



SAPIENZA  
UNIVERSITÀ DI ROMA

“SAPIENZA” UNIVERSITÀ DI ROMA  
INGEGNERIA DELL'INFORMAZIONE,  
INFORMATICA E STATISTICA  
DIPARTIMENTO DI INFORMATICA

---

# Automi, Calcolabilità e Complessità

---

Appunti integrati con il libro "Introduzione alla teoria  
della computazione", Michael Sipser

*Author*  
Simone Bianco

8 dicembre 2023

# Indice

<b>Informazioni e Contatti</b>	<b>1</b>
<b>1 Linguaggi regolari</b>	<b>2</b>
1.1 Linguaggi . . . . .	2
1.2 Determinismo . . . . .	5
1.3 Non determinismo . . . . .	9
1.3.1 Equivalenza tra NFA e DFA . . . . .	12
1.4 Chiusure dei linguaggi regolari . . . . .	15
1.5 Espressioni regolari . . . . .	20
1.5.1 NFA generalizzati . . . . .	23
1.5.2 Equivalenza tra espressioni e linguaggi regolari . . . . .	29
1.6 Pumping lemma per i linguaggi regolari . . . . .	30
1.7 Esercizi svolti . . . . .	33
<b>2 Linguaggi acontestuali</b>	<b>38</b>
2.1 Grammatiche acontestuali . . . . .	38
2.2 Linguaggi acontestuali ad estensione dei regolari . . . . .	42
2.3 Forma normale di Chomsky . . . . .	44
2.4 Automi a pila . . . . .	47
2.4.1 Equivalenza tra CFG e PDA . . . . .	50
2.5 Pumping lemma per i linguaggi acontestuali . . . . .	55
2.6 Chiusure dei linguaggi acontestuali . . . . .	60
<b>3 Decidibilità</b>	<b>67</b>
3.1 Macchine di Turing . . . . .	67
3.1.1 Varianti della macchina di Turing . . . . .	72
3.1.2 Tesi di Church-Turing . . . . .	77
3.2 Problemi decidibili . . . . .	78
3.3 Argomento diagonale di Cantor . . . . .	85
3.3.1 Esistenza di linguaggi non riconoscibili . . . . .	88
3.4 Problemi indecidibili . . . . .	89

# Informazioni e Contatti

Appunti e riassunti personali raccolti in ambito del corso di *Automi, Calcolabilità e Complessità* offerto dal corso di laurea in Informatica dell'Università degli Studi di Roma "La Sapienza".

Ulteriori informazioni ed appunti possono essere trovati al seguente link:

<https://github.com/Exyss/university-notes>. Chiunque si senta libero di segnalare incorrettezze, migliorie o richieste tramite il sistema di Issues fornito da GitHub stesso o contattando in privato l'autore :

- Email: [bianco.simone@outlook.it](mailto:bianco.simone@outlook.it)
- LinkedIn: [Simone Bianco](#)

Gli appunti sono in continuo aggiornamento, pertanto, previa segnalazione, si prega di controllare se le modifiche siano già state apportate nella versione più recente.

## Prerequisiti consigliati per lo studio:

Apprendimento del materiale relativo al corso *Progettazione di Algoritmi*.

## Licence:

These documents are distributed under the [GNU Free Documentation License](#), a form of copyleft intended for use on a manual, textbook or other documents. Material licensed under the current version of the license can be used for any purpose, as long as the use meets certain conditions:

- All previous authors of the work must be **attributed**.
- All changes to the work must be **logged**.
- All derivative works must be **licensed under the same license**.
- The full text of the license, unmodified invariant sections as defined by the author if any, and any other added warranty disclaimers (such as a general disclaimer alerting readers that the document may not be accurate for example) and copyright notices from previous versions must be maintained.
- Technical measures such as DRM may not be used to control or obstruct distribution or editing of the document.

# 1

## Linguaggi regolari

### 1.1 Linguaggi

#### Definizione 1: Alfabeto

Definiamo come **alfabeto** un insieme finito di elementi detti **simboli**

**Esempio:**

- L'insieme  $\Sigma = \{0, 1, x, y, z\}$  è un alfabeto
- L'insieme  $\Sigma = \{0, 1\}$  è un alfabeto. In particolare, tale alfabeto viene detto **alfabeto binario**

#### Definizione 2: Stringa

Data una sequenza di simboli  $w_1, \dots, w_n \in \Sigma$ , definiamo:

$$w := w_1 \dots w_n$$

come **stringa** (o **parola**) di  $\Sigma$

**Esempio:**

- Dato l'alfabeto  $\Sigma = \{0, 1, x, y, z\}$ , una stringa di  $\Sigma$  è  $0x1yyy0$

#### Definizione 3: Linguaggio

Dato un alfabeto  $\Sigma$ , definiamo come **linguaggio** di  $\Sigma$ , indicato come  $\Sigma^*$ , l'insieme delle stringhe di  $\Sigma$

**Definizione 4: Lunghezza di una stringa**

Data una stringa  $w \in \Sigma^*$ , definiamo la **lunghezza di**  $w$ , indicata come  $|w|$ , come il numero di simboli presenti in  $w$

**Definizione 5: Concatenazione**

Data la stringa  $x := x_1 \dots x_n \in \Sigma^*$  e la stringa  $y := y_1 \dots y_m \in \Sigma^*$ , definiamo come **concatenazione di**  $x$  **con**  $y$  la seguente operazione:

$$xy = x_1 \dots x_n y_1 \dots y_m$$

**Proposizione 1: Stringa vuota**

Indichiamo con  $\varepsilon$  la **stringa vuota**, ossia l'unica stringa tale che:

- $|\varepsilon| = 0$
- $\forall w \in \Sigma^* \quad w \cdot \varepsilon = \varepsilon \cdot w = w$
- $\Sigma^* \neq \emptyset \implies \varepsilon \in \Sigma^*$

**Definizione 6: Conteggio**

Data una stringa  $w \in \Sigma^*$  e un simbolo  $a \in \Sigma$  definiamo il **conteggio di**  $a$  **in**  $w$ , indicato come  $|w|_a$ , il numero di simboli uguali ad  $a$  presenti in  $w$

**Esempio:**

- Data la stringa  $w := 010101000 \in \{0, 1\}^*$ , si ha che  $|w|_0 = 6$  e  $|w|_1 = 3$

**Definizione 7: Stringa rovesciata**

Data una stringa  $w = a_1 \dots a_n \in \Sigma^*$ , dove  $a_1 \dots a_n \in \Sigma$ , definiamo la sua **stringa rovesciata**, indicata con  $w^R$ , come  $w^R = a_n \dots a_1$ .

**Esempio:**

- Data la stringa  $w := abcdefg \in \Sigma^*$ , si ha che  $w^R = gfedcba$

**Definizione 8: Potenza**

Data la stringa  $w \in \Sigma^*$  e dato  $n \in \mathbb{N}$ , definiamo come **potenza** la seguente operazione:

$$w^n = \begin{cases} \varepsilon & \text{se } n = 0 \\ ww^{n-1} & \text{se } n > 0 \end{cases}$$

**Proposizione 2: Operazioni sui linguaggi**

Dati i linguaggi  $L, L_1, L_2 \subseteq \Sigma^*$ , definiamo le seguenti operazioni:

- Operatore unione:

$$L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$$

- Operatore intersezione:

$$L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$$

- Operatore complemento:

$$\overline{L} = \{w \in \Sigma^* \mid w \notin L\}$$

- Operatore concatenazione:

$$L_1 \circ L_2 = \{xy \in \Sigma^* \mid x \in L_1, y \in L_2\}$$

- Operatore potenza:

$$L^n = \begin{cases} \{\varepsilon\} & \text{se } n = 0 \\ L \circ L^{n-1} & \text{se } n > 0 \end{cases}$$

- Operatore star di Kleene:

$$L^* = \{w_1 \dots w_k \in \Sigma^* \mid k \geq 0, \forall i \in [1, k] \ w_i \in L\} = \bigcup_{n \geq 0} L^n$$

- Operatore plus di Kleene:

$$L^+ = \{w_1 \dots w_k \in \Sigma^* \mid k \geq 1, \forall i \in [1, k] \ w_i \in L\} = \bigcup_{n \geq 1} L^n = L \circ L^*$$

**Teorema 1: Leggi di DeMorgan**

Dati due linguaggi  $L_1$  e  $L_2$ , si ha che:

$$L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$$

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

(dimostrazione omessa)

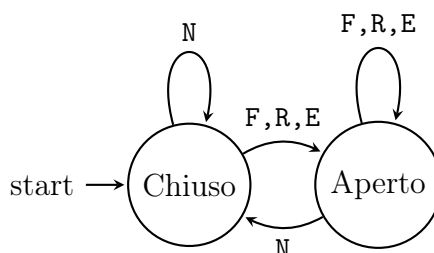
## 1.2 Determinismo

### Definizione 9: Automa

Un **automa** è un meccanismo di controllo (o macchina) progettato per seguire automaticamente una sequenza di operazioni o rispondere a istruzioni predeterminate, mantenendo informazioni relative allo **stato** attuale dell'automa stesso ed agendo di conseguenza, **passando da uno stato all'altro**.

#### Esempio:

- Un sensore che apre e chiude una porta può essere descritto tramite il seguente automa, dove **Chiuso** e **Aperto** sono gli stati dell'automa e N, F, R e E sono le operazioni di transizione tra i due stati indicanti rispettivamente:
  - N: il sensore non rileva alcuna persona da entrambi i lati della porta
  - F: il sensore rileva qualcuno nel lato frontale della porta
  - R: il sensore rileva qualcuno nel lato retrostante della porta
  - E: il sensore rileva qualcuno da entrambi i lati della porta



- L'automa appena descritto è in grado di interpretare una **stringa in input** che ne descriva la sequenza di operazioni da svolgere (es: la stringa NFNNNFRR terminerà l'esecuzione dell'automa sullo stato Aperto)

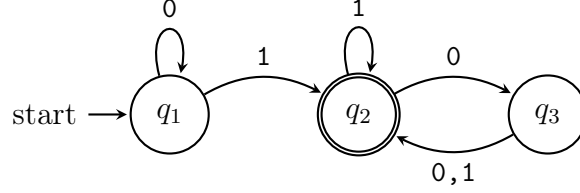
### Definizione 10: Deterministic Finite Automaton (DFA)

Un **Deterministic Finite Automaton (DFA)** (o *Automa Deterministico a Stati Finiti*) è una quintupla  $(Q, \Sigma, \delta, q_0, F)$  dove:

- $Q$  è l'**insieme finito degli stati** dell'automa
- $\Sigma$  è l'**alfabeto** dell'automa
- $\delta : Q \times \Sigma \rightarrow Q$  è la **funzione di transizione degli stati** dell'automa
- $q_0 \in Q$  è lo **stato iniziale** dell'automa
- $F \subseteq Q$  è l'**insieme degli stati accettanti** dell'automa, ossia l'insieme degli stati su cui, a seguito della lettura di una stringa in input, l'automa accetta la corretta terminazione

**Esempio:**

- Consideriamo il seguente DFA



dove:

- $Q = \{q_1, q_2, q_3\}$  è l'insieme degli stati dell'automa
- $\Sigma = \{0, 1\}$  è l'alfabeto dell'automa
- $\delta : Q \times \Sigma \rightarrow Q$  definita come

$\delta$	$q_1$	$q_2$	$q_3$
0	$q_1$	$q_3$	$q_2$
1	$q_2$	$q_2$	$q_2$

è la funzione di transizione degli stati dell'automa

- $q_1$  è lo stato iniziale dell'automa
- $F = \{q_2\}$  è l'insieme degli stati accettanti

**Definizione 11: Funzione di transizione estesa**

Sia  $D := (Q, \Sigma, \delta, q_0, F)$  un DFA. Definiamo  $\delta^* : Q \times \Sigma^* \rightarrow Q$  come **funzione di transizione estesa di  $D$**  la funzione definita ricorsivamente come:

$$\begin{cases} \delta^*(q, \varepsilon) = \delta(q, \varepsilon) = q \\ \delta^*(q, aw) = \delta^*(\delta(q, a), w), \text{ dove } a \in \Sigma, w \in \Sigma^* \end{cases}$$

**Proposizione 3: Stringa accettata in un DFA**

Sia  $D := (Q, \Sigma, \delta, q_0, F)$  un DFA. Data una stringa  $w \in \Sigma^*$ , diciamo che  $w$  è **accettata da  $D$**  se  $\delta^*(q_0, w) \in F$ , ossia l'interpretazione di tale stringa **termina su uno stato accettante**

**Esempio:**

- Consideriamo ancora il DFA dell'esempio precedente.
- La stringa 0101 è accettata da tale DFA, poiché:

$$\begin{aligned} \delta^*(q_1, 0101) &= \delta^*(\delta(q_1, 0), 101) = \delta^*(q_2, 101) = \delta^*(\delta(q_2, 1), 01) = \delta^*(q_2, 01) = \\ &= \delta^*(\delta(q_2, 0), 1) = \delta^*(q_3, 1) = \delta^*(\delta(q_3, 1), \varepsilon) = \delta^*(q_2, \varepsilon) = q_2 \in F \end{aligned}$$



- La stringa 1010, invece, non è accettata dal DFA, poiché:

$$\delta^*(q_1, 1010) = \delta^*(q_2, 010) = \delta^*(q_3, 10) = \delta^*(q_2, 0) = \delta^*(q_3, \varepsilon) = q_3 \notin F$$

### Definizione 12: Linguaggio di un automa

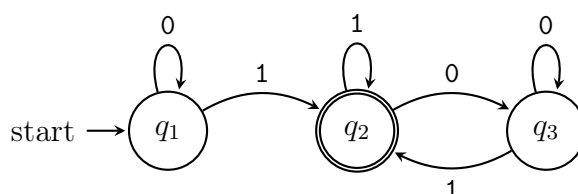
Sia  $A$  un automa. Definiamo come **linguaggio di  $A$** , indicato come  $L(A)$ , l'insieme di stringhe accettate da  $A$

$$L(A) = \{w \in \Sigma^* \mid A \text{ accetta } w\}$$

Inoltre, diciamo che  $D$  **riconosce**  $L(A)$

### Esempi:

- Consideriamo il seguente DFA  $D$



- Il linguaggio riconosciuto da tale DFA corrisponde a

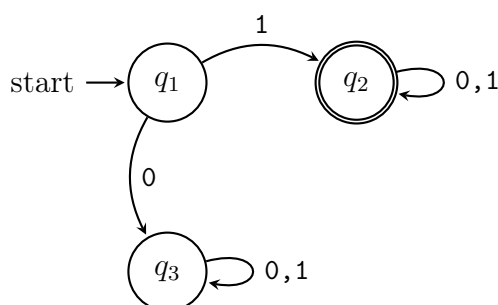
$$L(D) = \{x \in \{0, 1\}^* \mid x := y1, \exists y \in \{0, 1\}^*\}$$

ossia al linguaggio composto da tutte le stringhe terminanti con 1

- Consideriamo il seguente linguaggio

$$L = \{x \in \{0, 1\}^* \mid 1y, \exists y \in \{0, 1\}^*\}$$

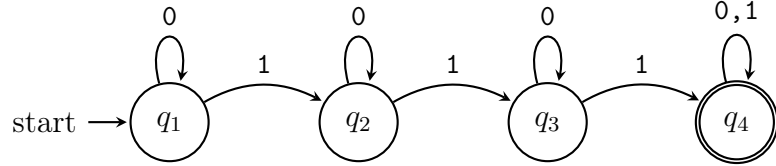
- Un DFA in grado di riconoscere tale linguaggio corrisponde a



3. • Consideriamo il seguente linguaggio

$$L = \{w \in \{0, 1\}^* \mid |w|_1 \geq 3\}$$

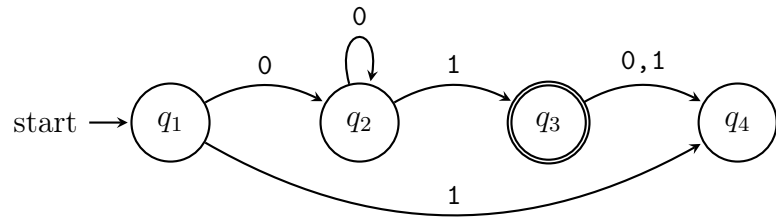
- Un DFA in grado di riconoscere tale linguaggio corrisponde a



