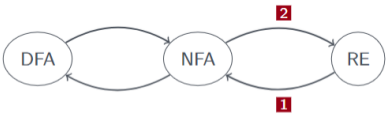
Argomenti trattati durante la lezione:

- Conversione per eliminazione di stati (buon riassunto qui) e GNFA
- Pumping lemma per linguaggi regolari

Equivalenza tra FA e RE (1)

Sappiamo già che DFA e NFA sono equivalenti.

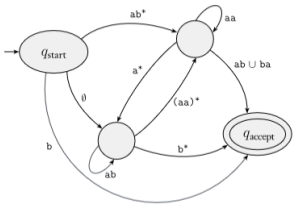


Gli FA sono equivalenti alle espressioni regolari:

1. Per ogni espressione regolare R esiste un *NFA A*, tale che $L(A) = L(R)$
2. Per ogni NFA A possiamo costruire un'espressione regolare R, tale che $L(R) = L(A)$

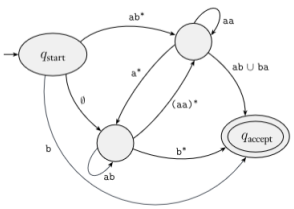
Automi nondeterministici generalizzati (GNFA)

- Sono NFA dove le transizioni sono etichettate con espressioni regolari
- Ogni transizione consuma un blocco di simboli dall'input che appartiene al linguaggio dell'espressione regolare

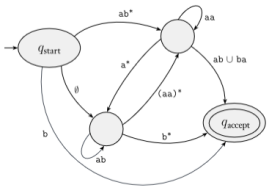


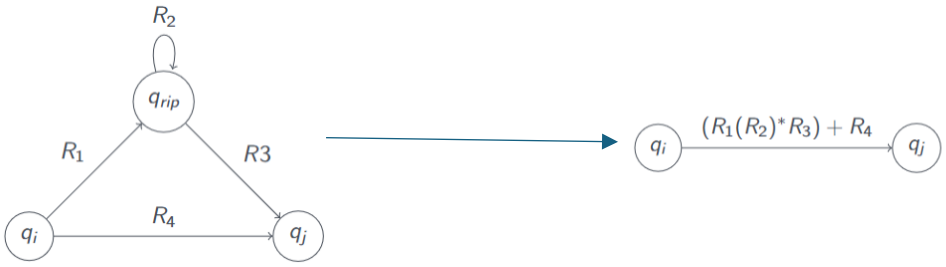
Primo passo: da NFA a GNFA

1. C'è una transizione dallo stato iniziale verso ogni altro stato, e nessuna transizione entrante
2. Un unico stato finale, senza transizioni uscenti e con una transizione proveniente da ogni altro stato
3. C'è sempre una transizione per ogni coppia di stati, ed un self loop dallo stato verso se stesso (eccetto stati iniziale e finale)

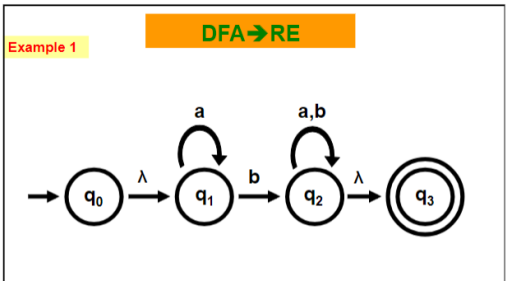


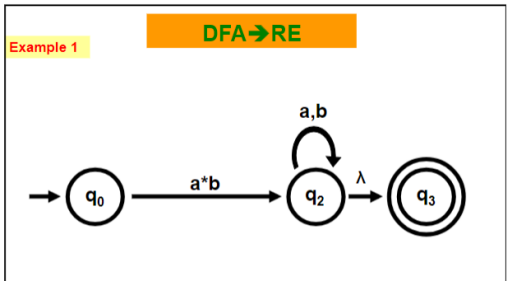
1. Nuovo stato iniziale q_{start} con transizione ε verso il vecchio q_0
2. Nuovo stato finale q_{accept} con transizione ε da tutti i vecchi stati finali $q \in F$
3. Rimpiazzo transizioni multiple tra due stati con l'unione delle etichette
4. Aggiungo transizioni etichettate con $\emptyset$ tra stati non collegati da transizioni





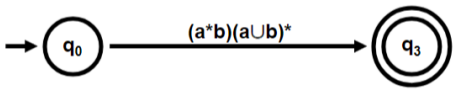
Vediamo alcuni esempi:





DFA→RE

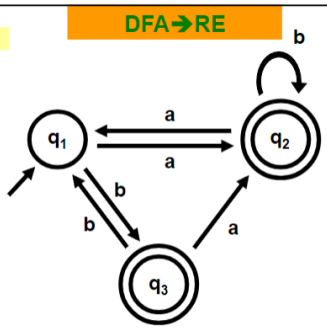
Example 1



$R(q_0,q_3) = (a^*b)(a \cup b)^*$

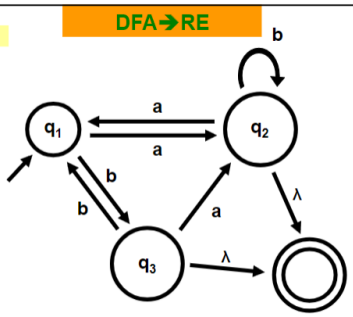
DFA→RE

Example 2



DFA→RE

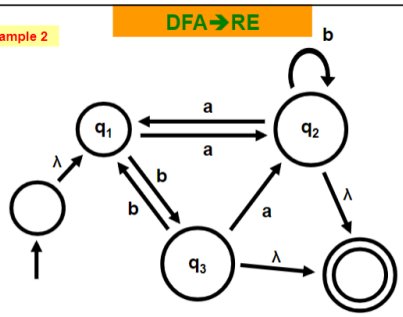
Example 2



19

DFA→RE

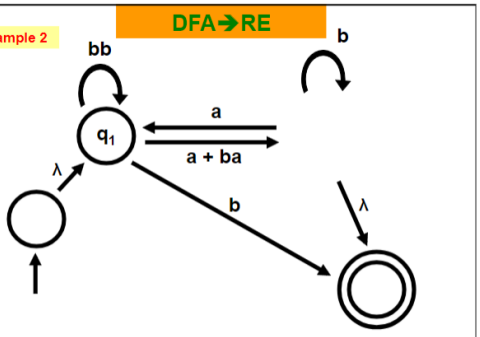
Example 2



20

DFA→RE

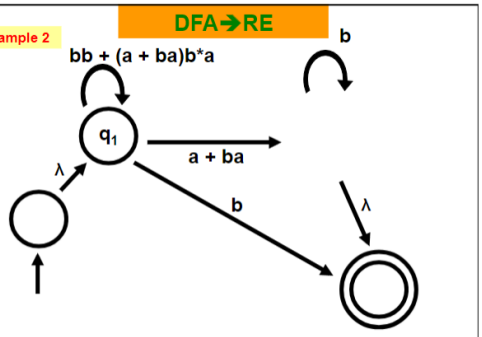
Example 2



25

DFA→RE

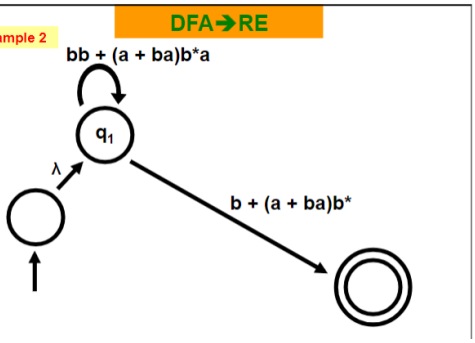
Example 2



26

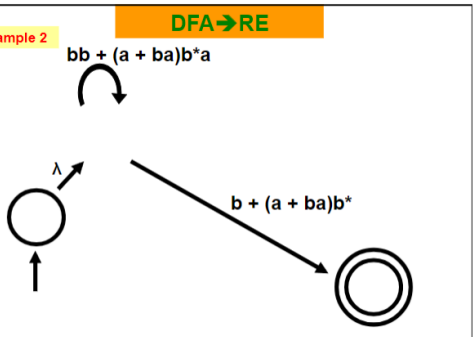
DFA→RE

Example 2



DFA→RE

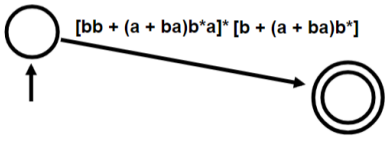
Example 2



28

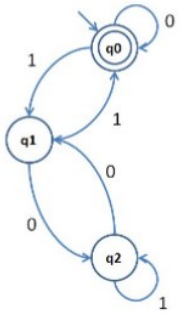
DFA→RE

Example 2

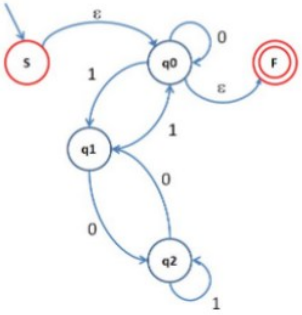


29

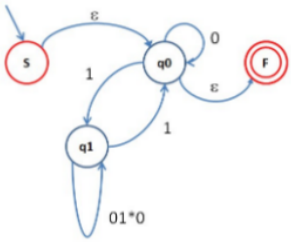
- DFA for binary numbers divisible by 3

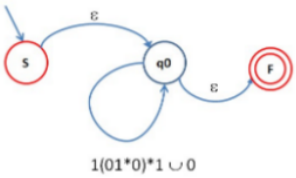


- Initial GNFA

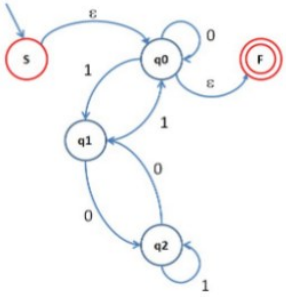


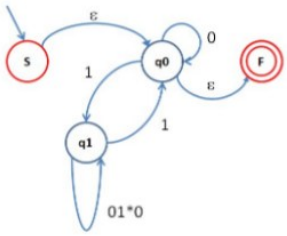
- Let's eliminate q_1





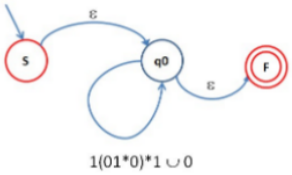
- Let's eliminate q_2

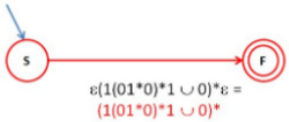




$q_i = q_1, q_j = q_1, q_k = q_2$

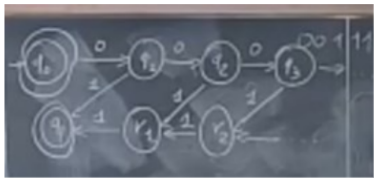
- Let's eliminate q_0





- So the regular expression we are looking for is $(\mathbf{1(01^*0)^*1 \cup 0})^*$

Consideriamo il linguaggio $L = \{0^n 1^n \mid n \geq 0\}$. Il linguaggio non è regolare.

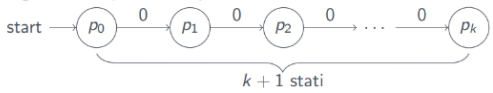


