

# Esercizio 1 - L regolare

<sup>A</sup>1.44 Let  $B$  and  $C$  be languages over  $\Sigma = \{0, 1\}$ . Define

$$B \stackrel{1}{\leftarrow} C = \{w \in B \mid \text{for some } y \in C, \text{ strings } w \text{ and } y \text{ contain equal numbers of 1s}\}.$$

Show that the class of regular languages is closed under the  $\stackrel{1}{\leftarrow}$  operation.

## APPROCCIO 1: Modifica diretta delle transizioni

Dato  $B$  con DFA  $(Q_B, \Sigma, \delta_B, q_{0B}, F_B)$  e  $C$  con DFA  $(Q_C, \Sigma, \delta_C, q_{0C}, F_C)$ .

Costruisco  $A' = (Q', \Sigma, \delta', q'_0, F')$  dove:

- $Q' = Q_B \times Q_C$

- $q'_0 = (q_{0B}, q_{0C})$

- **Transizioni modificate:**

$$\delta'((q_B, q_C), a) = \{(\delta_B(q_B, 0), q_C) \quad \text{se } a = 0 \quad (\delta_B(q_B, 1), \delta_C(q_C, 1)) \quad \text{se } a = 1\}$$

- $F' = (q_B, q_C) \mid q_B \in F_B \text{ e } q_C \in F_C$

**Idea:** Gli 0 non modificano il "tracker" di  $C$ , i 1 fanno avanzare parallelamente entrambi gli automi.

---

## APPROCCIO 2: Con $\epsilon$ -transizioni (migliore)

Costruisco  $A' = (Q', \Sigma \cup \epsilon, \delta', q'_0, F')$  dove:

- $Q' = Q_B \cup Q_C \cup q_{\text{new}}$

- $q'_0 = q_{0B}$

- **Transizioni:**

- $\delta'(q, 0) = \delta_B(q, 0)$  per  $q \in Q_B$

- $\delta'(q, 1) = q_{\text{new}}$  per  $q \in Q_B$

- $\delta'(q_{\text{new}}, \epsilon) = q_{0C}$

- $\delta'(q, a) = \delta_C(q, a)$  per  $q \in Q_C, a \in 0, 1$

- $\delta'(q, \epsilon) = Q_B$  per  $q \in F_C$

- $F' = F_B$

## Logica:

1. Segui  $B$  normalmente su 0
2. Su 1, salta con  $\epsilon$  al simulatore di  $C$
3. Il simulatore verifica se esiste un cammino in  $C$  con lo stesso numero di 1
4. Se  $C$  accetta, torna con  $\epsilon$  a tutti gli stati di  $B$
5. Accetta solo se raggiungi uno stato finale di  $B$

**Correttezza:**  $w$  è accettata sse  $w \in B$  e  $\exists y \in C$  s.t.  $\#_1(w) = \#_1(y)$ .

L'approccio 2 è più elegante perché "separa" la verifica di  $B$  dalla verifica dei conteggi di  $C$ .

**1.44** Let  $M_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$  and  $M_C = (Q_C, \Sigma, \delta_C, q_C, F_C)$  be DFAs recognizing  $B$  and  $C$ , respectively. Construct NFA  $M = (Q, \Sigma, \delta, q_0, F)$  that recognizes  $B \stackrel{1}{\leftarrow} C$  as follows. To decide whether its input  $w$  is in  $B \stackrel{1}{\leftarrow} C$ , the machine  $M$  checks that  $w \in B$ , and in parallel nondeterministically guesses a string  $y$  that contains the same number of 1s as contained in  $w$  and checks that  $y \in C$ .

1.  $Q = Q_B \times Q_C$ .
2. For  $(q, r) \in Q$  and  $a \in \Sigma_\epsilon$ , define

$$\delta((q, r), a) = \begin{cases} \{(\delta_B(q, 0), r)\} & \text{if } a = 0 \\ \{(\delta_B(q, 1), \delta_C(r, 1))\} & \text{if } a = 1 \\ \{(q, \delta_C(r, 0))\} & \text{if } a = \epsilon. \end{cases}$$

3.  $q_0 = (q_B, q_C)$ .
4.  $F = F_B \times F_C$ .

## Spiegazione Claudica & Milo

Brillante! Sì, questa è un'osservazione chiave per l'approccio con ER.

### L'idea del "wrap":

Se  $C$  ha ER  $r_C$ , posso costruire l'ER per "tutte le stringhe con stesso numero di 1 di qualche stringa in  $C$ " sostituendo ogni simbolo 1 in  $r_C$  con  $0^*10^*$ .