4. • Consideriamo il seguente linguaggio

$$L = \{w \in \{0, 1\}^* \mid w := 0^n 1, n \in \mathbb{N} - \{0\}\}$$

- Un DFA in grado di riconoscere tale linguaggio corrisponde a



### Definizione 13: Configurazione di un DFA

Sia  $D := (Q, \Sigma, \delta, q_0, F)$  un DFA. Definiamo la coppia  $(q, w) \in Q \times \Sigma^*$  come **configurazione di  $D$**

### Definizione 14: Passo di computazione in un DFA

Definiamo come **passo di computazione** la relazione binaria definita come

$$(p, aw) \vdash_D (q, w) \iff \delta(p, a) = q$$

### Definizione 15: Computazione deterministica

Definiamo una computazione come **deterministica** se ad ogni passo di computazione segue un'unica configurazione:

$$\forall (q, aw) \ \exists! (p, w) \mid (q, aw) \vdash_D (p, w)$$

**Proposizione 4: Chiusura del passo di computazione**

Sia  $D := (Q, \Sigma, \delta, q_0, F)$  un DFA. La **chiusura riflessiva e transitiva** di  $\vdash_D$ , indicata come  $\vdash_D^*$ , gode delle seguenti proprietà:

- $(p, aw) \vdash_D (q, w) \implies (p, aw) \vdash_D^* (q, w)$
- $\forall q \in Q, w \in \Sigma^* \quad (q, w) \vdash_D^* (q, w)$
- $(p, abw) \vdash_D (q, bw) \wedge (q, bw) \vdash_D (r, w) \implies (p, abw) \vdash_D^* (r, w)$

**Osservazione 1**

Sia  $D := (Q, \Sigma, \delta, q_0, F)$  un DFA. Dati  $q_i, q_f \in Q, w \in \Sigma^*$ , si ha che

$$\delta^*(q_i, w) = q_f \iff (q_i, w) \vdash_D^* (q_f, \varepsilon)$$

## 1.3 Non determinismo

**Definizione 16: Alfabeto epsilon**

Dato un alfabeto  $\Sigma$ , definiamo  $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$  come **alfabeto epsilon** di  $\Sigma$

**Definizione 17: Non-deterministic Finite Automaton (NFA)**

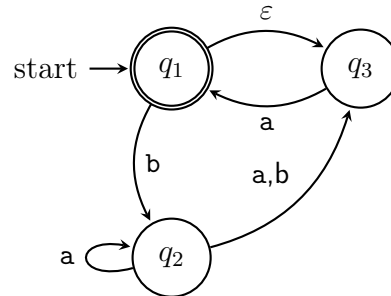
Un **Non-deterministic Finite Automaton (NFA)** (o *Automa Non-deterministico a Stati Finiti*) è una quintupla  $(Q, \Sigma, \delta, q_0, F)$  dove:

- $Q$  è l'**insieme finito degli stati** dell'automa
- $\Sigma$  è l'**alfabeto** dell'automa
- $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$  è la **funzione di transizione degli stati** dell'automa
- $q_0 \in Q$  è lo **stato iniziale** dell'automa
- $F \subseteq Q$  è l'**insieme degli stati accettanti** dell'automa

**Nota:**  $\mathcal{P}(Q)$  è l'insieme delle parti di  $Q$ , ossia l'insieme contenente tutti i suoi sottoinsiemi possibili

**Esempio:**

- Consideriamo il seguente NFA



dove:

- $Q = \{q_1, q_2, q_3\}$  è l'insieme degli stati dell'automa
- $\Sigma = \{a, b\}$  è l'alfabeto dell'automa
- $\delta : Q \times \Sigma \rightarrow Q$  definita come

$\delta$	$q_1$	$q_2$	$q_3$
$\varepsilon$	$\{q_3\}$	$\emptyset$	$\emptyset$
<b>a</b>	$\emptyset$	$\{q_2, q_3\}$	$\{q_1\}$
<b>b</b>	$\{q_2\}$	$\{q_3\}$	$\emptyset$

è la funzione di transizione degli stati dell'automa

- $q_1$  è lo stato iniziale dell'automa
- $F = \{q_1\}$  è l'insieme degli stati accettanti

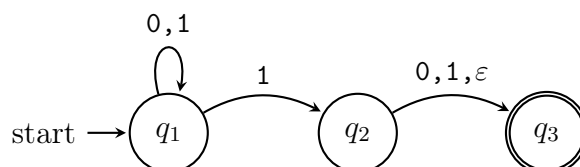
**Osservazione 2: Computazione in un NFA**

Sia  $N := (Q, \Sigma, \delta, q_0, F)$  un NFA. Data una stringa  $w \in \Sigma_\varepsilon$  in ingresso, la **computazione** viene eseguita nel seguente modo:

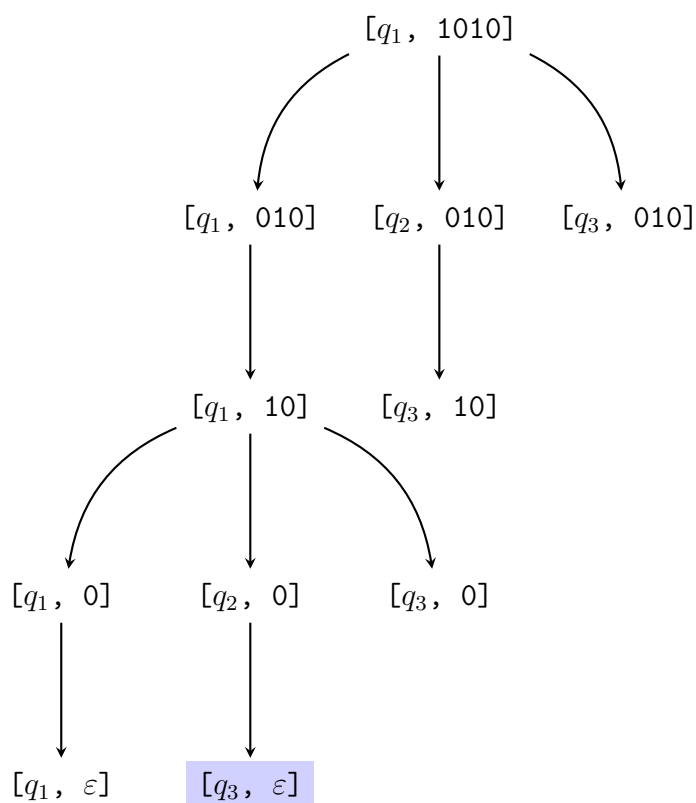
- Tutte le volte che uno stato potrebbe avere più transizioni per diversi simboli dell'alfabeto, l'automa  $N$  si duplica in **più copie**, ognuna delle quali segue il suo corso. Si vengono così a creare più **rami di computazione** indipendenti che sono eseguiti in **parallelo**.
- Se il prossimo simbolo della stringa da computare non si trova su nessuna delle transizioni uscenti dello stato attuale di un ramo di computazione, l'intero ramo **termina la sua computazione** (terminazione incorretta).
- Se almeno una delle copie di  $N$  termina correttamente su uno stato di accettazione, l'automa **accetta la stringa di partenza**.
- Quando a seguito di una computazione ci si ritrova in uno stato che possiede un  $\varepsilon$ -arco in uscita, la macchina si duplica in più copie: quelle che seguono gli  $\varepsilon$ -archi e quella che rimane nello stato raggiunto.

**Esempio:**

- Consideriamo il seguente NFA



- Supponiamo che venga computata la stringa  $w = 1010$ :



- Poiché esiste un ramo che termina correttamente, l'NFA descritto accetta la stringa  $w = 1010$

**Proposizione 5: Stringa accettata in un NFA**

Sia  $N := (Q, \Sigma, \delta, q_0, F)$  un NFA. Data una stringa  $w := w_0 \dots w_k \in \Sigma^*$ , dove  $w_0, \dots, w_k \in \Sigma_\varepsilon$ , diciamo che  $w$  è **accettata da**  $N$  se esiste una sequenza di stati  $r_0, r_1, \dots, r_{k+1} \in Q$  tali che:

- $r_0 = q_0$
- $\forall i \in [0, k] \quad r_{i+1} \in \delta(r_i, w_i)$
- $r_{k+1} \in F$

### 1.3.1 Equivalenza tra NFA e DFA

#### Definizione 18: Classe dei linguaggi riconosciuti da un DFA

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi di  $\Sigma$  riconosciuti da un DFA** il seguente insieme:

$$\mathcal{L}(\text{DFA}) = \{L \subseteq \Sigma^* \mid \exists \text{ DFA } D \text{ t.c. } L = L(D)\}$$

#### Definizione 19: Classe dei linguaggi riconosciuti da un NFA

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi di  $\Sigma$  riconosciuti da un NFA** il seguente insieme:

$$\mathcal{L}(\text{NFA}) = \{L \subseteq \Sigma^* \mid \exists \text{ NFA } N \text{ t.c. } L = L(N)\}$$

#### Teorema 2: Equivalenza tra NFA e DFA

Date le due classi di linguaggi  $\mathcal{L}(\text{DFA})$  e  $\mathcal{L}(\text{NFA})$ , si ha che:

$$\mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA})$$

*Dimostrazione.*

*Prima implicazione.*

- Dato  $L \in \mathcal{L}(\text{DFA})$ , sia  $D := (Q, \Sigma, \delta, q_0, F)$  il DFA tale che  $L = L(D)$
- Poiché il concetto di NFA è una generalizzazione del concetto di DFA, ne segue automaticamente che  $D$  sia anche un NFA, implicando che  $L \in \mathcal{L}(\text{NFA})$  e di conseguenza che:

$$\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{NFA})$$

*Seconda implicazione.*

- Dato  $L \in \mathcal{L}(\text{NFA})$ , sia  $N := (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$  il NFA tale che  $L = L(N)$
- Consideriamo quindi il DFA  $D := (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$  costruito tramite  $N$  stesso:

$$- Q_D = \mathcal{P}(Q_N)$$

- Dato  $R \in Q_D$ , definiamo l'estensione di  $R$  come:

$$E(R) = \{q \in Q_N \mid q \text{ è raggiungibile in } N \text{ da } q' \in R \text{ tramite } k \geq 0 \text{ } \varepsilon\text{-archi}\}$$

$$- q_{0_D} = E(\{q_{0_N}\})$$

$$- F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}$$

– Dati  $R \in Q_D$  e  $a \in \Sigma$ , definiamo  $\delta_D$  come:

$$\delta_D = (R, a) = \bigcup_{r \in R} E(\delta_N(r, a))$$

• A questo punto, per costruzione stessa di  $D$  si ha che:

$$w \in L = L(N) \iff w \in L(D)$$

implicando dunque che  $L \in \mathcal{L}(\text{DFA})$  e di conseguenza che:

$$\mathcal{L}(\text{NFA}) \subseteq \mathcal{L}(\text{DFA})$$

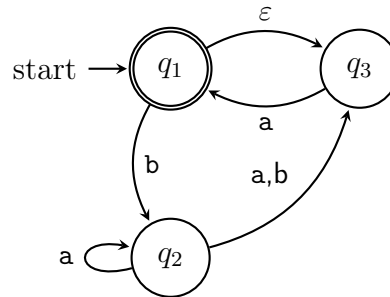
□

### Osservazione 3

Dato un NFA  $N$ , seguendo i passaggi della dimostrazione precedente è possibile definire un DFA  $D$  equivalente ad  $N$

**Esempio:**

• Consideriamo ancora il seguente NFA



• Definiamo quindi l'insieme degli stati del DFA equivalente a tale NFA:

$$Q_D = \{\emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_2, q_3\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}\} =$$

• Per facilitare la lettura, riscriviamo i vari stati con la seguente notazione

$$Q_D = \{\emptyset, q_1, q_2, q_3, q_{1,2}, q_{2,3}, q_{1,3}, q_{1,2,3}\}$$

• A questo punto, poniamo:

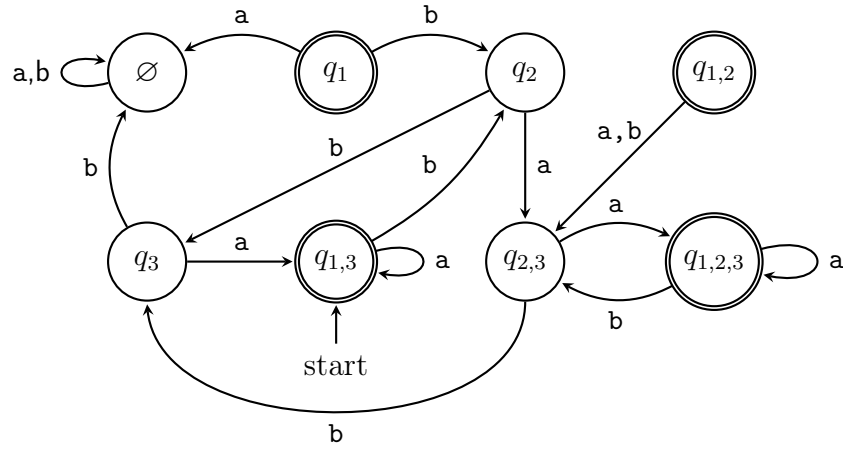
$$- q_{0_D} = E(\{q_{0_N}\}) = E(\{q_1\}) = \{q_1, q_3\} = q_{1,3}$$

$$- F_D = \{q_1, q_{1,2}, q_{1,3}, q_{1,2,3}\}$$

- Le transizioni del DFA corrisponderanno invece a:

- $\delta_D(\{q_1\}, a) = E(\delta_N(q_1, a)) = \emptyset$
- $\delta_D(\{q_1\}, b) = E(\delta_N(q_1, b)) = \{q_2\} = q_2$
- $\delta_D(\{q_2\}, a) = E(\delta_N(q_2, a)) = \{q_2, q_3\} = q_{2,3}$
- $\delta_D(\{q_2\}, b) = E(\delta_N(q_2, b)) = \{q_2\} = q_2$
- $\delta_D(\{q_1, q_2\}, a) = E(\delta_N(q_1, a)) \cup E(\delta_N(q_2, a)) = \emptyset \cup \{q_2, q_3\} = \{q_2, q_3\} = q_{2,3}$
- $\delta_D(\{q_1, q_2\}, b) = E(\delta_N(q_1, b)) \cup E(\delta_N(q_2, b)) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\} = q_{2,3}$
- ...

