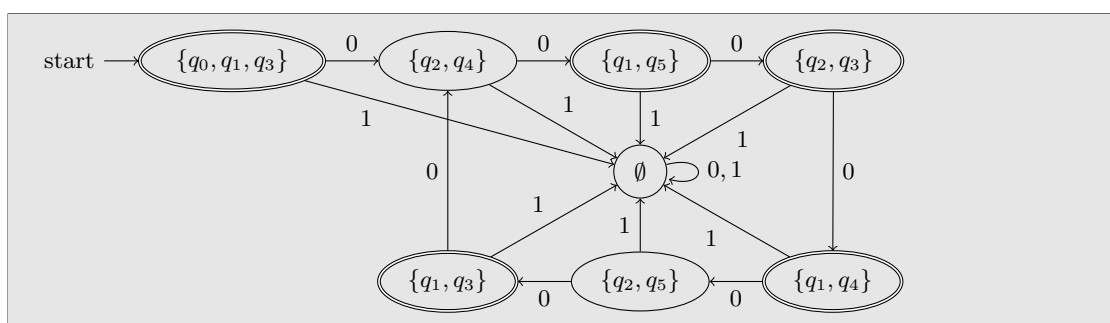


- (a) Descrivete il linguaggio riconosciuto dall'automa.

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 $w = 01^h 0^h$, 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:

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

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

Per dimostrare che $\text{suffixes}(L)$ è regolare basta mostrare come costruire un automa a stati finiti che riconosce $\text{suffixes}(L)$ 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 assegniamo 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

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

$$S \rightarrow S_1 C \mid A S_2$$

$$S_1 \rightarrow \varepsilon \mid a S_1 b$$

$$S_2 \rightarrow \varepsilon \mid b S_2 c$$

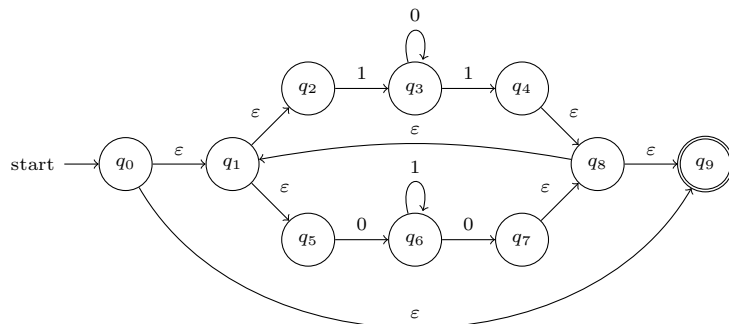
$$A \rightarrow \varepsilon \mid a A$$

$$C \rightarrow \varepsilon \mid c C$$

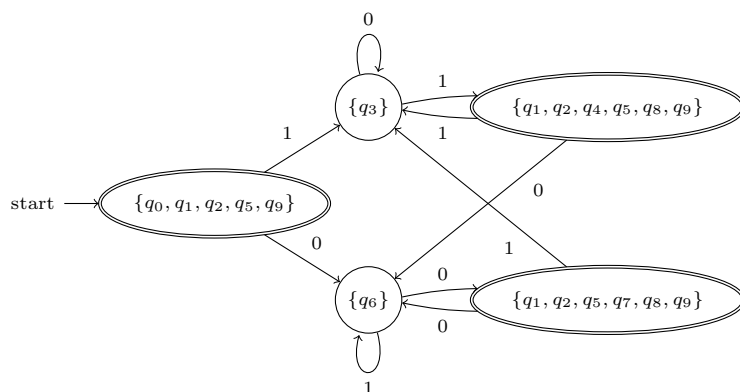
Tempo a disposizione: 2 ore

1. (a) Convertire l'espressione regolare $(10^*1 + 01^*0)^*$ in un ε -NFA (si può usare l'algoritmo visto a lezione oppure creare direttamente l'automa). **Importante:** l'automa deve essere nondeterministico e deve sfruttare le ε -transizioni per riconoscere il linguaggio.

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



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



2. Considerate il linguaggio $L = \{0^{2n}1^m0^n : n, m \geq 0\}$. 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 $w = 0^{2h}1^h0^h$, 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 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 $0^{2h+p}1^h0^h$, e quindi non appartiene al linguaggio perché il numero di 0 nella prima parte della parola non è uguale al doppio del numero di 0 nella seconda parte della parola.

Abbiamo trovato un assurdo quindi L_1 non può essere regolare.

3. Descrivere un automa a pila che accetti per stack vuoto il linguaggio che consiste di stringhe composte di a e b tali che, se n è il numero degli a , allora il numero dei b è tra n e $2n$. Il linguaggio contiene la stringa vuota. Si osservi che nelle stringhe del linguaggio, gli a e i b possono essere mescolati (cioè non necessariamente della forma $a^n b^m$).