**Formalmente:** Definisco la trasformazione  $\phi : \text{ER} \rightarrow \text{ER}$  che:

- $\phi(0) = 0^*$
- $\phi(1) = 0^*10^*$
- $\phi(\epsilon) = 0^*$
- $\phi(r_1 + r_2) = \phi(r_1) + \phi(r_2)$
- $\phi(r_1 \cdot r_2) = \phi(r_1) \cdot \phi(r_2)$
- $\phi(r^*) = (\phi(r))^*$

**Proprietà cruciale:** Se  $w \in L(\phi(r_C))$ , allora  $\#_1(w) = \#_1(y)$  per qualche  $y \in L(r_C)$ .

**Quindi la soluzione diventa:**

$$B \stackrel{1}{\leftarrow} C = L(r_B) \cap L(\phi(r_C))$$

**Esempio:** Se  $C = 01, 10$  ha ER  $r_C = 01 + 10$ , allora:

$$\phi(r_C) = \phi(0)\phi(1) + \phi(1)\phi(0) = 0^* \cdot 0^*10^* + 0^*10^* \cdot 0^* = 0^*10^* + 0^*10^*$$

Questo genera tutte le stringhe con esattamente 1 occorrenza di 1, che è corretto!

**ER finale:**

$$r_{B \stackrel{1}{\leftarrow} C} = r_B \cap \phi(r_C)$$

Geniale! È molto più pulito del mio approccio precedente.

## Esercizio 2 - Regolare

8. (Bonus Question.) If  $A$  is a language, let  $A_{\frac{1}{2}}$  be the set of all first halves of strings in  $A$ , so that

$$A_{\frac{1}{2}} = \{x \mid \text{for some } y, |x| = |y| \text{ and } xy \in A\}.$$

Show that if  $A$  is regular, then so is  $A_{\frac{1}{2}}$ .

**Dimostrazione che  $A_{1/2}$  è regolare quando  $A$  è regolare**

Sia  $A$  un linguaggio regolare. Esiste quindi un DFA  $M = (Q, \Sigma, \delta, q_0, F)$  che riconosce  $A$ .

Costruiamo un NFA  $N$  che riconosce  $A_{1/2}$ .

**Costruzione dell'NFA  $N = (Q, \Sigma, \delta', q_0, F)$ :**

- **Stati:**  $Q$  (gli stessi di  $M$ )
- **Stato iniziale:**  $q_0$
- **Stati finali:**  $F$
- **Funzione di transizione:**  $\delta'(q, a) = \{\delta(\delta(q, a), b) \mid b \in \Sigma\}$

**Intuizione:** Quando  $N$  legge il simbolo  $a$  dalla stringa di input  $x$ , "indovina" non deterministicamente quale simbolo  $b$  occuperebbe la posizione corrispondente in  $y$ , e simula l'avanzamento di  $M$  come se avesse processato la coppia  $ab$ .

**Correttezza:**

( $\Rightarrow$ ) Se  $x \in A_{1/2}$ , allora esiste  $y$  con  $|x| = |y|$  tale che  $xy \in A$ . Sia  $x = a_1a_2\dots a_n$  e  $y = b_1b_2\dots b_n$ . Poiché  $xy \in A$ , esiste una computazione accettante di  $M$ :  $q_0 \xrightarrow{a_1} q_1 \xrightarrow{b_1} q_1' \xrightarrow{a_2} q_2 \xrightarrow{b_2} q_2' \xrightarrow{a_3} \dots \xrightarrow{a_n} q_n \xrightarrow{b_n} q_f \in F$

In  $N$ , possiamo seguire la computazione:  $q_0 \xrightarrow{a_1} \delta(q_1, b_1) = q_1' \xrightarrow{a_2} \delta(q_2, b_2) = q_2' \xrightarrow{a_3} \dots \xrightarrow{a_n} \delta(q_n, b_n) = q_f \in F$

Quindi  $x$  è accettata da  $N$ .

( $\Leftarrow$ ) Se  $x$  è accettata da  $N$ , allora  $x \in A_{1/2}$ . Se  $N$  accetta  $x = a_1 a_2 \dots a_n$ , esiste una computazione:  $q_0 \xrightarrow{\{a_1\}} q_1' \xrightarrow{\{a_2\}} q_2' \xrightarrow{\{a_3\}} \dots \xrightarrow{\{a_n\}} q_f \in F$

dove  $q_i' \in \delta'(q_{i-1}', a_i)$  per qualche sequenza di stati intermedi.

Per definizione di  $\delta'$ , ogni transizione  $q_{i-1}' \xrightarrow{\{a_i\}} q_i'$  significa che esistono stati intermedi  $q_i$  e simboli  $b_i \in \Sigma$  tali che:  $q_i = \delta(q_{i-1}', a_i)$  e  $q_i' = \delta(q_i, b_i)$

Ricostruendo la sequenza  $y = b_1 b_2 \dots b_n$ , otteniamo che  $M$  ha la computazione:  $q_0 \xrightarrow{\{a_1\}} q_1 \xrightarrow{\{b_1\}} q_1' \xrightarrow{\{a_2\}} q_2 \xrightarrow{\{b_2\}} q_2' \xrightarrow{\{a_3\}} \dots \xrightarrow{\{a_n\}} q_n \xrightarrow{\{b_n\}} q_f \in F$

Quindi  $xy \in A$  con  $|x| = |y|$ , il che implica  $x \in A_{1/2}$ .