- Il DFA equivalente corrisponde dunque a:



### Definizione 20: Linguaggi regolari

Dato un alfabeto  $\Sigma$ , definiamo come **insieme dei linguaggi regolari di  $\Sigma$** , indicato con  $\text{REG}$ , l'insieme delle classi dei linguaggi riconosciuti da un DFA:

$$\text{REG} := \mathcal{L}(\text{DFA})$$

### Osservazione 4

Tramite il teorema dell'[Equivalenza tra NFA e DFA](#), si ha che:

$$\text{REG} := \mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA})$$



## 1.4 Chiusure dei linguaggi regolari

### Teorema 3: Chiusura dell'unione in REG

L'operatore unione è **chiuso in REG**, ossia:

$$\forall L_1, L_2 \in \text{REG} \quad L_1 \cup L_2 \in \text{REG}$$

*Dimostrazione I.*

- Dati  $L_1, L_2 \in \text{REG}$ , siano  $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  e  $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  i due DFA tali che  $L_1 = L(D_1)$  e  $L_2 = L(D_2)$
- Definiamo quindi il DFA  $D = (Q, \Sigma, \delta, q_0, F)$  tale che:

- $q_0 = (q_1, q_2)$
- $Q = Q_1 \times Q_2$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(r_1, r_2) \mid r_1 \in F_1 \vee r_2 \in F_2\}$
- $\forall (r_1, r_2) \in Q, a \in \Sigma$  si ha che:

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

- A questo punto, per costruzione stessa di  $D$  ne segue che:

$$w \in L_1 \cup L_2 \iff w \in L(D)$$

dunque che  $L_1 \cup L_2 = L(D) \in \text{REG}$

□

*Dimostrazione II.*

- Dati  $L_1, L_2 \in \text{REG}$ , siano  $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  e  $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  i due NFA tali che  $L_1 = L(N_1)$  e  $L_2 = L(N_2)$
- Definiamo quindi il NFA  $N = (Q, \Sigma, \delta, q_0, F)$  tale che:

- $q_0$  è un nuovo stato iniziale aggiunto
- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- $F = F_1 \cup F_2$
- $\forall q \in Q, a \in \Sigma$  si ha che:

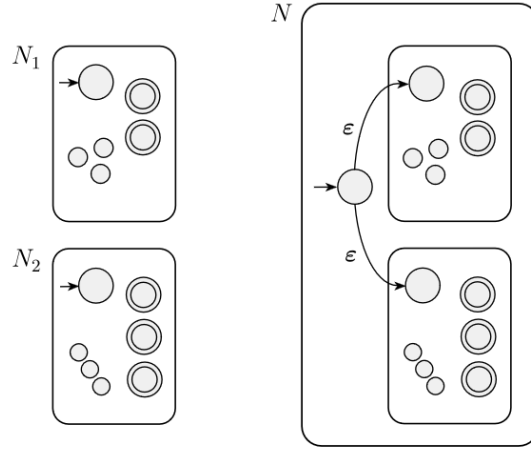
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se } q \in Q_1 \\ \delta_2(q, a) & \text{se } q \in Q_2 \\ \{q_1, q_2\} & \text{se } q = q_0 \wedge a = \varepsilon \\ \emptyset & \text{se } q = q_0 \wedge a \neq \varepsilon \end{cases}$$

- A questo punto, per costruzione stessa di  $N$  ne segue che:

$$w \in L_1 \cup L_2 \iff w \in L(N)$$

dunque che  $L_1 \cup L_2 = L(N) \in \text{REG}$

□



*Rappresentazione grafica della dimostrazione*

#### Teorema 4: Chiusura dell'intersezione in REG

L'operatore intersezione è **chiuso in REG**, ossia:

$$\forall L_1, L_2 \in \text{REG} \quad L_1 \cap L_2 \in \text{REG}$$

*Dimostrazione.*

- Dati  $L_1, L_2 \in \text{REG}$ , siano  $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  e  $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  i due DFA tali che  $L_1 = L(D_1)$  e  $L_2 = L(D_2)$
- Definiamo quindi il DFA  $D = (Q, \Sigma, \delta, q_0, F)$  tale che:

- $q_0 = (q_1, q_2)$
- $Q = Q_1 \times Q_2$
- $F = F_1 \times F_2 = \{(r_1, r_2) \mid r_1 \in F_1 \wedge r_2 \in F_2\}$
- $\forall (r_1, r_2) \in Q, a \in \Sigma$  si ha che:

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

- A questo punto, per costruzione stessa di  $D$  ne segue che:

$$w \in L_1 \cap L_2 \iff w \in L(D)$$

dunque che  $L_1 \cap L_2 = L(D) \in \text{REG}$

□

**Teorema 5: Chiusura del complemento in REG**

L'operatore complemento è **chiuso in REG**, ossia:

$$\forall L \in \text{REG} \quad \bar{L} \in \text{REG}$$

*Dimostrazione.*

- Dato  $L \in \text{REG}$ , sia  $D = (Q, \Sigma, \delta, q_0, F)$  il DFA tale che  $L = L(D)$
- Definiamo quindi il DFA  $D' = (Q, \Sigma, \delta, q_0, Q - F)$ , dunque il DFA uguale a  $D$  ma i cui stati accettanti sono invertiti. Per costruzione stessa di  $D'$  ne segue che:

$$w \in L \iff w \notin L(D)$$

dunque che  $\bar{L} = L(D') \in \text{REG}$

□

**Teorema 6: Chiusura della concatenazione in REG**

L'operatore concatenazione è **chiuso in REG**, ossia:

$$\forall L_1, L_2 \in \text{REG} \quad L_1 \circ L_2 \in \text{REG}$$

*Dimostrazione.*

- Dati  $L_1, L_2 \in \text{REG}$ , siano  $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  e  $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  i due NFA tali che  $L_1 = L(N_1)$  e  $L_2 = L(N_2)$
- Definiamo quindi il NFA  $N = (Q, \Sigma, \delta, q_0, F)$  tale che:
  - $q_0 = q_1$
  - $Q = Q_1 \cup Q_2$
  - $F = F_2$
  - $\forall q \in Q, a \in \Sigma$  si ha che:

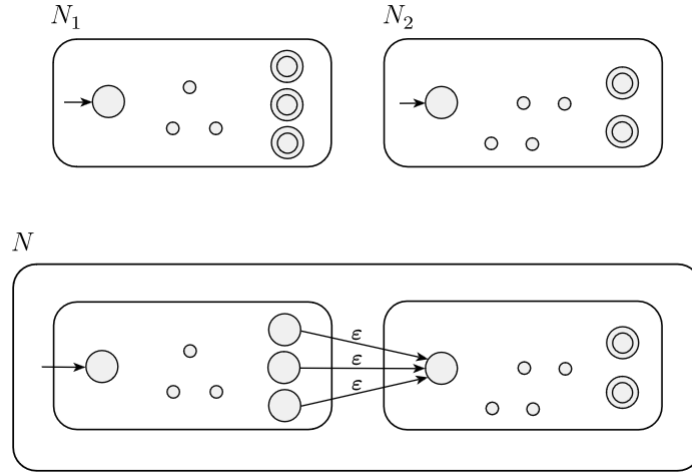
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se } q \in Q_1 - F_1 \\ \delta_1(q, a) & \text{se } q \in F_1 \wedge a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & \text{se } q \in F_1 \wedge a = \varepsilon \\ \delta_2(q, a) & \text{se } q \in Q_2 \end{cases}$$

- A questo punto, per costruzione stessa di  $N$  ne segue che:

$$w \in L_1 \circ L_2 \iff w \in L(N)$$

dunque che  $L_1 \circ L_2 = L(N) \in \text{REG}$

□



Rappresentazione grafica della dimostrazione

### Corollario 1: Chiusura della potenza in REG

L'operatore potenza è **chiuso in REG**, ossia:

$$\forall L \in \text{REG}, n \in \mathbb{N} \quad L^n \in \text{REG}$$

### Teorema 7: Chiusura di star in REG

L'operatore star è **chiuso in REG**, ossia:

$$\forall L \in \text{REG} \quad L^* \in \text{REG}$$

*Dimostrazione.*

- Dato  $L \in \text{REG}$ , sia  $N = (Q, \Sigma, \delta, q_0, F)$  il NFA tale che  $L = L(N)$
- Definiamo quindi il DFA  $N' = (Q', \Sigma, \delta', q_{0*}, F')$  tale che:
  - $q_{0*}$  è un nuovo stato iniziale aggiunto
  - $Q' = Q \cup \{q_{0*}\}$
  - $F' = F \cup \{q_{0*}\}$
  - $\forall q \in Q', a \in \Sigma$  si ha che:

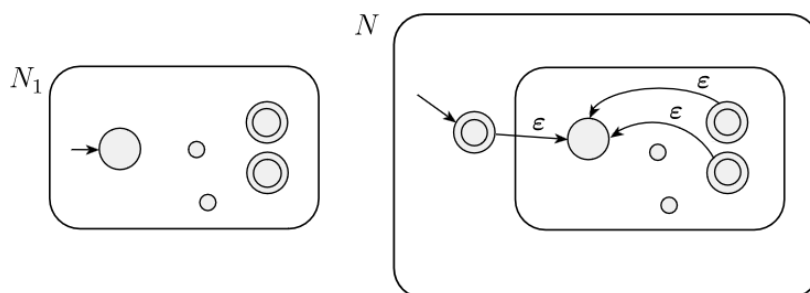
$$\delta'(q, a) = \begin{cases} \delta(q, a) & \text{se } q \in Q - F \\ \delta(q, a) & \text{se } q \in F \wedge a \neq \varepsilon \\ \delta(q, a) \cup \{q_{0*}\} & \text{se } q \in F \wedge a = \varepsilon \\ \{q_{0*}\} & \text{se } q = q_{0*} \wedge a = \varepsilon \\ \emptyset & \text{se } q = q_{0*} \wedge a \neq \varepsilon \end{cases}$$

- A questo punto, per costruzione stessa di  $N'$  ne segue che:

$$w \in L^* \iff w \in L(N')$$

dunque che  $L^* = L(N') \in \text{REG}$

□



*Rappresentazione grafica della dimostrazione*

### Corollario 2: Chiusura di plus in REG

L'operatore plus è **chiuso in REG**, ossia:

$$\forall L \in \text{REG} \quad L^+ \in \text{REG}$$

*Dimostrazione.*

- Analoga a quella dell'operatore star, rimuovendo tuttavia lo stato iniziale dall'insieme degli stati accettanti

□

## 1.5 Espressioni regolari

### Definizione 21: Espressione regolare

Dato un alfabeto  $\Sigma$ , definiamo come **espressione regolare di  $\Sigma$**  una stringa  $R$  rappresentante un linguaggio  $L(R) \subseteq \Sigma^*$ . In altre parole, ogni espressione regolare  $R$  rappresenta in realtà il linguaggio  $L(R)$  ad essa associata.

In particolare, definiamo l'**insieme delle espressioni regolari di  $\Sigma$** , indicato con  $\text{re}(\Sigma)$ , come:

- $\emptyset \in \text{re}(\Sigma)$
- $\varepsilon \in \text{re}(\Sigma)$
- $a \in \text{re}(\Sigma)$ , dove  $a \in \Sigma$
- $R_1, R_2 \in \text{re}(\Sigma) \implies R_1 \cup R_2 \in \text{re}(\Sigma)$
- $R_1, R_2 \in \text{re}(\Sigma) \implies R_1 \circ R_2 \in \text{re}(\Sigma)$
- $R \in \text{re}(\Sigma) \implies R^* \in \text{re}(\Sigma)$
- $R \in \text{re}(\Sigma) \implies R^+ \in \text{re}(\Sigma)$

### Osservazione 5

Data un'espressione regolare  $R \in \text{re}(\Sigma)$ , si ha che:

- $R = \emptyset \in \text{re}(\Sigma) \implies L(R) = \emptyset$
- $R = \varepsilon \in \text{re}(\Sigma) \implies L(R) = \{\varepsilon\}$
- $R = a \in \text{re}(\Sigma), a \in \Sigma \implies L(R) = \{a\}$
- $R = R_1 \cup R_2 \in \text{re}(\Sigma) \implies L(R) = L(R_1) \cup L(R_2)$
- $R = R_1 \circ R_2 \in \text{re}(\Sigma) \implies L(R) = L(R_1) \circ L(R_2)$
- $R = R_1^* \in \text{re}(\Sigma) \implies L(R) = L(R_1)^*$
- $R = R_1^+ \in \text{re}(\Sigma) \implies L(R) = L(R_1)^+$

### Esempi:

1.  $0 \cup 1$  rappresenta il linguaggio  $\{0\} \cup \{1\} = \{0, 1\}$
2.  $0^*10^*$  rappresenta il linguaggio  $\{0\}^* \circ \{1\} \circ \{0\}^* = \{x1y \mid x, y \in \{0\}^*\}$
3.  $\Sigma^*1\Sigma^*$  rappresenta il linguaggio  $\Sigma^* \circ \{1\} \circ \Sigma^* = \{x1y \mid x, y \in \Sigma^*\}$
4.  $(0 \cup 1000)^*$  rappresenta il linguaggio  $(\{0\} \cup \{1000\})^* = \{0, 1000\}^*$
5.  $\emptyset^*$  rappresenta il linguaggio  $\emptyset^* = \{\varepsilon\}$  (ricordiamo che per definizione stessa si ha che  $\forall L \subseteq \Sigma^* \quad L^0 = \{\varepsilon\}$ )

6.  $0^*\emptyset$  rappresenta il linguaggio  $\{0\}^* \circ \emptyset = \emptyset$
7.  $(0 \cup \varepsilon)(1 \cup \varepsilon)$  rappresenta il linguaggio  $\{\emptyset, 0, 1, 01\}$
8.  $\Sigma^+$  equivale all'espressione  $\Sigma\Sigma^*$

**Definizione 22: Classe dei linguaggi descritti da esp. reg.**

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi di  $\Sigma$  descritti da un'espressione regolare** il seguente insieme:

$$\mathcal{L}(\text{re}) = \{L \subseteq \Sigma^* \mid \exists R \in \text{re}(\Sigma) \text{ t.c. } L = L(R)\}$$

**Lemma 1: Conversione da espressione regolare a NFA**

Date le due classi di linguaggi  $\mathcal{L}(\text{re})$  e  $\mathcal{L}(\text{NFA})$ , si ha che:

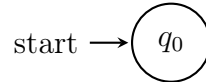
$$\mathcal{L}(\text{re}) \subseteq \mathcal{L}(\text{NFA})$$

*Dimostrazione.*

Procediamo per induzione strutturale, ossia dimostrando che se per ogni sotto-componente vale una determinata proprietà allora essa varrà anche per ogni componente formato da tali sotto-componenti

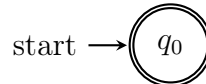
*Caso base.*

- Se  $R = \emptyset \in \text{re}(\Sigma)$ , definiamo il NFA  $N_\emptyset = (\{q_0\}, \Sigma, \delta, q_0, \emptyset)$ , ossia:



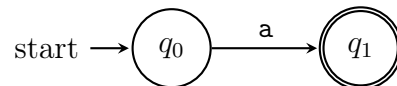
per cui si ha che  $w \in L(R) \iff w \in L(N_\emptyset)$  dunque  $L(R) = L(N_\emptyset) \in \mathcal{L}(\text{NFA})$

- Se  $R = \varepsilon \in \text{re}(\Sigma)$ , definiamo il NFA  $N_\varepsilon = (\{q_0\}, \Sigma, \delta, q_0, \{q_0\})$ , ossia:



per cui si ha che  $w \in L(R) \iff w \in L(N_\varepsilon)$  dunque  $L(R) = L(N_\varepsilon) \in \mathcal{L}(\text{NFA})$

- Se  $R = a \in \text{re}(\Sigma)$  con  $a \in \Sigma$ , definiamo il NFA  $N_a = (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$  dove per  $\delta$  è definita solo la coppia  $\delta(q_0, a) = q_1$ , ossia:



per cui si ha che  $w \in L(R) \iff w \in L(N_a)$  dunque  $L(R) = L(N_a) \in \mathcal{L}(\text{NFA})$

*Ipotesi induttiva.*

- Date  $R_1, R_2 \in \text{re}(\Sigma)$ , assumiamo che  $\exists \text{NFAN}_1, N_2 \mid L(R_1) = L(N_1), L(R_2) = L(N_2)$ , dunque che  $L(R_1), L(R_2) \in \mathcal{L}(\text{NFA})$

*Passo induttivo.*

- Se  $R = R_1 \cup R_2$ , tramite la **Chiusura dell'unione in REG**, otteniamo che:

$$L(R) = L(R_1) \cup L(R_2) = L(N_1) \cup L(N_2) \in \text{REG} = \mathcal{L}(\text{NFA})$$

- Se  $R = R_1 \circ R_2$ , tramite la **Chiusura della concatenazione in REG**, otteniamo che:

$$L(R) = L(R_1) \circ L(R_2) = L(N_1) \circ L(N_2) \in \text{REG} = \mathcal{L}(\text{NFA})$$