Il PDA P che riconosce L per stack vuoto ha 2 stati q_1 e q_2 e le seguenti transizioni:

$$\delta(q_1, a, Z) = \{(q_1, aZ), (q_1, aaZ)\}, \delta(q_1, a, a) = \{(q_1, aa), (q_1, aaa)\}, \delta(q_1, a, b) = \{(q_1, \epsilon), (q_2, \epsilon)\},$$

$$\delta(q_1, \epsilon, Z) = \{(q_1, \epsilon)\},$$

$$\delta(q_1, b, a) = \{(q_1, \epsilon)\}, \delta(q_1, b, Z) = \{(q_1, bZ)\}, \delta(q_1, b, b) = \{(q_1, bb)\}$$

$$\delta(q_2, \epsilon, b) = \{(q_1, \epsilon)\}, \delta(q_2, \epsilon, Z) = \{(q_1, aZ)\}$$

Lo stato q_2 serve quando si consuma l'input a si decide che deve contare doppio e la cima della pila contiene b , allora la transizione da q_1 a q_2 consuma il primo b della pila e lo stato q_2 vede quello che c'era sotto quel b . Se c'era un altro b , lo toglie, mentre se c'era Z , allora aggiunge a allo stack. In entrambi i casi torna in q_1 .

4. Dare la definizione precisa del linguaggio L_{ne} e successivamente dimostrare che esso è un linguaggio RE. La prova richiede di esibire una TM che riconosca in tempo finito tutte le stringhe del linguaggio L_{ne} .

$L_{ne} = \{M \mid L(M) \neq \emptyset\}$. Per dimostrare che L_{ne} è RE è necessario definire una TM M che lo riconosce. Gli input di L_{ne} sono tutte le stringhe binarie che vengono interpretate come TM. La TM M sarà come segue: data in input una stringa binaria che rappresenta la TM M' , M genera nondeterministicamente tutte le possibili stringhe binarie e per ogni tale stringa w , simula M' su w (per farlo usa la TM universale), se M' accetta qualcuna delle w prodotte, allora M accetta M' , infatti, in questo caso, M' appartiene a L_{ne} . Il punto centrale è la generazione nondeterministica di tutte le possibili stringhe binarie w . Ogni w è finita, e quindi verrà prodotta in un tempo finito e se esiste w che è accettata da M' , M accetterà M' in un tempo finito. Se invece M' non accetta alcuna w e quindi $M' \notin L_{ne}$, allora M continuerà il suo calcolo per sempre e questo è compatibile con il fatto che L_{ne} sia RE.

5. Stabilire se il seguente linguaggio $L = \{a^n b^m c^k \mid n = k, n \geq 0, m \geq 0, k \geq 0\}$ è CF oppure no. Se pensate che sia CF, esibite una CFG che lo generi oppure un PDA che lo riconosca (spiegando perché questo è vero). Se pensate che non sia CF, date un argomento che dimostri che effettivamente non lo sia.

L è CF. La seguente grammatica CF genera L :

$$S \rightarrow aSc \mid B$$

$$B \rightarrow bB \mid \epsilon$$

6. Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Considerate il seguente problema, che chiameremo HAM375: dato un grafo G con n vertici, trovare un ciclo che attraversa esattamente una volta $n - 375$ vertici del grafo (ossia tutti i vertici di G tranne 375).

- (a) Dimostrare che il problema HAM375 è in NP fornendo un certificato per il Sì che si può verificare in tempo polinomiale.

Sia n il numero di vertici del grafo che è istanza di HAM375. Un certificato per HAM375 è una sequenza ordinata di $n - 375$ vertici distinti. Occorre tempo polinomiale per verificare se ogni nodo è collegato al successivo e l'ultimo al primo.

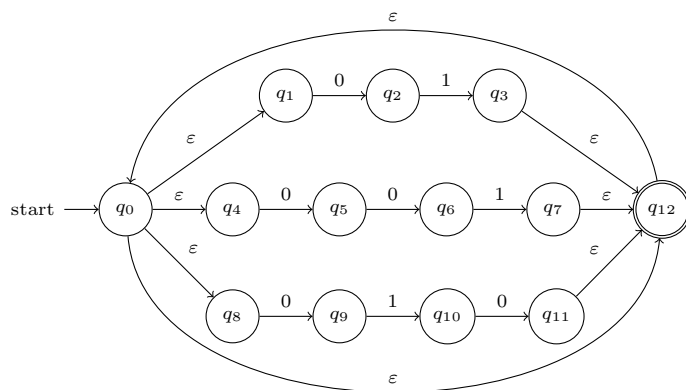
- (b) Mostrare come si può risolvere il problema del circuito Hamiltoniano usando il problema HAM375 come sottoprocedura.

Dato un qualsiasi grafo G che è un'istanza del problema del circuito Hamiltoniano, costruiamo in tempo polinomiale un nuovo grafo G' che è un'istanza di HAM375, aggiungendo a G 375 vertici isolati (senza archi). Chiaramente G' ha un ciclo che attraversa esattamente una volta $n - 375$ vertici del grafo se e solo se G ha un ciclo Hamiltoniano.