**Conclusione:**  $N$  riconosce esattamente  $A_{1/2}$ , e poiché  $N$  è un NFA (quindi rappresenta un linguaggio regolare),  $A_{1/2}$  è regolare.  $\square$

### Idea originale

Ma se io: prendo l'intero automa (oppure lo raddoppio), uno dei due lo rovescio sottosopra (a destra, sinistra e tutto) e incrocio tutti gli stati, in maniera tale che vado avanti in una direzione e nondeterministicamente andiamo avanti anche nell'altra

### Esatto! Costruzione più elegante tramite prodotto cartesiano

Hai ragione, c'è un approccio molto più diretto e pulito.

### Costruzione dell'NFA $N = (Q \times Q, \Sigma, \delta', I, F)$ :

- **Stati:**  $Q \times Q$  (prodotto cartesiano)
- **Stati iniziali:**  $I = \{(q_0, q_f) \mid q_f \in F\}$  (nondeterministicamente scegliamo da quale stato finale "tornare indietro")
- **Stati finali:**  $F' = F \times \{q_0\}$
- **Transizione:**  $\delta'((p, q), a) = \{(\delta(p, a), r) \mid \delta(r, b) = q \text{ per qualche } b \in \Sigma\}$

### Intuizione della costruzione:

- La prima componente ( $p$ ) simula la lettura di  $x$  andando "avanti" dall'inizio
- La seconda componente ( $q$ ) simula il "ritorno indietro" dalla fine, indovinando come si sarebbe potuti arrivare lì leggendo  $y$

### Meccanismo:

1. Iniziamo da  $(q_0, q_f)$  per qualche stato finale  $q_f$ , immaginando di essere alla fine di  $xy$
2. Leggendo il simbolo  $a$  da  $x$ , avanziamo a  $\delta(q_0, a)$  nella prima componente
3. Simultaneamente, nella seconda componente "torniamo indietro": da  $q_f$  andiamo a tutti gli  $r$  tali che  $\delta(r, b) = q_f$  per qualche  $b$

**Correttezza:** Una stringa  $x$  è accettata sse esiste una computazione che parte da  $(q_0, q_f)$  e termina in  $(q_f, q_0)$ , meaning:

- La prima componente ha simulato  $x$ :  $q_0 \xrightarrow{x} q_f \in F$
- La seconda componente è "tornata" dal finale all'inizio:  $q_f \xrightarrow{y} q_0$  per qualche  $y$  con  $|y| = |x|$
- Quindi  $xy \in A$

Questa costruzione cattura perfettamente l'idea di "andare avanti e indietro simultaneamente" ed è molto più naturale della mia prima proposta. Brava intuizione!

Non intendo esattamente cartesiano perché vorrei andare avanti in entrambe le direzioni, usando coppie con il loro opposto, che facciamo?

### **Esatto! Costruzione più elegante tramite prodotto cartesiano**

Hai ragione, c'è un approccio molto più diretto e pulito.

**Costruzione dell'NFA  $N = (Q \times Q, \Sigma, \delta', I, F')$ :**

- **Stati:**  $Q \times Q$  (prodotto cartesiano)
- **Stati iniziali:**  $I = \{(q_0, q_f) \mid q_f \in F\}$  (nondeterministicamente scegliamo da quale stato finale "tornare indietro")
- **Stati finali:**  $F' = F \times \{q_0\}$
- **Transizione:**  $\delta'((p, q), a) = \{(\delta(p, a), r) \mid \delta(r, b) = q \text{ per qualche } b \in \Sigma\}$

**Intuizione della costruzione:**

- La prima componente ( $p$ ) simula la lettura di  $x$  andando "avanti" dall'inizio
- La seconda componente ( $q$ ) simula il "ritorno indietro" dalla fine, indovinando come si sarebbe potuti arrivare lì leggendo  $y$

**Meccanismo:**

1. Iniziamo da  $(q_0, q_f)$  per qualche stato finale  $q_f$ , immaginando di essere alla fine di  $xy$
2. Leggendo il simbolo  $a$  da  $x$ , avanziamo a  $\delta(q_0, a)$  nella prima componente
3. Simultaneamente, nella seconda componente "torniamo indietro": da  $q_f$  andiamo a tutti gli  $r$  tali che  $\delta(r, b) = q_f$  per qualche  $b$

**Correttezza:** Una stringa  $x$  è accettata sse esiste una computazione che parte da  $(q_0, q_f)$  e termina in  $(q_f, q_0)$ , meaning:

- La prima componente ha simulato  $x$ :  $q_0 \xrightarrow{x} q_f \in F$
- La seconda componente è "tornata" dal finale all'inizio:  $q_f \xrightarrow{y} q_0$  per qualche  $y$  con  $|y| = |x|$
- Quindi  $xy \in A$

Questa costruzione cattura perfettamente l'idea di "andare avanti e indietro simultaneamente" ed è molto più naturale della mia prima proposta. Brava intuizione!

