Esercizi TdL

Indice

1. Esercizi lezione 01 e 02 [28/02]

1.1. Esercizio 01

Esercizio 1.1.1: Considerate l'alfabeto $\Sigma = \{a, b\}$.

Richiesta 1.1.1.1: Fornite una grammatica CF per il linguaggio delle stringhe palindrome di lunghezza pari su Σ, cioè per l'insieme $\text{PAL}_{\text{pari}} = \{ww^R \mid w \in \Sigma^*\}$.

Soluzione 1.1.1.1: Definisco G tale che $V = \{S\}$ e

$$P = \{S \longrightarrow \varepsilon \mid aSa \mid bSb\}.$$

Richiesta 1.1.1.2: Modificate la grammatica precedente per generare l'insieme PAL di tutte le stringhe palindrome su Σ.

Soluzione 1.1.1.2: Definisco G tale che $V = \{S\}$ e

$$P = \{S \longrightarrow \varepsilon \mid aSa \mid bSb \mid a \mid b\}.$$

Richiesta 1.1.1.3: Per ogni $k \in \{0, ..., 3\}$, rispondete alla domanda «Il linguaggio PAL è di tipo k?» giustificando la risposta.

Soluzione 1.1.1.3: Non è di tipo 3 per le produzioni $S \longrightarrow aSa \mid bSb$ ma è di tipo 2 visto che rispetta le restrizioni sulle produzioni. Di conseguenza, è anche di tipo 1 e di tipo 0.

Richiesta 1.1.1.4: Se sostituiamo l'alfabeto con $\Sigma = \{a, b, c\}$, le risposte al punto precedente cambiano? E se sostituiamo con $\Sigma = \{a\}$?

Soluzione 1.1.1.4: Se $\Sigma = \{a, b, c\}$ vanno aggiunte due produzioni che però sono nella forma di quelle precedenti, quindi le risposte non cambiano.

Se $\Sigma = \{a\}$ le uniche produzioni che abbiamo sono

$$S \longrightarrow \varepsilon \mid aS \mid a$$

e quindi la grammatica è di tipo 3. Di conseguenza, è anche di tipo 2, tipo 1 e tipo 0.

1.2. Esercizio 02

Esercizio 1.2.1: Considerate l'alfabeto $\Sigma = \{a, b\}$.

Richiesta 1.2.1.1: Scrivete una grammatica per generare il complemento di PAL.

Soluzione 1.2.1.1: Sia G tale che $V = \{S, D, B\}$ e P formato da

$$S \longrightarrow aSa \mid bSb \mid D$$
$$D \longrightarrow aDb \mid bDa \mid B$$
$$B \longrightarrow \varepsilon \mid a \mid b \mid S.$$

1.3. Esercizio 03

Esercizio 1.3.1: Sia $\Sigma = \{(,)\}$ un alfabeto i cui simboli sono la parentesi aperta e la parentesi chiusa.

Richiesta 1.3.1.1: Scrivete una grammatica CF che generi il linguaggio formato da tutte le sequenze di parentesi correttamente bilanciate, come ad esempio (()(()))().

Soluzione 1.3.1.1: Sia G una grammatica con $V = \{S\}$ e P formato da

$$S \longrightarrow \varepsilon \mid (S) \mid SS.$$

Richiesta 1.3.1.2: Risolvete il punto precedente per un alfabeto con due tipi di parentesi, come $\Sigma = \{(,), [,]\}$, nel caso non vi siano vincoli tra i tipi di parentesi (le tonde possono essere contenute tra quadre e viceversa). Esempio [()([])[]] ma non [[][(])()].

Soluzione 1.3.1.2: Sia G una grammatica con $V = \{S\}$ e P formato da

$$S \longrightarrow \varepsilon \mid (S) \mid [S] \mid SS.$$

Richiesta 1.3.1.3: Risolvete il punto precedente con $\Sigma = \{(,), [,]\}$, con il vincolo che le parentesi quadre non possano apparire all'interno di parentesi tonde. Esempio [()(())[][]](()()), ma non [()([])[]].

Soluzione 1.3.1.3: Sia G una grammatica con $V = \{S, T\}$ e P formato da

$$S \longrightarrow \varepsilon \mid [S] \mid SS \mid T$$
$$T \longrightarrow \varepsilon \mid (T) \mid TT.$$

1.4. Esercizio 04

Esercizio 1.4.1: Sia $G = (V, \Sigma, P, S)$ la grammatica con $V = \{S, B, C\}$, $\Sigma = \{a, b, c\}$ e P contenente le seguenti produzioni:

$$S \longrightarrow aSBC \mid aBC$$
$$CB \longrightarrow BC$$
$$aB \longrightarrow ab$$
$$bB \longrightarrow bb$$
$$bC \longrightarrow bc$$
$$cC \longrightarrow cc.$$

Richiesta 1.4.1.1: Dopo aver stabilito di che tipo è G, provate a derivare alcune stringhe. Riuscite a dire da quali stringhe è formato il linguaggio generato da G?

Soluzione 1.4.1.1: La grammatica G è di tipo 1.

Prima derivazione:

$$S \longrightarrow aBC \longrightarrow abC \longrightarrow abc.$$

Seconda derivazione:

$$\begin{aligned} S &\longrightarrow aSBC \longrightarrow aaBCBC \longrightarrow aabCBC \\ &\longrightarrow aabBCC \longrightarrow aabbCC \longrightarrow aabbcC \longrightarrow aabbcc. \end{aligned}$$

Possiamo dire che $L(G) = \{a^n b^n c^n \mid n \geq 1\}$.

1.5. Esercizio 05

Esercizio 1.5.1: Sia $G = (V, \Sigma, P, S)$ la grammatica con $V = \{S, B, C\}$, $\Sigma = \{a, b, c\}$ e P contenente le seguenti produzioni:

$$S \longrightarrow aBSc \mid abc$$
$$Ba \longrightarrow aB$$
$$Bb \longrightarrow bb.$$

Richiesta 1.5.1.1: Dopo aver stabilito di che tipo è G, provate a derivare alcune stringhe. Riuscite a dire da quali stringhe è formato il linguaggio generato da G?

Soluzione 1.5.1.1: La grammatica G è di tipo 1.

Prima derivazione:

$$S \longrightarrow abc.$$

Seconda derivazione:

$$S \longrightarrow aBSc \longrightarrow aBabcc \longrightarrow aaBbcc \longrightarrow aabbcc.$$

Come prima, possiamo dire che $L(G) = \{a^n b^n c^n \mid n \geq 1\}$.

1.6. Esercizio 06

Esercizio 1.6.1: Sia $G = (V, \Sigma, P, S)$ la grammatica con $V = \{S, A, B, C, D, E\}$, $\Sigma = \{a, b\}$ e P contenente le seguenti produzioni:

$$S \longrightarrow ABC$$
$$AB \longrightarrow aAD \mid bAE \mid \varepsilon$$
$$DC \longrightarrow BaC$$
$$EC \longrightarrow BbC$$
$$Da \longrightarrow aD$$
$$Db \longrightarrow bD$$
$$Ea \longrightarrow aE$$
$$Eb \longrightarrow bE$$
$$C \longrightarrow \varepsilon$$
$$aB \longrightarrow Ba$$
$$bB \longrightarrow Bb.$$

Richiesta 1.6.1.1: Dopo aver stabilito di che tipo è G, provate a derivare alcune stringhe. Riuscite a dire da quali stringhe è formato il linguaggio generato da G?

Suggerimento. Per ogni $w \in \{a, b\}^*$ è possibile costruire una derivazione $S \overset{*}{\Rightarrow} wABwC$ (provate a procedere per induzione sulla lunghezza di w cercando di capire il ruolo di ciascuna delle variabile nel processo di derivazione).

Soluzione 1.6.1.1: La grammatica G è di tipo 1.

Prima derivazione:

$$S \longrightarrow ABC \longrightarrow C \longrightarrow \varepsilon.$$

Seconda derivazione:

$$S \longrightarrow ABC \longrightarrow aADC \longrightarrow aABaC \longrightarrow aaC \longrightarrow aa.$$

Terza derivazione:

$$\begin{aligned} S \longrightarrow ABC \longrightarrow aADC \longrightarrow aABaC \longrightarrow abAEaC \longrightarrow abAaEC \\ \longrightarrow abAaBbC \longrightarrow abABabC \longrightarrow ababC \longrightarrow abab. \end{aligned}$$

Possiamo dire che $L(G) = \{ww \mid w \in \Sigma^*\}$.

1.7. Esercizio 07

Esercizio 1.7.1: Sia $G = (V, \Sigma, P, S)$ la grammatica con $V = \{S, A, B, C, X, Y, L, R\}$, $\Sigma = \{a\}$ e P contenente le seguenti produzioni:

$$\begin{gathered} S \longrightarrow LXR \\ LX \longrightarrow LYYA \mid aC \\ AX \longrightarrow YYA \\ AR \longrightarrow BR \\ YB \longrightarrow BX \\ LB \longrightarrow L \\ CX \longrightarrow aC \\ CR \longrightarrow \varepsilon. \end{gathered}$$

Richiesta 1.7.1.1: Riuscite a stabilire da quali stringhe è formato il linguaggio generato da G?

Suggerimento. Si può osservare che $LX^iR \overset{*}{\Rightarrow} LY^{2i}AR \Rightarrow LX^{2i}R$ per ogni $i > 0$. Inoltre dal simbolo iniziale si ottiene la forma LXR. Le ultime tre produzioni sono utili per sostituire variabili in una forma sentenziale con occorrenze di terminali.

Soluzione 1.7.1.1: La grammatica G è di tipo 0.

Prima derivazione:

$$S \longrightarrow LXR \longrightarrow aCR \longrightarrow a.$$

Seconda derivazione:

$$\begin{aligned} S &\longrightarrow LXR \longrightarrow LYYAR \longrightarrow LYYBR \longrightarrow LYBXR \\ &\longrightarrow LBXXR \longrightarrow LXXR \longrightarrow aCXR \longrightarrow aaCR \longrightarrow aa. \end{aligned}$$

Terza derivazione:

$$\begin{aligned} S &\longrightarrow LXR \longrightarrow LYYAR \longrightarrow LYYBR \longrightarrow LYBXR \longrightarrow LBXXR \longrightarrow LXXR \\ &\longrightarrow LYYAXR \longrightarrow LYYYYAR \longrightarrow LYYYYBR \longrightarrow LYYYBXR \\ &\longrightarrow LYYBXXR \longrightarrow LYBXXXR \longrightarrow LBXXXXR \longrightarrow LXXXXR \\ &\longrightarrow aCXXXR \longrightarrow aaCXXR \longrightarrow aaaCXR \longrightarrow aaaaCR \longrightarrow aaaa. \end{aligned}$$

Possiamo dire che $L(G) = \{a^{2^n} \mid n \geq 0\}$.

1.8. Esercizio 08

Esercizio 1.8.1:

Richiesta 1.8.1.1: Modificate la grammatica dell'esercizio 7 in modo da ottenere una grammatica di tipo 1 che generi lo stesso linguaggio.

Soluzione 1.8.1.1: La produzione che dà problemi è $LB \longrightarrow L$. La facciamo diventare

$$LB \longrightarrow CRL.$$

In questo modo rispettiamo tutti i vincoli delle grammatiche di tipo 1 e non modifichiamo la grammatica, visto che CR non genera problemi con la L o con la a quando facciamo le sostituzioni finali.

1.9. Esercizio 09

Esercizio 1.9.1:

Richiesta 1.9.1.1: Dimostrate che la grammatica $G = (\{A, B, S\}, \{a, b\}, P, S)$, con l'insieme delle produzioni P elencate sotto, genera il linguaggio $\{w \in \{a, b\}^* \mid \forall x \in \{a, b\}^* \quad w \neq xx\}$.

$$S \longrightarrow AB \mid BA \longrightarrow A \mid B$$
$$A \longrightarrow aAa \mid aAb \mid bAa \mid bAb \mid a$$
$$B \longrightarrow aBa \mid aBb \mid bBa \mid bBb \mid b$$

Soluzione 1.9.1.1: Eseguendo come prima produzione $S \longrightarrow A \mid B$ si ottengono delle stringhe di lunghezza dispari, che quindi non possono essere scritte come concatenazione di due stringhe uguali.

Eseguendo invece come prima produzione $S \longrightarrow AB \mid BA$ e facendo un numero di sostituzioni uguali per entrambe le parti, le due stringhe risultanti di ugual lunghezza avranno almeno una posizione differente, generate dall'ultimo cambio di A e B.

Eseguendo invece come prima produzione $S \longrightarrow AB \mid BA$ e facendo un numero di sostituzioni diverso per le due parti, non lo so dimostrare, secondo Martino lo dobbiamo fare ma io non lo farò, baci.

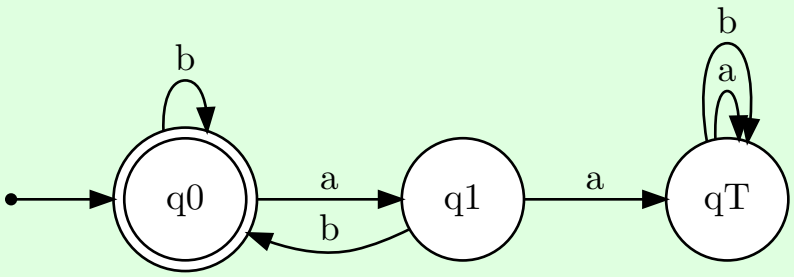
2. Esercizi lezioni 03, 04 e 05 [12/03]

2.1. Esercizio 01

Esercizio 2.1.1:

Richiesta 2.1.1.1: Costruite un automa a stati finiti che riconosca il linguaggio formato da tutte le stringhe sull'alfabeto $\{a, b\}$ nelle quali ogni a è seguita immediatamente da una b.

Soluzione 2.1.1.1:



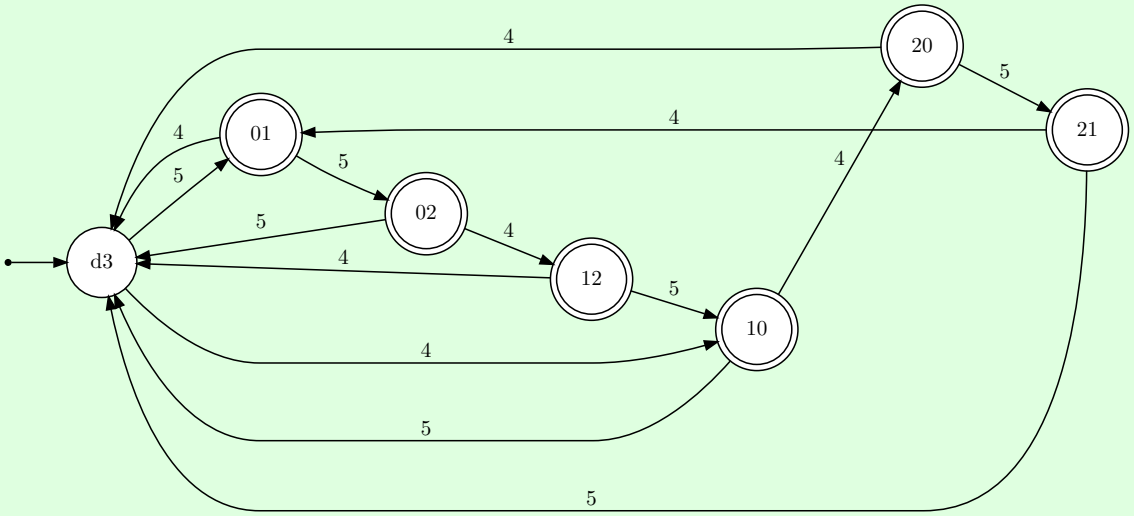
2.2. Esercizio 02

Esercizio 2.2.1:

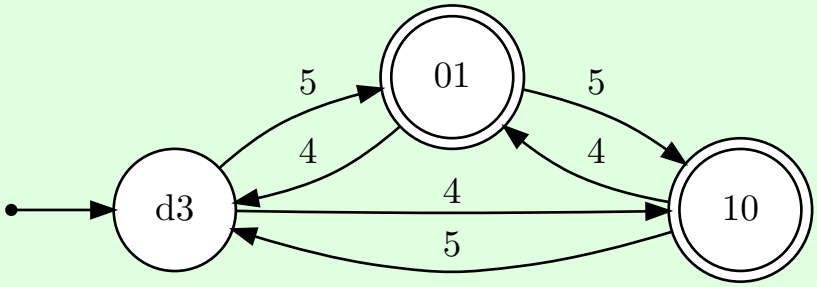
Richiesta 2.2.1.1: Costruite un automa a stati finiti che riconosca il linguaggio formato da tutte le stringhe sull'alfabeto $\{4, 5\}$ che, interpretate come numeri in base 10, rappresentano interi che non sono divisibili per 3.

Suggerimento. Un numero intero è divisibile per 3 se e solo se la somma delle sue cifre in base 10 è divisibile per 3.

Soluzione 2.2.1.1: Prima versione



Soluzione 2.2.1.2: Seconda versione

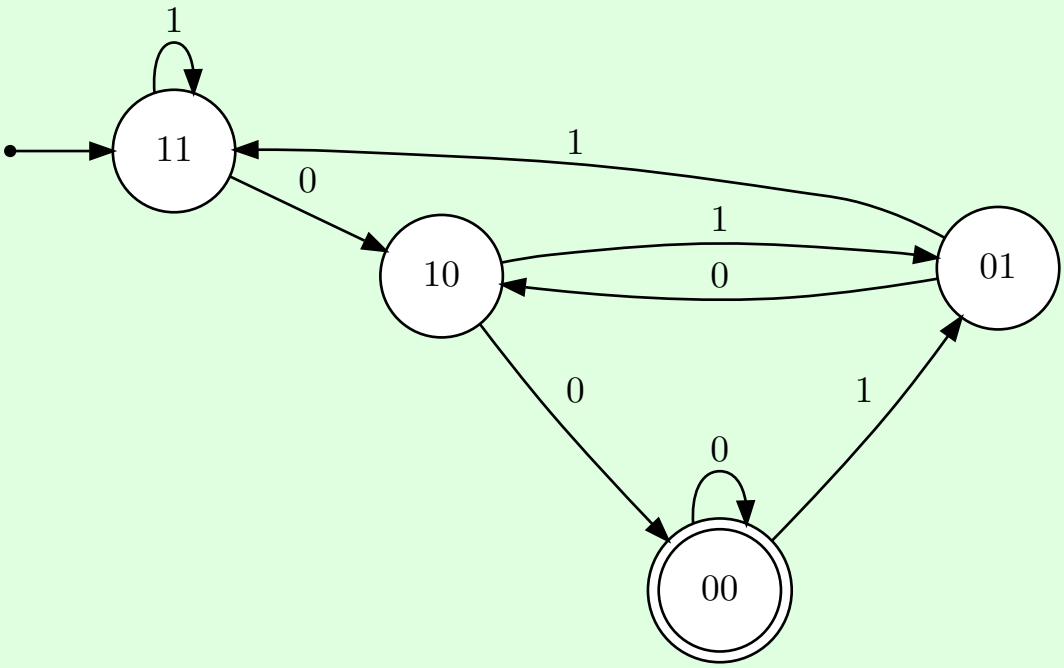


2.3. Esercizio 03

Esercizio 2.3.1:

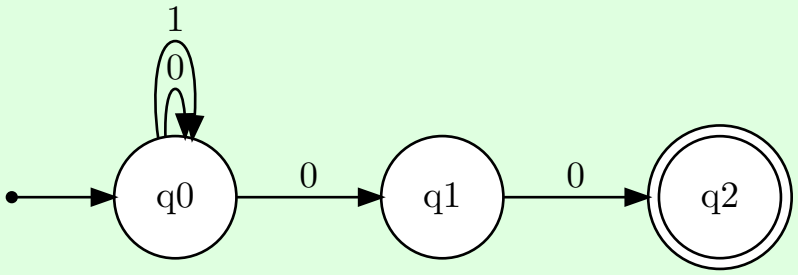
Richiesta 2.3.1.1: Costruite un automa a stati finiti deterministico che riconosca il linguaggio formato da tutte le stringhe sull'alfabeto $\{0, 1\}$ che, interpretate come numeri in notazione binaria, denotano multipli di 4.