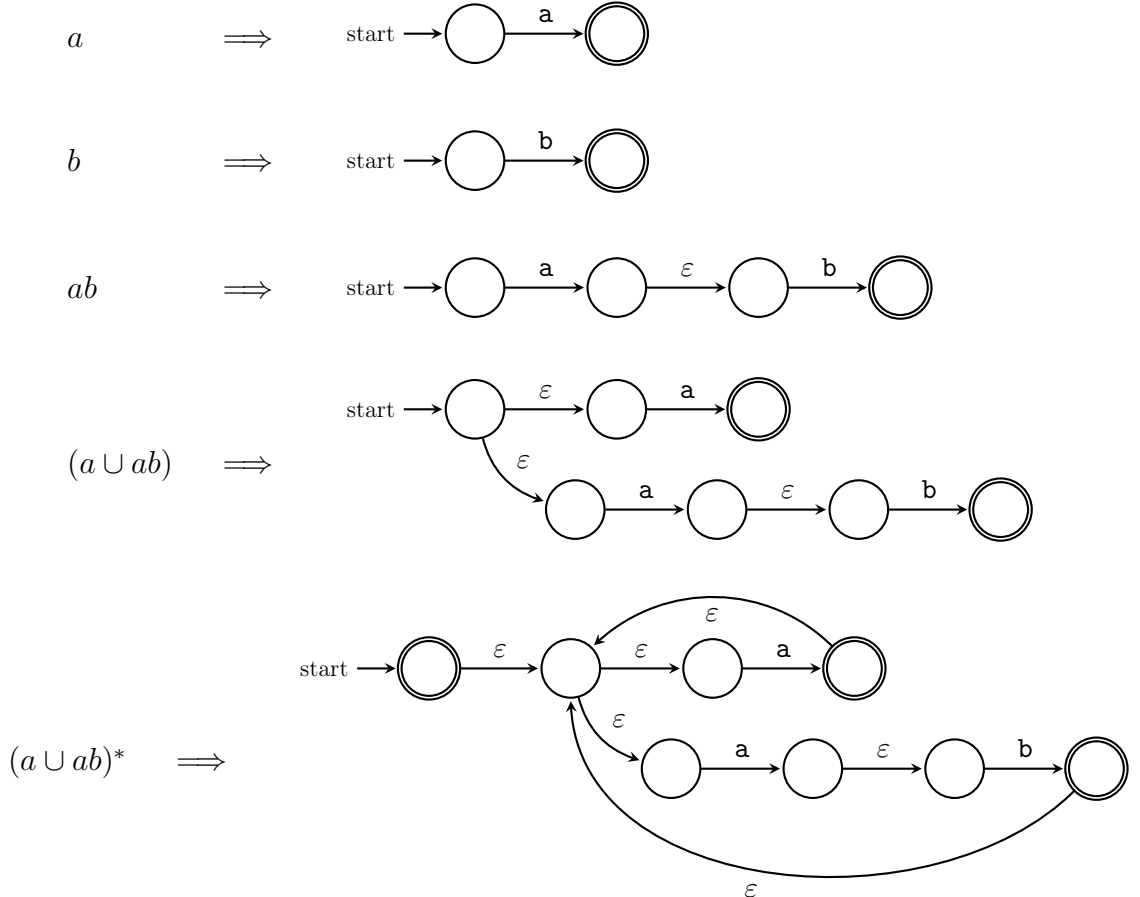
- Se  $R = R_1^*$ , tramite la **Chiusura di plus in REG**, otteniamo che:

$$L(R) = L(R_1)^* = L(N_1)^* \in \text{REG} = \mathcal{L}(\text{NFA})$$

□

**Esempio:**

- Consideriamo l'espressione regolare  $(a \cup ab)^*$
- Costruiamo il NFA corrispondente a tale espressione partendo dai suoi sotto-componenti





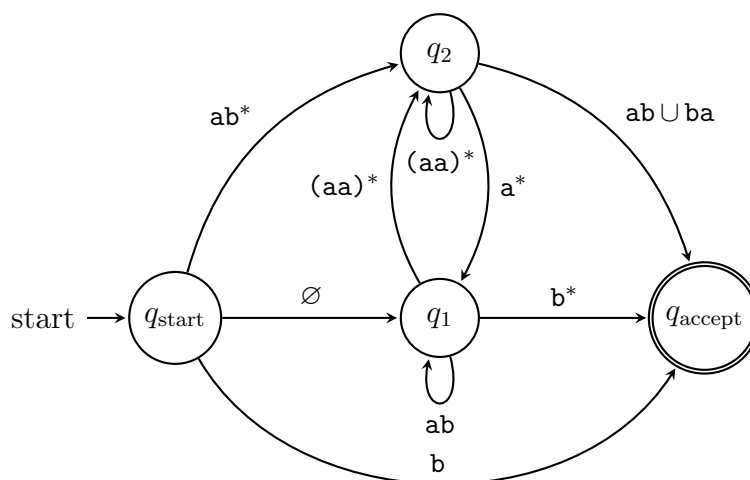
### 1.5.1 NFA generalizzati

#### Definizione 23: Generalized NFA (GNFA)

Un **Generalized NFA (GNFA)** è una quintupla  $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$  dove:

- $Q$  è l'insieme finito degli stati dell'automa dove  $|Q| \geq 2$
- $\Sigma$  è l'alfabeto dell'automa
- $q_{\text{start}} \in Q$  è lo **stato iniziale** dell'automa
- $q_{\text{accept}} \in Q$  è l'**unico stato accettante** dell'automa
- $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \text{re}(\Sigma)$  è la **funzione di transizione degli stati** dell'automa, implicando che:
  - Lo stato  $q_{\text{start}}$  abbia solo transizioni **uscenti**
  - Lo stato  $q_{\text{accept}}$  abbia solo transizioni **entranti**
  - Tra **tutte le possibili coppie di stati**  $q, q' \in Q$  (incluso il caso in cui  $q = q'$ ) vi sia una transizione  $q \rightarrow q'$  ed una transizione  $q' \rightarrow q$
  - Le "etichette" delle transizioni sono delle **espressioni regolari**

**Esempio:**



#### Osservazione 6

In un GNFA, il risultato  $\delta(q, q') = R$  può essere interpretato come "l'espressione regolare che effettua la transizione da  $q$  a  $q'$  è  $R$ ". Di conseguenza, possiamo immaginare un GNFA come un NFA che legga la stringa in input **blocco per blocco**

**Proposizione 6: Stringa accettata in un GNFA**

Sia  $G := (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$  un GNFA. Data una stringa  $w := w_0 \dots w_k \in \Sigma^*$ , dove  $w_0, \dots, w_k \in \Sigma^*$  (ossia sono delle sottostringhe), diciamo che  $w$  è **accettata da  $G$**  se esiste una sequenza di stati  $r_0, r_1, \dots, r_{k+1} \in Q$  tali che:

- $r_0 = q_{\text{start}}$
- $\forall i \in [0, k] \quad w_i \in L(\delta(r_i, r_{i+1}))$
- $r_{k+1} = q_{\text{accept}}$

**Esempio:**

- Il GNFA dell'esempio precedente accetta la stringa **ababaaaba**, poiché:
  - $\delta(q_{\text{start}}, q_1) = \mathbf{ab^*}$ , dunque viene letta in blocco la sottostringa **abab**
  - $\delta(q_1, q_1) = \mathbf{aa^*}$ , dunque viene letta in blocco la sottostringa **aa**
  - $\delta(q_1, q_{\text{accept}}) = \mathbf{ab \cup ba}$ , dunque viene letta in blocco la sottostringa **ba**

**Corollario 3**

Una transizione con "etichetta" pari a  $\emptyset$  è una **transizione inutilizzabile** in quanto  $L(\emptyset) = \emptyset$

**Definizione 24: Classe dei linguaggi riconosciuti da un GNFA**

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi di  $\Sigma$  riconosciuti da un GNFA** il seguente insieme:

$$\mathcal{L}(\text{GNFA}) = \{L \subseteq \Sigma^* \mid \exists \text{ GNFA } G \text{ t.c. } L = L(G)\}$$

**Lemma 2: Conversione da DFA a GNFA**

Date le due classi di linguaggi  $\mathcal{L}(\text{DFA})$  e  $\mathcal{L}(\text{GNFA})$ , si ha che:

$$\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{GNFA})$$

*Dimostrazione.*

- Dato  $L \in \mathcal{L}(\text{DFA})$ , sia  $D := (Q, \Sigma, \delta, q_0, F)$  il DFA tale che  $L(D) = L$
- Consideriamo quindi il GNFA  $G := (Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$  costruito tramite  $D$  stesso:
  - $Q' = Q \cup \{q_{\text{start}}, q_{\text{accept}}\}$
  - $\delta'(q_{\text{start}}, q_0) = \varepsilon$
  - $\forall q \in F \quad \delta'(q, q_{\text{accept}}) = \varepsilon$

- Per ogni transizione con etichetta multipla in  $D$ , in  $G$  esiste una transizione equivalente con etichetta corrispondente all'unione di tali etichette multiple
- Per ogni coppia di stati per cui non esiste una transizione entrante o uscente in  $D$ , viene aggiunta una transizione con etichetta  $\emptyset$
- A questo punto, per costruzione stessa di  $G$  si ha che:

$$w \in L = L(D) \implies L(G)$$

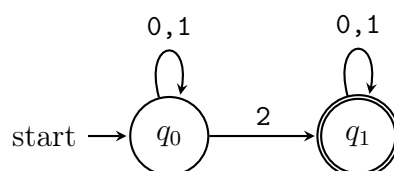
implicando dunque che  $L(D) \in \mathcal{L}(\text{DFA})$  e di conseguenza che:

$$\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{GNFA})$$

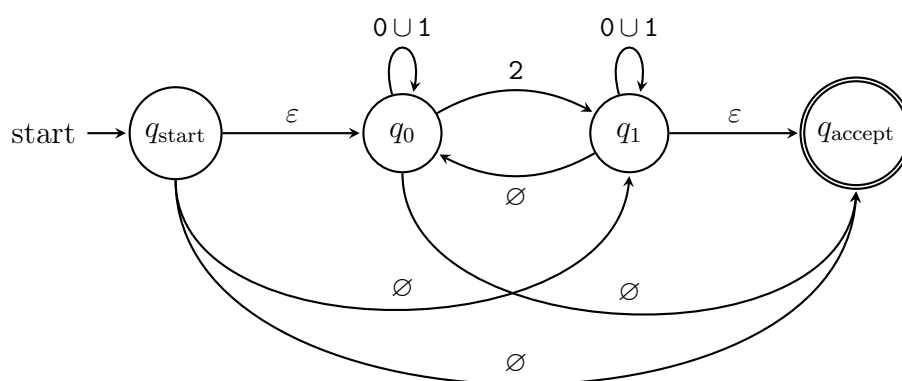
□

### Esempio:

- Consideriamo il seguente DFA:



- Il suo GNFA equivalente corrisponde a:



**Algoritmo 1: Riduzione minimale di un GNFA**

Dato un GNFA  $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ , il seguente algoritmo restituisce un GNFA  $G'$  avente solo due stati e tale che  $L(G) = L(G')$ :

```

function REDUCEGNFA( $G$ )
  if  $|Q| == 2$  then
    return  $G$ 
  else if  $|Q| > 2$  then
     $q := q \in Q - \{q_{\text{start}}, q_{\text{accept}}\}$ 
     $Q' := Q - \{q\}$ 
    for  $q_i \in Q' - \{q_{\text{accept}}\}$  do
      for  $q_j \in Q' - \{q_{\text{start}}\}$  do
         $\delta'(q_i, q_j) := \delta(q_i, q)\delta(q, q)^*\delta(q, q_j) \cup \delta(q_i, q_j)$ 
      end for
    end for
     $G' := (Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$ 
    return reduceGNFA( $G'$ )
  end if
end function

```

*Dimostrazione.*

Siano  $G_0, \dots, G_n$  i vari GNFA prodotti dalla ricorsione dell'algoritmo, implicando che  $G_0 = G$  e che  $G_n$  sia l'output. Procediamo per induzione sul numero  $k \in \mathbb{N}$  di riduzioni effettuate, mostrando che  $L(G) = L(G_0) = \dots = L(G_n)$

*Caso base.*

- Se  $k = 0$ , allora  $G_0 = G$ , dunque  $L(G) = L(G_0)$

*Ipotesi induttiva.*

- Dato  $k \in \mathbb{N}$ , assumiamo che per il GNFA  $G_k := (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$  si abbia che  $L(G) = L(G_k)$

*Passo induttivo.*

- Consideriamo quindi il GNFA  $G_{k+1} := (Q', \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$  ottenuto rimuovendo uno stato  $q \in Q$  (dunque  $Q' = Q - \{q\}$ ) e ponendo

$$\delta'(q_i, q_j) := \delta(q_i, q)\delta(q, q)^*\delta(q, q_j) \cup \delta(q_i, q_j)$$

per ogni  $q_i \in Q' - \{q_{\text{accept}}\}, q_j \in Q' - \{q_{\text{start}}\}$

- Data una stringa  $w := w_0 \dots w_m \in L(G_k)$ , dove  $w_0, \dots, w_m \in \Sigma^*$ , esiste una sequenza di stati  $q_0, \dots, q_m \in Q$  tali che:

- $q_0 = q_{\text{start}}$  e  $q_m = q_{\text{accept}}$
- $\forall i \in [0, m-1] \quad w_i \in L(\delta(q_i, q_{i+1}))$

- A questo punto, consideriamo la costruzione della funzione  $\delta'$ :

$$\delta'(q_i, q_j) = \delta(q_i, q)\delta(q, q)^*\delta(q, q_j) \cup \delta(q_i, q_j)$$

- Se  $q \notin \{q_0, \dots, q_m\}$ , allora tramite l'unione si ha che  $w_i \in L(\delta(q_i, q_j)) \implies w \in L(\delta'(q_i, q_j))$ , dunque tutte le possibili sottostringhe passanti per le transizioni dirette da  $q_i$  a  $q_j$  vengono riconosciute
- Se  $q \in \{q_0, \dots, q_m\}$ , allora la concatenazione  $\delta(q_i, q)\delta(q, q)^*\delta(q, q_j)$  permette il riconoscimento di tutti i cammini da  $q_i$  a  $q_j$  passanti per  $q$ , implicando che  $w \in L(\delta'(q_i, q_j))$
- Viceversa, poiché ogni  $\delta'(q_i, q_j)$  è definito come la combinazione di tutti i cammini possibili da  $q_i$  a  $q_j$  (dunque passando per  $q$  o non), ne segue automaticamente che  $w \in L(G_{k+1}) \implies w \in L(G_k)$
- Esprimendo il tutto graficamente, risulta evidente che le seguenti transizioni siano del tutto equivalenti:

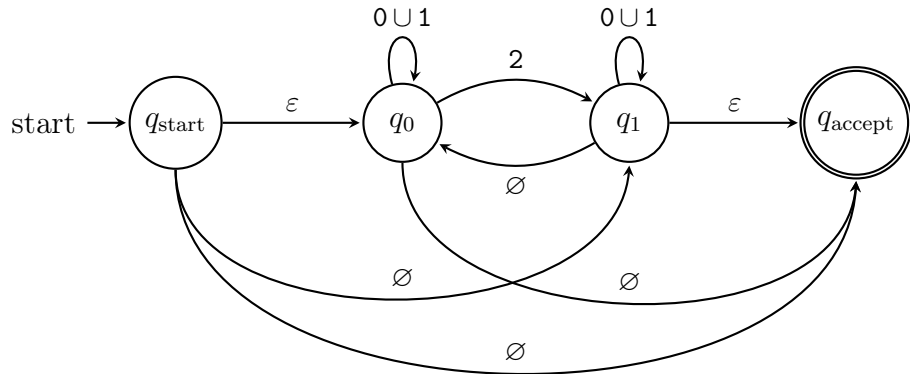


- Di conseguenza, otteniamo che  $w \in L(G_k) \iff w \in L(G_{k+1})$ , concludendo quindi, per ipotesi induttiva, che  $L(G) = L(G_k) = L(G_{k+1})$

□

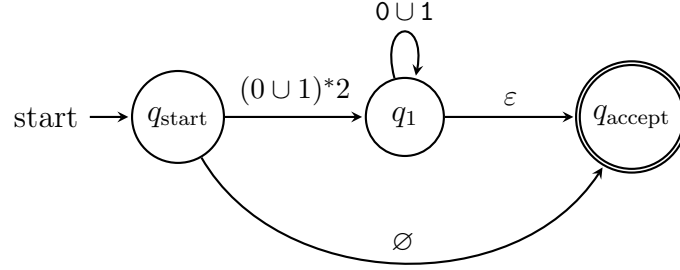
### Esempio:

- Consideriamo nuovamente il seguente GNFA, applicando su esso l'algoritmo **reduceGNFA**:



- Rimuoviamo quindi lo stato  $q_0$  calcolando le nuove transizioni:

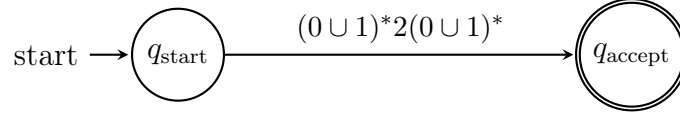
$$\begin{aligned}\delta'(q_{\text{start}}, q_1) &= \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_{\text{start}}, q_1) = \varepsilon(0 \cup 1)^*2 \cup \emptyset = (0 \cup 1)^*2 \\ \delta'(q_{\text{start}}, q_{\text{accept}}) &= \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_{\text{accept}}) \cup \delta(q_{\text{start}}, q_{\text{accept}}) = \varepsilon(0 \cup 1)^*\emptyset \cup \emptyset = \emptyset \\ \delta'(q_1, q_1) &= \delta(q_1, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_1, q_1) = \emptyset(0 \cup 1)^*2 \cup (0 \cup 1) = 0 \cup 1 \\ \delta'(q_1, q_{\text{accept}}) &= \delta(q_1, q_0)\delta(q_0, q_0)^*\delta(q_0, q_{\text{accept}}) \cup \delta(q_1, q_{\text{accept}}) = \emptyset(0 \cup 1)^*\emptyset \cup \varepsilon = \varepsilon\end{aligned}$$



- Infine, rimuoviamo lo stato  $q_1$  calcolando le nuove transizioni:

$$\begin{aligned}\delta''(q_{\text{start}}, q_{\text{accept}}) &= \delta'(q_{\text{start}}, q_1)\delta'(q_1, q_1)^*\delta'(q_1, q_{\text{accept}}) \cup \delta'(q_{\text{start}}, q_{\text{accept}}) = \\ &= (0 \cup 1)^*2(0 \cup 1)^*\varepsilon \cup \emptyset = (0 \cup 1)^*2(0 \cup 1)^*\end{aligned}$$

- Il GNFA minimale, dunque, corrisponde a:



#### Corollario 4: Conversione da GNFA ad espressione regolare

Date le due classi di linguaggi  $\mathcal{L}(\text{GNFA})$  e  $\mathcal{L}(\text{re})$ , si ha che:

$$\mathcal{L}(\text{GNFA}) \subseteq \mathcal{L}(\text{re})$$

*Dimostrazione.*

- Dato  $L \in \mathcal{L}(\text{GNFA})$ , sia  $G := (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$  il GNFA tale che  $L(G) = L$
- Dato il GNFA  $G'$  ottenuto applicando **reduceGNFA**, sia  $R \in \text{re}(\Sigma)$  l'espressione regolare tale che  $R = \delta'(q_{\text{start}}, q_{\text{accept}})$ . Essendo l'unica transizione di  $G'$  ed essendo  $G'$  equivalente a  $G$ , ne segue automaticamente che:

$$L = L(G) = L(G') = L(R) \in \text{re}(\Sigma)$$

da cui traiamo che:

$$\mathcal{L}(\text{GNFA}) \subseteq \mathcal{L}(\text{re})$$

□

### 1.5.2 Equivalenza tra espressioni e linguaggi regolari

#### Teorema 8: Equivalenza tra espressioni e linguaggi regolari

Date le due classi di linguaggi  $\mathcal{L}(\text{re})$  e REG, si ha che:

$$\mathcal{L}(\text{re}) = \text{REG}$$

*Dimostrazione.*

*Prima implicazione.*

- Tramite la [Conversione da espressione regolare a NFA](#), otteniamo che:

$$\mathcal{L}(\text{re}) \subseteq \mathcal{L}(\text{NFA}) = \text{REG}$$

- Inoltre, in quando un NFA è anche un GNFA, ne segue automaticamente che:

$$\mathcal{L}(\text{NFA}) \subseteq \mathcal{L}(\text{GNFA})$$

*Seconda implicazione.*

- Tramite la [Conversione da DFA a GNFA](#) e [Conversione da GNFA ad espressione regolare](#), otteniamo che:

$$\text{REG} = \mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{GNFA}) \subseteq \mathcal{L}(\text{re})$$

□

#### Proposizione 7: Classi dei linguaggi regolari

Dato un alfabeto  $\Sigma$ , si ha che:

$$\text{REG} := \mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA}) = \mathcal{L}(\text{GNFA}) = \mathcal{L}(\text{re})$$

In altre parole, per ogni linguaggio regolare  $L$  esistono un DFA, un NFA e un GNFA che lo riconoscono e un'espressione regolare che lo descrive

## 1.6 Pumping lemma per i linguaggi regolari

Consideriamo il seguente linguaggio composto dalle stringhe aventi un numero uguale di simboli 0 ed 1:

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$

Nel provare a costruire un automa che riconosca tale linguaggio, notiamo che sarebbe necessario che l'automa avesse **infiniti stati**, in quanto esso dovrebbe memorizzare la quantità di simboli 0 ed 1 letti. Di conseguenza, non è possibile costruire un **automa a stati finiti** (dunque un DFA, NFA o GNFA) che riconosca tale linguaggio.

### Lemma 3: Pumping lemma per i linguaggi regolari

Dato un linguaggio  $L$ , se  $L \in \text{REG}$  allora  $\exists p \in \mathbb{N}$ , detto **lunghezza del pumping**, tale che  $\forall w := xyz \in L$ , con  $|w| \geq p$  e  $x, y, z \in \Sigma^*$  (ossia sono sue sottostringhe), si ha che:

- $\forall i \in \mathbb{N} \quad xy^i z \in L$ , ossia è possibile concatenare  $y$  per  $i$  volte rimanendo in  $L$
- $|y| > 0$ , dunque  $y \neq \varepsilon$
- $|xy| \leq p$ , ossia  $y$  deve trovarsi nei primi  $p$  simboli di  $w$

*Dimostrazione.*

- Dato  $L \in \text{REG}$ , sia  $D := (Q, \Sigma, \delta, q_0, F)$  il DFA tale che  $L = L(D)$
- Consideriamo quindi  $p := |Q|$ . Data la stringa  $w := w_1 \dots w_n \in L$  dove  $w_1, \dots, w_n \in \Sigma$  e dove  $n \geq p$ , consideriamo la sequenza di stati  $r_1, \dots, r_{n+1}$  tramite cui  $w$  viene accettata da  $D$ :

$$\forall k \in [1, n] \quad \delta(r_k, w_k) = r_{k+1}$$

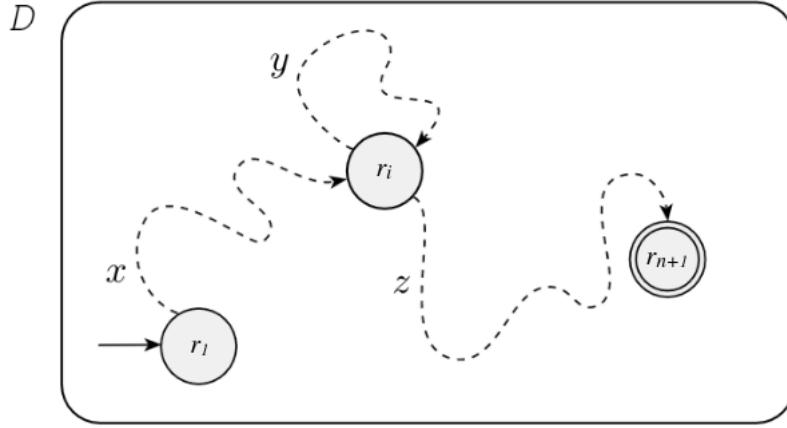
- Notiamo quindi che  $|r_1, \dots, r_{n+1}| = n + 1$ , ossia che il numero di stati attraversati sia  $n + 1$ . Inoltre, in quanto  $n \geq p$ , ne segue automaticamente che  $n + 1 \geq p + 1$ . Tuttavia, poiché  $p := |Q|$  e  $n + 1 \geq p + 1$ , ne segue necessariamente che  $\exists i, j \mid 1 \leq i < j \leq p + 1 \wedge r_i = r_j$ , ossia che tra i primi  $p + 1$  stati della sequenza vi sia almeno uno stato ripetuto
- A questo punto, consideriamo le seguenti sottostringhe di  $w$ :
  - $x = w_1 \dots w_{i-1}$ , tramite cui si ha che  $\delta^*(r_1, x) = r_i$
  - $y = w_i \dots w_{j-1}$ , tramite cui si ha che  $\delta^*(r_i, y) = r_j = r_i$
  - $z = w_j \dots w_n$ , tramite cui si ha che  $\delta^*(r_j, z) = r_n$
- Poiché  $\delta^*(r_i, y) = r_i$ , ossia  $y$  porta sempre  $r_i$  in se stesso, ne segue automaticamente che

$$\forall k \in \mathbb{N} \quad \delta^*(r_i, y^k) = r_i \implies \delta(r_1, xy^k z) \in F \implies xy^k z \in L(D) = L$$



- Inoltre, ne segue direttamente che  $|y| > 0$  in quanto  $i < j$  e che  $|xy| \leq p$  in quanto  $j \leq p + 1$

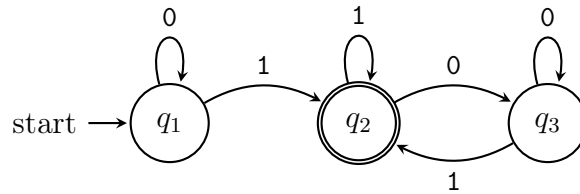
□



Rappresentazione grafica della dimostrazione

**Esempio:**

- Consideriamo il linguaggio  $L = \{x \in \{0, 1\}^* \mid x := y1, \exists y \in \{0, 1\}^*\}$
- Tale linguaggio risulta essere regolare in quanto il seguente DFA è in grado di riconoscerlo:



- Essendo un linguaggio regolare, per esso vale il [Pumping lemma per i linguaggi regolari](#). Ad esempio, preso  $p = 5$  e la stringa  $w := 0100010101 \in L$ , è possibile separare  $w$  in tre sottostringhe  $x := 010$ ,  $y = 00$  e  $z = 10101$  tali che:

- $xy^0z = 01010101 \in L$
- $xy^1z = 0100010101 \in L$
- $xy^2z = 010000010101 \in L$
- $xy^3z = 01000000010101 \in L$
- ...

**Osservazione 7: Dimostrazione di non regolarità**

Il **Pumping lemma per i linguaggi regolari** può essere utilizzato per dimostrare che un linguaggio **non è regolare**

**Esempi:**

- Consideriamo il linguaggio  $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
- Supponiamo per assurdo che  $L$  sia regolare. In tal caso, ne segue che per esso debba valere il pumping lemma, dove  $p$  è la lunghezza del pumping
- Consideriamo quindi la stringa  $w := 0^p 1^p \in L$ . Poiché  $|w| \geq p$ , possiamo suddividerla in tre sottostringhe  $x, y, z \in \Sigma^*$  tali che  $w = xyz$ , per poi procedere con uno dei due seguenti approcci:

**1. Approccio enumerativo:**

- Se  $y$  è composta da soli 0, allora ogni stringa generata dal pumping non sarà in  $L$  in quanto il numero di 0 sarà superiore al numero di 1
- Se  $y$  è composta da soli 1, allora ogni stringa generata dal pumping non sarà in  $L$  in quanto il numero di 1 sarà superiore al numero di 0
- Se  $y$  è composta sia da 0 che da 1, allora ogni stringa generata dal pumping non sarà in  $L$  in quanto esse assumeranno la forma  $0000 \dots 101010 \dots 1111$
- Di conseguenza, poiché in ogni caso viene contraddetto il pumping lemma, ne segue necessariamente che  $L$  non sia regolare

**2. Approccio condizionale:**

- Poiché la terza condizione del pumping lemma impone che  $|xy| \leq p$  e poiché  $w := 0^p 1^p$ , ne segue che  $xy = 0^m$  e  $z = 0^{p-m} 1^p$ , dove  $m \in [1, p]$
- Inoltre, per la seconda condizione, si ha che  $|y| > 0$ , dunque necessariamente si ha che  $x = 0^{m-k}$  e  $y = 0^k$ , dove  $k \in [1, m]$
- A questo punto, consideriamo la stringa  $xy^0 z$ . Notiamo immediatamente che

$$xy^0 z = 0^{m-k} (0^k)^0 0^{p-m} 1^p = 0^{m-k} 0^{p-m} 1^p = 0^{p-k} 1^p$$

implicando dunque che  $xy^0 z \notin L$ , contraddicendo la prima condizione del lemma per cui si ha che  $\forall i \in \mathbb{N} \quad xy^i z \in L$

- Dunque, ne segue necessariamente che  $L$  non sia regolare

## 1.7 Esercizi svolti

### Problema 1: Linguaggio rovesciato

Dato un linguaggio  $L$  e il suo linguaggio rovesciato  $L^R = \{w^R \mid w \in L\}$ , dimostrare che

$$L \in \text{REG} \implies L^R \in \text{REG}$$

*Dimostrazione.*

- Dato  $L \in \text{REG}$ , sia  $D = (Q, \Sigma, \delta, q_0, F)$  il DFA tale che  $L = L(D)$
- Definiamo quindi un primo NFA  $N = (Q', \Sigma, \delta', q_0, \{q_f\})$  tale che:
  - $q_f$  è il nuovo unico stato accettante aggiunto
  - $Q' = Q \cup \{q_f\}$
  - $\forall q \in Q, a \in \Sigma \quad \delta'(q, a) = \delta(q, a)$ , ossia tutti gli archi rimangono invariati
  - $\forall q \in F \quad \delta'(q, \varepsilon) = q_f$ , ossia tutti gli stati finali precedenti hanno un  $\varepsilon$ -arco verso  $q_f$
- A questo punto, per costruzione stessa di  $N$  ne segue che:

$$w \in L = L(D) \iff w \in L(N)$$

dunque che  $L = L(D) = L(N)$

- Definiamo quindi un secondo NFA  $N^R = (Q', \Sigma, \delta'', q_f, \{q_0\})$  tale che:

$$\forall p, q \in Q', a \in \Sigma \quad \delta'(p, a) = q \implies \delta''(q, a) = p$$

ossia avente tutti gli archi invertiti rispetto ad  $N$

- A questo punto, per costruzione stessa di  $N^R$  ne segue che:

$$w \in L = L(N) \iff w^R \in L(N^R)$$

dunque che  $L^R = L(N)^R = L(N^R) \in \text{REG}$

□

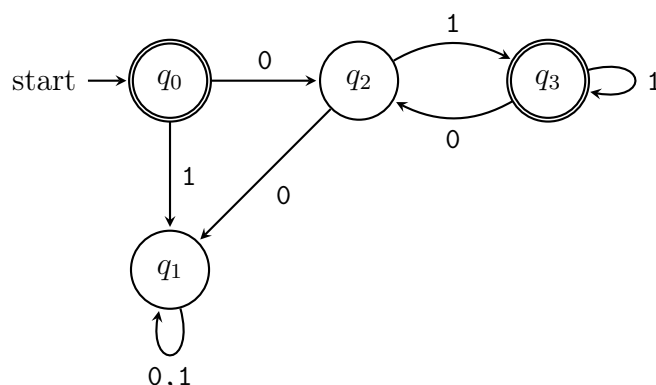
**Problema 2: Complemento di un'espressione regolare**

Data l'espressione regolare  $R = (01^+)^*$ , costruire il DFA  $D$  tale che:

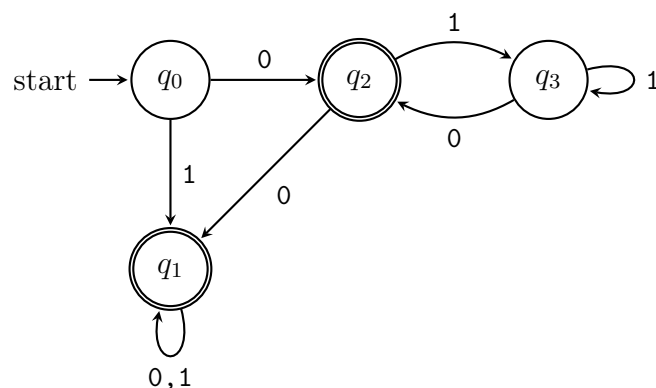
$$L(D) = \{w \in \{0, 1\}^* \mid w \notin L(R)\}$$