Let's take the language $B = \{0^n 1^n \mid n \geq 0\}$. If we attempt to find a DFA that recognizes B, we discover that the machine seems to need to remember how many 0s have been seen so far as it reads the input. Because the number of 0s isn't limited, the machine will have to keep track of an unlimited number of possibilities. But it cannot do so with any finite number of states.

Costruire un automa e vedere che il suo numero di stati non è finito non è sufficiente.

'Si usa normalmente la dimostrazione per assurdo.

- Supponiamo che $L_{01} = \{0^n 1^n : n \geq 0\}$ sia regolare
- Allora deve essere accettato da un DFA A con un certo numero k di stati
- Cosa succede quando A legge 0^k?
- Seguirà una qualche sequenza di transizioni:



- Siccome ci sono $k+1$ stati nella sequenza, esiste uno stato che si ripete: esistono $i < j$ tali che $p_i = p_j$
- Chiamiamo q questo stato

Dimostrazione:

- Supponiamo che L sia un linguaggio regolare
- Allora è riconosciuto da un DFA con, supponiamo, k stati
- Consideriamo una parola $w = a_1 a_2 \dots a_n \in L$ di lunghezza $n \geq k$
- Consideriamo gli stati nella computazione di A per w:

$$p_0 p_1 p_2 \dots p_k \dots p_n$$

- Siccome in $p_0, p_1, \dots, p_k$ ci sono $k+1$ stati, ne esiste uno che si ripete:

esistono $l < m$ tali che $p_l = p_m$ e $m \leq k$

- Cosa succede quando l'automa A legge 1^i partendo da q?
- Se l'automa finisce la lettura in uno stato finale:
  - allora accetta, sbagliando, la parola $0^i 1^i$
- Se l'automa finisce la lettura in uno stato non finale:
  - allora rifiuta, sbagliando, la parola $0^i 1^i$
- In entrambi i casi abbiamo ingannato l'automa, quindi L_{01} non può essere regolare