Soluzione 2.3.1.1:



Richiesta 2.3.1.2: Utilizzando il non determinismo si riesce a costruire un automa con meno stati?

Soluzione 2.3.1.2: Utilizzando il non determinismo riusciamo ad utilizzare 1 stato in meno, se non inseriamo uno stato trappola per le transizioni dagli stati q_1 e q_2.



Richiesta 2.3.1.3: Generalizzate l'esercizio a multipli di 2^k, dove $k > 0$ è un intero fissato.

Soluzione 2.3.1.3: Per i DFA che riconoscono i multipli di 2^k dobbiamo ricordarci una finestra di k caratteri. Tutte le possibili combinazioni di queste finestre sono 2^k, quindi anche il DFA che riconosce quel linguaggio ha 2^k stati.

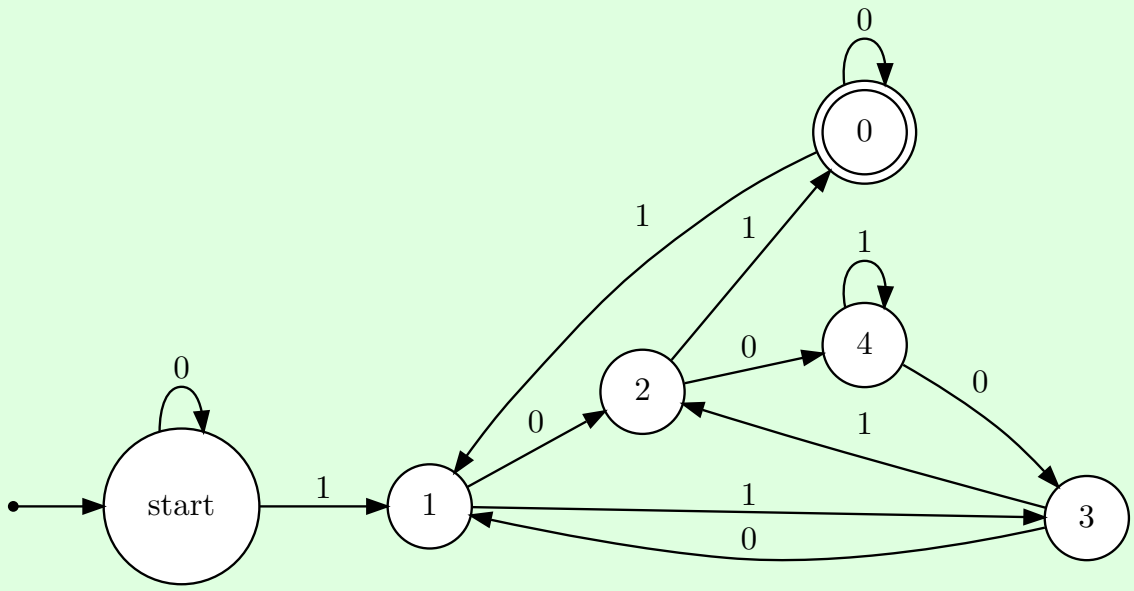
Per gli NFA che riconoscono i multipli di 2^k dobbiamo utilizzare $k+1$ stati, di cui k leggono gli ultimi k zeri e uno che fa da «stato scommettitore».

2.4. Esercizio 04

Esercizio 2.4.1:

Richiesta 2.4.1.1: Costruite un automa a stati finiti che riconosca il linguaggio formato da tutte le stringhe sull'alfabeto $\{0, 1\}$ che, interpretate come numeri in notazione binaria, rappresentano multipli di 5.

Soluzione 2.4.1.1:



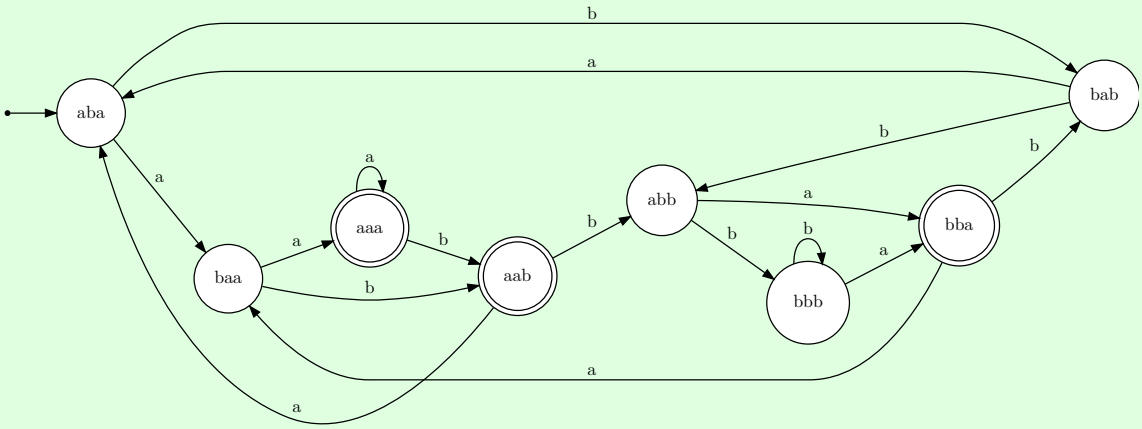
2.5. Esercizio 05

Esercizio 2.5.1: Considerate il seguente linguaggio:

$$L = \{w \in \{a, b\}^* \mid \text{il penultimo e il terzultimo simbolo di } w \text{ sono uguali}\}.$$

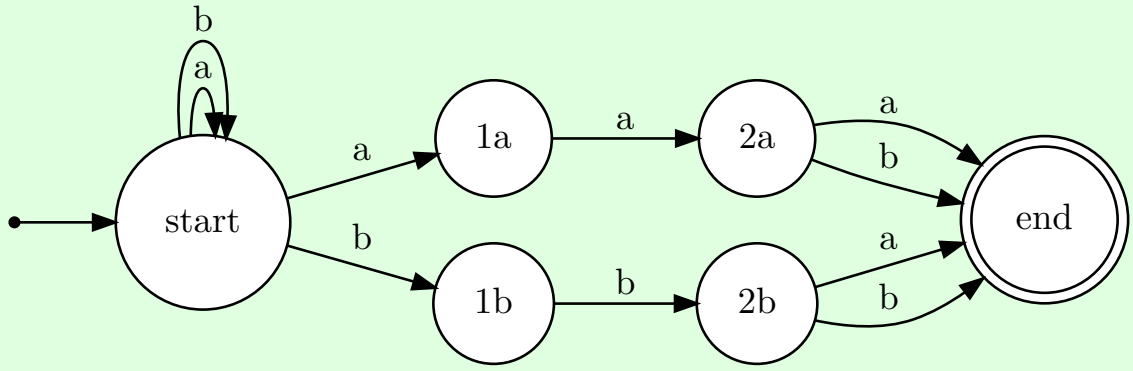
Richiesta 2.5.1.1: Costruite un automa a stati finiti deterministico che accetta L.

Soluzione 2.5.1.1: Secondo me andrebbero usati più stati per le stringhe di 1 e 2 caratteri, oppure si dovrebbe imporre il riconoscimento di stringhe lunghe almeno 3 caratteri.



Richiesta 2.5.1.2: Costruite un automa a stati finiti non deterministico che accetta L.

Soluzione 2.5.1.2:



Richiesta 2.5.1.3: Dimostrare che per il linguaggio L:

- tutte le stringhe di lunghezza 3 sono distinguibili tra loro;
- la parola vuota è distinguibile da tutte le stringhe di lunghezza 3.

Soluzione 2.5.1.3: Sia $X = \{a, b\}^3$. Date due stringhe $\sigma, \gamma \in X$ esse possono avere un carattere diverso in 3 posizioni:

- se $\sigma_1 \neq \gamma_1$:
  - se $\sigma_2 = \gamma_2$ usiamo $z = \varepsilon$;
  - se $\sigma_2 \neq \gamma_2$ usiamo $z = a^2$ oppure $z = b^2$;
- se $\sigma_2 \neq \gamma_2$:
  - se $\sigma_3 = \gamma_3$ usiamo $z \in \{a, b\}$;
  - se $\sigma_3 \neq \gamma_3$ usiamo $z = a^2$ oppure $z = b^2$;
- se $\sigma_3 \neq \gamma_3$ usiamo $z = a^2$ oppure $z = b^2$.

Non voglio dimostrare perché funziona, ma funziona, fidatevi di me.

Inoltre, la stringa vuota è distinguibile da ogni stringa di lunghezza 3 perché basta aggiungere una stringa z formata dall'ultimo carattere della stringa σ che stiamo considerando ripetuto due volte.

Richiesta 2.5.1.4: Utilizzando i risultati precedenti, ricavate un limite inferiore per il numero di stati di ogni automa deterministico che accetta L.

Soluzione 2.5.1.4: Grazie al teorema sulla distinguibilità, ogni DFA per il linguaggio L deve usare almeno 8 stati.

2.6. Esercizio 06

Esercizio 2.6.1: Costruite un insieme di stringhe distinguibili tra loro per ognuno dei seguenti linguaggi.

Richiesta 2.6.1.1: $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$.

Soluzione 2.6.1.1: $X = \{a^n \mid n \geq 0\}$.

Richiesta 2.6.1.2: $\{a^n b^n \mid n \geq 0\}$.

Soluzione 2.6.1.2: $X = \{a^n \mid n \geq 0\}$.

Richiesta 2.6.1.3: $\{ww^R \mid w \in \{a, b\}^*\}$ dove, per ogni stringa w, w^R indica la stringa w scritta al contrario.

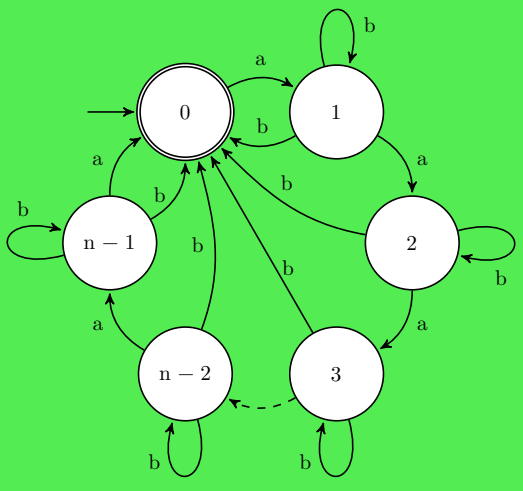
Soluzione 2.6.1.3: $X = \{(ab)^n \mid n \geq 0\}$.

Richiesta 2.6.1.4: Per alcuni di questi linguaggi riuscite ad ottenere insiemi di stringhe distinguibili di cardinalità infinita? Cosa significa ciò?

Soluzione 2.6.1.4: Significa che non sono dei linguaggi di tipo 3.

2.7. Esercizio 07

Esercizio 2.7.1: Considerate l'automa di Meyer e Fischer M_n presentato nella Lezione 5 (caso peggiore della costruzione per sottoinsiemi) e mostrato nella seguente figura:



Richiesta 2.7.1.1: Descrivete a parole la proprietà che deve soddisfare una stringa per essere accettata da M_n. Riuscite a costruire un automa non deterministico, diverso da

M_n, per lo stesso linguaggio, basandovi su tale proprietà? (Potete usare un numero di stati diverso da n, ma non esponenziale, e stati iniziali multipli.)

Soluzione 2.7.1.1: No non ci riesco.

3. Esercizi lezione 06 [14/03]

3.1. Esercizio 01

Esercizio 3.1.1: Considerate il linguaggio

$$\text{DOUBLE}_k = \{ww \mid w \in \{a,b\}^k\},$$

dove $k > 0$ è un intero fissato.

Richiesta 3.1.1.1: Trovare un fooling set di cardinalità 2^k per questo linguaggio. Riuscite a trovare un fooling set o un extended fooling set di cardinalità maggiore?

Soluzione 3.1.1.1: Definiamo il fooling set

$$P = \{(x,x) \mid x \in \{a,b\}^k\}.$$

Esso è un fooling set per DOUBLE_k:

- la stringa $z = xx$ appartiene al linguaggio;
- presi due elementi di lunghezza k da due coppie diverse, essi saranno diversi in almeno una posizione e quindi la stringa risultate non appartiene al linguaggio.

Non so se riusciamo a trovare un fooling set o un extended set di cardinalità maggiore, non ti bastava questo qua?

3.2. Esercizio 02

Esercizio 3.2.1: Considerate il linguaggio

$$\text{PAL}_k = \{w \in \{a,b\}^k \mid w = w^R\},$$

dove k è un intero fissato.

Richiesta 3.2.1.1: Qual è l'extended fooling set per PAL_k di cardinalità maggiore che riuscite a trovare?

Soluzione 3.2.1.1: Definiamo l'insieme

$$P = \left\{(w, w^R) \mid w \in \{a,b\}^{\frac{k}{2}}\right\}.$$

Esso è un fooling set per il linguaggio PAL_k:

- la coppia (w, w^R) appartiene al linguaggio;
- ogni altra stringa formata da elementi di due coppie diverse non appartiene al linguaggio perché esiste almeno una coppia di caratteri con la stessa distanza dagli estremi che sono diversi.

P ha cardinalità $2^{\frac{k}{2}}$, quindi ogni NFA per PAL_k ha almeno $2^{\frac{k}{2}}$ stati.

3.3. Esercizio 03

Esercizio 3.3.1: Considerate il linguaggio

$$K_k = \{w \mid w = x_1 x_2 \cdots x_m x \mid m > 0, x_1, \ldots x_m \in \{a, b\}^k, \exists i \in \{1, \ldots, m\} \mid x_i = x\},$$

dove k è un intero fissato. Si può osservare che ogni stringa w di questo linguaggio è la concatenazione di blocchi di lunghezza k, in cui l'ultimo blocco coincide con uno dei blocchi precedenti.

Richiesta 3.3.1.1: Riuscite a costruire un (extended) fooling set di cardinalità 2^k o maggiore per il linguaggio K_k?

Suggerimento. Ispiratevi all'esercizio 1.

Soluzione 3.3.1.1: Definiamo il fooling set

$$P = \{(x, x) \mid x \in \{a, b\}^k\}.$$

Esso è un fooling set per K_k:

- la stringa $z = xx$ appartiene al linguaggio;
- presi due elementi di lunghezza k da due coppie diverse, essi rappresentano due blocchi diversi perché diversi in almeno una posizione, quindi la stringa risultate non appartiene al linguaggio.

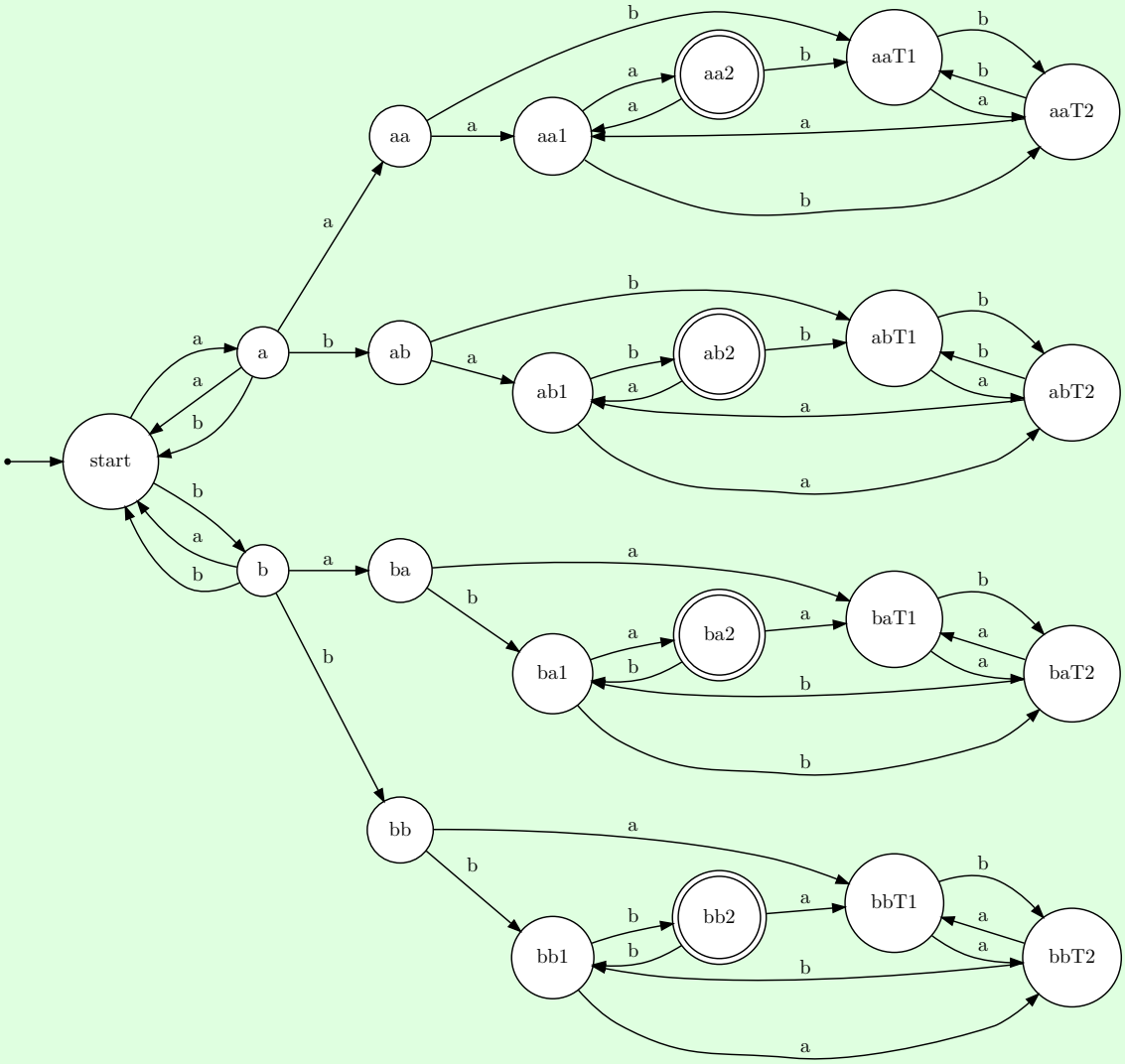
La cardinalità di P è 2^k, quindi ogni NFA per K_k ha almeno 2^k stati.