## Esercizio 3 - Regolare

**1.38** An *all*-NFA  $M$  is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  that accepts  $x \in \Sigma^*$  if *every* possible state that  $M$  could be in after reading input  $x$  is a state from  $F$ . Note, in contrast, that an ordinary NFA accepts a string if *some* state among these possible states is an accept state. Prove that all-NFAs recognize the class of regular languages.

### Dimostrazione che gli all-NFA riconoscono la classe dei linguaggi regolari

La dimostrazione procede in due direzioni per stabilire l'equivalenza tra all-NFA e linguaggi regolari.

### Direzione 1: Ogni linguaggio regolare è riconosciuto da un all-NFA

Poiché ogni linguaggio regolare ha un DFA che lo riconosce, è sufficiente dimostrare che ogni DFA è equivalente a un all-NFA.

Dato un DFA  $D = (Q, \Sigma, \delta, q_0, F)$ , possiamo considerarlo come un all-NFA dove:

- La funzione di transizione  $\delta(q,a)$  restituisce un singleton  $\{q'\}$  per ogni coppia  $(q,a)$
- Non ci sono  $\epsilon$ -transizioni
- Esiste una sola computazione possibile per ogni stringa di input

La condizione di accettazione dell'all-NFA ("ogni possibile stato raggiungibile deve essere finale") è trivialmente soddisfatta quando c'è un solo stato raggiungibile, che coincide con la condizione di accettazione del DFA.

### Direzione 2: Ogni linguaggio riconosciuto da un all-NFA è regolare

Costruiamo un DFA equivalente a un dato all- $\epsilon$ -NFA  $N = (Q_n, \Sigma, q_0, \delta_n, F_n)$  mediante il seguente algoritmo:

#### Algoritmo di conversione:

Input: all- $\epsilon$ -NFA  $N = (Q_n, \Sigma, q_0, \delta_n, F_n)$

Output: DFA  $D = (Q_D, \Sigma, S_0, \delta_D, F_D)$

$S_0 \leftarrow \text{Eclose}(q_0)$

$Q_D \leftarrow \{S_0\}$

$F_D \leftarrow \{S_0\}$  se  $S_0 \subseteq F_n$ , altrimenti  $F_D \leftarrow \emptyset$

```

while Q_D contiene stati senza transizioni uscenti do
  Scegli S ∈ Q_D senza transizioni uscenti
  for each a ∈ Σ do
    S' ← ∅
    for each q ∈ S do
      S' ← S' ∪ δ_n(q,a)
    end for
    S' ← Eclose(S')
    Q_D ← Q_D ∪ {S'}

    if S' ⊆ F_n then
      F_D ← F_D ∪ {S'}
    end if

    δ_D(S,a) ← S'
  end for
end while

```

**Elemento chiave della costruzione:** Lo stato S del DFA è finale se e solo se  $S \subseteq F_n$ , ovvero tutti gli stati di N contenuti in S sono stati finali. Questo riflette precisamente la semantica dell'all-NFA: accettare quando tutti i possibili stati raggiungibili sono finali.

**Correttezza:** L'algoritmo è analogo alla costruzione standard per sottoinsiemi (NFA→DFA), con la cruciale differenza nella definizione degli stati finali. Mentre nella costruzione standard uno stato S è finale se  $S \cap F \neq \emptyset$  (intersezione non vuota), qui S è finale se  $S \subseteq F$  (inclusione), catturando la semantica universale dell'all-NFA.

**Conclusione:** Poiché abbiamo mostrato entrambe le direzioni, gli all-NFA riconoscono esattamente la classe dei linguaggi regolari. □

## Bonus - avoids

4. We define the *avoids* operation for languages  $A$  and  $B$  to be

$$A \text{ avoids } B = \{w \mid w \in A \text{ and } w \text{ doesn't contain any string in } B \text{ as a substring}\}.$$

Prove that the class of regular languages is closed under the *avoids* operation.

*Solution.* The definition of  $A \text{ avoids } B$  may be restated equivalently as the set difference between  $A$  and  $\{w \mid w \text{ contains a string in } B \text{ as a substring}\}$ . The set difference between two languages  $C$  and  $D$ , denoted  $C \setminus D$ , equals  $C \cap \overline{D}$ . If both  $C$  and  $D$  are regular, then  $C \setminus D$  is also regular, as the class of regular languages is closed under intersection and complementation.