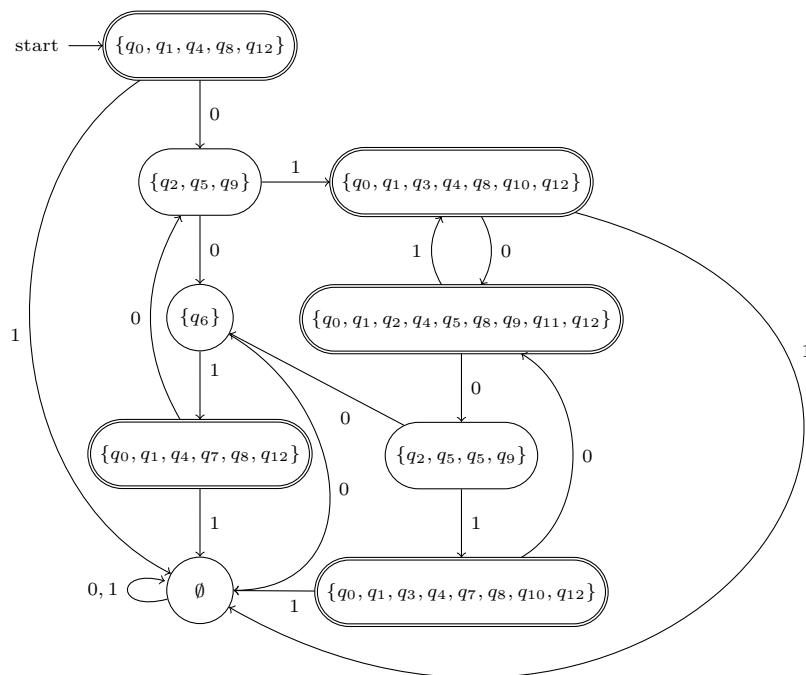
Tempo a disposizione: 2 h 30 m

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

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



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



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

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

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

Abbiamo trovato un assurdo quindi L non può essere regolare.

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

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

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

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

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

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

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

e la seconda derivazione è:

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

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

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

Il PDA ha 2 stati q_1 e q_2 e le seguenti transizioni:

$\delta(q_1, a, Z_0) = \{((q_1, aZ_0))\}$, $\delta(q_1, b, a) = \{((q_1, ba))\}$, $\delta(q_1, a, b) = \{((q_1, ab))\}$, $\delta(q_1, c, b) = \{((q_2, \epsilon))\}$,

$\delta(q_2, c, b) = \{((q_2, \epsilon))\}$, $\delta(q_2, a, a) = \{((q_2, \epsilon))\}$, $\delta(q_2, \epsilon, Z_0) = \{((q_2, \epsilon))\}$.

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

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

$L_U = \{(M, w) \mid w \in L(M)\}$. Esiste una macchina di Turing capace di accettare L_U . Questa macchina di Turing ha diversi nastri e prende in input una qualsiasi coppia (M, w) , scrive M su uno dei nastri, poi scrive w su un altro e $q_0 = 0$ sul terzo e simula il calcolo di M su w cercando le mosse sul primo nastro. Insomma sfrutta l'idea della macchina universale, che, avendo una qualsiasi macchina di Turing su un nastro, è capace di simulare il suo calcolo. Da questo argomento segue che L_U è RE. Per dimostrare che L_U non è ricorsivo, usiamo il fatto che, se lo fosse, allora anche \hat{L}_U lo sarebbe e, visto che potremmo ridurre L_d a \hat{L}_U , avremmo un assurdo. Per capire la riduzione è sufficiente capire che $\hat{L}_U = \{(M, w) \mid w \notin L(M)\}$. L_d ha come istanza una qualsiasi macchina di Turing M . Da una tale istanza di L_d , la corrispondente istanza di \hat{L}_U è semplicemente (M, M) . Ovviamente se \hat{L}_U fosse ricorsivo, sarebbe possibile determinare se $M \notin L(M)$ e quindi sarebbe possibile decidere L_d . Assurdo.

6. Il problema SETPARTITIONING chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 . Sappiamo che questo problema è NP-completo.

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

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

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

- (b) Mostrare come si può risolvere il problema SETPARTITIONING usando il problema SUBSETSUM come sottoprocedura.

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

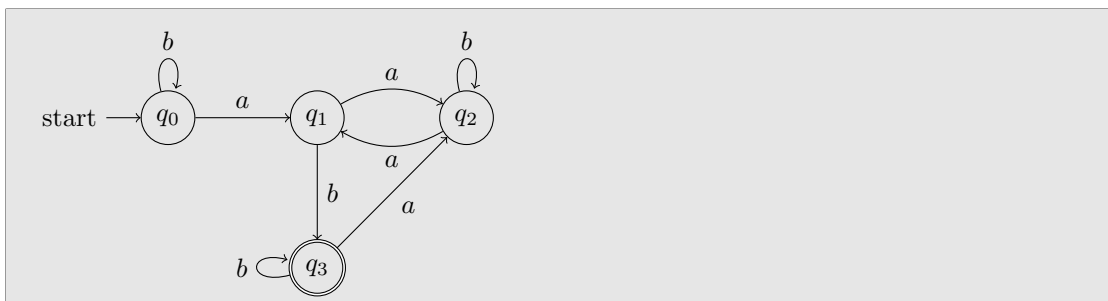
In questo modo, se S' è una soluzione di SUBSETSUM, allora la somma dei valori contenuti nell'insieme $S - S'$ sarà pari alla metà della somma degli elementi di S . Quindi ho diviso S in sue sottoinsiemi $S_1 = S'$ e $S_2 = S - S'$ tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 .