Richiesta 3.3.1.2: Quale è l'informazione principale che un automa non deterministico può scegliere di ricordare nel proprio controllo a stati finiti durante la lettura di una stringa per riuscire a riconoscere K_k?

Suggerimento. Che «scommessa» può fare l'automa mentre legge la stringa in ingresso e come può verificare tale scommessa leggendo l'ultimo blocco?

Soluzione 3.3.1.2: Un automa non deterministico dovrebbe scommettere che ha appena finito di leggere il blocco che poi sarà ripetuto alla fine. Il controllo che fa l'automa alla fine è quello di uguaglianza con il blocco scommesso, che viene fatta in maniera deterministica ma richiede un grande numero di stati. Va creato quindi un albero che in maniera non deterministica mi manda indietro allo start prima dell'ultimo nodo.

Ad esempio, se $k = 2$ l'automa non deterministico ha questa forma.



Richiesta 3.3.1.3: Supponete di costruire un automa deterministico per riconoscere K_k. Cosa ha necessità di ricordare l'automa nel proprio controllo a stati finiti mentre legge la stringa in input?

Soluzione 3.3.1.3: Un automa deterministico si deve ricordare i blocchi di lunghezza k che ha incontrato fino a quel momento. Questo però fa esplodere il numero di stati, perché dobbiamo calcolare praticamente ogni combinazione possibile.

Richiesta 3.3.1.4: Utilizzando il concetto di distinguibilità, dimostrate che ogni automa deterministico che riconosce K_k deve avere almeno 2^{2^k} stati.

Soluzione 3.3.1.4: Sia $T = \{a, b\}^k$ insieme di tutte le stringhe di lunghezza k costruite sull'alfabeto $\{a, b\}$. Definiamo l'insieme

$$X = \mathcal{P}(T)$$

insieme di tutti i sottoinsiemi di T, ovvero l'insieme delle parti di T. Supponiamo che ogni sottoinsieme venga rappresentato come una stringa ottenuta dalla concatenazione dei suoi elementi.

Questo insieme è formato da stringhe distinguibili tra loro: infatti, prese due stringhe di X, esse hanno almeno un blocco lungo k che appartiene a una delle due ma non all'altra. Ma allora usando quell'elemento come stringa z che distingue noi otteniamo l'appartenenza per la stringa che contiene quell'elemento e la non appartenenza per l'altra.

La cardinalità di X è $2^{|T|}$, ovvero 2^{2^k}, quindi ogni DFA per K_k ha almeno 2^{2^k} stati.

3.4. Esercizio 04

Esercizio 3.4.1: Considerate il linguaggio

$$J_k = \{w \mid w = xx_1 \cdots x_m \mid m > 0, x_1, ...x_m, x \in \{a, b\}^k, \exists i \in \{1, ..., m\} \mid x_i = x\},$$

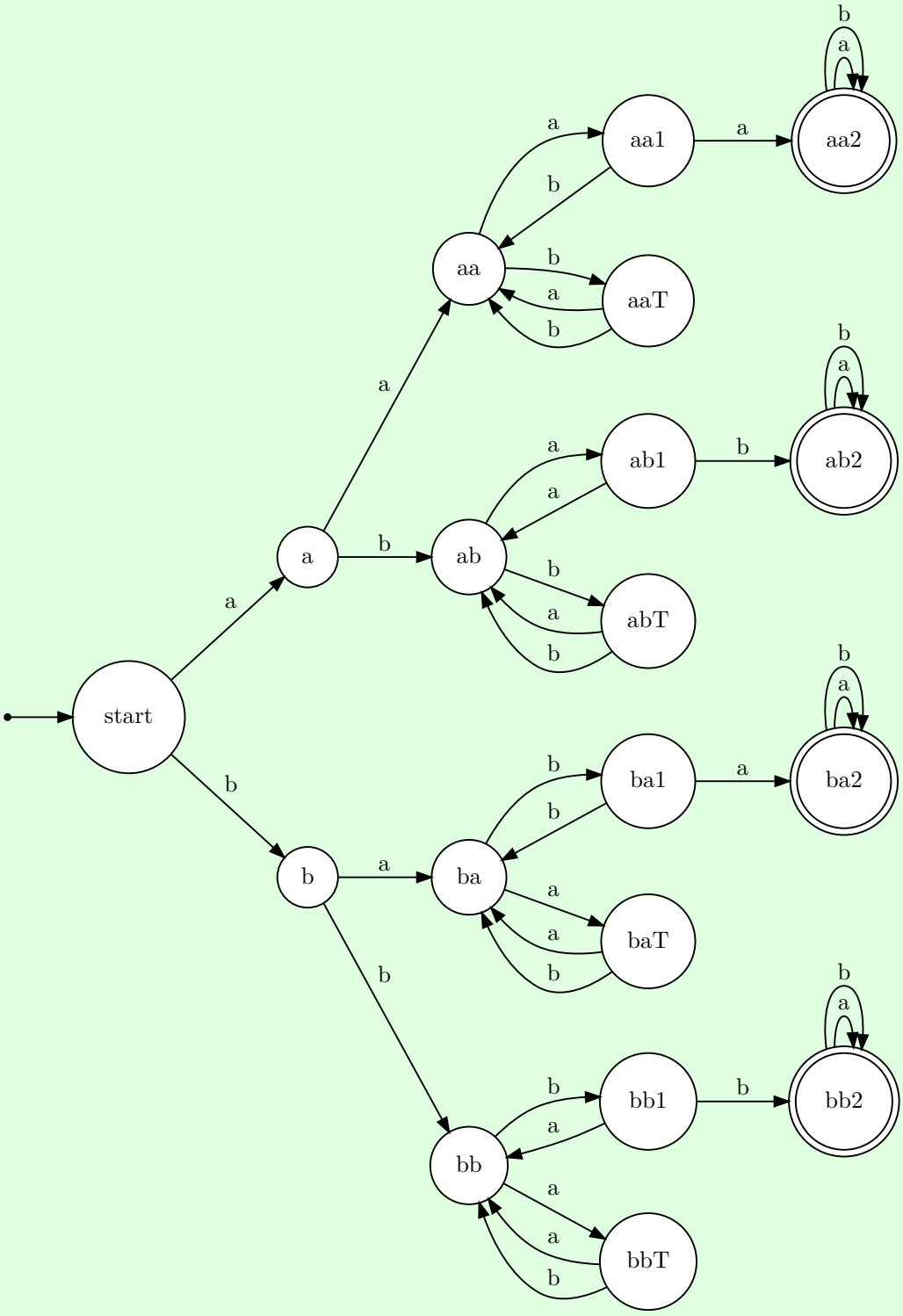
dove k è un intero fissato. Si può osservare che ogni stringa w di questo linguaggio è la concatenazione di blocchi di lunghezza k, in cui il primo blocco coincide con uno dei blocchi successivi; ogni stringa di J_k si ottiene «rovesciando» una stringa del linguaggio K_k dell'esercizio 3.

Richiesta 3.4.1.1: Supponete di costruire automi a stati finiti per J_k. Valgono ancora gli stessi limiti inferiori ottenuti per K_k o si riescono a costruire automi più piccoli? Rispondete sia nel caso di automi deterministici sia in quello di automi non deterministici.

Soluzione 3.4.1.1: Un DFA per J_k usa molti meno stati di un DFA per K_k: infatti, un DFA per J_k deve fare un albero per vedere quale stringa lunga k viene letta all'inizio e poi deve eseguire un controllo con altri $2k-1$ stati per ogni stato foglia. Il numero di stati è quindi

$$2^{k+1} - 1 + 2^k(2k-1) = 2^{k+1} - 1 + k2^{k+1} - 2^k = O\big(k2^k\big).$$

Vediamo un esempio con $k = 2$ per semplicità.



Per il caso non deterministico, ogni NFA deve comunque generare l'albero iniziale per vedere le possibili combinazioni. Nei nodi delle combinazioni potremmo inserire la scommessa, ma questo ci farebbe accettare più stringhe di quelle che vorremmo accettare.

Quindi non lo so RIP, ci penserò più avanti, magari c'è un fooling set da trovare.

3.5. Esercizio 05

Esercizio 3.5.1:

Richiesta 3.5.1.1: Ispirandovi all'esercizio 3, fornite limiti inferiori per il numero di stati degli automi che riconoscono il seguente linguaggio:

$$E_k = \{w \mid w = x_1 \cdots x_m \mid m > 0, x_1, ..., x_m \in \{a,b\}^k, \exists i < j \in \{1, ..., m\} \mid x_i = x_j\},$$

dove k è un intero fissato. Considerate sia il caso deterministico che quello non deterministico.

Soluzione 3.5.1.1: Definiamo l'insieme

$$P = \{(x, x) \mid x \in \{a,b\}^k\}.$$

Questo è un fooling set per il linguaggio E_k:

- la stringa $z = xx$ appartiene a al linguaggio;
- la stringa $z = xy$ non appartiene al linguaggio.

Allora ogni NFA per E_k ha almeno $|P| = 2^k$ stati.

Per i DFA, come per K_k, essi si devono ricordare i blocchi lunghi k che hanno incontrato fino a quel momento, e questo fa esplodere il numero di stati. Infatti, definiamo l'insieme

$$X = \mathcal{P}(\{a,b\}^k).$$

Esso è un insieme di stringhe distinguibili per E_k: presi due sottoinsiemi A e B, allora prendiamo un elemento $x \in A/B$ e usiamolo per distinguere le due stringhe (generate dalla concatenazione di tutte le stringhe contenute in un sottoinsieme).

Ma allora ogni DFA per E_k ha almeno $|X| = 2^{2^k}$ stati.

4. Esercizi lezione 07 [19/03]

4.1. Esercizio 01

Esercizio 4.1.1: Per tutti i linguaggi di questo esercizio l'alfabeto è $\{a, b\}$.

Richiesta 4.1.1.1: Considerate l'insieme L formato dalle stringhe in cui sia il numero di a che il numero di b sono pari.

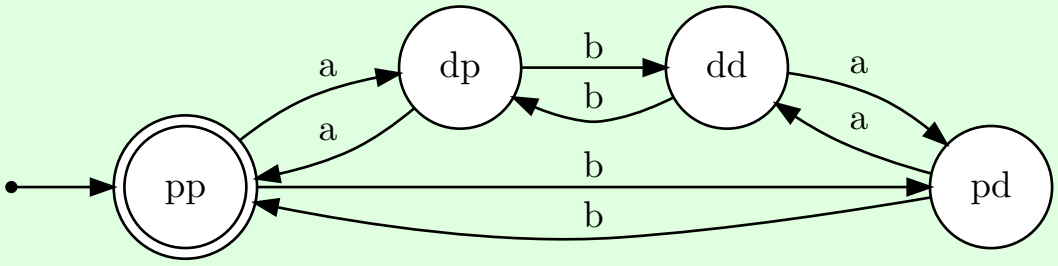
Richieste:

- Determinate le classi di equivalenza della relazione R di Myhill-Nerode associata a L.
- Costruite l'automa deterministico minimo corrispondente.
- Determinate un insieme X di cardinalità massima che contenga stringhe tutte distinguibili tra loro.
- La relazione cambia nel caso si chieda che le stringhe del linguaggio abbiano un numero di a pari, un numero di b pari ed entrambi siano positivi? E l'automa?

Soluzione 4.1.1.1: Le classi di equivalenza di R sono 4:

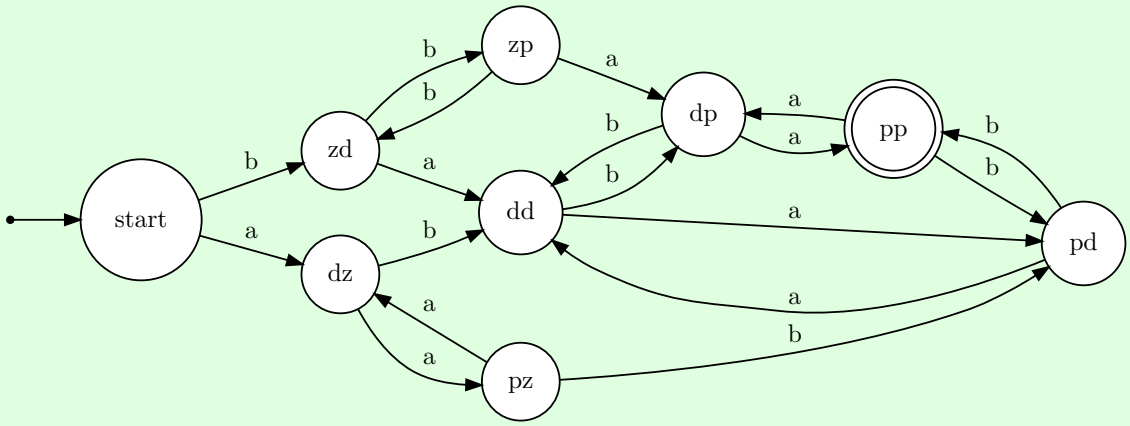
- $[pp]_R$ per a e b pari;
- $[dd]_R$ per a e b dispari;
- $[pd]_R$ per a pari e b dispari;
- $[dp]_R$ per a dispari e b pari.

L'automa minimo per L è il seguente.



L'insieme $X = \{\varepsilon, a, b, ab\}$ è un insieme di stringhe distinguibili, non voglio dimostrarlo.

Richiedendo a e b pari in numero positivo la relazione sale a 9 classi di equivalenza, e quindi anche l'automa passa ad avere 9 stati.



Richiesta 4.1.1.2: Considerate ora il linguaggio L' formato dalle stringhe in cui il numero di a sia pari e il numero di b sia dispari.

Richieste:

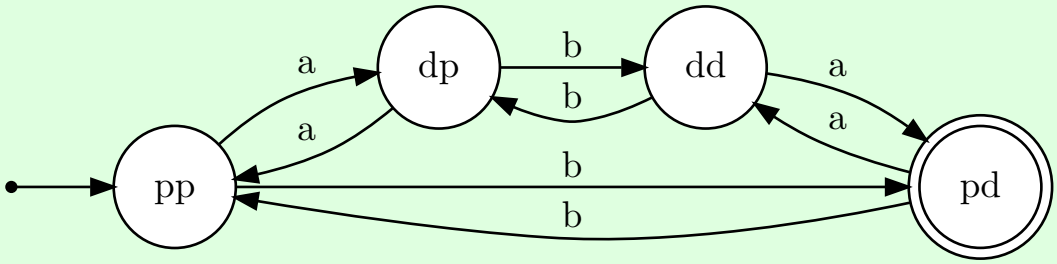
- Verificate che la relazione R' di Myhill-Nerode associata a L' coincide con la relazione R associata a L.
- Cosa hanno in comune e cosa hanno di diverso i rispettivi automi deterministici minimi.

Soluzione 4.1.1.2: Le classi di equivalenza di R' sono 4:

- $[pp]_R$ per a e b pari;
- $[dd]_R$ per a e b dispari;
- $[pd]_R$ per a pari e b dispari;
- $[dp]_R$ per a dispari e b pari.

Sono le stesse classi di equivalenza della relazione R della richiesta precedente.

L'automa minimo per L' è lo stesso identico a quello per L (la prima versione) tranne lo stato finale, che nei due automi è differente.

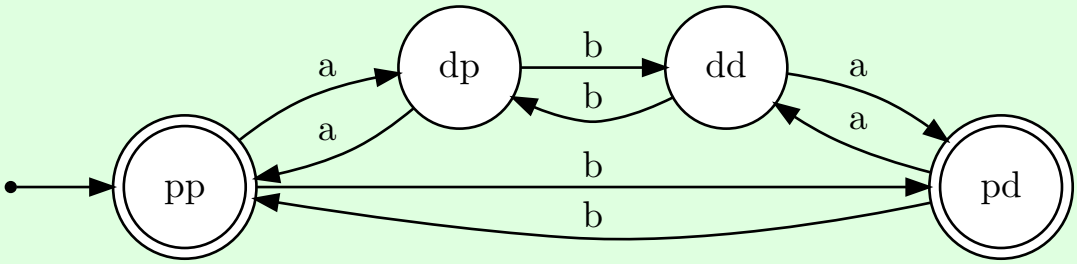


Richiesta 4.1.1.3: Considerate ora il linguaggio L'' formato dalle stringhe in cui il numero di a sia pari e il numero di b sia qualunque.

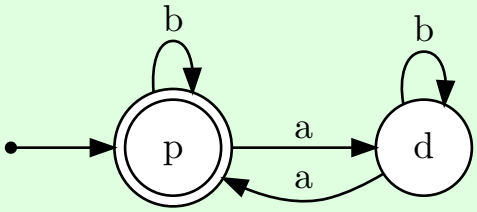
Richieste:

- Verificate che L'' è l'unione di alcune classi di equivalenza della relazione R precedente.
- Costruite un automa che accetti L'' i cui stati corrispondano alle classi di R.
- Determinate le classi di equivalenza della relazione R'' di Myhill-Nerode associata a L''.
- Costruite l'automa deterministico minimo corrispondente.
- Che legame c'è tra R e R'' e tra i due automi ottenuti?

Soluzione 4.1.1.3: Possiamo definire L'' come l'unione delle classi di equivalenza $[pp]_R$ e $[pd]_R$, che vanno contenere tutte le stringhe che hanno un numero di a pari.



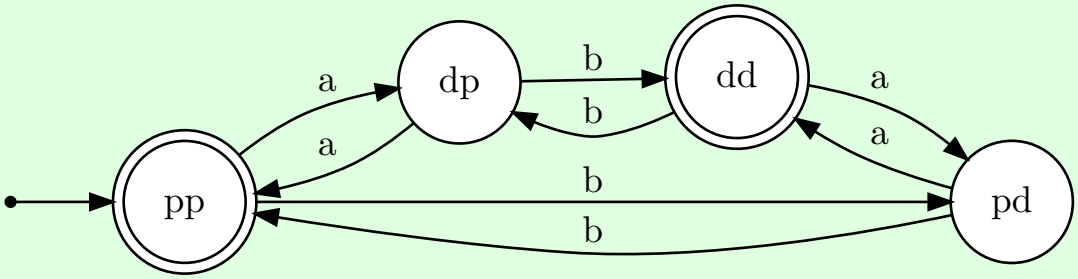
Le classi di equivalenza della relazione R'' sono $[p]_{R''}$ e $[d]_{R''}$, che contengono rispettivamente le stringhe con un numero di a pari e le stringhe con un numero di a dispari.