- Possiamo spezzare w in tre parti $w = xyz$:
  1. $x = a_1 a_2 \dots a_l$
  2. $y = a_{l+1} a_{l+2} \dots a_m$
  3. $z = a_{m+1} a_{m+2} \dots a_n$
- che rispettano le condizioni del Lemma:
  - $y \neq \varepsilon$ perché $l < m$
  - $|xy| \leq k$ perché $m \leq k$

The Pumping Lemma as a 2-Person Game

1. You pick the language L to be proved nonregular.
2. Your adversary picks n, but does not reveal to you what n is. You must devise a move for all possible n's.
3. You pick w, which may depend on n. $|w| \geq n$.
4. Your adversary picks a factoring of $w = xyz$. Your adversary does not reveal what the factors are, only that they satisfy the constraints of the theorem: $|y| > 0$ and $|xy| \leq n$.
5. You "win" by picking k, which may be a function of n, x, y, and z, such that $xy^k z \notin L$.

- L'avversario sceglie la lunghezza k
- Noi scegliamo una parola w
- L'avversario spezza w in xyz
- Noi scegliamo i tale che $xy^i z \notin L$
- allora abbiamo vinto

- Ogni linguaggio regolare soddisfa il Pumping Lemma.
- Un linguaggio che falsifica il Pumping Lemma non può essere regolare:
  - per ogni lunghezza $k \geq 0$
  - esiste una parola $w \in L$ di lunghezza $|w| \geq k$ tale che
  - per ogni suddivisione $w = xyz$ tale che:
    1. $y \neq \varepsilon$ (il secondo pezzo è non vuoto)
    2. $|xy| \leq k$ (i primi due pezzi sono lunghi al max k)
  - esiste un $i \geq 0$ tale che $xy^i z \notin L$ (possiamo "pompare" y ed uscire da L)
- Attenzione: esistono linguaggi non regolari che rispettano il Pumping Lemma!

Esercizio 1

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

Dimostrazione formale

1. **Assunzione**: Supponiamo, per assurdo, che L sia regolare.
2. Per il Pumping Lemma, esiste $p > 0$ (pumping length).
3. Consideriamo la stringa
   $$w = a^{p+1} b^p$$
   che appartiene a L (poiché ci sono $p+1$ a e p b, quindi la quantità di a è strettamente maggiore di quella di b). Inoltre $|w| = (p+1) + p = 2p + 1 \geq p$.
4. Per dimostrare la contraddizione, prendiamo *qualunque* suddivisione $w = xyz$ con $|xy| \leq p$ e $|y| > 0$.
   - In w, i primi p simboli sono tutti a. Poiché $|xy| \leq p$, la parte y cade interamente nella sezione di a.
   - Di conseguenza, $y = a^q$ per qualche $q > 0$.
5. **Scelta di i**: Poniamo $i = 0$, cioè "rimuoviamo" y. La stringa risultante è
   $$xy^0 z = xz = a^{p+1-q} b^p.$$
   Ora, se $q \geq 1$, allora $(p+1-q) \leq p$. Può capitare perfino $(p+1-q) < p$ se $q > 1$. In ogni caso, non abbiamo più a *strettamente più numerose* di b, e se $q = 1$, i numeri di a e b sono uguali, oppure se $q > 1$, le a possono diventare addirittura meno delle b. In tutte le situazioni, $\#a(xz) \leq \#b(xz)$, quindi $xz \notin L$.
6. **Contraddizione**: La condizione (3) del Pumping Lemma (tutti $xy^i z$ in L) fallisce. Quindi L non è regolare.