Viceversa, se S_1 e S_2 sono una soluzione di SETPARTITIONING, allora la somma dei numeri contenuti in S_1 sarà uguale alla somma dei numeri in S_2 , e quindi uguale alla metà della somma dei numeri contenuti in S . Quindi sia S_1 che S_2 sono soluzione di SUBSETSUM per il valore obiettivo t specificato sopra.

Soluzioni della Parte I – Linguaggi Regolari

1. Considerare il linguaggio $L = \{w \mid w \in \{a, b\}^* \text{ con un numero dispari di } a \text{ e che terminano con } b\}$

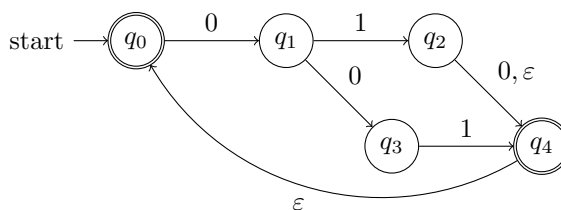
(a) Dare un automa a stati finiti *deterministico* che accetti il linguaggio L .



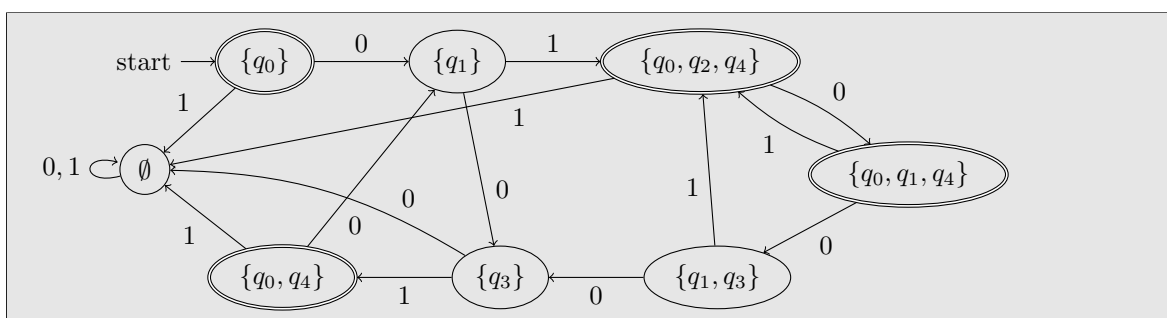
(b) Dare un'espressione regolare che rappresenti il linguaggio L .

$b^* a (b^* a b^* a)^* b^* b$

2. Dato il seguente NFA



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



3. Minimizzare il DFA che avete ottenuto come soluzione dell'esercizio 2 usando l'algoritmo riempi-tabella.

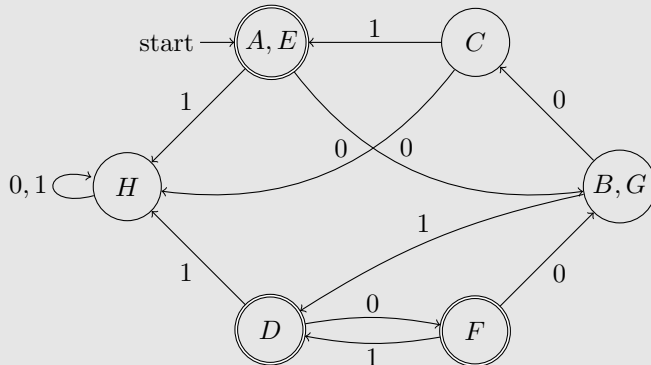
Rinominiamo gli stati del DFA soluzione dell'esercizio 2 come segue:

$A = \{q_0\}$	$B = \{q_1\}$	$C = \{q_3\}$	$D = \{q_0, q_2, q_4\}$
$E = \{q_0, q_4\}$	$F = \{q_0, q_1, q_4\}$	$G = \{q_1, q_3\}$	$H = \emptyset$

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

B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X	X	X		
G	X		X	X	X	X	
H	X	X	X	X	X	X	X
	A	B	C	D	E	F	G

Fondendo le due coppie di stati equivalenti $A \equiv E$ e $B \equiv G$ otteniamo il DFA minimo:



4. Sia $\Sigma = \{a, b, =\}$ e considerate il linguaggio

$$EQ = \{w=w \mid w \in \{a, b\}^*\}$$

Per esempio, la stringa **abab=abab** appartiene ad EQ perché la stringa a destra dell'uguale è identica alla stringa a sinistra dell'uguale. Viceversa, la stringa **aaaa=abb** non appartiene ad EQ perché la stringa a destra dell'uguale è diversa dalla stringa a sinistra dell'uguale.

- (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 $h = 4$

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

$$w = \text{aaaa=aaaa}$$

Giocatore 1: suddivide w in

- $x = a$
- $y = aa$
- $z = a=aaaa$

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

Giocatore 2: sceglie una potenza $k = 2$

La parola $xy^kz = \text{aaaaaa=aaaa} \notin EQ$: vince il **Giocatore 2**

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

Supponiamo per assurdo che EQ sia regolare:

- sia h la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = a^h=a^h$, che appartiene ad EQ 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 posto prima dell'uguale, 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 $a^{h+p}=a^h$, e non appartiene al linguaggio perché la stringa a destra dell'uguale è diversa dalla stringa a sinistra dell'uguale.