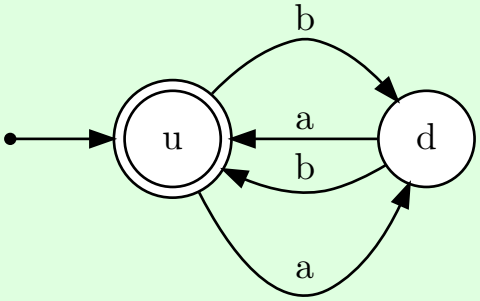
Possiamo dire che R è un raffinamento di R'', e quindi che l'automa di R'' è più compatto e conciso di quello di R, che infatti ha il doppio degli stati.

Richiesta 4.1.1.4: Considerate ora il linguaggio L''' formato dalle stringhe in cui il numero di a e il numero di b siano entrambi pari o entrambi dispari. Rispondete alle domande del punto c precedente, considerando L''' e la rispettiva relazione R''' di Myhill-Nerode al posto di L'' e R''.

Soluzione 4.1.1.4: Possiamo definire L''' come l'unione delle classi di equivalenza $[pp]_R$ e $[dd]_R$, che vanno contenere tutte le stringhe che hanno un numero di a e b entrambi pari o entrambi dispari.



Le classi di equivalenza della relazione R''' sono $[u]_{R''}$ e $[d]_{R''}$, che contengono rispettivamente le stringhe con un numero di a e b entrambi pari o entrambi dispari e le stringhe con un numero di a dispari/pari e di b pari/dispari.



Possiamo dire che R è un raffinamento di R''', e quindi che l'automa di R''' è più compatto e conciso di quello di R, che infatti ha il doppio degli stati.

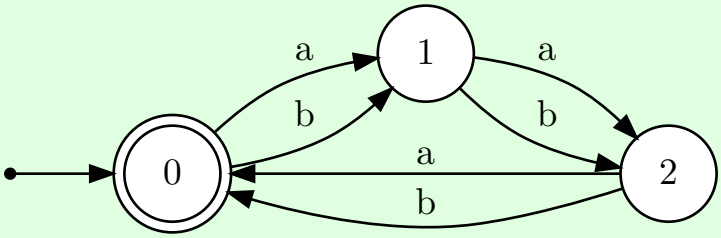
4.2. Esercizio 02

Esercizio 4.2.1: Calcolate le classi d'equivalenza della relazione di Myhill-Nerode associata a ciascuno dei seguenti linguaggi e, nel caso la relazione sia di indice finito, costruite l'automa deterministico minimo corrispondente.

Richiesta 4.2.1.1: $L = \{w \in \{a,b\}^* \mid$ la somma del numero di a e del numero di b è multiplo di $3\}$.

Soluzione 4.2.1.1: Le classi di equivalenza della relazione R_L sono 3:

- $[0]_{R_L}$ per $a + b$ multiplo di 3;
- $[1]_{R_L}$ per $a + b$ multiplo di $3 + 1$;
- $[2]_{R_L}$ per $a + b$ multiplo di $3 + 2$.



Richiesta 4.2.1.2: $J = \{a^n b^n \mid n \geq 0\}$.

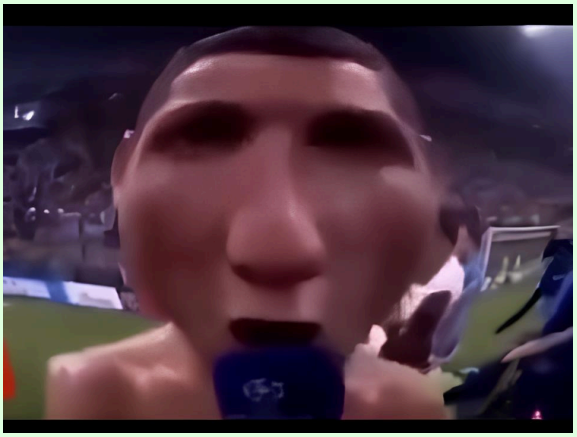
Soluzione 4.2.1.2: Il linguaggio J non è regolare, quindi non siamo sicuri che l'indice della relazione R_J sia finito. E infatti non lo è: le classi di equivalenza sono nella forma

$$[a^n]_{R_J} \mid n \geq 0$$

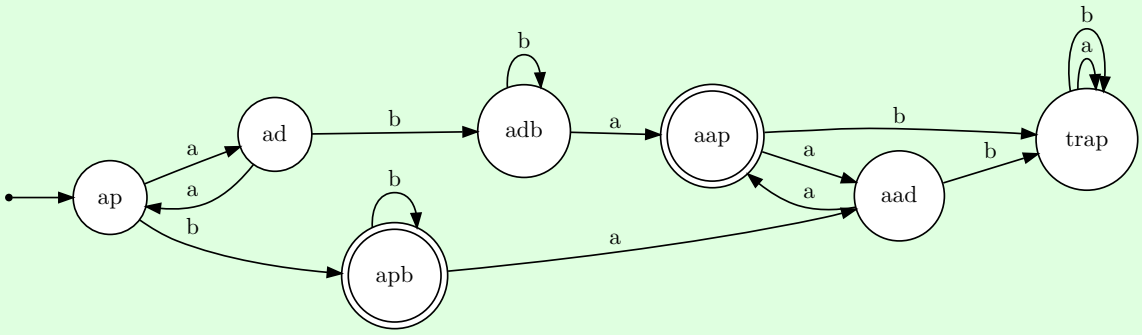
ma il numero di queste classi è infinito, quindi l'indice di R_J è infinito.

Richiesta 4.2.1.3: $K = \{a^i b^n a^j \mid n > 0 \wedge i + j \text{ è pari}\}$.

Soluzione 4.2.1.3: Visto che sono furbo, prima ho costruito l'automa minimo e ora scrivo quali sono le classi di equivalenza, urlo del sium.



Le classi di equivalenza della relazione R_L quindi sono 7, che riprendono i nomi degli stati dell'automa successivo, molto facili.



Sono sicuro che questo sia l'automa minimo, ho controllato con l'algoritmo di Hopcroft.

5. Esercizi lezioni 08, 09 e 10 [28/03]

5.1. Esercizio 01

Esercizio 5.1.1:

Richiesta 5.1.1.1: Scrivete un'espressione regolare per il linguaggio formato da tutte le stringhe sull'alfabeto $\{0, 1\}$ che, interpretate come numeri in notazione binaria, rappresentano potenze di 2.

Soluzione 5.1.1.1:

$$L = 10^*.$$

5.2. Esercizio 02

Esercizio 5.2.1:

Richiesta 5.2.1.1: Scrivete un'espressione regolare per il linguaggio formato da tutte le stringhe sull'alfabeto $\{0, 1\}$ che, interpretate come numeri in notazione binaria, non rappresentano potenze di 2.

Soluzione 5.2.1.1:

$$L = 0 + 1(0 + 1)^*1(0 + 1)^*.$$

5.3. Esercizio 03

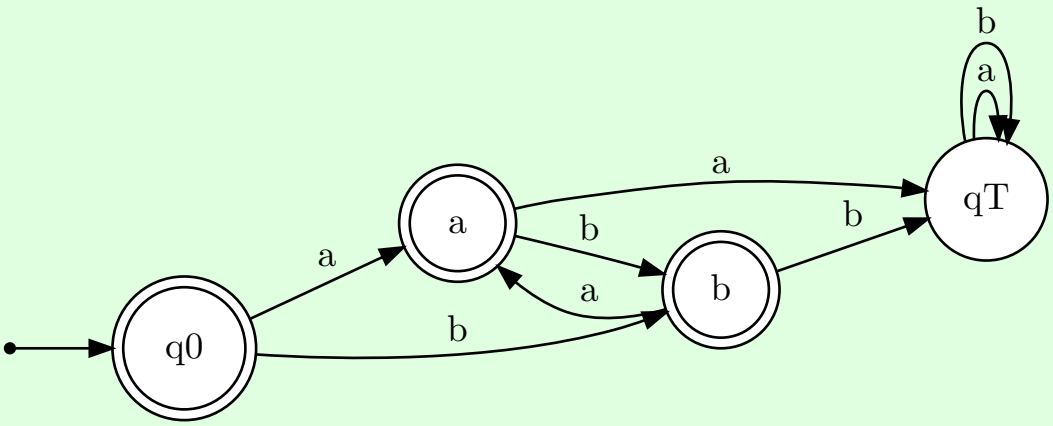
Esercizio 5.3.1:

Richiesta 5.3.1.1: Scrivete un'espressione regolare per il linguaggio formato da tutte le stringhe sull'alfabeto $\{a, b\}$ in cui le a e le b si alternano (come $abab$, bab, b, ecc.). Disegnate poi un automa per lo stesso linguaggio.

Soluzione 5.3.1.1:

$$L = \varepsilon + (ab)^+ + (ba)^+ + a(ba)^* + b(ab)^*.$$

Diamo un DFA per questo linguaggio.



5.4. Esercizio 04

Esercizio 5.4.1:

Richiesta 5.4.1.1: Scrivere un'espressione regolare per il linguaggio formato da tutte le stringhe sull'alfabeto $\{a, b\}$ nelle quali ogni a è seguita immediatamente da una b.

Soluzione 5.4.1.1:

$$L = b^*(ab^+)^*.$$

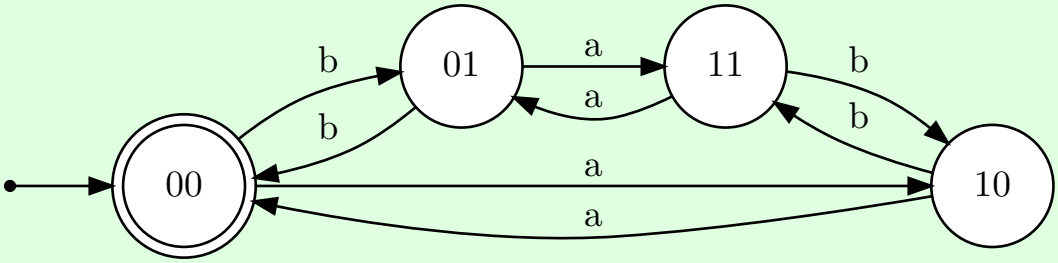
5.5. Esercizio 05

Esercizio 5.5.1:

Richiesta 5.5.1.1: Scrivete un'espressione regolare per il linguaggio formato da tutte le stringhe sull'alfabeto $\{a, b\}$ che contengono un numero di a pari e un numero di b pari.

Suggerimento. Se avete difficoltà nell'ottenere direttamente l'espressione, potete costruire prima un automa a stati finiti e, successivamente, ricavare l'espressione regolare servendovi di una delle trasformazioni da automi a espressioni presentate a lezione.

Soluzione 5.5.1.1: Visto che non mi viene subito l'espressione regolare, usiamo un DFA.



Scriviamo il sistema di equazioni associato e risolviamo. Buona fortuna.

$$\begin{cases} X_0 = aX_2 + bX_1 + \varepsilon \\ X_1 = aX_3 + bX_0 \\ X_2 = aX_0 + bX_3 \\ X_3 = aX_1 + bX_2 \end{cases}$$

$$\begin{cases} X_0 = aX_2 + bX_1 + \varepsilon \\ X_1 = a(aX_1 + bX_2) + bX_0 \\ X_2 = aX_0 + b(aX_1 + bX_2) \end{cases}$$

$$\begin{cases} X_0 = aX_2 + bX_1 + \varepsilon \\ X_1 = aaX_1 + abX_2 + bX_0 \\ X_2 = bbX_2 + aX_0 + baX_1 \end{cases}$$

$$\begin{cases} X_0 = aX_2 + bX_1 + \varepsilon \\ X_1 = aaX_1 + abX_2 + bX_0 \\ X_2 = (bb)^*(aX_0 + baX_1) \end{cases}$$

$$\begin{cases} X_0 = a(bb)^*(aX_0 + baX_1) + bX_1 + \varepsilon \\ X_1 = aaX_1 + ab(bb)^*(aX_0 + baX_1) + bX_0 \end{cases}$$

$$\begin{cases} X_0 = a(bb)^*aX_0 + a(bb)^*baX_1 + bX_1 + \varepsilon \\ X_1 = aaX_1 + ab(bb)^*aX_0 + ab(bb)^*baX_1 + bX_0 \end{cases}$$

$$\begin{cases} X_0 = a(bb)^*aX_0 + (a(bb)^*ba + b)X_1 + \varepsilon \\ X_1 = (aa + ab(bb)^*ba)X_1 + (ab(bb)^*a + b)X_0 \end{cases}$$

$$\begin{cases} X_0 = a(bb)^*aX_0 + (a(bb)^*ba + b)X_1 + \varepsilon \\ X_1 = (aa + ab(bb)^*ba)^*(ab(bb)^*a + b)X_0 \end{cases}$$

$$X_0 = a(bb)^*aX_0 + (a(bb)^*ba + b)(aa + ab(bb)^*ba)^*(ab(bb)^*a + b)X_0 + \varepsilon.$$

Possiamo ora sistemare quest'ultima espressione e trovare il risultato finale

$$X_0 = (a(bb)^*a + (a(bb)^*ba + b)(aa + ab(bb)^*ba)^*(ab(bb)^*a + b))X_0 + \varepsilon$$
$$L = (a(bb)^*a + (a(bb)^*ba + b)(aa + ab(bb)^*ba)^*(ab(bb)^*a + b))^*.$$

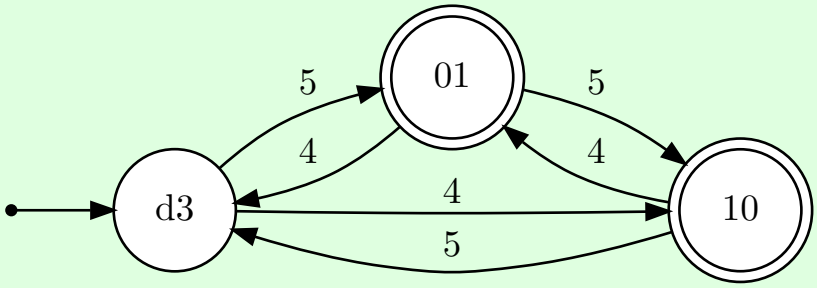
5.6. Esercizio 06

Esercizio 5.6.1:

Richiesta 5.6.1.1: Scrivete un'espressione regolare per il linguaggio formato da tutte le stringhe sull'alfabeto $\{4, 5\}$ che, interpretate come numeri in base 10, rappresentano interi che non sono divisibili per 3.

Suggerimento. Riguardate gli esercizi delle lezioni 4-5.

Soluzione 5.6.1.1: Riprendiamo l'automa per questo linguaggio.



Scriviamo il sistema di equazioni associato e risolviamolo.

$$\begin{cases} X_0 = 4X_2 + 5X_1 \\ X_1 = 4X_0 + 5X_2 + \varepsilon \\ X_2 = 4X_1 + 5X_0 + \varepsilon \end{cases}$$

$$\begin{cases} X_0 = 4(4X_1 + 5X_0 + \varepsilon) + 5X_1 \\ X_1 = 4X_0 + 5(4X_1 + 5X_0 + \varepsilon) + \varepsilon \end{cases}$$

$$\begin{cases} X_0 = 44X_1 + 45X_0 + 4 + 5X_1 \\ X_1 = 54X_1 + 4X_0 + 55X_0 + 5 + \varepsilon \end{cases}$$

$$\begin{cases} X_0 = 45X_0 + (44 + 5)X_1 + 4 \\ X_1 = (54)^*((4 + 55)X_0 + 5 + \varepsilon) \end{cases}$$

$$X_0 = 45X_0 + (44 + 5)(54)^*((4 + 55)X_0 + 5 + \varepsilon) + 4.$$

Come prima, risolviamo quest'ultima espressione per ottenere il risultato.

$$X_0 = 45X_0 + (44+5)(54)^*(4+55)X_0 + (44+5)(54)^*5 + (44+5)(54)^* + 4$$
$$X_0 = (45 + (44+5)(54)^*(4+55))X_0 + (44+5)(54)^*5 + (44+5)(54)^* + 4$$
$$L = (45 + (44+5)(54)^*(4+55))^*((44+5)(54)^*5 + (44+5)(54)^* + 4).$$

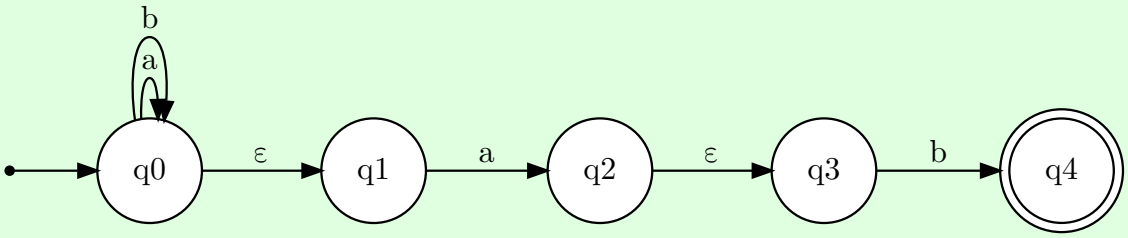
5.7. Esercizio 07

Esercizio 5.7.1: Utilizzando le costruzioni presentate a lezione, disegnare un automa a stati finiti con ε-transizioni equivalente a ciascuna delle seguenti espressioni regolari. Ricavate poi degli automi a stati finiti deterministici equivalenti.