Esercizio 2

$$L = \{ a^l b^m a^n \mid l + m = n \}.$$

Dimostrazione formale

1. **Assunzione**: Supponiamo per assurdo che L sia regolare.
2. Esiste un pumping length $p > 0$.
3. **Stringa**: Prendiamo
$$w = a^p\, b^0\, a^p = a^p a^p = a^{2p}.$$
In questa stringa, $l = p$, $m = 0$, $n = p$, dunque $l + m = p + 0 = p = n$, quindi $w \in L$. Inoltre $|w| = 2p \geq p$.
4. **Suddivisione**: Sia $w = xyz$ con $|xy| \leq p$, $|y| > 0$. Poiché i primi p simboli sono tutti a, y è *un blocco di a*, diciamo $y = a^q$.
5. **Scelta di** i: Mettiamo $i = 0$ (rimozione di y). Allora
$$xy^0z = a^{\,2p-q}.$$
Ora, in questa nuova stringa, la parte iniziale e finale di a non mantengono più la relazione $l + m = n$. Infatti se consideriamo la forma generica $a^{l'}b^{m'}a^{n'}$, in xy^0z non ci sono b (poiché $m = 0$ inizialmente), e la lunghezza totale di a risulta $2p - q$. Se volessimo interpretare $l' + m' = n'$, e $m' = 0$, avremmo bisogno che $l' = n'$. Ma qui $l' + n' < 2p$ a seconda di come interpretiamo la separazione. In ogni caso è evidente che la condizione $l' + m' = n'$ non regge più se $l' + n' \neq 2p$ correttamente. Quindi $xy^0z \notin L$.
6. **Contraddizione**: Concludiamo che L non è regolare.

Esercizio 3

$$L = \{\, a^l b^m a^n \mid l + m \equiv n \pmod 3 \}.$$

Dimostrazione formale

1. Supponiamo, per assurdo, che L sia regolare.
2. Esiste un pumping length $p > 0$.
3. **Stringa**: Scegliamo
$$w = a^p\, b^0\, a^p = a^{2p}.$$
In questa stringa, $l = p$, $m = 0$, $n = p$. Poiché $p + 0 - p = 0$, $(p + m) - n = 0 \equiv 0 \pmod 3$. Quindi $w \in L$ e $|w| = 2p \geq p$.
4. **Suddivisione**: Sia $w = xyz$ con $|xy| \leq p$, $|y| > 0$. Di nuovo, y contiene solo a.
5. **Pompaggio**: Scegliamo $i = 0$. Allora
$$xy^0z = a^{\,2p-q}.$$
Ora, se definissimo l' e n' come i numeri di a nella parte iniziale e finale (non necessariamente uguali), quasi certamente $l' + m' - n' \neq 0 \pmod 3$. Se rimuoviamo un certo blocco q, la lunghezza di a totali non soddisfa più la congruenza mod 3 prestabilita. In sostanza, $xy^0z \notin L$.
6. Contraddizione. Dunque L non è regolare.

Esercizio 4

$$L = \{\, 0^{n^2} \mid n \geq 0 \}.$$

Dimostrazione formale

1. Supponiamo, per assurdo, che L sia regolare.
2. Esiste un pumping length $p > 0$.
3. **Stringa**: consideriamo $w = 0^{q^2}$ con $q \geq p$. Così $|w| = q^2 \geq p$ e $w \in L$.
4. **Suddivisione**: $w = xyz$ con $|xy| \leq p$ e $|y| > 0$. Allora y è un blocco di 0, e $|y| \leq p$.
5. **Pompaggio**: Scegliamo $i = 2$ (per esempio, raddoppiamo la parte y). La nuova stringa è
$$xy^2z = 0^{q^2+|y|}.$$
Ora, la lunghezza $q^2+|y|$ è strettamente fra q^2 e $(q+1)^2$, infatti $(q+1)^2 = q^2+2q+1$. Siccome $|y| \leq p \leq q$, la differenza fra $q^2 + |y|$ e $(q + 1)^2$ è almeno $2q + 1 - |y| > 0$ (fino a $2q$ circa). Comunque, la lunghezza $q^2+|y|$ non coincide con nessun quadrato intero, *quindi* la stringa xy^2z non ha forma 0^{n^2} per qualche n, ed esce da L.
6. Contraddizione. L non è regolare.

2. *Considera il linguaggio*

$$L_2 = \{w \in \{0,1\}^* \mid w \textit{ contiene lo stesso numero di } 00 \textit{ e di } 11\}.$$

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 $w = 0^k1^k$, che appartiene ad L_2 ed è di lunghezza maggiore di k;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq k$;
- poiché $|xy| \leq k$, allora x e y sono entrambe contenute nella sequenza di 0. Inoltre, siccome $y \neq \emptyset$, abbiamo che $x = 0^q$ e $y = 0^p$ per qualche $q \geq 0$ e $p > 0$. z contiene la parte rimanente della stringa: $z = 0^{k-q-p}1^k$. Consideriamo l'esponente $i = 0$: la parola xy^0z ha la forma
$$xy^0z = xz = 0^q0^{k-q-p}1^k = 0^{k-p}1^k$$
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.