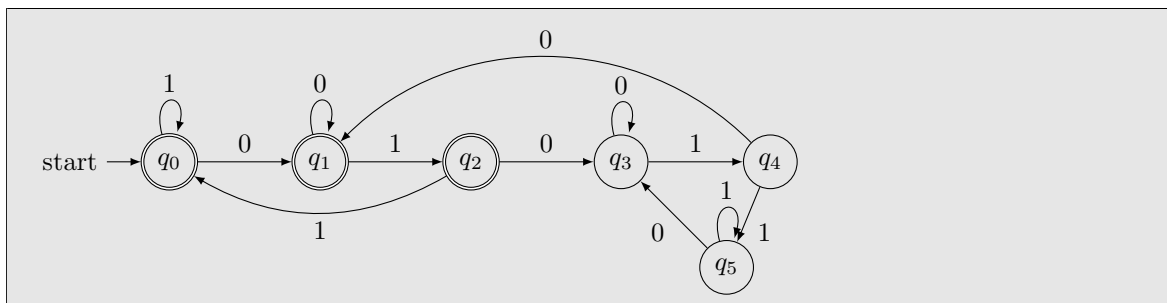
Abbiamo trovato un assurdo quindi EQ non può essere regolare.

Tempo a disposizione: 1 h 30 min

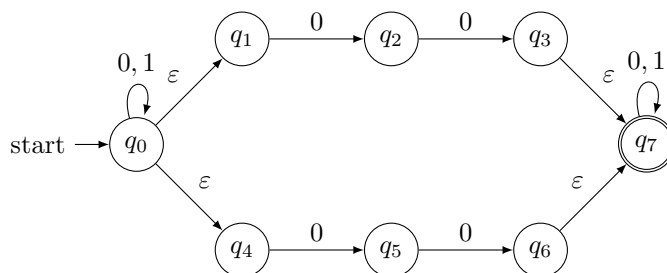
1. Definire un automa a stati finiti (di qualsiasi tipologia) che riconosca il linguaggio

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

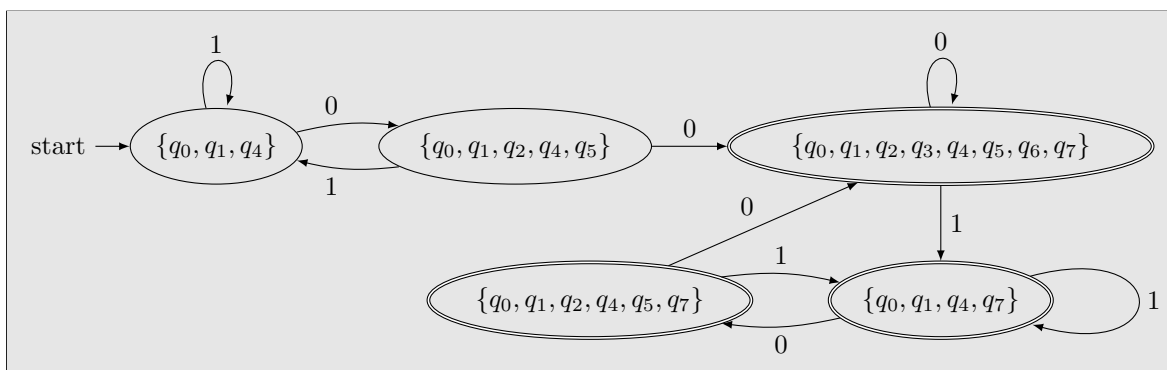
Per esempio, la parola 0100010 appartiene al linguaggio perché contiene due occorrenze di 010, la parola 01111 appartiene al linguaggio perché contiene zero occorrenze di 010, mentre la parola 0101010 non appartiene al linguaggio perché contiene tre occorrenze di 010.



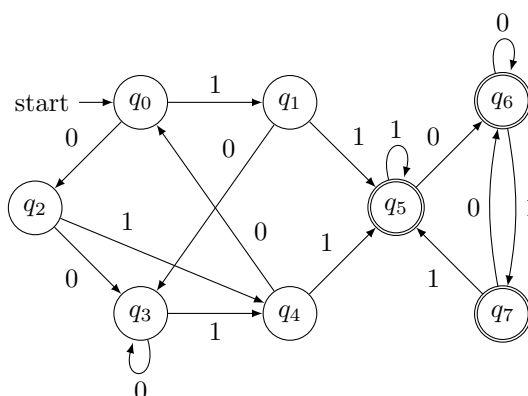
2. Dato il seguente ε -NFA



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



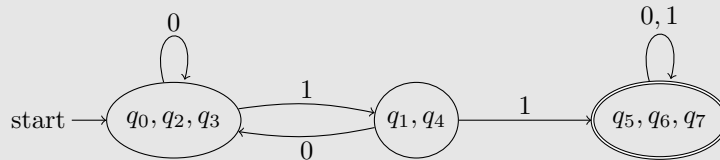
3. Minimizzare il seguente DFA usando l'algoritmo riempi-tabella.