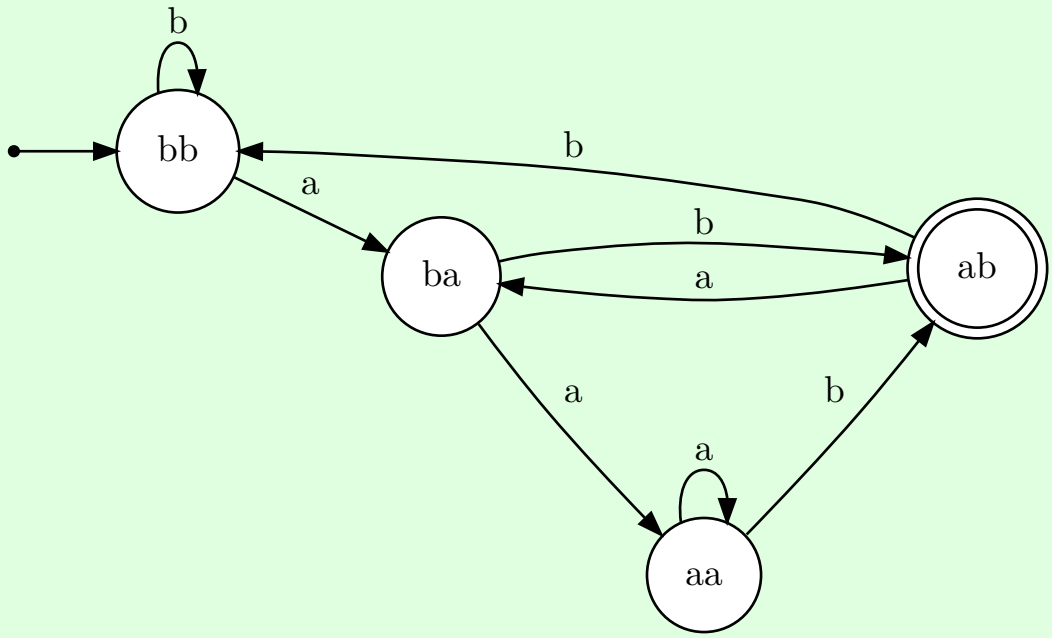
Disclaimer: non so se le costruzioni che ho usato sono corrette, anche se mi sembra di sì.

Richiesta 5.7.1.1: $(a+b)^*ab$

Soluzione 5.7.1.1: Automa ottenuto con la costruzione:

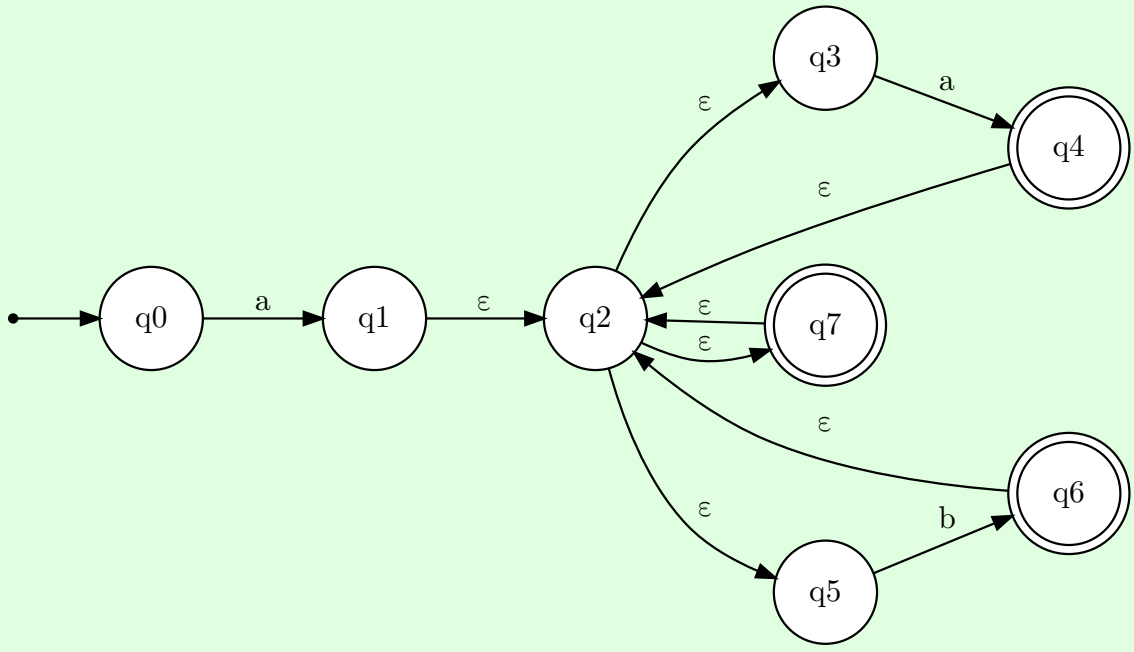


Automa deterministico equivalente:

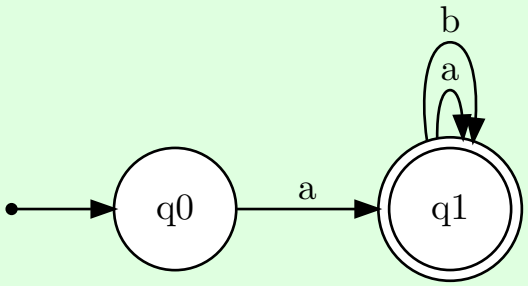


Richiesta 5.7.1.2: $a(a+b+\varepsilon)^*$

Soluzione 5.7.1.2: Automa ottenuto con la costruzione:

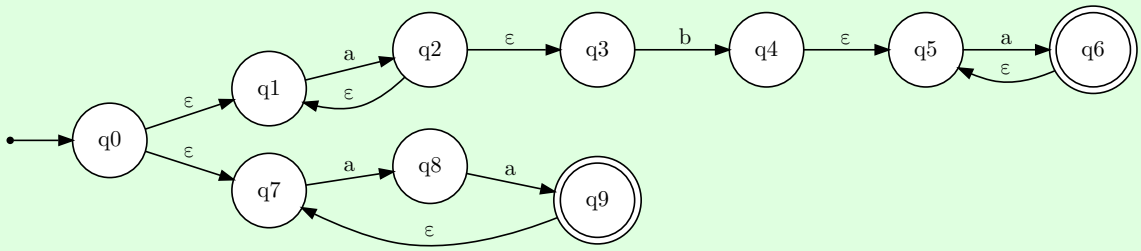


Automa deterministico equivalente:

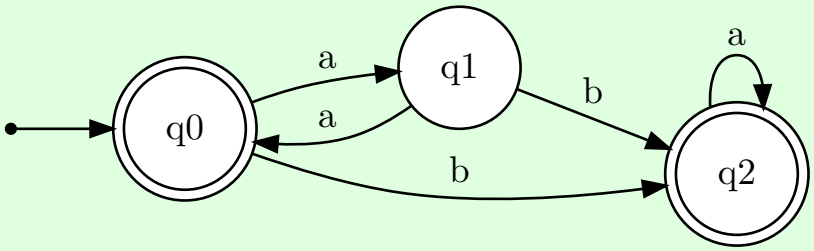


Richiesta 5.7.1.3: $a^*ba^* + (aa)^*$

Soluzione 5.7.1.3: Automa ottenuto con la costruzione:



Automa deterministico equivalente:



5.8. Esercizio 08

Esercizio 5.8.1:

Richiesta 5.8.1.1: Dimostrate che per ogni $n', n'' > 0$ esistono due linguaggi L' e L'' con $\mathrm{sc}(L') = n'$ e $\mathrm{sc}(L'') = n''$ tali che $\mathrm{sc}(L' \cup L'') = \mathrm{sc}(L' \cap L'') = n'n''$.

Suggerimento. Per $n' = n'' = 2$ potete considerare $L' = \{w \in \{a,b\}^* \mid \text{il numero di } a \text{ in } w \text{ è pari}\}$ e $L'' = \{w \in \{a,b\}^* \mid \text{il numero di } b \text{ in } w \text{ è pari}\}$, due automi deterministici A' e A'' con due stati che li riconoscono e dimostrare che ogni automa deterministico per l'unione o l'intersezione deve avere almeno $n'n'' = 4$ stati (conviene costruire l'automa prodotto di A' e A'' e da esso ricavare 4 stringhe tutte distinguibili tra loro). Potete poi generalizzare ad altri valori di n' e n''.

Soluzione 5.8.1.1: Consideriamo i due linguaggi

$$L_{n'} = \{w \in \{a,b\}^* \mid \#_a(w) \bmod n' = 0\}$$
$$L_{n''} = \{w \in \{a,b\}^* \mid \#_b(w) \bmod n'' = 0\}$$

riconosciuti da due DFA A' e A'' rispettivamente di n' e n'' stati. L'automa prodotto per $L' \cup L''$ o per $L' \cap L''$ ha almeno (si dimostra poi essere esattamente) $n'n''$ stati.

L'automa prodotto per $L' \cup L''$ accetta se una stringa sta in almeno uno dei due linguaggi, mentre la versione per $L' \cap L''$ accetta se una stringa sta in entrambi i linguaggi.

Definiamo l'insieme

$$X = \{a^p b^q \mid p \in \{0, ..., n' - 1\} \wedge q \in \{0, ..., n'' - 1\}\}.$$

Questo insieme è formato da stringhe distinguibili tra loro per $L' \cup L''$:

- date due stringhe w' e w'' esse possono avere:
  - $p' \neq p'' \wedge q' \neq q''$:
    - se nessuna stringa è accettata, aggiungiamo un numero di a tale da rendere la prima stringa in L';
    - se una sola delle due stringhe è accettata grazie ad uno dei linguaggi, scegliamo $z = \varepsilon$;
    - se entrambe le stringhe sono accettate, la prima per L' e la seconda per L'', aggiungiamo un numero di b tale da rendere la seconda fuori da L'';
  - $p' = p'' \vee q' = q''$:
    - se il valore nel quale sono uguali le due stringhe le fa accettare allora aggiungiamo:
      - una copia della lettera riferita a quel valore;
      - un numero k dell'altra lettera tale che k fa accettare una delle due stringhe;
    - se il valore nel quale sono uguali le due stringhe non le fa accettare allora aggiungiamo un numero di lettere riferite all'altro valore tale da rendere una delle due stringhe accettata.

X è un insieme di stringhe distinguibili e ogni automa per $L' \cup L''$ ha almeno $n'n''$ stati.

Possiamo fare la stessa cosa per $L' \cap L''$:

- date due stringhe w' e w'' esse possono avere:
  - $p' \neq p'' \wedge q' \neq q''$:

- se nessuna stringa è accettata, aggiungiamo un numero di a e di b tale da rendere la prima stringa in L';
- se una sola delle due stringhe è accettata, scegliamo $z = \varepsilon$;

▸ $p' = p'' \vee q' = q''$: aggiungiamo un numero di lettere riferite all'elemento uguale per arrivare ad accettare il linguaggio di quella lettera e un numero di altre lettere per far accettare solo una delle due stringhe.

X è un insieme di stringhe distinguibili e ogni automa per $L' \cap L''$ ha almeno $n'n''$ stati

5.9. Esercizio 09

Esercizio 5.9.1: Richieste:

- Ispirandovi alle costruzioni presentate a lezione, fornite delle costruzioni che, dati due automi deterministici A' e A'' che accettano i linguaggi L' e L'', producano automi deterministici per i seguenti linguaggi.
- Se gi automi A' e A'' hanno rispettivamente n' e n'' stati, quanti sono gli stati degli automi risultanti? Ritenete che si possa fare di meglio (considerate l'esercizio 8)?
- Le costruzioni che avete fornito sono corrette anche nel caso uno o entrambi gli automi dati siano non deterministici? Se la risposta è negativa fornite delle costruzioni alternative e discutete cosa si ottiene rispetto al numero di stati.

Richiesta 5.9.1.1: $L = L'/L''$.

Soluzione 5.9.1.1: Possiamo costruire l'automa prodotto che abbiamo visto a lezione dove modifichiamo l'insieme degli stati finali: per avere la differenza, la stringa in input deve stare in L' e non in L'', quindi

$$F = \{(q, p) \mid q \in F' \wedge p \in Q''/F''\}.$$

Abbiamo detto che non si può fare meglio dell'automa prodotto in questi casi, quindi il numero di stati per questo automa è $n'n''$.

Nel caso in cui uno dei due automi sia un NFA, possiamo eseguire una costruzione simile a quella della concatenazione: la prima componente dell'automa prodotto manda avanti il DFA, la seconda componente invece simula la costruzione per sottoinsiemi. Il numero di stati di questo automa è $\leq n'2^{n''}$.

Nel caso in cui entrambi gli automi sono NFA, dobbiamo eseguire una doppia costruzione per sottoinsiemi e costruire l'automa prodotto, ottenendo un numero di stati $\leq 2^{n'}2^{n''}$.

Richiesta 5.9.1.2: $L = L'\Delta L''$ (differenza simmetrica di L' e L'').

Soluzione 5.9.1.2: Possiamo costruire l'automa prodotto che abbiamo visto a lezione dove modifichiamo l'insieme degli stati finali: per avere la differenza simmetrica, la stringa in input deve stare in L' e non in L'' o viceversa, quindi

$$F = \{(q,p) \mid q \in F' \wedge p \in Q''/F''\} \cup \{(q,p) \mid q \in Q'/F' \wedge p \in F''\}.$$

Abbiamo detto che non si può fare meglio dell'automa prodotto in questi casi, quindi il numero di stati per questo automa è $n'n''$.

Nel caso in cui uno dei due automi sia un NFA, possiamo eseguire una costruzione simile a quella della concatenazione: la prima componente dell'automa prodotto manda avanti il DFA, la seconda componente invece simula la costruzione per sottoinsiemi. Il numero di stati di questo automa è $\leq n'2^{n''}$.

Nel caso in cui entrambi gli automi sono NFA, dobbiamo eseguire una doppia costruzione per sottoinsiemi e costruire l'automa prodotto, ottenendo un numero di stati $\leq 2^{n'}2^{n''}$.

5.10. Esercizio 10

Esercizio 5.10.1:

Richiesta 5.10.1.1: Scrivete un'espressione regolare estesa (in cui cioè avete a disposizione anche operatori per intersezione e complemento) equivalente all'espressione $(aba)^*$, nella quale non si utilizzi l'operatore $*$.

Suggerimento. L'insieme Σ^* di tutte le stringhe su un alfabeto Σ che non contengono un certo fattore α può essere espresso come complemento di $\overline{\Sigma}\alpha\overline{\Sigma}$. L'insieme delle stringhe che iniziano con il prefisso α può essere espresso come $\alpha\overline{\Sigma}$. Quali fattori non possono comparire nelle stringhe del linguaggio che stiamo considerando? Quale prefisso e quale suffisso hanno in comune tutte le stringhe non vuote di questo linguaggio?

Soluzione 5.10.1.1: Non mi viene in mente per ora.

5.11. Esercizio 11

Esercizio 5.11.1: Dato l'alfabeto $\Sigma = \{a_0, a_1, a_2, ..., a_n\}$, considerate la stringa

$$w_n = \left(... \left(\left((a_0^2 a_1)^2 a_2 \right)^2 a_3 \right)^2 ... a_n \right)^2,$$

dove, per una stringa x, x^2 è un'abbreviazione di xx.

Richiesta 5.11.1.1: Provate a scrivere alcune stringhe come w_1, w_2, w_3, ... e a esprimere w_i in funzione di w_{i-1}, per $i > 0$, scegliendo opportunamente w_0.

Soluzione 5.11.1.1: Calcoliamo qualche stringa w_i:

$$
\begin{aligned}
w_0 &= a_0^2 \\
w_1 &= \left(a_0^2 a_1\right)^2 = \left(w_0 a_1\right)^2 \\
w_2 &= \left(\left(a_0^2 a_1\right)^2 a_2\right)^2 = \left(w_1 a_2\right)^2 \\
w_3 &= \left(\left(\left(a_0^2 a_1\right)^2 a_2\right)^2 a_3\right)^2 = \left(w_2 a_3\right)^2 \\
w_4 &= \left(\left(\left(\left(a_0^2 a_1\right)^2 a_2\right)^2 a_3\right)^2 a_4\right)^2 = \left(w_3 a_4\right)^2.
\end{aligned}
$$

Notiamo una struttura ricorsiva: definiamo quindi w_i induttivamente come

$$
w_i = \begin{cases} a_0^2 & \text{se } i = 0 \\ \left(w_{i-1} a_i\right)^2 & \text{se } i > 0 \end{cases}.
$$

Richiesta 5.11.1.2: Quanto è lunga w_n? (Potete scrivere un'equazione di ricorrenza e risolverla.)

Soluzione 5.11.1.2: La lunghezza di w_n è definita dall'equazione di ricorrenza

$$
|w_n| = \begin{cases} 2|a_0| & \text{se } n = 0 \\ 2|w_{n-1}| + 2|a_n| & \text{se } n > 0 \end{cases}.
$$

Risolvendo questa equazione di ricorrenza si ottiene

$$
\sum_{i=0}^{n} 2^{n+1-i} |a_i|.
$$

Richiesta 5.11.1.3: Osservate che ogni espressione regolare per il linguaggio $L_n = \{w_n\}$ (costituito da w_n come unica stringa) deve essere lunga almeno quanto w_n.

Soluzione 5.11.1.3: Soluzione nel commento, secondo me non è giusta.

Richiesta 5.11.1.4: Supponete di avere a disposizione anche l'operatore di intersezione. Riuscite a trovare un'espressione per L_n di lunghezza $O(n^2)$?

Suggerimento. Si potrebbe «imbottire» un'espressione come $\left(...\left(\left((a_0^* a_1)^* a_2\right)^* a_3\right)^* ... a_n\right)^*$ con intersezioni con linguaggi opportuni, in modo che, per ogni occorrenza dell'operatore $*$, tutto ciò che viene ottenuto con un numero di ripetizioni diverso da 2 venga eliminato.

Soluzione 5.11.1.4: Sincero non so nemmeno da che parte sono girato.

5.12. Esercizio 12

Esercizio 5.12.1: Dimostrate che se L è un linguaggio regolare, allora lo sono anche i seguenti linguaggi. In ognuno dei casi, fornite una costruzione per ottenere un automa che accetta il linguaggio considerato a partire da un automa che accetta L. Discutete se le costruzioni fornite preservino il determinismo o no.

Richiesta 5.12.1.1: $\text{Pref}(L) = \{x \in \Sigma^* \mid \exists y \in \Sigma^* \mid xy \in L\}$.

Soluzione 5.12.1.1: Non lo so, forse.

Richiesta 5.12.1.2: $\text{Suff}(L) = \{x \in \Sigma^* \mid \exists y \in \Sigma^* \mid yx \in L\}$.

Soluzione 5.12.1.2: Non lo so, forse.

Richiesta 5.12.1.3: $\text{Fact}(L) = \{x \in \Sigma^* \mid \exists y, z \in \Sigma^* \mid yxz \in L\}$.

Soluzione 5.12.1.3: Questo veramente non lo so.

Richiesta 5.12.1.4: $\min(L) = \{w \in L \mid \forall x, y \in \Sigma^* \quad xy = w \land y \neq \varepsilon \implies x \notin L\}$.

Soluzione 5.12.1.4: Non so nemmeno questo, forse.

Richiesta 5.12.1.5: $\max(L) = \{w \in L \mid \forall x \in \Sigma^+ \quad wx \notin L\}$.

Soluzione 5.12.1.5: Oggi terribile.