1. (8 punti) *Considera il linguaggio*

$$L = \{0^m1^n \mid m/n \textit{ è un numero intero}\}.$$

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
$$xy^0z = xz = 0^q0^{k+1-q-p}1^{k+1} = 0^{k+1-p}1^{k+1}.$$
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.

La chiusura per le operazioni regolari rende possibile anche capire se un certo linguaggio è regolare.

5. For languages A and B, let the *perfect shuffle* of A and B be the language

$$\{w \mid w = a_1 b_1 \cdots a_k b_k, \text{ where } a_1 \cdots a_k \in A \text{ and } b_1 \cdots b_k \in B, \text{ each } a_i, b_i \in \Sigma\}.$$

Show that the class of regular languages is closed under perfect shuffle.

Answer: Let $D_A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ and $D_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ be two DFAs that recognize A and B, respectively. Here, we shall construct a DFA $D = (Q, \Sigma, \delta, q, F)$ that recognizes the perfect shuffle of A and B.

The key idea is to design D to alternately switch from running D_A and running D_B after each character is read. Therefore, at any time, D needs to keep track of (i) the current states of D_A and D_B and (ii) whether the next character of the input string should be matched in D_A or in D_B. Then, when a character is read, depending on which DFA should match the character, D makes a move in the corresponding DFA accordingly. After the whole string is processed, if both DFAs are in the accept states, the input string is accepted; otherwise, the input string is rejected.

Formally, the DFA D can be defined as follows:

(a) $Q = Q_A \times Q_B \times \{A, B\}$, which keeps track of all possible current states of D_A and D_B, and which DFA to match.

(b) $q = (q_A, q_B, A)$, which states that D starts with D_A in q_A, D_B in q_B, and the next character read should be in D_A.

(c) $F = F_A \times F_B \times \{A\}$, which states that D accepts the string if both D_A and D_B are in accept states, and the next character read should be in D_A (i.e., last character was read in D_B).

(d) δ is as follows:

i. $\delta((x, y, A), a) = (\delta_A(x, a), y, B)$, which states that if current state of D_A is x, the current state of D_B is y, and the next character read is in D_A, then when a is read as the next character, we should change the current state of A to $\delta_A(x, a)$, while the current state of B is not changed, and the next character read will be in D_B.

ii. Similarly, $\delta((x, y, B), b) = (x, \delta_B(y, b), A)$.

1.55 The pumping lemma says that every regular language has a pumping length p, such that every string in the language can be pumped if it has length p or more. If p is a pumping length for language A, so is any length $p' \geq p$. The ***minimum pumping length*** for A is the smallest p that is a pumping length for A. For example, if $A = \mathtt{01}^*$, the minimum pumping length is 2. The reason is that the string $s = \mathtt{0}$ is in A and has length 1 yet s cannot be pumped, but any string in A of length 2 or more contains a 1 and hence can be pumped by dividing it so that $x = \mathtt{0}$, $y = \mathtt{1}$, and z is the rest. For each of the following languages, give the minimum pumping length and justify your answer.

A **a.** $\mathtt{0001}^*$	**f.** ε
A **b.** $\mathtt{0}^*\mathtt{1}^*$	**g.** $\mathtt{1}^*\mathtt{01}^*\mathtt{01}^*$
c. $\mathtt{001} \cup \mathtt{0}^*\mathtt{1}^*$	**h.** $\mathtt{10}(\mathtt{11}^*\mathtt{0})^*\mathtt{0}$
A **d.** $\mathtt{0}^*\mathtt{1}^*\mathtt{0}^*\mathtt{1}^* \cup \mathtt{10}^*\mathtt{1}$	**i.** $\mathtt{1011}$
e. $(\mathtt{01})^*$	**j.** Σ^*

Soluzioni e minime pumping length

a) $000\,1^*$

Le stringhe ammesse sono "000" seguita da zero o più '1'. - La stringa 000 (lunga 3) non è pompabile (se la riduciamo di un 0, usciremmo dal linguaggio). - Quindi, se volessimo $p \leq 3$, saremmo costretti a pompare anche 000. Contraddizione. - Invece, con $p = 4$, tutte le stringhe di lunghezza ≥ 4 sono $000\,1^k$ con $k \geq 1$, e si possono pompare prendendo il loop solo tra le 1.