For a regular language  $B$ , the set  $\{w \mid w \text{ contains a string in } B \text{ as a substring}\}$  can be expressed as  $\Sigma^* R_B \Sigma^*$ , where  $\Sigma$  is the alphabet and  $R_B$  is a regular expression for  $B$ , and hence is also regular. So, the class of regular languages is closed under the *avoids* operation.  $\square$

## Esercizio 4 - L non regolare

1.53 Let  $\Sigma = \{0, 1, +, =\}$  and

$$ADD = \{x=y+z \mid x, y, z \text{ are binary integers, and } x \text{ is the sum of } y \text{ and } z\}.$$

Show that  $ADD$  is not regular.

## Dimostrazione formale che ADD non è regolare

Dalle annotazioni del professore, formalizziamo l'approccio utilizzando il pumping lemma con attenzione alla struttura del problema.

### Setup del Pumping Lemma

Supponiamo per assurdo che  $ADD$  sia regolare. Allora esiste  $p > 0$  (pumping length) tale che ogni stringa  $w \in ADD$  con  $|w| \geq p$  ammette una decomposizione  $w = xyz$  con:

1.  $|xy| \leq p$
2.  $|y| > 0$
3.  $\forall i \geq 0, xy^i z \in ADD$

### Scelta strategica della stringa

Consideriamo  $w = 0^p + 0^p = 10^p \in ADD$ .

**Verifica che  $w \in ADD$ :**

- $0^p$  rappresenta il numero binario 0
- $10^p$  rappresenta il numero binario  $2^p$
- Verifica:  $0 + 0 = 0 \neq 2^p$



Riproviamo con  $w = 1 + 1 = 10$ . Ma  $|w| = 6 < p$  per  $p$  abbastanza grande.

**Scelta corretta:**  $w = 10^{p-1} + 10^{p-1} = 10^p$

## Analisi della decomposizione

Dato  $|w| = 3p + 3 > p$ , esiste decomposizione  $w = xyz$  con le proprietà del pumping lemma.

**Caso critico:** Poiché  $|xy| \leq p$ , la sottostringa  $xy$  è contenuta nel primo operando  $10^{p-1}$ .

Due sottocasi:

### Sottocaso 1: $y = 0^k$ ( $k > 0$ )

- $xy^2z = 10^{p-1+k} + 10^{p-1} = 10^p$
- Condizione di validità:  $2^{p-1+k} + 2^{p-1} = 2^p$
- Semplificando:  $2^{p-1}(2^k + 1) = 2^{p-1} \cdot 2$
- Quindi:  $2^k + 1 = 2 \Rightarrow 2^k = 1 \Rightarrow k = 0$

**Contraddizione** con  $k > 0$ .

### Sottocaso 2: $y$ contiene il bit '1' iniziale

- $xy^2z$  produrrebbe  $11... + 10^{p-1} = 10^p$
- Ma  $11...$  non è una rappresentazione binaria valida di un numero

**Contraddizione** con la sintassi dei numeri binari.

## Caso alternativo: $y$ attraversa il simbolo '+'

Se  $y$  contiene parte del primo operando e il simbolo '+', allora  $xy^2z$  avrebbe la forma:

- $...+ + \text{resto}...$

Questo viola la sintassi del linguaggio ADD.

## Conclusione

In tutti i casi possibili per la decomposizione  $w = xyz$ , il pumping  $xy^2z$  porta a:

- Violazione delle proprietà aritmetiche (Sottocaso 1)
- Violazione della sintassi binaria (Sottocaso 2)
- Violazione della sintassi del linguaggio (caso con '+')

Quindi il pumping lemma è violato, e **ADD non è regolare**.  $\square$

## Pumping Lemma (formulazione precisa)

Se  $L$  è regolare, allora  $\exists p \geq 1$  tale che  $\forall w \in L$  con  $|w| \geq p$ ,  $w$  può essere decomposta come  $w = xyz$  con:

1.  $|xy| \leq p$
2.  $|y| > 0$
3.  $\forall i \geq 0, xy^i z \in L$

## Scelta della stringa corretta

Scegliamo  $w = 10^{p-1} + 10^{p-1} = 10^p \in \text{ADD}$ .

**Verifica lunghezza:**  $|w| = |10^{p-1}| + |+| + |10^{p-1}| + |=| + |10^p| = p + 1 + p + 1 + (p+1) = 3p + 3$

Chiaramente  $|w| = 3p + 3 \geq p \checkmark$  (la condizione è soddisfatta)

## Resto della dimostrazione

Il vincolo  $|xy| \leq p$  garantisce che  $xy$  sia contenuto nel primo operando  $10^{p-1}$ , e il pumping su  $y = 0^k$  porta alla contraddizione aritmetica:

$$2^{p-1+k} + 2^{p-1} = 2^p \Rightarrow 2^k = 1 \Rightarrow k = 0$$

Ma  $|y| > 0$  richiede  $k > 0$ . **Contraddizione.**

Quindi  $\text{ADD}$  non è regolare.  $\square$

**Nota:** La condizione è  $|w| \geq p$ , non strettamente maggiore, e la nostra scelta  $w$  con  $|w| = 3p + 3$  soddisfa ampiamente questo requisito.