5.13. Esercizio 13

Esercizio 5.13.1: Date due stringhe $x, y \in \Sigma^*$, lo shuffle di x e y, indicato come $\text{shuffle}(x, y)$ è il linguaggio che si ottiene «fondendo» in tutti i modi possibili le due stringhe, i.e.,

$$\text{shuffle}(x, y) = \{x_0 b_1 x_1 b_2 x_2 ... x_k b_k \mid x_0, x_1, ..., x_k \in \Sigma^* \land x_0 x_1 ... x_k = x \land b_1, b_2, ..., b_k \in \Sigma^* \land b_1 b_2 ... b_k = y\}.$$

Lo shuffle di due linguaggi $L', L'' \subseteq \Sigma^*$ è il linguaggio formato da tutte le stringhe ottenibili come shuffle di stringhe di L' con stringhe di L'', cioè:

$$\text{shuffle}(L', L'') = \bigcup_{x \in L' \land y \in L''} \text{shuffle}(x, y).$$

Richiesta 5.13.1.1: Fornite una costruzione che dati due automi deterministici per L' e L'' permetta di ottenere un automa per $\text{shuffle}(L', L'')$ (ispiratevi alla costruzione dell'automa prodotto utilizzata per unione e intersezione).

Soluzione 5.13.1.1: Vedi teoria.

Richiesta 5.13.1.2: L'automa ottenuto è deterministico?

Soluzione 5.13.1.2: No, l'automa ottenuto non è deterministico perché ad ogni iterazione deve scommettere qualche automa mandare avanti.

Richiesta 5.13.1.3: Cosa succede nel caso in cui L' e L'' siano definiti su alfabeti disgiunti?

Soluzione 5.13.1.3: Nel caso di linguaggi definiti su alfabeti disgiunti andiamo a finire in un automa deterministico.

5.14. Esercizio 14

Esercizio 5.14.1: Lo shuffle perfetto di due stringhe x e y della stessa lunghezza è la stringa che si ottiene alternando i simboli di x con quelli di y, i.e., se $x = a_1a_2...a_n$ e $y = b_1b_2...b_n$, con $a_ib_i \in \Sigma$, $i = 1, ..., n$, allora

$$\text{perfectShuffle}(x, y) = a_1b_1a_2b_2...a_nb_n.$$

Dati due linguaggi L' e L'' definiamo:

$$\text{perfectShuffle}(L', L'') = \{\text{perfectShuffle}(x, y) \mid x \in L' \wedge y \in L'' \wedge |x| = |y|\}.$$

Richiesta 5.14.1.1: Svolgete l'esercizio 13 per l'operazione di shuffle perfetto.

Soluzione 5.14.1.1: Non mi viene in mente niente.

6. Esercizi lezione 11 [02/04]

6.1. Esercizio 01

Esercizio 6.1.1: Considerate l'insieme delle stringhe sull'alfabeto $\{a, b\}$ il cui simbolo centrale è una a, cioè il linguaggio

$$L = \left\{ w \in \{a,b\}^* \mid \text{il simbolo in posizione } \left\lceil \frac{|x|}{2} \right\rceil \text{ di } w \text{ è una } a \right\}.$$

Richiesta 6.1.1.1: Utilizzate il pumping lemma per dimostrare che L non è regolare.

Soluzione 6.1.1.1: Per assurdo sia L regolare, e sia N la costante del pumping lemma. Prendiamo la stringa

$$z = b^N a b^N$$

di lunghezza $|z| = 2N + 1 \geq N$ e scomponiamola come $z = uvw$. Sappiamo che $|uv| \leq N$, quindi uv è formata da solo b, ovvero

$$u = b^i \wedge v = b^j \mid i + j \leq N \wedge j \geq 1.$$

Sappiamo anche che possiamo ripetere v un numero arbitrario di volte: se scegliamo $k \geq 2$ otteniamo una stringa z' che viene gonfiata di b all'inizio, spostando il carattere centrale mano a mano verso sinistra. Visto che i caratteri a sinistra del centro sono solo b, il nuovo elemento centrale sarà una b, ma questo è assurdo perché la ripetizione arbitraria dovrebbe mantenerci in L, quindi quest'ultimo non è regolare.

Richiesta 6.1.1.2: Cosa si può dire nel caso l'alfabeto sia formato dal solo simbolo a?

Soluzione 6.1.1.2: Nel caso l'alfabeto sia unario, il linguaggio che stiamo riconoscendo è a^+, quindi lo possiamo fare con un DFA e quindi il linguaggio è regolare.

6.2. Esercizio 02

Esercizio 6.2.1: Utilizzando il pumping lemma, dimostrate che i seguenti linguaggi non sono regolari.

Richiesta 6.2.1.1: $\text{PAL} = \{w \in \{a,b\}^* \mid w = w^R\}$.

Soluzione 6.2.1.1: Per assurdo sia L regolare, e sia N la costante del pumping lemma. Prendiamo la stringa

$$z = b^N a b^N$$

di lunghezza $|z| = 2N + 1 \geq N$ e scomponiamola come $z = uvw$. Sappiamo che $|uv| \leq N$, quindi uv è formata da solo b, ovvero

$$u = b^i \wedge v = b^j \mid i + j \leq N \wedge j \geq 1.$$

Sappiamo anche che possiamo ripetere v un numero arbitrario di volte: se scegliamo $k = 0$ otteniamo una stringa z' che contiene un numero iniziale di b minore di N perché ne abbiamo cancellata almeno una, ma questo è assurdo perché la ripetizione arbitraria dovrebbe mantenerci in L, quindi quest'ultimo non è regolare.

Richiesta 6.2.1.2: $\{a^m b^n a^n \mid m \geq 1 \wedge n \geq 0\}$.

Soluzione 6.2.1.2: Per assurdo sia L regolare, e sia N la costante del pumping lemma. Prendiamo la stringa

$$z = a b^N a^N$$

di lunghezza $|z| = 2N + 1 \geq N$ e scomponiamola come $z = uvw$. Sappiamo che $|uv| \leq N$, quindi per uv abbiamo 3 scomposizioni possibili:

- $u = \varepsilon \wedge v = a$: basta scegliere $k = 0$ nella ripetizione arbitraria per far cadere $m \geq 1$;
- $u = \varepsilon \wedge v = ab^i \mid i \in \{1, ..., N-1\}$: basta scegliere $k = 0$ nella ripetizione arbitraria per far cadere $m \geq 0$ e l'uguaglianza finale delle a e delle b;
- $u = ab^i \wedge v = b^j \mid i + j \leq N - 1 \wedge j \geq 1$: basta scegliere $k = 0$ nella ripetizione arbitraria per far cadere l'uguaglianza finale delle a e delle b.

Ma questo è assurdo perché la ripetizione arbitraria dovrebbe mantenerci in L, quindi quest'ultimo non è regolare.

Richiesta 6.2.1.3: $\{a^n b^m a^n \mid m \geq 1 \wedge n \geq 0\}$.

Soluzione 6.2.1.3: Per assurdo sia L regolare, e sia N la costante del pumping lemma. Prendiamo la stringa

$$z = a^N b a^N$$

di lunghezza $|z| = 2N + 1 \geq N$ e scomponiamola come $z = uvw$. Sappiamo che $|uv| \leq N$, quindi uv è formata da solo a, ovvero

$$u = a^i \wedge v = a^j \mid i + j \leq N \wedge j \geq 1.$$

Sappiamo anche che possiamo ripetere v un numero arbitrario di volte: se scegliamo $k = 0$ otteniamo una stringa z' che contiene un numero iniziale di a minore di N perché ne abbiamo cancellata almeno una, ma questo è assurdo perché la ripetizione arbitraria dovrebbe mantenerci in L, quindi quest'ultimo non è regolare.

Richiesta 6.2.1.4: $\{a^p \mid p \text{ è un numero primo}\}$.

Soluzione 6.2.1.4: Per assurdo sia L regolare, e sia N la costante del pumping lemma. Prendiamo la stringa

$$z = a^{N+k} \mid k \geq 0$$

di lunghezza $|z| = N + k \geq N$ tale che $N + k$ è primo. Scomponiamo z come $z = uvw$. Ogni pezzo di z è formato da solo a, ma siamo sicuri che v ne abbia almeno una perché $v \neq \varepsilon$.

Sia quindi $v = a^i \mid i \geq 1$. Abbiamo due casi per $N + k$:

- se $N + k$ pari vuol dire che $N + k = 2$, unico primo pari, ma allora:
  - se $i = 1$ scegliamo $k = 3$ ottenendo $z' = aaaa$ che non appartiene a L;
  - se $i = 2$ scegliamo $k = 2$ ottenendo ancora $z' = aaaa$ che non appartiene a L;
- se $N + k$ dispari allora:
  - se i è dispari andiamo a scegliere $k = 0$ nella ripetizione arbitraria per rimuovere un numero dispari di a da $N + k$, ottenendo quindi un numero pari, che non è primo;
  - se i è pari andiamo a scegliere k che mi renda il numero di a non primo, e siamo sicuri che esista questo numero perché esistono infiniti numeri dispari che non sono primi.

Ma ogni caso visto è assurdo perché la ripetizione arbitraria dovrebbe mantenerci in L, quindi quest'ultimo non è regolare.

Richiesta 6.2.1.5: $\{a^{2^n} \mid n \geq 0\}$.

Soluzione 6.2.1.5: Vedi teoria.

Richiesta 6.2.1.6: $\{a^{n!} \mid n \geq 0\}$.

Soluzione 6.2.1.6: Per assurdo sia L regolare, e sia N la costante del pumping lemma. Prendiamo la stringa

$$z = a^{N!}$$

di lunghezza $|z| = N! \geq N$ e scomponiamola come $z = uvw$. Ogni pezzo di z è formato da solo a, ma siamo sicuri che v ne abbia almeno una perché $v \neq \varepsilon$. Sia quindi $v = a^i \mid i \geq 1$.

Potendo ripetere arbitrariamente v, scegliamo $k = 2$ e andiamo a scrivere

$$\left|uv^2w\right| = N! + |v| = N! + i.$$

Il fattoriale successivo a $N!$ è $(N+1)!$, ma per ottenere questo dovremmo avere

$$\begin{gathered} N! + i < (N+1)! \\ i < (N+1)N! - N! \\ i < (N+1-1)N! \\ i < NN! \end{gathered}$$

TODO: **DA CHIEDERE A MARTINO**

6.3. Esercizio 03

Esercizio 6.3.1: Dimostrate che il linguaggio delle parentesi bilanciate non è regolare in tre modi.

Richiesta 6.3.1.1: Utilizzando il pumping lemma.

Soluzione 6.3.1.1: Per assurdo sia L regolare, e sia N la costante del pumping lemma. Prendiamo la stringa

$$z = ({}^N)^N$$

di lunghezza $|z| = 2N \geq N$ e scomponiamola come $z = uvw$. Sappiamo che $|uv| \leq N$, quindi uv è formata da solo (, ovvero

$$u = ({}^i \land v = ({}^j \mid i + j \leq N \land j \geq 1.$$

Sappiamo anche che possiamo ripetere v un numero arbitrario di volte: se scegliamo $k = 0$ otteniamo una stringa z' che contiene un numero iniziale di (minore di N perché

ne abbiamo cancellata almeno una, ma questo è assurdo perché la ripetizione arbitraria dovrebbe mantenerci in L, quindi quest'ultimo non è regolare.

Richiesta 6.3.1.2: Utilizzando il concetto di distinguibilità.

Soluzione 6.3.1.2: Definiamo l'insieme

$$X = \{(^k \mid k \geq 0\}$$

formato da stringhe distinguibili tra loro per L. Infatti, prese due stringhe x_i e x_j, con $i \neq j$, avremo

$$\#_{(}(x_i) = i \neq j = \#_{(}(x_j)$$

che possono essere distinte con la stringa $z =)^i$.

Ma allora ogni DFA per L ha almeno $|X| = \infty$ stati, ma quindi L non è regolare.

Richiesta 6.3.1.3: Sfruttando le proprietà di chiusura della classe dei linguaggi regolari per mostrare che la sua regolarità implicherebbe quella del linguaggio $\{a^n b^n \mid n \geq 0\}$.

Soluzione 6.3.1.3: Per assurdo sia L regolare, allora esso è chiuso tramite l'operazione di intersezione. Andiamo a calcolare

$$L \cap (^*)^*$$

andiamo a ottenere il linguaggio

$$\{(^n)^n \mid n \geq 0\}$$

perché tra tutte le stringhe di parentesi bilanciate vado a prendere quelle che hanno tutte le parentesi aperte all'inizio seguite da tutte quelle chiuse per, appunto, chiudere quelle aperte.

Però il linguaggio $\{(^n)^n \mid n \geq 0\}$ non è regolare, ed essendo $(^*)^*$ regolare, deve essere per forza L non regolare.

6.4. Esercizio 04

Esercizio 6.4.1: Considerate il seguente linguaggio $L \subseteq \{a, b, c\}^*$:

$$L = \{a^m b^n c^n \mid m \geq 1 \wedge n \geq 0\} \cup \{b^m c^n \mid m, n \geq 0\}.$$

Richiesta 6.4.1.1: Verificate che il linguaggio L soddisfa la condizione del pumping lemma per i linguaggi regolari.

Soluzione 6.4.1.1: Non ci riesco ora.

Richiesta 6.4.1.2: Dimostrate che L non è regolare utilizzando il concetto di distinguibilità.

Soluzione 6.4.1.2: Definiamo l'insieme

$$X = \{ab^k \mid k \geq 0\}$$

formato da stringhe distinguibili tra loro per L. Infatti, prese due stringhe x_i e x_j, con $i \neq j$, avremo

$$\#_b(x_i) = i \neq j = \#_b(x_j)$$

che possono essere distinte con la stringa $z = c^i$.

Ma allora ogni DFA per L ha almeno $|X| = \infty$ stati, ma quindi L non è regolare.

Richiesta 6.4.1.3: Dimostrate che L non è regolare sfruttando le proprietà di chiusura della classe dei linguaggi regolari.

Soluzione 6.4.1.3: Per assurdo sia L regolare, allora esso è chiuso tramite l'operazione di intersezione. Andiamo a calcolare

$$L \cap ab^*c^*$$

andiamo a ottenere il linguaggio

$$\{ab^n c^n \mid n \geq 0\}$$

perché andiamo a cancellare tutte le stringhe del secondo insieme e di tutte le prime teniamo quelle che hanno solo una a, seguita da tutte le b e tutte le c, che però devono essere in numero uguale per la definizione del primo insieme.

Però il linguaggio $\{ab^n c^n \mid n \geq 0\}$ non è regolare, ed essendo ab^*c^* regolare, deve essere per forza L non regolare.

7. Esercizi lezione 12 [04/04]

7.1. Esercizio 01

Esercizio 7.1.1: Dato un linguaggio L indichiamo con $\text{cycle}(L)$ il linguaggio ottenuto ruotando ciclicamente in tutti i modi possibili le stringhe di L, i.e.,

$$\text{cycle}(L) = \{vu \in \Sigma^* \mid uv \in L\}.$$

Richiesta 7.1.1.1: Dimostrate che se L è regolare allora anche $\text{cycle}(L)$ è regolare, fornendo una costruzione per ottenere un automa che accetti $\text{cycle}(L)$ a partire da un automa che accetta L.

Soluzione 7.1.1.1: Non so da che parte sono girato.

7.2. Esercizio 02

Esercizio 7.2.1: Dato un linguaggio L indichiamo con $\frac{1}{2}L$ il linguaggio formato dalla prima metà delle stringhe di L (di lunghezza pari), i.e.,

$$\frac{1}{2}L = \{x \in \Sigma^* \mid \exists y \in \Sigma^* \mid |y| = |x| \wedge xy \in L\}.$$

Richiesta 7.2.1.1: Dimostrate che se L è regolare allora anche $\frac{1}{2}L$ è regolare, fornendo una costruzione per ottenere un automa che accetti $\frac{1}{2}L$ a partire da un automa che accetta L.

Soluzione 7.2.1.1: Vedi teoria.

7.3. Esercizio 03

Esercizio 7.3.1: Dimostrate che se L è un linguaggio regolare, allora sono regolari anche i seguenti linguaggi.

Richiesta 7.3.1.1: $1\frac{1}{3}L = \{x \in \Sigma^* \mid \exists y, z \in \Sigma^* \mid |x| = |y| = |z| \wedge xyz \in L\}$.

Soluzione 7.3.1.1: Dato l'automa A per L, costruiamo l'automa

$$A' = (Q', \Sigma, \delta', q_0', F')$$

definito da:

- l'**insieme degli stati**

  $$Q' = Q^3 \times 2^Q \times 2^Q$$

  è formato da una serie di quintuple nella forma

  $$[p, s_1, s_2, q, r]$$

  con:
  - p stato nel quale si trova l'automa A;
  - s_1 scommessa sullo stato nel quale si trova A dopo aver letto x;
  - s_2 scommessa sullo stato nel quale si trova A dopo aver letto y partendo dallo stato q
  - q stati nei quali si trova l'automa A partendo da s_1 mentre crea y;
  - r stati nei quali si trova l'automa A partendo da s_2 mentre crea z;
- lo **stato iniziale** è un insieme di quintuple

  $$q_0' = \{[q_0, p, q, \{p\}, \{q\}] \mid \forall p, q \in Q\};$$
- la **funzione di transizione** deve mandare avanti l'automa A su x in maniera deterministica, mentre y e z le deve creare, come nel caso di $\frac{1}{2}L$, quindi

  $$\delta'([p, s_1, s_2, q, r], a) = \left[\delta(p, a), s_1, s_2, \bigcup_{q_1 \in q \wedge \sigma \in \Sigma} \delta(q_1, \sigma), \bigcup_{r_1 \in r \wedge \sigma \in \Sigma} \delta(r_1, \sigma)\right];$$
- l'**insieme degli stati finali** deve controllare le scommesse fatte, ovvero il primo automa finisce in s_1, il secondo automa finisce in s_2, il terzo automa finisce in uno stato finale. Formalmente lo scriviamo come

  $$F' = \{[s_1, s_1, s_2, q, r] \mid s_2 \in q \wedge r \cap F \neq \oslash\}.$$

Richiesta 7.3.1.2: $2\frac{1}{3}L = \{y \in \Sigma^* \mid \exists x, z \in \Sigma^* \mid |x| = |y| = |z| \wedge xyz \in L\}$.