*Soluzione:*

- Prima di tutto, costruiamo un DFA  $D_R$  tale che  $L(D_R) = L(R)$ :



- A questo punto, ci basta costruire il DFA  $D$  tale che  $L(D) = \overline{L(D_R)}$  utilizzando la [Chiusura del complemento in REG](#):

**Problema 3**

Dato il linguaggio  $L = \{w \in \{0, 1\}^* \mid |w|_0 = |w|_1\}$ , dimostrare che  $L \notin \text{REG}$

*Dimostrazione.*

- Supponiamo per assurdo che  $L$  sia regolare, implicando che per esso debba valere il pumping lemma, dove  $p$  è la lunghezza del pumping
- Consideriamo quindi la stringa  $w := 0^p 1^p \in L$ . Poiché  $|w| \geq p$ , possiamo suddividerla in tre sottostringhe  $x, y, z \in \Sigma^*$  tali che  $w = xyz$

- Poiché la terza condizione del pumping lemma impone che  $|xy| \leq p$  e poiché  $w := 0^p 1^p$ , ne segue che  $xy = 0^m$  e  $z = 0^{p-m} 1^p$ , dove  $m \in [1, p]$
- Inoltre, per la seconda condizione, si ha che  $|y| > 0$ , dunque necessariamente si ha che  $x = 0^{m-k}$  e  $y = 0^k$ , dove  $k \in [1, m]$
- A questo punto, consideriamo la stringa  $xy^0z$ . Notiamo immediatamente che

$$xy^0z = 0^{m-k}(0^k)^0 0^{p-m} 1^p = 0^{m-k} 0^{p-m} 1^p = 0^{p-k} 1^p$$

$$\implies |xy^0z|_0 \neq |xy^0z|_1 \implies xy^0z \notin L$$

contraddicendo la prima condizione del lemma per cui si ha che  $\forall i \in \mathbb{N} \quad xy^i z \in L$

- Dunque, ne segue necessariamente che  $L$  non sia regolare

□

#### Problema 4

Dato il linguaggio  $L = \{1^{n^2} \mid n \in \mathbb{N}\}$ , dimostrare che  $L \notin \text{REG}$

*Dimostrazione.*

- Supponiamo per assurdo che  $L$  sia regolare, implicando che per esso debba valere il pumping lemma, dove  $p$  è la lunghezza del pumping
- Consideriamo quindi la stringa  $w := 1^{p^2} \in L$ . Poiché  $|w| \geq p$ , possiamo suddividerla in tre sottostringhe  $x, y, z \in \Sigma^*$  tali che  $w = xyz$
- Poiché la terza condizione del lemma impone che  $|xy| \leq p$  e poiché  $w := 1^{p^2}$ , ne segue che  $xy = 1^m$  e  $z = 1^{p^2-m}$ , dove  $m \in [1, p]$
- Inoltre, per la seconda condizione del lemma, si ha che  $|y| > 0$ , dunque necessariamente si ha che  $x = 1^{m-k}$  e  $y = 1^k$ , dove  $k \in [1, m]$
- A questo punto, consideriamo la stringa  $xy^0z$ . Notiamo immediatamente che

$$xy^0z = 1^{m-k}(1^k)^0 1^{p^2-m} = 1^{p^2-k}$$

- Tuttavia, poiché  $k \in [1, p]$ , ne segue che  $\nexists n \in \mathbb{N} \mid n^2 = p^2 - k$ , implicando dunque che  $xy^0z \notin L$ , contraddicendo la prima condizione del lemma per cui si ha che  $\forall i \in \mathbb{N} \quad xy^i z \in L$
- Dunque, ne segue necessariamente che  $L$  non sia regolare

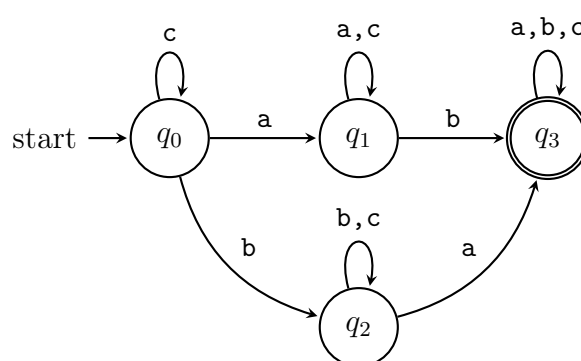
□

**Problema 5**

Sia  $\Sigma = \{a, b, c\}$ . Determinare un'espressione regolare  $R \in \text{re}(\Sigma)$  descrivente il linguaggio di  $\Sigma$  composto dalle stringhe contenenti almeno una  $a$  ed almeno una  $b$ . Determinare inoltre un DFA  $D$  che riconosca lo stesso linguaggio.

*Soluzione:*

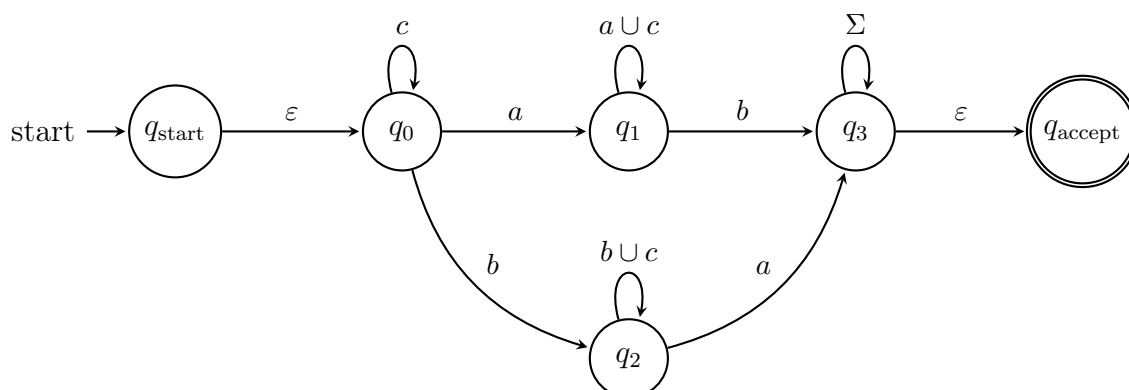
- Nonostante il problema inviti alla determinazione dell'espressione regolare e poi del DFA ad essa equivalente, trovare quest'ultimo risulta molto più rapido
- Difatti, il DFA  $D$  in grado di riconoscere il linguaggio richiesto corrisponde a:



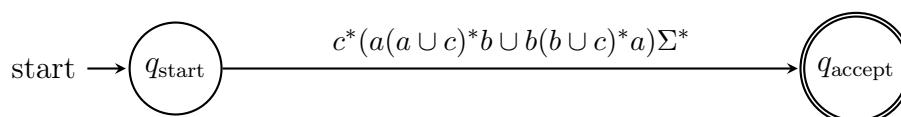
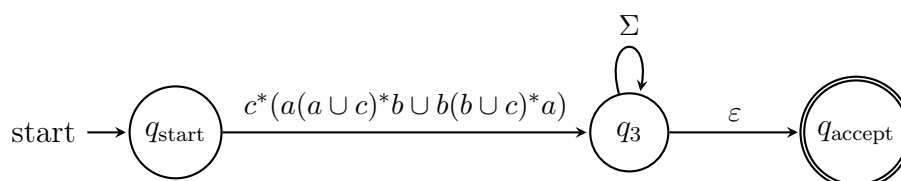
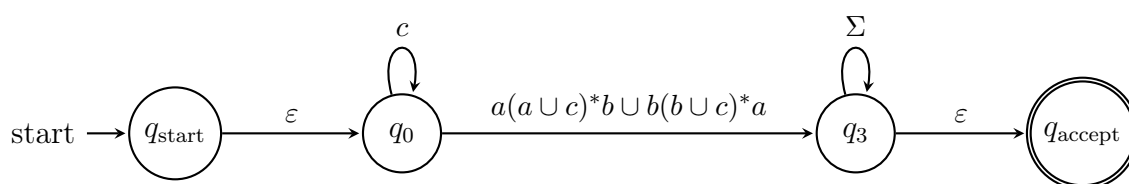
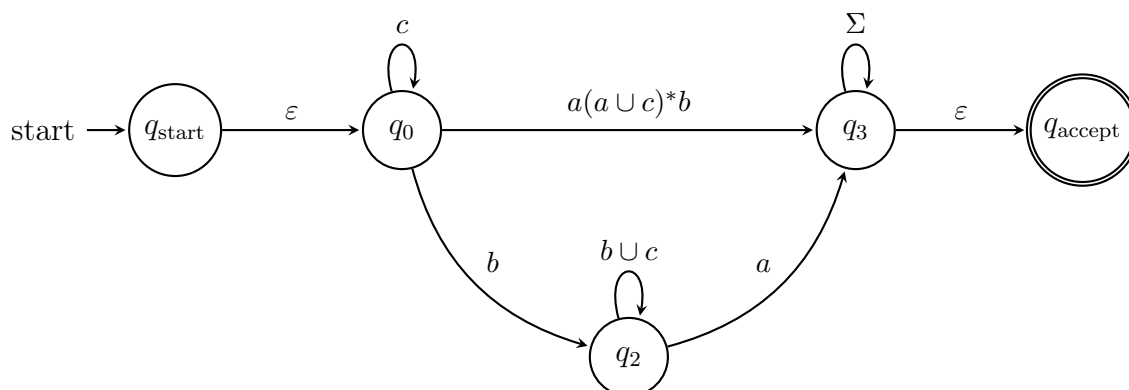
- A questo punto, osservando il DFA possiamo già notare che l'espressione regolare ad esso equivalente corrisponde a:

$$c^*(a(a \cup c)^*b \cup b(a \cup c)^*a)\Sigma^*$$

- Volendo procedere più rigorosamente, possiamo ricavare tale espressione regolare convertendo il DFA costruito nel suo GNFA equivalente, per poi ridurre al minimo tale GNFA, ottenendo l'espressione regolare
- Definiamo quindi il GNFA equivalente (del quale vengono omesse le sue transizioni etichettate con  $\emptyset$ ):



- Procediamo quindi con la riduzione:



- Come anticipato, l'espressione regolare ottenuta corrisponde a:

$$c^*(a(a \cup c)^*b \cup b(b \cup c)^*a)\Sigma^*$$

# Linguaggi acontestuali

## 2.1 Grammatiche acontestuali

### Definizione 25: Context-free Grammar (CFG)

Una **Context-free Grammar (CFG)** (o *Grammatica acontestuale*) è una quadrupla  $(V, \Sigma, R, S)$  dove:

- $V$  è l'insieme delle **variabili** della grammatica
- $\Sigma$  è l'insieme dei **terminali** della grammatica e
- $R$  è l'insieme delle **regole** o **produzioni** della grammatica
- $S \in V$  è la **variabile iniziale** della grammatica
- $V \cap \Sigma = \emptyset$ , ossia variabili e terminali sono tutti distinti tra loro

Le **regole** in  $R$  assumono la forma  $A \rightarrow X$ , dove  $A \in V$ , ossia è una variabile, e  $X \in (V \cup \Sigma_\epsilon)^*$ , ossia è una stringa composta da una o più variabili e/o terminali.

### Esempio:

- La seguente quadrupla  $G = (\{A, B\}, \{0, 1, \#\}, R, A)$  è una CFG dove in  $R$  sono definite le seguenti regole:

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$



**Osservazione 8: Acontestualità**

Con **acontestualità** intendiamo la condizione secondo cui il lato sinistro delle regole della grammatica è composto sempre e solo da **una singola variabile**.

**Esempio:**

- La regola  $A \rightarrow B$  può appartenere ad una CFG
- La regola  $AB \rightarrow B$  non può appartenere ad una CFG

**Osservazione 9: Notazione contratta per le regole**

Data una CFG  $G = (V, \Sigma, R, S)$ , se in  $R$  esistono più regole  $A \rightarrow X_1, X_2, \dots, A \rightarrow X_n$  definite sulla stessa variabile  $A$ , è possibile indicare tali regole con la seguente notazione contratta:

$$A \rightarrow X_1 \mid X_2 \mid \dots \mid X_n$$

**Esempio:**

- Le regole della CFG dell'esempio precedente possono essere contratte in:

$$A \rightarrow 0A1 \mid B$$

$$B \rightarrow \#$$

**Definizione 26: Produzione**

Sia  $G = (V, \Sigma, R, S)$  una CFG. Se  $u, v, w$  sono stringhe di variabili o terminali ed esiste la regola  $A \rightarrow w$ , allora la stringa  $uAv$  **produce** la stringa  $uwv$ , denotato come  $uAv \Rightarrow uwv$ .

$$u, v, w \in (V \cup \Sigma)^*, A \rightarrow w \in R \implies uAv \Rightarrow uwv$$

**Esempio:**

- Consideriamo la grammatica  $G = (\{A, B\}, \{0, 1, \#\}, R, A)$  dove:

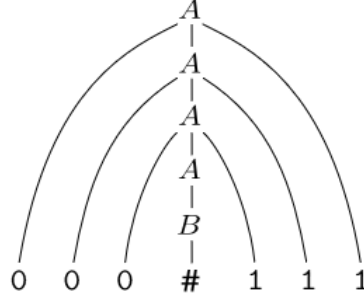
$$A \rightarrow 0A1 \mid B$$

$$B \rightarrow \#$$

- Tramite le regole di  $G$  è possibile ottenere la stringa  $000\#111$  attraverso la seguente catena di produzioni:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000\#111$$

- Tale catena può anche essere descritta graficamente dal seguente **albero di produzione**:



### Definizione 27: Derivazione

Sia  $G = (V, \Sigma, R, S)$  un CFG. Date  $u, v \in (V \cup \Sigma)^*$ , diciamo che  $u$  **deriva**  $v$ , denotato come  $u \Rightarrow^* v$ , se  $u = v$  oppure se  $\exists u_1, \dots, u_k \in (V \cup \Sigma)^*$  tali che:

$$u \Rightarrow u_1 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$$

### Definizione 28: Context-free Language (CFL)

Sia  $G = (V, \Sigma, R, S)$  una CFG. Definiamo come **Context-free Language (CFL)** (o *Linguaggio acontestuale*) **generato da**  $G$ , indicato come  $L(G)$ , l'insieme di stringhe derivate dalle regole di  $G$  tramite la variabile  $S$ :

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

### Esempi:

1. Data la CFG  $G = (\{S\}, \{a, b\}, R, S)$ , dove:

$$S \rightarrow \varepsilon \mid aSb \mid SS$$

si ha che:

- $S \Rightarrow aSb \Rightarrow a\varepsilon b = ab$ , dunque  $ab \in L(G)$
- $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\varepsilon bb = aabb$ , dunque  $aabb \in L(G)$
- $S \Rightarrow SS \xRightarrow{*} aSbaSb \xRightarrow{*} a\varepsilon ba\varepsilon b = abab$ , dunque  $abab \in L(G)$