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

q_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		
	q_0	q_1	q_2	q_3	q_4	q_5	q_6

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 $h = 2$

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

$$w = 00000011$$

Giocatore 1: suddivide w in

- $x = 0$
- $y = 0$
- $z = 000011$

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

Giocatore 2: sceglie una potenza $k = 2$

La parola $xy^kz = 0000000111 \notin M3N$: 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 : $S \rightarrow aA$, $A \rightarrow Bb \mid b$, $B \rightarrow aA \mid \epsilon$

- (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 $L(G) \subseteq L$.

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

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

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

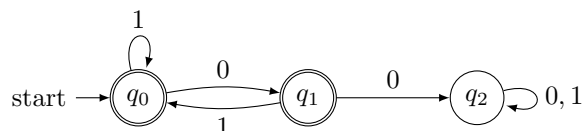
Base per $L(B)$: lunghezza 1. $B \Rightarrow \epsilon$, soddisfa la tesi.

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 $k > 0$.

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:



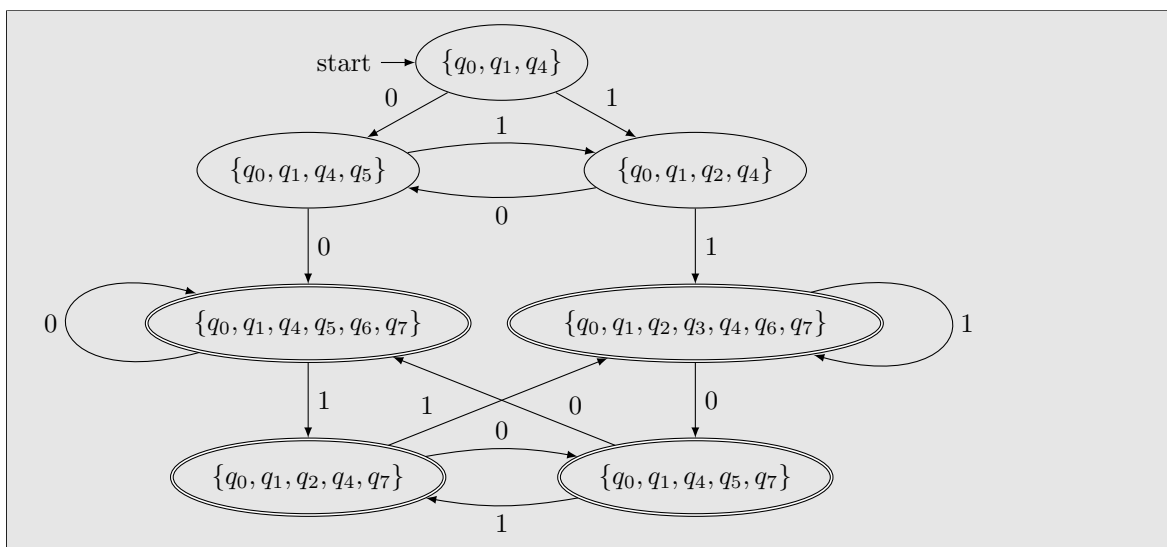
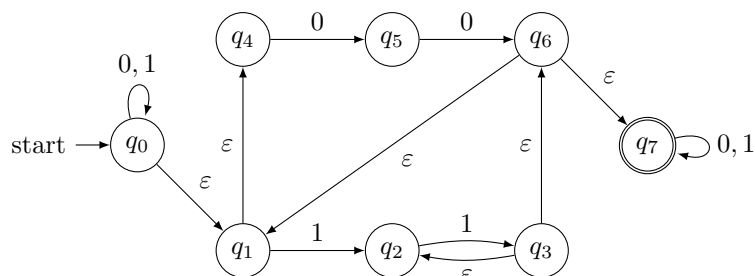
Trasformate il DFA in una Espressione Regolare usando l'algoritmo di eliminazione degli stati.

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 $h = 4$

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

$$w = 0000011000000$$

Giocatore 1: suddivide w in

- $x = 0$
- $y = 00$
- $z = 0011000000$

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

Giocatore 2: sceglie una potenza $k = 2$

La parola $xy^kz = 000000011000000 \notin LMN$: 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 : $S \rightarrow aBb$, $B \rightarrow aBb \mid BB \mid \varepsilon$

(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 $w'' \in L(B)$. $S \rightarrow aBb$ 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.

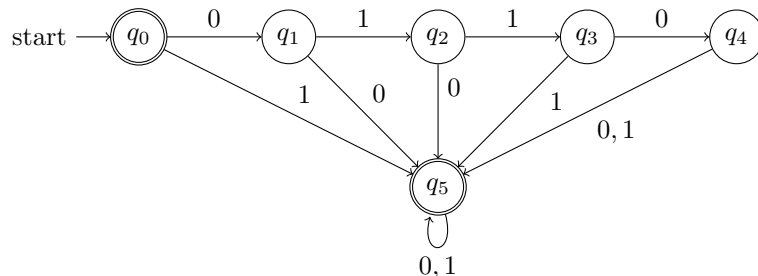
Base. Lunghezza 1: $B \Rightarrow \epsilon$ ed ϵ è ben parentesizzata.