Soluzione 7.3.1.2: Dato l'automa A per L, costruiamo l'automa

$$A' = (Q', \Sigma, \delta', q_0', F')$$

definito da:

- l'**insieme degli stati**

  $$Q' = 2^Q \times Q^3 \times 2^Q$$

  è formato da una serie di quintuple nella forma

$$[p, s_1, s_2, q, r]$$

con:

- p stati nei quali si trova l'automa A mentre crea x;
- s_1 scommessa sullo stato nel quale si trova A dopo aver letto x;
- s_2 scommessa sullo stato nel quale si trova A dopo aver letto y partendo dallo stato q
- q stato nel quale si trova l'automa A partendo da s_1 mentre riconosce y;
- r stati nei quali si trova l'automa A partendo da s_2 mentre crea z;

- lo **stato iniziale** è un insieme di quintuple

$$q_0' = \{[\{q_0\}, p, q, p, \{q\}] \mid \forall p, q \in Q\};$$

- la **funzione di transizione** deve creare x e z considerando tutti i possibili cammini, mentre manda avanti l'automa per y considerando solo il carattere che viene letto, quindi

$$\delta'([p, s_1, s_2, q, r], a) = \left[\bigcup_{p_1 \in p \wedge \sigma \in \Sigma} \delta(p_1, \sigma), s_1, s_2, \delta(q, a), \bigcup_{r_1 \in r \wedge \sigma \in \Sigma} \delta(r_1, \sigma)\right];$$

- l'**insieme degli stati finali** deve controllare le scommesse fatte, ovvero il primo automa finisce in s_1, il secondo automa finisce in s_2, il terzo automa finisce in uno stato finale. Formalmente lo scriviamo come

$$F' = \{[p, s_1, s_2, s_2, r] \mid s_1 \in p \wedge r \cap F \neq \varnothing\}.$$

Richiesta 7.3.1.3: $3\frac{1}{3}L = \{z \in \Sigma^* \mid \exists x, y \in \Sigma^* \mid |x| = |y| = |z| \wedge xyz \in L\}$.

Soluzione 7.3.1.3: Dato l'automa A per L, costruiamo l'automa

$$A' = (Q', \Sigma, \delta', q_0', F')$$

definito da:

- l'**insieme degli stati**

$$Q' = 2^Q \times Q^2 \times 2^Q \times Q$$

è formato da una serie di quintuple nella forma

$$[p, s_1, s_2, q, r]$$

con:

- p stati nei quali si trova l'automa A mentre crea x;
- s_1 scommessa sullo stato nel quale si trova A dopo aver letto x;
- s_2 scommessa sullo stato nel quale si trova A dopo aver letto y partendo dallo stato q
- q stati nei quali si trova l'automa A partendo da s_1 mentre crea y;

- r stato nel quale si trova l'automa A partendo da s_2 mentre riconosce z;

- lo **stato iniziale** è un insieme di quintuple
$$q'_0 = \{[\{q_0\}, p, q, \{p\}, q] \mid \forall p, q \in Q\};$$
- la **funzione di transizione** deve creare x e y considerando tutti i possibili cammini, mentre manda avanti l'automa per z considerando solo il carattere che viene letto, quindi
$$\delta'([p, s_1, s_2, q, r], a) = \left[\bigcup_{p_1 \in p \wedge \sigma \in \Sigma} \delta(p_1, \sigma), s_1, s_2, \bigcup_{q_1 \in q \wedge \sigma \in \Sigma} \delta(q_1, \sigma), \delta(r, a)\right];$$
- l'**insieme degli stati finali** deve controllare le scommesse fatte, ovvero il primo automa finisce in s_1, il secondo automa finisce in s_2, il terzo automa finisce in uno stato finale. Formalmente lo scriviamo come
$$F' = \{[p, s_1, s_2, q, r] \mid s_1 \in p \wedge s_2 \in q \wedge r \in F\}.$$

7.4. Esercizio 04

Esercizio 7.4.1:

Richiesta 7.4.1.1: Dimostrate che se L è un linguaggio regolare, allora è regolare anche il seguente linguaggio:
$$\log(L) = \left\{x \in \Sigma^* \mid \exists y \in \Sigma^* \mid |y| = 2^{|x|} \wedge xy \in L\right\}.$$

Soluzione 7.4.1.1: Non so da che parte sono girato.

7.5. Esercizio 05

Esercizio 7.5.1:

Richiesta 7.5.1.1: Dimostrate che se L è un linguaggio regolare, allora è regolare anche il seguente linguaggio:
$$\text{root}(L) = \left\{w \in \Sigma^* \mid w^{|w|} \in L\right\}.$$

Soluzione 7.5.1.1: Ancora peggio.

8. Esercizi lezione 13 [09/04]

8.1. Esercizio 01

Esercizio 8.1.1: Fissato un intero $n > 0$, sia K_n il linguaggio sull'alfabeto $\{a, b\}$ denotato dall'espressione $(a+b)^*a(a+b)^{n-1}a(a+b)^*$.

Richiesta 8.1.1.1: A lezione abbiamo discusso come riconoscere il linguaggio K_n con un automa deterministico two-way con $O(n)$ stati. Scrivete formalmente la funzione di transizione di tale automa (rispetto al parametro n).

Soluzione 8.1.1.1: Usiamo lo stato r per ricercare le a:

$$\delta(r, a) = (q_0, +1)$$
$$\delta(r, b) = (r, +1).$$

Con lo stato r andiamo avanti a fare la ricerca delle a, mentre da q_0 devo leggere $n-1$ caratteri:

$$\delta(q_i, a/b) = (q_{i+1}, +1) \quad \forall i \in \{0, ..., n-2\}.$$

Nello stato q_{n-1} vediamo se abbiamo una a:

$$\delta(q_{n-1}, a) = \big(q_f, +1\big).$$

In caso contrario, torniamo indietro di $n-1$ caratteri e ripartiamo con la ricerca delle a a partire dallo stato r. Se $n = 2$ torniamo direttamente nello stato r andando indietro di 1, invece negli altri casi facciamo

$$\delta(q_{n-1}, b) = (p_1, -1)$$
$$\delta(p_i, a/b) = (p_{i+1}, -1) \quad \forall i \in \{1, ..., n-3\}$$
$$\delta(p_{n-2}, a/b) = (r, -1).$$

Dallo stato q_f vado avanti a leggere fino a quando non sforo l'end marker destro.

Infine, vanno aggiunti un po' di regole per vedere se, mentre andiamo avanti di n caratteri, andiamo a sforare l'end marker: in questo caso andiamo in uno stato trappola e ciao a tutti.

Richiesta 8.1.1.2: Abbiamo inoltre accennato a come riconoscere K_n con un automa sweeping con $O(n^2)$ stati, cioè un automa two-way che può cambiare la direzione della testina di input solo sugli end-marker. (Nell'i-esima «passata» del nastro di input, l'automa ispeziona i simboli nelle posizioni k tali che $k \bmod n = i$.) Scrivete la funzione di transizione di tale automa nel caso $n = 3$ e generalizzatela poi a ogni n fissato.

Soluzione 8.1.1.2: Non mi viene in mente ora.

8.2. Esercizio 02

Esercizio 8.2.1:

Richiesta 8.2.1.1: Modificate l'automa sweeping per K_n ottenuto nell'esercizio 1 in modo da ridurre il numero degli stati a $O(n)$.

Suggerimento. In ciascuna passata si utilizza un contatore modulo n per individuare i simboli da ispezionare. Il contatore può essere inizializzato in funzione del valore finale del contatore nella passata precedente, evitando di contare le passate.

Soluzione 8.2.1.1: Non mi viene nemmeno questo.

8.3. Esercizio 03

Esercizio 8.3.1:

Richiesta 8.3.1.1: Svolgete l'esercizio 1 per il linguaggio formato da tutte le stringhe in cui esistono due simboli a distanza n tra loro che sono uguali.

Soluzione 8.3.1.1: Vedi esercizio 1 ma con gli stati di avanzamento doppi, in base alla lettera che viene letta dallo stato r.

8.4. Esercizio 04

Esercizio 8.4.1:

Richiesta 8.4.1.1: Svolgete l'esercizio 1 per il linguaggio formato da tutte le stringhe in cui tutti i simboli a distanza n tra loro sono uguali.

Soluzione 8.4.1.1: Non ho voglia di formalizzarlo, quindi lo farò scritto:

1. partiamo da uno stato r_0, andiamo avanti di n posti e vediamo se abbiamo lo stesso carattere;
2. in caso negativo andiamo in loop in qualche stato trappola;
3. in caso positivo torniamo indietro di $n-1$ caratteri e ripetiamo dal punto iniziale;
4. dopo aver fatto n iterazioni andiamo a sforare l'end marker destro.

8.5. Esercizio 05

Esercizio 8.5.1:

Richiesta 8.5.1.1: Rivedete la simulazione di automi two-way deterministici mediante automi one-way deterministici presentata a lezione e studiate come possa essere modificata nel caso l'automa di partenza sia two-way non deterministico.

Soluzione 8.5.1.1: Ho un dubbio amletico, fino a quando non lo risolvo non posso fare questo esercizio matto.

8.6. Esercizio 06

Esercizio 8.6.1: Supponete di disporre di un automa one-way deterministico A che riconosca un linguaggio L. Fornite, partendo da A, la costruzione di automa two-way deterministico per i seguenti linguaggi (cercate di ottenere automi con un numero di stati lineare o polinomiale rispetto a quelli dell'automa A).

Richiesta 8.6.1.1: $\beta(L) = \{x \mid xx^R \in L\}$.

Soluzione 8.6.1.1: Dobbiamo costruire A' un 2DFA per $\beta(L)$.

Questo automa ha in input x, che legge per intero e simula l'esecuzione di A. Quando raggiunge l'end marker destro allora si rimette sull'ultimo carattere e continua la lettura dallo stesso stato nel quale eravamo alla fine di x.

Ora leggiamo la x al contrario fino all'end marker sinistro, dove controlliamo lo stato nel quale siamo: se esso è finale allora sfondiamo l'end marker destro.

Come numero di stati abbiamo:

- lo stesso numero di stati di A per simularlo;
- lo stesso numero di stati di A per simularlo, ma al contrario, quindi servono regole che vanno indietro e non avanti;
- uno stato finale e uno stato trappola.

Quindi abbiamo lo stesso ordine di grandezza degli stati di A.

Richiesta 8.6.1.2: $\alpha(L) = \{x \mid xx \in L\}$.

Soluzione 8.6.1.2: Dobbiamo costruire A' un 2DFA per $\alpha(L)$.

Questo automa ha in input x, che legge per intero e simula l'esecuzione di A. Quando raggiunge l'end marker destro allora fa uno sweeping e si rimette sul primo carattere di x e continua la lettura dallo stesso stato nel quale eravamo alla fine di x.

Ora leggiamo la x di nuovo fino all'end marker destro, dove controlliamo lo stato nel quale siamo: se esso è finale allora sfondiamo l'end marker destro.

Come numero di stati abbiamo:

- lo stesso numero di stati di A per simularlo;
- lo stesso numero di stati di A per tornare indietro, perché dobbiamo tenere traccia dello stato nel quale eravamo;
- lo stesso numero di stati di A per simularlo di nuovo, quindi servono regole che poi si fermano all'end marker;
- uno stato finale e uno stato trappola.

Quindi abbiamo lo stesso ordine di grandezza degli stati di A.

8.7. Esercizio 07

Esercizio 8.7.1: Supponete di disporre di un automa one-way deterministico A che riconosca un linguaggio L.

Richiesta 8.7.1.1: Fornite, partendo da A, la costruzione di un automa two-way non deterministico per il seguente linguaggio:

$$\frac{1}{2}L = \{x \in \Sigma^* \mid \exists y \in \Sigma^* \mid |x| = |y| \wedge xy \in L\}.$$

Cercate di ottenere un automa two-way non deterministico con un numero di stati lineare o polinomiale rispetto a quelli dell'automa A.

Soluzione 8.7.1.1: Dobbiamo costruire A' un 2NFA per $\frac{1}{2}L$.

Questo automa ha in input x, che legge per intero e simula l'esecuzione di A. Quando raggiunge l'end marker destro allora fa uno sweeping e si rimette sul primo carattere di x con lo stesso stato che abbiamo raggiunto a fine x.

Ora leggiamo la x di nuovo fino all'end marker destro, ma **non deterministicamente** generiamo tutti i possibili cammini di lunghezza $|x|$. In poche parole, leggiamo i caratteri di x ma solo per sapere quanto è lunga.

Accettiamo se sull'end marker destro abbiamo almeno uno stato finale.

Come numero di stati abbiamo:

- lo stesso numero di stati di A per simularlo;
- lo stesso numero di stati di A per tornare indietro, perché dobbiamo tenere traccia dello stato nel quale eravamo;
- non oltre $2^{|A|}$ stati per via della costruzione per sottoinsiemi;
- uno stato finale e uno stato trappola.

Richiesta 8.7.1.2: Cosa riuscite a dire rispetto al numero di stati di un automa two-way deterministico? (Anche se non riuscite a calcolare il numero di stati necessario e sufficiente, provate a capire come si potrebbe ottenere facilmente un upper bound sul numero di stati di un automa two-way deterministico. Alla luce di quanto sarà discusso nella lezione 14, cercate di capire quali implicazioni vi sarebbero se tale numero fosse anche necessario.)

Soluzione 8.7.1.2: Ho fatto la lezione 14 prima di questi esercizi, RIP.

9. Esercizi lezione 14 [11/04]

9.1. Esercizio 01

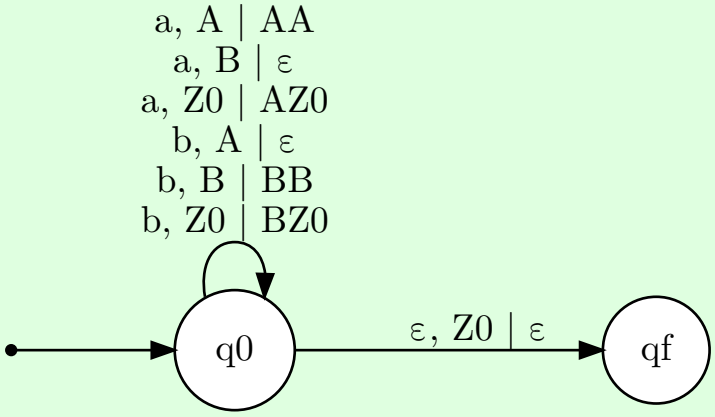
Esercizio 9.1.1:

Richiesta 9.1.1.1: Descrivete e costruite un automa a pila che accetti l'insieme delle stringhe sull'alfabeto $\{a, b\}$ che contengono lo stesso numero di a e b, come ad esempio *aabb*, *abba*, *baba*.

Soluzione 9.1.1.1: L'idea dietro un automa a pila per questo linguaggio è quella di mettere sulla pila i caratteri A e B per contare quante ne abbiamo, e poi vedere se i due numeri sono uguali. Per contare usiamo un semplice algoritmo:

- se leggo una a e sulla pila ho una A allora ne metto un'altra, altrimenti se ho una B la andiamo a cancellare perché vuol dire che ho letto una B poco prima, quindi la a di ora e la b di prima sono in egual numero;
- stesso discorso per la b;
- ogni volta che leggo Z_0 sulla pila, con una ε-mossa mi sposto in uno stato finale scommettendo che l'input sia finito e che quindi tutte le A e le B si sono cancellate sulla pila, avendo quindi un numero di a e di b uguale.

Andiamo ad accettare per pila vuota. Va bene anche l'accettazione per stato finale, rendendo q_f finale, ma mi piaceva di più pila vuota.



Con l'accorgimento della ε-mossa riusciamo a riconoscere anche la stringa vuota.

9.2. Esercizio 02

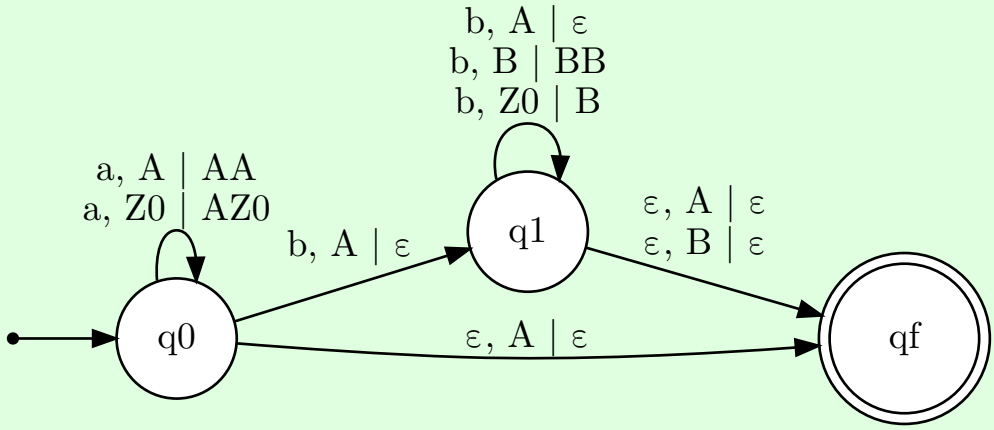
Esercizio 9.2.1:

Richiesta 9.2.1.1: Descrivete e costruite un automa a pila che accetti il linguaggio $\{a^n b^m \mid n \neq m\}$.

Soluzione 9.2.1.1: Per questo automa andiamo a riempire tutta la pila con un numero n di A, poi andiamo a togliere una A per ogni b che troviamo. Se:

- $n > m$ allora alla fine arriviamo con delle A sulla pila, e non deterministicamente andiamo in uno stato finale;
- $n < m$ vuol dire che leggendo delle b arriviamo a Z_0, ma in questo caso andiamo a mettere delle B sulla pila e poi a fine stringa ci spostiamo nello stato finale.

Andiamo ad accettare per stati finali.



Manca da aggiungere la transizione

$$b, Z_0 \mid Z_0$$

ma non ho voglia di cambiare l'immagine ora.

9.3. Esercizio 03

Esercizio 9.3.1: Sia $\Sigma = \{(,)\}$ un alfabeto i cui simboli sono la parentesi aperta e la parentesi chiusa.

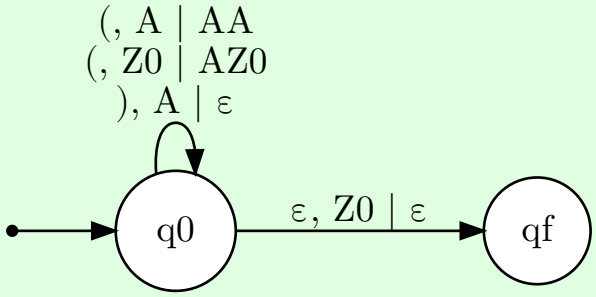
Richiesta 9.3.1.1: Descrivete e costruite un automa a pila che riconosca il linguaggio formato da tutte le sequenze di parentesi correttamente bilanciate, come ad esempio (()(()))().

Soluzione 9.3.1.1: Costruiamo un automa a pila che:

- ogni volta che legge una parentesi aperta mettiamo una A sulla pila;
- ogni volta che legge una parentesi chiusa toglie una A dalla pila, se può.

Se non possiamo togliere una A dalla pila l'automa si blocca. Poi, non deterministicamente, ogni volta che leggiamo Z_0 sulla pila andiamo in uno stato finale.