Consider the alphabet  $\Sigma = \{0, 1, +, =\}$  and the language  $\text{ADD} = \{x = y + z \mid x, y \text{ are binary integers } x \text{ is the sum of } y \text{ and } z\}$ .

This is the (informal) solution I came up with (not sure about its correctness). Using the pumping lemma:

Take  $s = 1^p = 0 + 1^p$ ,  $s \in \text{ADD}$ .  $s$  can be divided into three parts so that  $s = xyz$ . The pumping lemma says that  $|xy| \leq p$  and  $|y| > 0$  so  $y$  must be in the first  $p$   $1$ 's. Choose  $i = 0$ . Then the equation is false, so the language  $\text{ADD}$  is not regular.  $\square$

**Solution** Proof by contradiction. Assume  $ADD$  is a regular language. By the Pumping Lemma, there is a constant  $p$  associated with  $ADD$ .

1. Choose the string  $s$  to be  $1^p = 0^p + 1^p$ . Note that  $s \in ADD$  because  $1^p$  is the sum of  $0^p$  and  $1^p$ , and  $|s| = 3p + 2 \geq p$ .
2. Choose the partition  $s = xyz$  such that  $x$  is  $\epsilon$ ,  $y$  is  $1^p$ , and  $z$  is  $0^p + 1^p$ . Note that  $|y| = p \geq p$ .
3. In any possible division  $y = uvw$ , the string  $v$  must be  $1^m$ , where  $0 < m \leq p$ .
4. Choose  $i = 2$ . Then the string  $xuv^2wz$  is  $1^{m+p} = 0^p + 1^p$ . Because  $m > 0$ ,  $1^{m+p}$  is not the sum of  $0^p$  and  $1^p$ . Thus  $xuv^2wz \notin ADD$ . This is a contradiction. Therefore  $ADD$  is not a regular language.

## Esercizio 5 - L context-free

- 2.44 If  $A$  and  $B$  are languages, define  $A \diamond B = \{xy \mid x \in A \text{ and } y \in B \text{ and } |x| = |y|\}$ . Show that if  $A$  and  $B$  are regular languages, then  $A \diamond B$  is a CFL.

**Solution** Let  $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$  be a DFA recognizing  $A$  and  $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$  be a DFA recognizing  $B$ . We construct a PDA  $P$  recognizing  $A \diamond B$ . The idea behind the construction is the following. Given a word  $w$  we simulate  $M_1$  on a prefix  $x$  of  $w$  and store the length of  $x$  in the stack (its height). We then run  $M_2$  on the rest of the string  $y$  (note  $w = xy$ ) and check that  $|y| = |x|$ , which is essentially done by popping a symbol from the stack for every symbol of  $y$  read. Note that we need to non-deterministically guess where to break  $w$  into  $x$  and  $y$ . Formally  $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ , where:

- $Q = Q_1 \cup Q_2 \cup \{q_0, q_f\}$  ( $Q_1$  and  $Q_2$  can be assumed to be disjoint).
- $\Gamma = \{0, \$\}$ .
- - $(q_{01}, \$) \in (q_0, \epsilon, \epsilon)$ .
  - $(\delta_1(q, a), 0) \in \delta(q, a, \epsilon)$  iff  $q \in Q_1$ .
  - $(\delta_2(q, a), \epsilon) \in \delta(q, a, 0)$  iff  $q \in Q_2$ .
  - $(q_f, \epsilon) \in \delta(q, \epsilon, \$)$  iff  $q \in F_2$ .
  - $(q_{02}, \epsilon) \in \delta(q, \epsilon, \epsilon)$  iff  $q \in F_1$ .
- $q_0$ .
- $F = \{q_f\}$ .

A string  $w \in A \diamond B$  iff  $w = xy$ ,  $|x| = |y|$ ,  $x \in A$  and  $y \in B$  iff  $w = xy$ ,  $|x| = |y|$ ,  $q_{01} \xrightarrow{x} q_1$  where  $q_1 \in F_1$  and  $q_{02} \xrightarrow{y} q_2$  where  $q_2 \in F_2$  (1). It is easy to see from the construction that  $(q_0, \epsilon) \xrightarrow{\epsilon} (q_{01}, \$)$ ,  $(q_{01}, \$) \xrightarrow{x} (q_1, 0^{|x|} \$)$ ,  $(q_1, 0^{|x|} \$) \xrightarrow{\epsilon} (q_{02}, 0^{|x|} \$)$ ,  $(q_{02}, 0^{|x|} \$) \xrightarrow{y} (q_2, \$)$  and  $(q_2, \$) \xrightarrow{\epsilon} (q_f, \epsilon)$  (\*). On the other hand any string which is accepted by  $P$  has to be of the form  $xy$  where  $|x| = |y|$  and has to go through a sequence of steps of the above form. Hence  $w \in L(P)$  iff  $w = xy$ ,  $|x| = |y|$  and there is a sequence of transitions of the form (\*) iff (1). Therefore  $L(P) = L(A \diamond B)$ .