2. Data la CFG  $G = (\{S, T\}, \{0, 1\}, R, S)$ , dove:

$$S \rightarrow T1T1T1T$$

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

si ha che:

$$L(G) = \{w \in \{0, 1\}^* \mid |w|_1 \geq 3\}$$

3. Data la CFG  $G = (\{S\}, \{0, 1\}, R, S)$ , dove:

$$S \rightarrow \varepsilon \mid 0S0 \mid 1S1$$

si ha che:

$$L(G) = \{w \in \{0, 1\}^* \mid w = w^R \wedge |w| \equiv 0 \pmod{2}\}$$

4. Data la CFG  $G = (\{S, T\}, \{a, b, c\}, R, S)$ , dove:

$$S \rightarrow aSc \mid T$$

$$T \rightarrow bTc \mid \varepsilon$$

si ha che:

$$L(G) = \{a^i b^j c^{i+j} \in \Sigma^* \mid i, j \in \mathbb{N}\}$$

### Osservazione 10

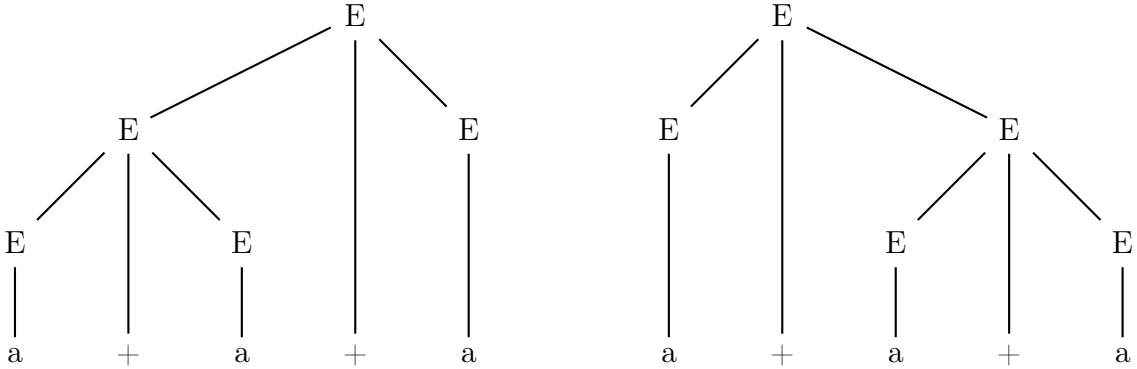
Sia  $G$  una CFG. Data la stringa  $w \in L(G)$ , possono esistere più derivazioni di  $w$

**Esempio:**

- Data la CFG

$$E \rightarrow E + E \mid E \cdot E \mid (E) \mid a$$

la stringa  $a + a + a$  può essere derivata in due modi:



### Definizione 29: Derivazione a sinistra

Data una CFG  $G = (V, \Sigma, R, S)$ , definiamo la derivazione  $S \xRightarrow{*} w$  come **derivazione sinistra** se ad ogni produzione interna alla derivazione viene valutata la variabile più a sinistra

**Esempio:**

- Riprendiamo la CFG dell'esempio precedente:

$$E \rightarrow E + E \mid E \cdot E \mid (E) \mid a$$

- Per maggior chiarezza, riscriviamo tali regole come:

$$E \rightarrow E + F \mid E \cdot E \mid (E) \mid a$$

$$F \rightarrow E$$

ottenendo una CFG del tutto equivalente alla precedente

- Una derivazione sinistra della stringa  $a + a + a$  corrisponde a:

$$E \Rightarrow E + F \Rightarrow E + F + F \Rightarrow a + F + F \Rightarrow a + E + F \Rightarrow a + a + F \Rightarrow a + a + E \Rightarrow a + a + a$$

### Osservazione 11

L'uso delle derivazioni a sinistra permette di fissare un "ordine", rimuovendo la maggior parte delle derivazioni multiple per una stessa stringa.

Tuttavia, in alcune grammatiche possono esistere più di una derivazione a sinistra per la stessa stringa.

### Definizione 30: Grammatica ambigua

Definiamo una grammatica  $G$  come **ambigua** se  $\exists w \in L(G)$  tale che esistono almeno due derivazioni a sinistra per  $w$

## 2.2 Linguaggi acontestuali ad estensione dei regolari

### Definizione 31: Classe dei linguaggi acontestuali

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi acontestuali di  $\Sigma$**  il seguente insieme:

$$\text{CFL} = \{L \subseteq \Sigma^* \mid \exists \text{ CFG } G \text{ t.c. } L = L(G)\}$$

### Lemma 4: Conversione da DFA a CFG

Date le due classi di linguaggi REG e CFL, si ha che:

$$\text{REG} \subseteq \text{CFL}$$

*Dimostrazione.*

- Dato  $L \in \text{REG}$ , sia  $D = (Q, \Sigma, \delta, q_0, F)$  il DFA tale che  $L = L(D)$
- Consideriamo quindi la CFG  $G = (V, \Sigma, R, S)$  tale che:
  - Esiste una funzione biettiva  $\varphi : Q \rightarrow V : q_i \mapsto V_i$

$$- S = \varphi(q_0) = V_0$$

- Dati  $q_i, q_j \in Q$  e  $a \in \Sigma$ , si ha che:

$$\delta(q_i, a) = q_j \implies \varphi(q_i) \rightarrow a\varphi(q_j) \implies V_i \rightarrow aV_j$$

$$- q_f \in F \implies \varphi(q_f) \rightarrow \varepsilon \implies V_f \rightarrow \varepsilon$$

• A questo punto, per costruzione stessa di  $G$  si ha che:

$$w \in L(D) \implies w \in L(G)$$

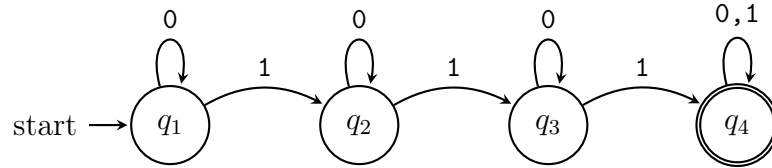
implicando dunque che  $L(D) \in \text{CFL}$  e di conseguenza che:

$$\text{REG} \subseteq \text{CFL}$$

□

**Esempio:**

• Consideriamo il seguente DFA



• Una CFG  $G = (V, \Sigma, R, S)$  equivalente è costituita da:

$$- V = \{V_1, V_2, V_3, V_4\}$$

$$- S = V_1$$

-  $R$  definito come:

$$V_1 \rightarrow 0V_1 \mid 1V_2$$

$$V_2 \rightarrow 0V_2 \mid 1V_3$$

$$V_3 \rightarrow 0V_3 \mid 1V_4$$

$$V_4 \rightarrow 0V_4 \mid 1V_4 \mid \varepsilon$$

• Difatti, sia il DFA sia la CFG descrivono il seguente linguaggio:

$$L = \{w \in \Sigma^* \mid |w|_1 \geq 3\}$$

**Teorema 9: Ling. acontestuali estensione dei ling. regolari**

Date le due classi di linguaggi REG e CFL, si ha che:

$$\text{REG} \subsetneq \text{CFL}$$

*Dimostrazione.*

- Tramite la [Conversione da DFA a CFG](#), sappiamo che  $\text{REG} \subseteq \text{CFL}$
- Consideriamo quindi il linguaggio  $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
- Tale linguaggio è generabile dalla grammatica  $G = (\{S\}, \{0, 1\}, R, S)$ , dove:

$$S \rightarrow 0S1 \mid \varepsilon$$

dunque abbiamo che  $L = L(G) \in \mathcal{L}(\text{CFG})$

- Tuttavia, abbiamo già dimostrato nella sezione [1.6](#) che  $L$  non sia regolare, dunque abbiamo che  $L \notin \text{REG}$
- Di conseguenza, concludiamo che:

$$\text{REG} \subsetneq \text{CFL}$$

□

## 2.3 Forma normale di Chomsky

**Definizione 32: Chomsky's Normal Form (CNF)**

Una CFG  $G = (V, \Sigma, R, S)$  viene detta in **Chomsky's Normal Form (CNF)** (o *Forma Normale di Chomsky*) se tutte le regole in  $R$  assumono una delle seguenti tre forme:

$$A \rightarrow BC \qquad A \rightarrow a \qquad S \rightarrow \varepsilon$$

dove  $A \in V$ ,  $a \in \Sigma$  e  $B, C \in V - \{S\}$

**Teorema 10: Conversione in Forma Normale di Chomsky**

Per ogni CFG  $G$ , si ha che:

$$\exists \text{ CFG } G' \text{ in CNF} \mid L(G) = L(G')$$

*Dimostrazione.*

- Data una CFG  $G = (V, \Sigma, R, S)$ , costruiamo una CFG  $G'$  in CNF equivalente a  $G$ :
  1. Vengono aggiunte una variabile  $S_0$  e una regola  $S_0 \rightarrow S$ , dove  $S_0$  è la **nuova variabile iniziale**
  2. Finché in  $R$  esiste una  $\varepsilon$ -**regola**  $A \rightarrow \varepsilon$  dove  $A \in V - \{S_0\}$ , tale regola viene **eliminata** e per ogni regola in  $R$  contenente delle occorrenze di  $A$  vengono **aggiunte** delle regole in cui vengono eliminate tutte le possibili combinazioni di occorrenze di  $A$ 

(es: se viene rimossa  $A \rightarrow \varepsilon$  e in  $R$  esiste  $B \rightarrow uAvAw \mid u, v, w \in (V \cup \Sigma)^*$ , vengono aggiunte le regole  $B \rightarrow uvAw \mid uAvw \mid uvw$ )
  3. Ogni regola nella forma  $A \rightarrow B$  (dette **regole unitarie**) per cui esiste una regola nella forma  $B \rightarrow u \mid u \in (V \cup \Sigma)^*$  viene **sostituita** con la regola  $A \rightarrow u$
  4. Per ogni regola  $A \rightarrow u_1 \dots u_k$  dove  $k \geq 3$  e  $u \in (V \cup \Sigma)$ , vengono **aggiunte** le variabili  $A_1, \dots, A_k$  e le seguenti regole:

$$A \rightarrow u_1 A_1 \quad \dots \quad A_{k-3} \rightarrow u_{k-2} A_{k-2} \quad A_{k-2} \rightarrow u_{k-1} u_k$$

per poi eliminare la regola iniziale  $A \rightarrow u_1 u_2 \dots u_k$

5. Per ogni regola rimanente nella forma  $A \rightarrow u_1 u_2 \mid u_1, u_2 \in (V \cup \Sigma)$ , se  $u_1 \in \Sigma$  allora viene aggiunta una variabile  $U_1$  ed una regola  $U_1 \rightarrow u_1$ , sostituendo la regola  $A \rightarrow u_1 u_2$  con la regola  $A \rightarrow U_1 u_2$ . Analogamente, lo stesso viene svolto se  $u_2 \in \Sigma$ .
- Poiché le operazioni svolte dall'algoritmo non modificano le stringhe generabili dalla CFG, ne segue automaticamente che  $L(G) = L(G')$

□

**Esempio:**

- Consideriamo la seguente grammatica  $G$  non in CNF, dove  $S$  è la variabile iniziale:

$$\begin{aligned} G: \quad S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

- Aggiungiamo la nuova variabile iniziale  $S_0$  e la regola  $S_0 \rightarrow S$ :

$$\begin{aligned} G: \quad S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

- Eliminiamo la  $\varepsilon$ -regola  $B \rightarrow \varepsilon$ :

$$\begin{aligned} G : S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid \mathbf{a} \\ A &\rightarrow B \mid S \mid \varepsilon \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

- Eliminiamo la  $\varepsilon$ -regola  $A \rightarrow \varepsilon$ :

$$\begin{aligned} G : S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid a \mid \mathbf{SA} \mid \mathbf{AS} \mid S \\ A &\rightarrow B \mid S \mid \varepsilon \\ B &\rightarrow b \end{aligned}$$

- Eliminiamo la regola unitaria  $S \rightarrow S$ :

$$\begin{aligned} G : S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

- Eliminiamo la regola unitaria  $S_0 \rightarrow S$ :

$$\begin{aligned} G : S_0 &\rightarrow S \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

- Eliminiamo le regole unitarie  $A \rightarrow B$  e  $A \rightarrow S$ :

$$\begin{aligned} G : S_0 &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow B \mid S \mid \mathbf{b} \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\ B &\rightarrow b \end{aligned}$$

- Separiamo ogni regola con tre o più elementi a destra in regole con massimo due elementi a destra:

$$\begin{aligned} G : S_0 &\rightarrow ASA \mid \mathbf{AA_1} \mid aB \mid a \mid SA \mid AS \\ S &\rightarrow ASA \mid \mathbf{AA_1} \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow b \mid ASA \mid \mathbf{AA_1} \mid aB \mid a \mid SA \mid AS \\ \mathbf{A_1} &\rightarrow \mathbf{SA} \\ B &\rightarrow b \end{aligned}$$



- Infine, convertiamo tutte le regole aventi due elementi a destra di cui almeno uno è un terminale:

$$\begin{aligned}
G: \quad S_0 &\rightarrow AA_1 \mid \mathbf{aB} \mid \mathbf{UB} \mid a \mid SA \mid AS \\
S &\rightarrow AA_1 \mid \mathbf{aB} \mid \mathbf{UB} \mid a \mid SA \mid AS \\
A &\rightarrow b \mid AA_1 \mid \mathbf{aB} \mid \mathbf{UB} \mid a \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow \mathbf{a} \\
B &\rightarrow b
\end{aligned}$$

- La grammatica finale ottenuta risulta sia equivalente a quella iniziale sia in forma normale di Chomsky:

$$\begin{aligned}
G: \quad S_0 &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
S &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
A &\rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow a \\
B &\rightarrow b
\end{aligned}$$

## 2.4 Automi a pila

### Definizione 33: Pushdown Automaton (PDA)

Un **Pushdown Automaton (PDA)** (o *Automa a pila*) è una sestupla  $(Q, \Sigma, \Gamma, \delta, q_0, F)$  dove:

- $Q$  è l'**insieme finito degli stati** dell'automa
- $\Sigma$  è l'**alfabeto** dell'automa
- $\Gamma$  è l'**alfabeto** dello stack (o *pila*) dell'automa
- $q_0 \in Q$  è lo **stato iniziale** dell'automa
- $F \subseteq Q$  è l'**insieme degli stati accettanti** dell'automa
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$  è la **funzione di transizione** dell'automa, dove se  $(q, c) \in \delta(p, a, b)$  si ha che:
  - Viene letto il simbolo  $a$  dalla stringa in input e se il simbolo  $b$  è in cima allo stack allora l'automa passa dallo stato  $p$  allo stato  $q$  e il simbolo  $b$  viene sostituito dal simbolo  $c$
  - L'etichetta della transizione da  $p$  a  $q$  viene indicata come  $a; b \rightarrow c$

**Osservazione 12**

Dato  $(q, c) \in \delta(p, a, b)$  dove  $\delta$  è la funzione di transizione di un PDA, si ha che:

- Se  $b, c = \varepsilon$  (dunque  $a; \varepsilon \rightarrow \varepsilon$ ) allora l'automa leggerà  $a$  dalla stringa e passerà direttamente dallo stato  $p$  allo stato  $q$ , senza modificare lo stack
- Se  $b = \varepsilon$  e  $c \neq \varepsilon$  (dunque  $a; \varepsilon \rightarrow c$ ) allora l'automa leggerà  $a$  dalla stringa, passerà direttamente dallo stato  $p$  allo stato  $q$  e in cima allo stack viene aggiunto il simbolo  $c$  (**push**)
- Se  $b \neq \varepsilon$  e  $c = \varepsilon$  (dunque  $a; b \rightarrow \varepsilon$ ) allora l'automa leggerà  $a$  e se in cima allo stack vi è  $b$ , l'automa passerà dallo stato  $p$  allo stato  $q$  e rimuoverà  $b$  dalla cima dello stack (**pop**)

**Esempio:**

- Consideriamo il seguente PDA:



- Data la stringa **aab**, uno dei possibili rami di computazione del PDA procede nel seguente ordine:
  1. Viene letta la prima **a** e viene inserita la prima **c** in cima allo stack, rimanendo nello stato  $q_1$ .
  2. Viene letta la seconda **a** e viene inserita la seconda **c** in cima allo stack, rimanendo nello stato  $q_1$ .
  3. Viene letta la **b**, passando da  $q_1$  a  $q_2$  e lasciando lo stack inalterato
  4. Viene "letta" la prima  $\varepsilon$ , rimuovendo la seconda **c** dallo stack (poiché essa è in cima), rimanendo nello stato  $q_2$ .
  5. Viene "letta" la seconda  $\varepsilon$ , rimuovendo la prima **c** dallo stack (poiché essa è in cima), rimanendo nello stato  $q_2$ .
  6. Sia la stringa che lo stack sono vuoti, dunque la computazione termina necessariamente poiché non vi sono transizioni percorribili
- Notiamo in particolare che, in tal caso, la stringa verrebbe accettata anche se la computazione si fermasse al terzo passo
- Difatti, lo stack non deve necessariamente esser vuoto affinché la stringa possa essere accettata