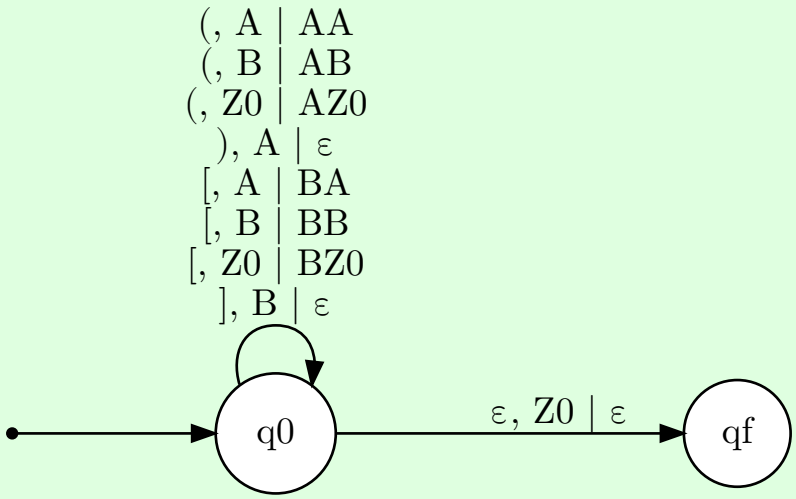
Andiamo ad accettare per pila vuota. Va bene anche l'accettazione per stato finale, rendendo q_f finale, ma mi piaceva di più pila vuota.



Richiesta 9.3.1.2: Risolvete il punto precedente per un alfabeto con due tipi di parentesi, come $\Sigma = \{(,), [,]\}$, nel caso non vi siano vincoli tra i tipi di parentesi (le tonde possono essere contenute tra quadre e viceversa). Esempio [()([])[]], ma non [[][(])()].

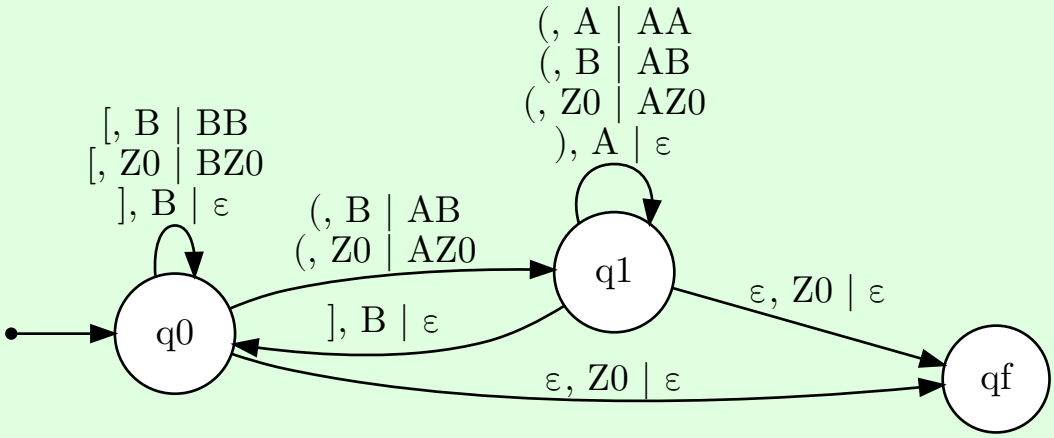
Soluzione 9.3.1.2: L'automa è uguale a quello di prima, solo con delle regole in più per le parentesi quadre, rappresentate dal carattere B nella pila.

Andiamo ad accettare per pila vuota. Va bene anche l'accettazione per stato finale, rendendo q_f finale, ma mi piaceva di più pila vuota.



Richiesta 9.3.1.3: Risolvete il punto precedente con $\Sigma = \{(,), [,]\}$, con il vincolo che le parentesi quadre non possano mai apparire all'interno delle parentesi tonde. Esempio [()(())[][]](()()) ma non [()([])[]].

Soluzione 9.3.1.3: Con lo stato q_0 andiamo a riconoscere tutte le parentesi quadre, mentre appena leggiamo una parentesi tonda ci spostiamo in q_1, nel quale non possiamo leggere una quadra aperta.



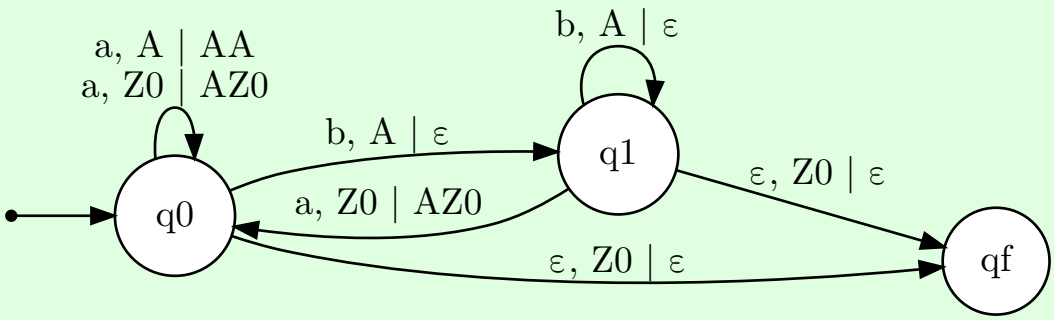
9.4. Esercizio 04

Esercizio 9.4.1:

Richiesta 9.4.1.1: Basandovi su uno degli automi a pila presentati a lezione per il linguaggio $L = \{a^n b^n \mid n \geq 1\}$, costruite un automa a pila per la chiusura di Kleene di L, cioè il linguaggio $\{a^{n_1} b^{n_1} a^{n_2} b^{n_2} \cdots a^{n_k} b^{n_k} \mid k \geq 0 \wedge n_1, n_2, ..., n_k > 0\}$.

Soluzione 9.4.1.1: All'automa a pila che abbiamo visto a lezione per L aggiungiamo una transizione direttamente nello stato finale, visto che dobbiamo accettare anche ε. Inoltre, ogni volta che in q_1 torniamo sulla base della pila, se leggiamo una a allora torniamo in q_0 per iniziare un nuovo riconoscimento.

Andiamo ad accettare per pila vuota. Va bene anche l'accettazione per stato finale, rendendo q_f finale, ma mi piaceva di più pila vuota.

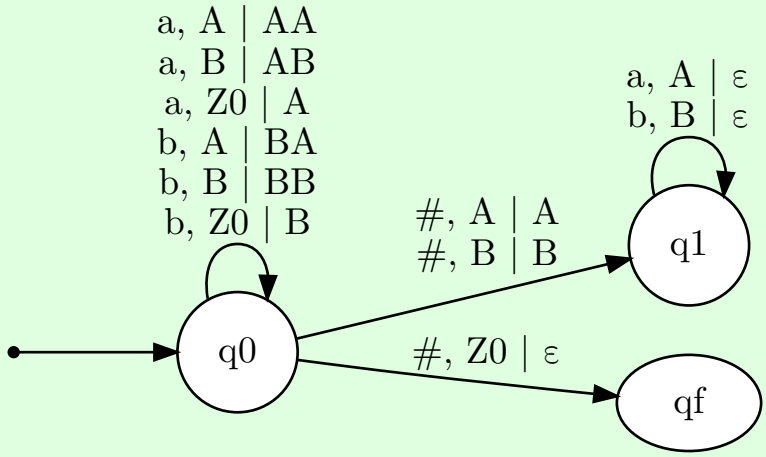


9.5. Esercizio 05

Esercizio 9.5.1: Considerate l'alfabeto $\Sigma = \{a, b\}$.

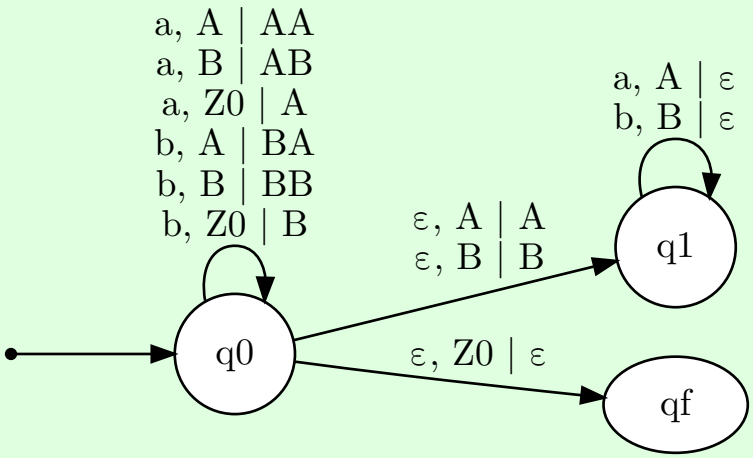
Richiesta 9.5.1.1: Descrivete e costruite un automa a pila per il linguaggio $\text{PAL}_{\#} = \{w\#w^R \mid w \in \Sigma^*\}$, dove $\#$ è un nuovo simbolo, cioè $\# \notin \Sigma$.

Soluzione 9.5.1.1: Un automa a pila per $\text{PAL}_{\#}$ legge la stringa, la mette nella pila, appena trova $\#$ controlla se tirandola fuori si ottiene la stessa stringa che si sta leggendo ora.



Richiesta 9.5.1.2: Modificate l'automa a pila ottenuto al punto precedente in modo che accetti il linguaggio delle stringhe palindrome di lunghezza pari su Σ, cioè per l'insieme $\text{PAL}_{\text{pari}} = \{ww^R \mid w \in \Sigma^*\}$.

Soluzione 9.5.1.2: Un automa a pila per PAL_{pari} in maniera non deterministica si sposta nello stato q_1 scommettendo di essere arrivato a metà stringa.



Richiesta 9.5.1.3: Riuscite a ottenere automi a pila deterministici per $\text{PAL}_{\#}$ e per PAL_{pari}?

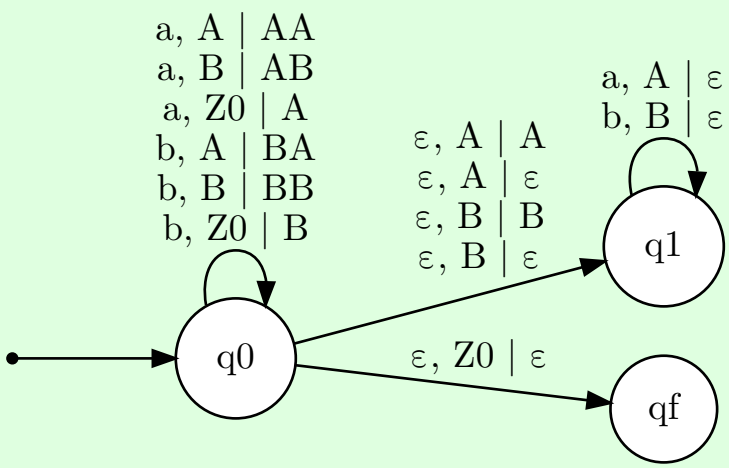
Soluzione 9.5.1.3: Per $\text{PAL}_{\#}$ abbiamo un automa deterministico, mentre per PAL_{pari} no perché dobbiamo scommettere di essere arrivati a metà stringa.

9.6. Esercizio 06

Esercizio 9.6.1: Considerate l'alfabeto $\Sigma = \{a, b\}$ e gli automi a pila ottenuti nell'esercizio precedente.

Richiesta 9.6.1.1: Modificate gli automi a pila ottenuti nell'esercizio precedente in modo da riconoscere il linguaggio di tutte le stringhe palindrome su Σ, cioè l'insieme $\text{PAL} = \{w \in \Sigma^* \mid w = w^R\}$.

Soluzione 9.6.1.1: L'automa a pila deve iniziare a caricare la pila con alcuni caratteri, poi non deterministicamente deve spostarsi nello stato di check della stringa e vedere se ha scommesso bene. Inoltre, non deterministicamente si sposta nello stato di check rimuovendo un elemento dalla pila per indicare che l'elemento tolto è quello centrale di una stringa lunga dispari.



Richiesta 9.6.1.2: Come si possono modificare gli automi a pila per i linguaggi $\text{PAL}_{\#}$, PAL_{pari} e PAL se $\Sigma = \{a, b, c\}$? In quali casi si riescono ad ottenere dispositivi deterministici e in quali no?

Soluzione 9.6.1.2: Basta aggiungere agli automi le stesse regole che già abbiamo per le a e le b ma usando i caratteri c nell'alfabeto di input e C nell'alfabeto della pila.

Come prima, $\text{PAL}_{\#}$ è l'unico automa che è deterministico, mentre gli altri due non lo sono perché dobbiamo scommettere di essere arrivati a metà stringa (entrambi) o di essere arrivato nel carattere centrale della stringa (secondo).

Richiesta 9.6.1.3: Come si possono modificare gli automi a pila per i linguaggi $\text{PAL}_{\#}$, PAL_{pari} e PAL se $\Sigma = \{a, b, \#\}$? In quali casi si riescono ad ottenere dispositivi deterministici e in quali no?

Soluzione 9.6.1.3:

Richiesta 9.6.1.4: Considerate ora i linguaggi $\text{PAL}_{\#}$, PAL_{pari} e PAL nel caso di alfabeto $\Sigma = \{a\}$. Per ognuno di essi cercate di ottenere un dispositivo riconoscitore più semplice possibile e discutete se sia necessario l'uso del non determinismo.

Soluzione 9.6.1.4: Il secondo e il terzo linguaggio mantengono i loro automi: abbiamo cambiato il carattere c con il carattere #, quindi rimaniamo non deterministici con il nostro bellissimo automa a pila.

Nel primo caso dobbiamo aggiungere del non determinismo che indovina di essere arrivato nel carattere # che divide la stringa nelle due stringhe una il rovescio dell'altra.

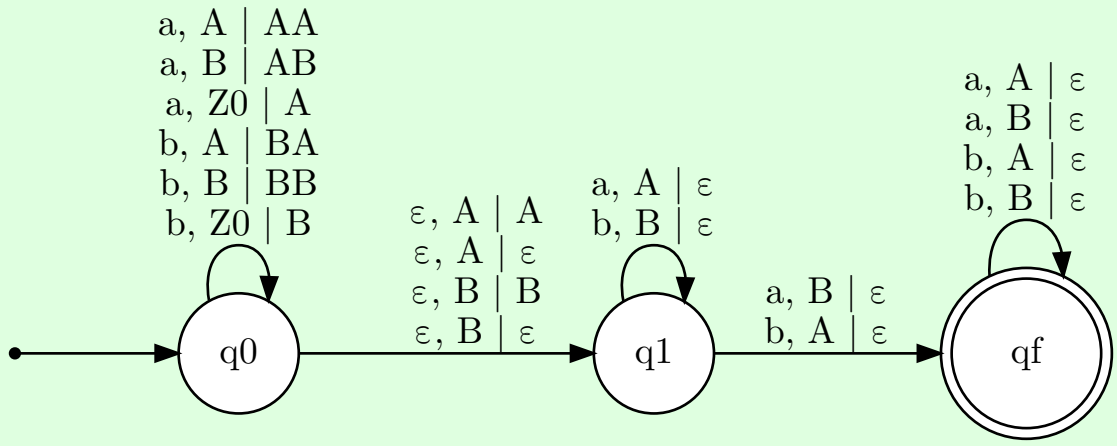
9.7. Esercizio 07

Esercizio 9.7.1:

Richiesta 9.7.1.1: Definite un automa a pila che riconosca tutte le stringhe sull'alfabeto $\{a, b\}$ che non sono palindrome.

Soluzione 9.7.1.1: L'automa a pila inizia a caricare la pila con A e B e poi deve scommettere di essere arrivato a metà, spostandosi nello stato di check, cancellando eventualmente un carattere perché potremmo avere stringhe di lunghezza dispari.

Nello stato di check, se troviamo una corrispondenza di caratteri diversi andiamo in uno stato finale e svuotiamo la pila. In questo caso accettiamo per stati finali.



10. Esercizi lezione 15 [23/04]

10.1. Esercizio 01

Esercizio 10.1.1: Sia $\Sigma = \{a, b\}$ e $L = \{w \in \Sigma^+ \mid \#_a(w) = \#_b(w)\}$, dove $\#_\sigma(w)$ indica il numero di occorrenze del simbolo σ nella stringa w, con $\sigma \in \Sigma$. Siano inoltre $L_a = \{w \in \Sigma^+ \mid \#_a(w) = \#_b(w) + 1\}$ e $L_b = \{w \in \Sigma^+ \mid \#_b(w) = \#_a(w) + 1\}$.

Richiesta 10.1.1.1: Si fornisca una definizione induttiva per le stringe nei linguaggi L, L_a, L_b.

Suggerimento. Ispiratevi all'esempio presentato a lezione per il linguaggio delle parentesi bilanciate, utilizzando affermazioni come «Una stringa di L inizia con una b seguita da una stringa di L_a, oppure inizia con ...».

Soluzione 10.1.1.1: Vediamo le definizioni induttive dei tre linguaggi.

Per L_a possiamo dire che:
- a è stringa di L_a;
- se X è una stringa di L_a, allora anche aXb e bXa e baX e abX e Xab e Xba lo sono.

Per L_b possiamo dire che:
- b è stringa di L_b;
- se X è una stringa di L_b, allora anche aXb e bXa e baX e abX e Xab e Xba lo sono.

Per L possiamo dire che:
- ab e ba sono stringhe di L;
- se X è una stringa di L, allora anche aXb e bXa e baX e abX e Xab e Xba lo sono;
- se X è una stringa di L_a, allora bX e Xb sono in L;
- se X è una stringa di L_b, allora aX e Xa sono in L;
- se X è una stringa di L_a e Y è una stringa di L_b allora XY e YX sono in L.

In realtà questa ultima mi sa che si può evitare, o si può evitare la prima.

Richiesta 10.1.1.2: Dalla definizione ricavate una grammatica per il linguaggio L.

Suggerimento. Per ricavare la grammatica potete considerare una variabile per ognuno dei tre linguaggi. Una delle possibili soluzioni vi permetterà di ottenere la grammatica per L presentata a lezione.

Soluzione 10.1.1.2: Definiamo la grammatica $G = (\{S, A, B\}, \{a, b\}, P, S)$ con regole di produzione nella forma

$$S \longrightarrow aB \mid bA \mid abS \mid baS \mid Sab \mid Sba \mid aSb \mid bSa$$
$$A \longrightarrow a \mid aAb \mid bAa \mid abA \mid baA \mid Aab \mid Aba$$
$$B \longrightarrow b \mid aBb \mid bBa \mid abB \mid baB \mid Bab \mid Bba.$$

10.2. Esercizio 02

Esercizio 10.2.1:

Richiesta 10.2.1.1: Ripetete l'esercizio precedente, sostituendo $w \in \Sigma^*$ a $w \in \Sigma^+$ nella definizione dei linguaggi. Osservate che è sufficiente un unico caso base in tutta la definizione ricorsiva.

Soluzione 10.2.1.1: Aggiungiamo che in L abbiamo anche ε e come regola aggiuntiva mettiamo $S \longrightarrow \varepsilon$, molto molto semplice (troppo semplice per essere giusto).