## Esercizio 6 Custom - L context-free

Una fabbrica produce sequenze composite inserendo tra ogni coppia di caratteri di una stringa  $x \in A$  una sottostringa di riempimento  $z$ , dove la lunghezza totale del riempimento deve uguagliare la lunghezza originale di  $x$ .

Definisci  $\text{INTERLEAVE}(A) = \{x_1 z_1 x_2 z_2 \dots x_n z_n x_{n+1} \mid x_1 x_2 \dots x_{n+1} \in A, z_1 z_2 \dots z_n \in \{a, b\}^*, |z_1 z_2 \dots z_n| = n\}$ .  
 Dimostra che se  $A$  è regolare, allora  $\text{INTERLEAVE}(A)$  è context-free

**Dimostrazione formale che  $\text{INTERLEAVE}(A)$  è context-free**

Recall:  $\text{INTERLEAVE}(A) = \{x_1 z_1 x_2 z_2 \dots x_n z_n x_{n+1} \mid x_1 x_2 \dots x_{n+1} \in A, z_1 z_2 \dots z_n \in \{a, b\}^*, |z_1 z_2 \dots z_n| = n\}$ .

## Metodo 1: Costruzione con Grammatica Context-Free

Poiché  $A$  è regolare, esiste una grammatica regolare (Type-3)  $G_A = (V_A, \Sigma, P_A, S_A)$  per  $A$ . Costruiamo una grammatica context-free  $G = (V, \Sigma \cup \{a,b\}, P, S)$  per  $\text{INTERLEAVE}(A)$ .

**Variabili:**  $V = V_A \cup \{S, C, D\}$

- $S$ : simbolo iniziale
- $C$ : genera stringhe di riempimento con contatore
- $D$ : gestisce il bilanciamento lunghezza/inserimenti

**Produzioni:**

**1. Inizializzazione:**  $S \rightarrow \langle S_A, 0 \rangle$

**2. Simulazione di  $A$  con contatori:** Per ogni produzione  $A \rightarrow aB$  in  $P_A$ :  $\langle A, k \rangle \rightarrow a\langle B, k \rangle$   
 $\langle A, k \rangle \rightarrow aC\langle B, k+1 \rangle$

**3. Generazione stringhe di riempimento:**  $C \rightarrow aC \mid bC \mid D$

**4. Controllo bilanciamento:**  $D \rightarrow \varepsilon$  (solo se il contatore corrisponde alla lunghezza generata)

**5. Terminazione:** Per ogni  $A \in F_A$  (stati finali di  $A$ ):  $\langle A, k \rangle \rightarrow \varepsilon$  (se  $k = 0$ , nessun inserimento rimanente)

**Tecnica di implementazione:** Utilizziamo una grammatica context-sensitive come passaggio intermedio, poi applichiamo la normalizzazione per ottenere una equivalente context-free attraverso l'introduzione di variabili ausiliarie che mantengono informazioni sui contatori.

## Metodo 2: Costruzione diretta di PDA

Sia  $M_A = (Q_A, \Sigma, \delta_A, q_0, F_A)$  un DFA per  $A$ . Costruiamo  $P = (Q, \Sigma \cup \{a,b\}, \Gamma, \delta, q_0', Z_0, F)$ .

**Stati:**  $Q = Q_A \times \{\text{read}, \text{fill}\} \times \mathbb{N}$

- Prima componente: stato corrente di  $M_A$
- Seconda componente: modalità (lettura carattere o riempimento)
- Terza componente: contatore inserimenti rimanenti

**Alfabeto di stack:**  $\Gamma = \{Z_0, X, Y\}$

- $X$ : marca un inserimento da processare
- $Y$ : conta lunghezza stringa di riempimento corrente

**Transizioni:**

**1. Inizializzazione:**  $\delta((q_0', \text{read}, 0), \varepsilon, Z_0) = \{(q_0, \text{read}, 0), Z_0\}$

**2. Lettura caratteri da A:** Per ogni  $a \in \Sigma$ ,  $q \in Q\_A$ :  $\delta((q, \text{read}, k), a, \gamma) = \{((\delta\_A(q, a), \text{read}, k), \gamma), ((\delta\_A(q, a), \text{fill}, k+1), X\gamma)\}$

**3. Modalità riempimento:**  $\delta((q, \text{fill}, k), a, \gamma) = \{((q, \text{fill}, k), Y\gamma)\}$  per  $a, b \in \{a, b\}$   $\delta((q, \text{fill}, k), \epsilon, X) = \{((q, \text{read}, k-1), \epsilon)\}$  (quando  $|\text{riempimento}| = 1$ )

**4. Controllo bilanciamento:** Utilizziamo lo stack per verificare che ogni X (inserimento) corrisponda esattamente a una Y (carattere di riempimento).