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 $B \rightarrow (B)$ oppure $B \rightarrow BB$. Consideriamo il primo caso: $B \Rightarrow (B) \Rightarrow^n (w')$ e quindi $B \Rightarrow^n w'$, e allora, per ipotesi induttiva, w' è ben 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.

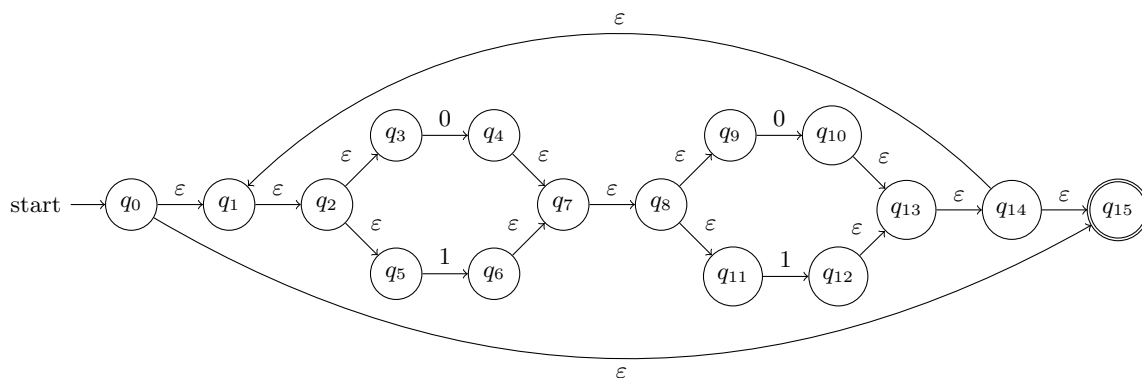
Tempo a disposizione: 1 h 30 min

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

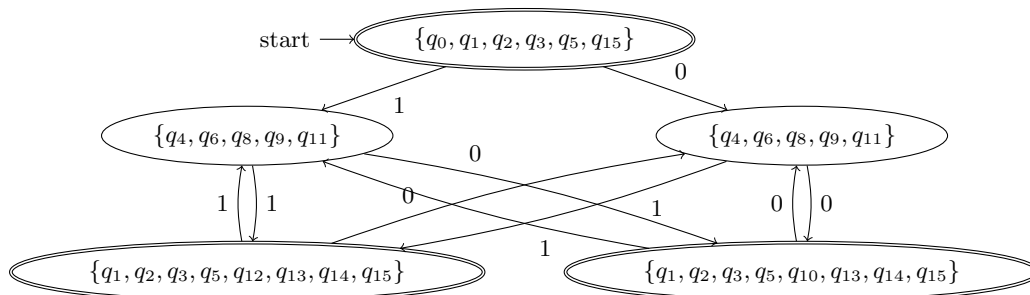
$$L = \{w \in \{0,1\}^* \mid w \neq 0110\}$$



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.

- Dire quale dei due linguaggi è regolare e quale non è regolare.
 - Per il linguaggio regolare, dare un automa a stati finiti o un'espressione regolare che lo rappresenta.
 - 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 $w = 1^n 0 1^n$, 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 $1^{n+p} 0 1^n$, 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 $L = \{a^n b^m c^k \mid \text{con } n = m \text{ o } m = k \text{ e } n, m \text{ e } k \geq 0\}$. Dimostrare che per la grammatica G che proponete, vale $L(G) \subseteq L$.

La grammatica G è come segue:

$$S \rightarrow AC \mid BD$$

$$A \rightarrow aAb \mid \epsilon$$

$$C \rightarrow cC \mid \epsilon$$

$$B \rightarrow aB \mid \epsilon$$

$$D \rightarrow bDc \mid \epsilon$$

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.