$$\boxed{p = 4 \text{ (minimo)}}$$

b) $0^*\,1^*$

Qualunque stringa di almeno 1 simbolo (0 o 1) può essere pompata facilmente scegliendo come `y` un solo simbolo (0 o 1). Rimuoverlo o duplicarlo lascia comunque una forma "blocchi di 0 seguiti da blocchi di 1". Anche `0` e `1` (lunghezza 1) sono pompabili.

$$\boxed{p = 1 \text{ (minimo)}}$$

c) $001 \cup 0^*1^*$

Il linguaggio è l'unione della stringa "001" e di 0^*1^*. - Per 0^*1^*, come in (b), $p = 1$ funziona. - La stringa "001" (lunga 3) si può pompare: scegli ad es. $x = \texttt{0}, y = \texttt{0}, z = \texttt{1}$. Rimuovendo `0` si ottiene "01", che è in 0^*1^*; duplicandola si ottiene "0001", pure in 0^*1^*.

$$\boxed{p = 1 \text{ (minimo)}}$$

d) $0^*1^*0^*1^* \cup 1\,0^*1$

Il linguaggio è piuttosto grande: la prima parte ammette (0–1–0–1) in qualsiasi ordine di blocchi; la seconda parte è "1" seguito da qualche zero, poi "1". - Anche qui, sostanzialmente ogni stringa lunga almeno 1 si può pompare con un `y` monocarattere, conservando la forma. - Si verifica che `1` o `10` non danno problemi con $p = 1$.

$$\boxed{p = 1 \text{ (minimo)}}$$

e) $(01)^*$

Le stringhe sono concatenazioni di '01': ε, "01", "0101", "010101", ... - La stringa "01" di lunghezza 2 non è pompabile (se rimuovi un simbolo, non resta un multiplo di 2, e la forma si rompe). Quindi $p \neq 1$ e $p \neq 2$. - Con $p = 3$, la stringa di lunghezza 2 ('01') è esclusa (non deve essere pompata). La prima stringa ≥ 3 nel linguaggio è "0101" (lunghezza 4), che *è* pompabile scegliendo come 'y' il primo "01".

$$\boxed{p = 3 \text{ (minimo)}}$$

f) $\{\varepsilon\}$

Il linguaggio contiene solo la stringa vuota, di lunghezza 0. - Non esistono stringhe con lunghezza ≥ 1, dunque la condizione del lemma ("tutte le stringhe di lunghezza $\geq p$ si pompano") è triviale.

$$\boxed{p = 1 \text{ (minimo, o qualunque } p > 0)}$$

g) $1^*0\,1^*0\,1^*$

Serve esattamente **due** `0`, in quell'ordine, e un po' di `1` in giro. Alcune stringhe brevi, come "00" (lunghezza 2) e "001" (3) sono nel linguaggio e *non* pompabili se $p \leq 3$. Infatti, es. "00" non puoi rimuovere un `0` e restare con un solo zero. - Con $p = 4$, tutte le stringhe di lunghezza ≥ 4 contengono almeno i due zero e abbastanza `1` per potersi "pompizzare" nel blocco di `1`.

$$\boxed{p = 4 \text{ (minimo)}}$$

h) $10\,(11^*0)^*\,0$

Stringhe del tipo: inizio con "10", poi zero o più blocchi `11*0`, e infine un '0'. - La stringa più breve è "100", lunga 3. Essa non si può pompare con $p \leq 3$. Per es., se rimuovi un simbolo `0`, la stringa non termina più con '0'. - Con $p = 4$, le stringhe ≥ 4 si possono *sempre* pompare (di solito raddoppiando o rimuovendo `1` in quei blocchi).

$$\boxed{p = 4 \text{ (minimo)}}$$

i) $\{\texttt{1011}\}$

Linguaggio costituito da un'unica stringa "1011" di lunghezza 4. - Se $p \leq 4$, allora la stringa "1011" di lunghezza 4 *deve* poter essere pompata. Ma in un linguaggio di un solo elemento, pompare produce altre stringhe (es. rimuovere 1, o duplicarlo) che non esistono nel linguaggio. Contraddizione. - Se $p = 5$, allora non esistono stringhe di lunghezza ≥ 5 in L, quindi la condizione è vacuamente vera.

$$\boxed{p = 5 \text{ (minimo)}}$$