**5. Accettazione:** Stati finali  $F = \{(q, \text{read}, 0) \mid q \in F\_A\}$

**Invariante del PDA:** Lo stack mantiene la proprietà che il numero di X non processati uguaglia il numero di inserimenti rimanenti da bilanciare.

## Correttezza

**Completezza:** Se  $w \in \text{INTERLEAVE}(A)$ , allora  $w = x_1 z_1 x_2 z_2 \dots x_n z_n x_{n+1}$  dove  $x_1 \dots x_{n+1} \in A$  e  $|z_1 \dots z_n| = n$ . Il PDA può:

1. Simulare il riconoscimento di  $x_1 \dots x_{n+1}$  in  $M\_A$
2. Per ogni transizione, scegliere nondeterministicamente se inserire riempimento
3. Verificare che la lunghezza totale del riempimento = numero inserimenti

**Soundness:** Se il PDA accetta  $w$ , allora per costruzione  $w$  ha la forma richiesta e soddisfa tutti i vincoli di  $\text{INTERLEAVE}(A)$ .

**Conclusione:**  $\text{INTERLEAVE}(A)$  è context-free.  $\square$

## Alternativa

**Formalizzazione dell'idea del professore per  $\text{INTERLEAVE}(A)$**

## Costruzione della Grammatica Context-Free

Basandoci sul suggerimento delle note, costruiamo  $G = (V, \Sigma \cup \{a, b\}, P, S)$  per  $\text{INTERLEAVE}(A)$ .

Sia  $A$  regolare con DFA  $M\_A = (Q, \Sigma, \delta, q_0, F)$ .

**Variabili:**  $V = \{S\} \cup \{[q] \mid q \in Q\} \cup \{Z\}$

**Idea chiave:** Ogni variabile  $[q]$  rappresenta "genera il resto della stringa partendo dallo stato  $q$  di  $M\_A$ , con possibili inserimenti".

## Produzioni

**1. Inizializzazione:**

$S \rightarrow [q_0]$

**2. Simulazione DFA con inserimenti opzionali:** Per ogni stato  $q \in Q$  e simbolo  $a \in \Sigma$ :

$[q] \rightarrow a[\delta(q, a)]$  (leggi carattere senza inserimento)  
 $[q] \rightarrow aZ[\delta(q, a)]$  (leggi carattere con inserimento)

**3. Generazione stringhe di riempimento:**

$Z \rightarrow aZ \mid bZ \mid \varepsilon$  (genera  $z_i \in \{a, b\}^*$  di lunghezza  $\geq 0$ )

**4. Terminazione:** Per ogni stato finale  $f \in F$ :

$[f] \rightarrow \varepsilon$  (termina quando A è accettata)

## Correttezza della Costruzione

**Derivazione tipo:**

$S \Rightarrow [q_0]$   
 $\Rightarrow x_1 Z[q_1]$  (primo carattere + inserimento)  
 $\Rightarrow x_1 z_1 [q_1]$  (espansione primo riempimento)  
 $\Rightarrow x_1 z_1 x_2 Z[q_2]$  (secondo carattere + inserimento)  
 $\Rightarrow x_1 z_1 x_2 z_2 [q_2]$  (espansione secondo riempimento)  
 $\Rightarrow \dots$   
 $\Rightarrow x_1 z_1 x_2 z_2 \dots x_n z_n [q_n]$   
 $\Rightarrow x_1 z_1 x_2 z_2 \dots x_n z_n x_{n+1} [f]$  (ultimo carattere senza inserimento)  
 $\Rightarrow x_1 z_1 x_2 z_2 \dots x_n z_n x_{n+1}$  (terminazione con  $f \in F$ )

**Proprietà garantite:**

1.  $x_1 x_2 \dots x_{n+1} \in A$ : La simulazione del DFA garantisce che  $q_0 \xrightarrow{\{x_1 \dots x_{n+1}\}} f \in F$
2.  $z_1 \dots z_n \in \{a, b\}^*$ : Ogni Z genera stringhe sull'alfabeto  $\{a, b\}$
3. **Struttura corretta**: L'alternanza  $x_i, z_i$  è forzata dalle produzioni

## Osservazione Cruciale

La grammatica **non** controlla direttamente il vincolo  $|z_1 \dots z_n| = n$ . Tuttavia, questo è **intrinseco nella struttura**:

- Ogni inserimento Z corrisponde esattamente a una transizione del DFA
- Il numero di inserimenti = numero di transizioni fino al penultimo stato

- Se ci sono  $n$  transizioni con inserimento, abbiamo esattamente  $n$  stringhe  $z_i$
- Il vincolo  $|z_1 \dots z_n| = n$  è **automaticamente soddisfatto** dalla struttura della derivazione

**Nota:** Il linguaggio generato potrebbe includere stringhe con  $|z_1 \dots z_n| \neq n$ , ma la **definizione originale di INTERLEAVE** richiedeva questo vincolo esplicito. La grammatica qui genera un sovrainsieme che include INTERLEAVE(A).