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

```

 $S_0 \leftarrow \text{ECLOSE}(q_0)$                                 ▷ 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- $\varepsilon$ -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
         $S' \leftarrow \text{ECLOSE}(S')$                         ▷ Chiusura di  $S'$  per  $\varepsilon$ -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:

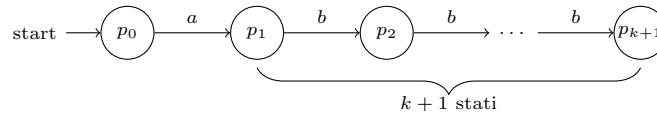
- l'insieme degli stati è l'insieme delle parti di Q_N : $Q_D = \{S \mid S \subseteq Q_N\}$;
- lo stato iniziale è la ε -chiusura di q_0 : $S_0 = \text{ECLOSE}(q_0)$;
- la funzione di transizione “simula” le transizioni di N :

$$\delta_D(S, a) = \text{ECLOSE}\left(\bigcup_{p \in S} \delta_N(p, a)\right)$$

- 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, \dots, 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 $xy^2 z$ non appartiene ad L' perché contiene due a ;
 - y contiene solamente b . In questo caso la parola $xy^2 z$ 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.

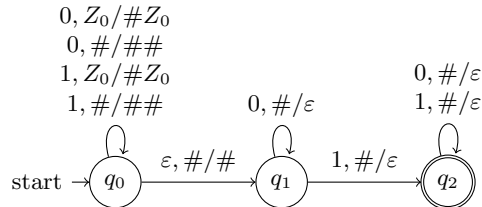
- Poniamo come lunghezza del pumping $k = 2$.
- Data una qualsiasi parola $w = a^\ell b^m c^n \in L_1$ di lunghezza maggiore o uguale a 2, si possono presentare vari casi, secondo il numero di a presenti nella parola:
 - se c'è una sola a , allora $w = ab^m c^m$. Scegliamo la suddivisione $x = \varepsilon$,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

$$S \rightarrow 0S0 \mid 1S0 \mid 0T1 \mid 1T1$$

$$T \rightarrow 0T \mid 1T \mid \varepsilon$$

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 .

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

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

L'automa che accetta $\text{faro}(L, M)$ 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_{uff} e M_{uff} 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 $t \in \{L, M, L_{\text{uff}}, M_{\text{uff}}\}$, e dobbiamo aggiungere le seguenti transizioni all'automa:

- $(r_L, r_M, L) \xrightarrow{a} (s_L, r_M, L_{\text{uff}})$ se $\delta_L(r_L, a) = s_L$ e r_M è uno stato finale di A_M ;
- $(r_L, r_M, M) \xrightarrow{a} (r_L, s_M, M_{\text{uff}})$ se $\delta_M(r_M, a) = s_M$ e r_L è uno stato finale di A_L ;
- $(r_L, r_M, L_{\text{uff}}) \xrightarrow{a} (s_L, r_M, L_{\text{uff}})$ se $\delta_L(r_L, a) = s_L$;
- $(r_L, r_M, M_{\text{uff}}) \xrightarrow{a} (r_L, s_M, M_{\text{uff}})$ se $\delta_M(r_M, a) = s_M$.

Si può notare che se il turno diventa uguale a L_{uff} allora A prosegue simulando solamente le transizioni di A_L , senza cambiare lo stato r_M . Viceversa, quando il turno diventa uguale a M_{uff} 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

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

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

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

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 $A \rightarrow u$ 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 $A \rightarrow R + S$ con le due regole $A \rightarrow R$ e $A \rightarrow S$;
- (b) per ogni regola $A \rightarrow R.S$, aggiungi due nuove variabili A_R e A_S e rimpiazza la regola con le regole $A \rightarrow A_R A_S$, $A \rightarrow R$ e $A \rightarrow S$;
- (c) per ogni regola $A \rightarrow S^*$, aggiungi una nuova variabile A_S e rimpiazza la regola con le regole $A \rightarrow A_S A$, $A \rightarrow \varepsilon$ e $A_S \rightarrow S$;
- (d) rimpiazza ogni regola $A \rightarrow \emptyset$ con la regola $A \rightarrow A$;
- (e) ripeti da (a) finché non rimangono solamente regole del tipo $A \rightarrow u$ dove u è una stringa di variabili e terminali, oppure $A \rightarrow \varepsilon$.

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” $A \rightarrow u$, senza operatori regolari.

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:

$$\begin{aligned}\delta((r_L, r_M), 0) &= \{(\delta_L(r_L, 0), \delta_M(r_M, 0)), (\delta_L(r_L, 1), \delta_M(r_M, 0)), (\delta_L(r_L, 0), \delta_M(r_M, 1))\} \\ \delta((r_L, r_M), 1) &= \{(\delta_L(r_L, 1), \delta_M(r_M, 1))\}\end{aligned}$$

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 $xy^0 z$ ha la forma

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

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 Chomsky. Di conseguenza le regole di G sono solamente di due tipi: $A \rightarrow BC$, con A, B, C simboli non terminali, oppure $A \rightarrow b$ con b simbolo nonterminale.

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

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

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 $V' \rightarrow V$ e $V' \rightarrow \varepsilon$ appartengono a G' ;
- per ogni regola $V \rightarrow AB$ di G , le regole $V' \rightarrow AB'$ e $V' \rightarrow A'$ 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 $\Gamma = \{0, 1, _ \}$. 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 le Turing machine con alfabeto binario 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 $w = w_1w_2 \dots w_n$ 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:

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

LOAD BALANCE è il problema di trovare un assegnamento con makespan minore o uguale al limite superiore k :

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

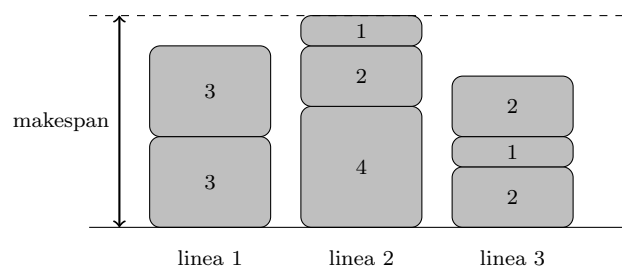


Figura 1: Esempio di assegnamento dei lavori $T = \{1, 1, 2, 2, 2, 3, 3, 4\}$ su 3 linee con makespan 7.

(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 $\text{makespan}(A)$: 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 $\langle T \rangle \in \text{SETPARTITIONING}$, 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 $\langle 2, T, k \rangle \in \text{LOADBALANCE}$, 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.