**Proposizione 8: Stringa accettata in un PDA**

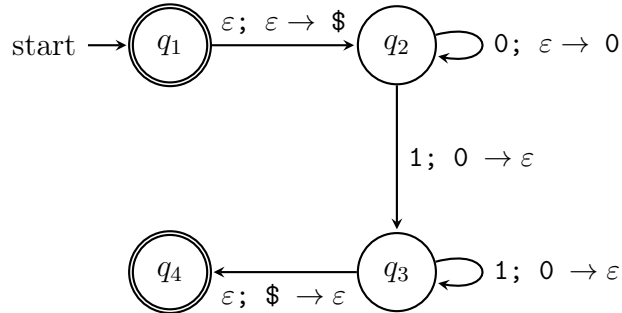
Sia  $P := (Q, \Sigma, \Gamma, \delta, q_0, F)$  un PDA. Data una stringa  $w := w_0 \dots w_k \in \Sigma^*$ , dove  $w_0, \dots, w_k \in \Sigma_\varepsilon$ , diciamo che  $w$  è **accettata da  $P$**  se esiste una sequenza di stati  $r_0, r_1, \dots, r_{k+1} \in Q$  ed una sequenza di stringhe  $s_1, \dots, s_n \in \Gamma^*$  tali che:

- $r_0 = q_0$
- $r_{k+1} \in F$
- $s_0 = \varepsilon$ , dunque lo stack è inizialmente vuoto
- $\forall i \in [0, k]$  si abbia che:
  - $(r_{i+1}, b) \in \delta(r_i, w_i, a)$
  - $s_i = at$
  - $s_{i+1} = bt$

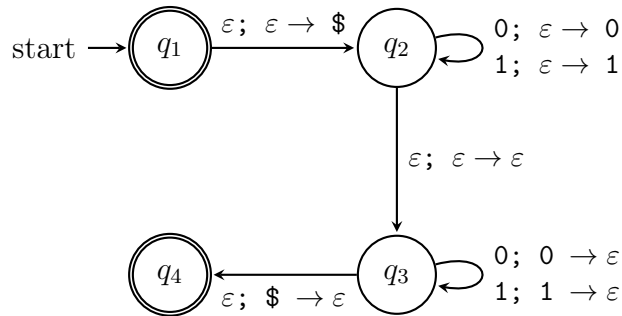
dove  $a, b \in \Gamma_\varepsilon$  e dove  $t \in \Gamma^*$  è la stringa composta dai caratteri nello stack

**Esempi:**

- Il seguente automa riconosce il linguaggio  $L = \{0^n 1^n \mid n \in \mathbb{N}\}$



- Il seguente automa riconosce il linguaggio  $L = \{ww^R \mid w \in \{0, 1\}^*\}$



### 2.4.1 Equivalenza tra CFG e PDA

#### Definizione 34: Classe dei linguaggi riconosciuti da un PDA

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi di  $\Sigma$  riconosciuti da un PDA** il seguente insieme:

$$\mathcal{L}(\text{PDA}) = \{L \subseteq \Sigma^* \mid \exists \text{ PDA } P \text{ t.c. } L = L(P)\}$$

#### Proposizione 9: Scrittura di una stringa sullo stack

Sia  $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$  un PDA. Dati  $u_1, \dots, u_k \in \Gamma$ , introduciamo una notazione per cui  $\delta$  possa ammettere la scrittura diretta sullo stack della stringa  $u := u_1 \dots u_k$ .

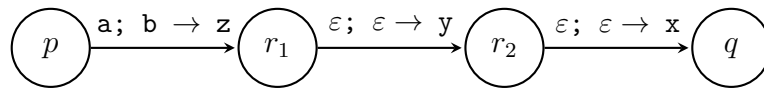
Formalmente, diciamo che:

$$(q, u_1 \dots u_k) \in \delta(p, a, b) \iff \exists r_1, \dots, r_{k-1} \in Q \text{ tali che:}$$

- $\delta(p, a, b) \ni (r_1, u_k)$
- $\delta(r_1, \varepsilon, \varepsilon) = \{(r_2, u_{k-1})\}$
- ...
- $\delta(r_{k-1}, \varepsilon, \varepsilon) = \{(q, u_1)\}$

**Esempio:**

- Dato  $(q, xyz) \in \delta(p, a, b)$  si ha che:



#### Lemma 5: Conversione da CFG a PDA

Date le due classi di linguaggi CFL e  $\mathcal{L}(\text{PDA})$ , si ha che:

$$\text{CFL} \subseteq \mathcal{L}(\text{PDA})$$

*Dimostrazione.*

- Dato  $L \in \text{CFL}$ , sia  $G = (V, \Sigma, R, S)$  la CFG tale che  $L = L(G)$
- Consideriamo quindi il PDA  $P = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, F)$  tale che:
  - $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup Q_\delta$ , dove  $Q_\delta$  sono i minimi stati aggiunti affinché la sua funzione  $\delta$  sia ben definita (vedi i punti successivi)
  - $\Gamma = V \cup \Sigma$

–  $F = \{q_{\text{accept}}\}$

– Dato  $q_{\text{start}} \in Q$  si ha che

$$\delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{(q_{\text{loop}}, S\$)\}$$

–  $\forall A \in V$  si ha che

$$\delta(q_{\text{loop}}, \varepsilon, A) = \{(q_{\text{loop}}, u) \mid (A \rightarrow u) \in R, u \in \Gamma^*\}$$

–  $\forall a \in \Sigma$  si ha che

$$\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \varepsilon)\}$$

– Dato  $q_{\text{accept}} \in Q$  si ha che

$$\delta(q_{\text{loop}}, \varepsilon, \$) = \{(q_{\text{accept}}, \varepsilon)\}$$

- A questo punto, per costruzione stessa di  $P$  si ha che:

$$w \in L = L(G) \iff w \in L(P)$$

dunque che  $L = L(P) \in \mathcal{L}(\text{PDA})$

□

### Esempio:

- Consideriamo la seguente grammatica:

$$\begin{aligned} G : \quad S &\rightarrow aTb \mid b \\ T &\rightarrow Ta \mid \varepsilon \end{aligned}$$

- Il PDA in grado di riconoscere  $L(G)$  corrisponde a:



**Lemma 6: Conversione da PDA a CFG**

Date le due classi di linguaggi  $\mathcal{L}(\text{PDA})$  e CFL, si ha che:

$$\mathcal{L}(\text{PDA}) \subseteq \text{CFL}$$

*Dimostrazione.*

- Dato  $L \in \mathcal{L}(\text{PDA})$ , sia  $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$  il PDA tale che  $L = L(P)$
- Consideriamo il PDA  $P' = (Q', \Sigma, \Gamma, \delta', q_0, \{q_{\text{accept}}\})$  tale che:
  - Ogni transizione effettua solo un'operazione di push o di pop, ma mai una sostituzione diretta:

$$(q, c) \in \delta(p, a, b) \implies \exists r \in Q' \mid (r, \varepsilon) \in \delta'(p, a, b) \wedge \delta'(r, \varepsilon, \varepsilon) = \{(q, c)\}$$

- $Q' = Q \cup Q_{\delta'} \cup \{q_{\text{accept}}\}$ , dove  $Q_{\delta'}$  sono gli stati aggiunti per il punto precedente
- $q_{\text{accept}} \in Q'$  è il nuovo unico stato accettante:

$$\forall q \in F \quad (q_{\text{accept}}, \varepsilon) \in \delta'(q, \varepsilon, \varepsilon)$$

- Lo stack deve essere svuotato prima di poter accettare una stringa:

$$\forall q \in F, a \in \Sigma \quad (q, \varepsilon) \in \delta'(q, \varepsilon, a)$$

- A questo punto, per costruzione stessa di  $P'$  si ha che:

$$w \in L(P) \iff w \in L(P')$$

dunque che  $L = L(P) = L(P')$

- Consideriamo quindi la CFG  $G = (V, \Sigma, R, S)$  tale che:
  - $V = \{A_{p,q} \mid p, q \in Q'\}$
  - $S = A_{q_0, q_{\text{accept}}}$
  - Ogni variabile  $A_{p,q}$  è grado di derivare tutte le stringhe generabili passando dallo stato  $p$  allo stato  $q$ :

- \*  $\forall p \in Q'$  si ha che:

$$(A_{p,p} \rightarrow \varepsilon) \in R$$

- \*  $\forall p, q, r, s \in Q', u \in \Gamma$  e  $a, b \in \Sigma_\varepsilon$  si ha che:

$$(r, u) \in \delta'(p, a, \varepsilon) \wedge (q, \varepsilon) \in \delta(s, b, u) \iff (A_{p,q} \rightarrow aA_{r,s}b) \in R$$

- \*  $\forall p, q, r \in Q'$  si ha che:

$$(A_{p,q} \rightarrow A_{p,r}A_{r,q}) \in R$$

- **Affermazione:** dati  $p, q \in Q'$  e  $x \in \Sigma^*$ , se  $A_{p,q} \xRightarrow{*} x$  allora  $x$  porta il PDA  $P'$  dallo stato  $p$  allo stato  $q$  con uno stack vuoto:

*Dimostrazione.*

Procediamo per induzione sul numero  $n$  di produzioni che compongono la derivazione  $A_{p,q} \xRightarrow{*} x$

*Caso base.*

- Per  $n = 1$ , la derivazione è composta da una sola produzione. Di conseguenza, l'unica regola possibile affinché  $A_{p,q} \Rightarrow x$  è la regola  $A_{p,q} \rightarrow \varepsilon$ , implicando che  $p = q$  e che  $x = \varepsilon$ , dunque la stringa  $x$  porta correttamente il PDA  $P'$  dallo stato  $p$  allo stato  $q$  con uno stack vuoto

*Ipotesi induttiva forte.*

- Assumiamo che per ogni stringa  $x \in \Sigma^*$  derivabile da  $A_{p,q}$  (dunque tale che  $A_{p,q} \xRightarrow{*} x$ ) tramite  $k \leq n$  produzioni, tale stringa  $x$  porti il PDA  $P'$  da  $p$  a  $q$  con uno stack vuoto

*Passo induttivo.*

- Consideriamo la derivazione  $A_{p,q} \xRightarrow{*} x$  composta da  $n + 1$  produzioni. Poiché tale derivazione è composta da almeno due produzioni, la prima produzione deve essere necessariamente data dalla regola  $A_{p,q} \rightarrow aA_{r,s}b$  o dalla regola  $A_{p,q} \rightarrow A_{p,r}A_{r,q}$

- (a) Consideriamo il caso in cui  $A_{p,q} \Rightarrow aA_{r,s}b \xRightarrow{*} x$ .

Sia  $x = ayb$ , dove  $A_{r,s} \xRightarrow{*} y$ . Poiché  $A_{r,s} \xRightarrow{*} y$  è composta da  $n$  produzioni, per ipotesi induttiva la stringa  $y$  porta il PDA  $P'$  da  $r$  ad  $s$  con uno stack vuoto.

Inoltre, per costruzione stessa di  $G$ , tale regola di derivazione si ha che:

$$(r, u) \in \delta'(p, a, \varepsilon) \wedge (q, \varepsilon) \in \delta(s, b, u) \iff (A_{p,q} \rightarrow aA_{r,s}b) \in R$$

dunque concludiamo che:

$$\left. \begin{array}{l} a \text{ porta } P' \text{ da } p \text{ in } r \\ y \text{ porta } P' \text{ da } r \text{ in } s \\ b \text{ porta } P' \text{ da } s \text{ in } q \end{array} \right\} \implies x = ayb \text{ porta } P' \text{ da } p \text{ in } q$$

- (b) Consideriamo il caso in cui  $A_{p,q} \Rightarrow A_{p,r}A_{r,q} \xRightarrow{*} x$ .

Sia  $x = yz$ , dove  $A_{p,r} \xRightarrow{*} y$  e  $A_{r,q} \xRightarrow{*} z$ . Poiché  $A_{p,r} \xRightarrow{*} y$  è composta da  $m \leq n$  produzioni e  $A_{r,q} \xRightarrow{*} z$  da  $n - m \leq n$  produzioni, per ipotesi induttiva le stringhe  $y$  e  $z$  portano il PDA  $P'$  rispettivamente da  $p$  ad  $r$  e da  $r$  a  $q$  con uno stack vuoto, dunque concludiamo che:

$$\left. \begin{array}{l} y \text{ porta } P' \text{ da } p \text{ in } r \\ z \text{ porta } P' \text{ da } r \text{ in } q \end{array} \right\} \implies x = yz \text{ porta } P' \text{ da } p \text{ in } q$$

- **Affermazione:** dati  $p, q \in Q'$  e  $x \in \Sigma^*$ , se la stringa  $x$  porta il PDA  $P'$  dallo stato  $p$  allo stato  $q$  con uno stack vuoto allora  $A_{p,q} \xRightarrow{*} x$

*Dimostrazione.*

Procediamo per induzione sul numero  $n$  di transizioni percorse da  $P'$  durante la lettura di  $x$

*Caso base.*

- Per  $n = 0$ , il PDA percorre zero transizioni, dunque  $x = \varepsilon$  e  $x$  porta il PDA da  $p$  a  $p$ . Pertanto, la regola  $A_{p,p} \rightarrow \varepsilon$  soddisfa la derivazione  $A_{p,p} \Rightarrow x$

*Ipotesi induttiva forte.*

- Assumiamo che per ogni stringa  $x \in \Sigma^*$  che porta il PDA  $P'$  da  $p$  a  $q$  con uno stack vuoto percorrendo  $k \leq n$  transizioni, si abbia che  $A_{p,q} \xRightarrow{*} x$

*Passo induttivo.*

- Consideriamo la stringa  $x \in \Sigma^*$  che porta il PDA  $P'$  da  $p$  a  $q$  con uno stack vuoto percorrendo  $n + 1$  transizioni. A seconda dell'evolvere dello stack durante la computazione, abbiamo due casi:

- (a) Se lo stack risulta vuoto solo all'inizio e alla fine della computazione, ciò implica che  $\exists u \in \Gamma$  inserito nella prima transizione e rimosso solo nell'ultima.

Sia quindi  $a \in \Sigma_\varepsilon$  il simbolo letto durante tale prima transizione. In tal caso,  $\exists r, s \in Q'$  tali che:

$$(r, u) \in \delta(p, a, \varepsilon) \wedge (q, \varepsilon) \in \delta(s, b, u)$$

Sia quindi  $x = ayb$ , dove  $y$  è una stringa che porta  $P'$  da  $r$  a  $s$ . Affinché la computazione di  $x$  termini con lo stack vuoto, è necessario che ciò valga anche per la computazione di  $y$ .

Poiché la computazione di  $y$  percorre  $n - 1$  transizioni, per ipotesi induttiva abbiamo che  $A_{r,s} \xRightarrow{*} y$ , dunque data la regola  $A_{p,q} \rightarrow aA_{r,s}b$  concludiamo che:

$$A_{p,q} \Rightarrow aA_{r,s}b \xRightarrow{*} ayb = x$$

- (b) Se lo stack si svuota durante la computazione, ciò implica che  $\exists r \in Q'$  percorso durante la computazione di  $x$  in cui ciò accade.

Sia quindi  $x = yz$ , dove  $y$  e  $z$  sono due stringhe che portano  $P'$  rispettivamente da  $p$  a  $r$  e da  $r$  a  $q$ .

Poiché le computazioni di  $y$  e  $z$  percorrono rispettivamente  $m \leq n$  e  $n - m \leq n$  transizioni, per ipotesi induttiva abbiamo che  $A_{p,r} \xRightarrow{*} y$  e  $A_{r,q} \xRightarrow{*} z$ , dunque data la regola  $A_{p,q} \rightarrow A_{p,r}A_{r,q}$  concludiamo che:

$$A_{p,q} \Rightarrow A_{p,r}A_{r,q} \xRightarrow{*} yz = x$$



- Tramite le due affermazioni, abbiamo che:

$$A_{q_0, q_{\text{accept}}} \xRightarrow{*} x \iff x \text{ porta } P' \text{ da } q_0 \text{ in } q_{\text{accept}} \text{ con uno stack vuoto}$$

da cui concludiamo che:

$$x \in L(G) \iff A_{q_0, q_{\text{accept}}} \iff x \in L(P')$$

dunque che  $L = L(P) = L(P') = L(G) \in \text{CFL}$

□

### Teorema 11: Equivalenza tra CFG e PDA

Date le due classi di linguaggi  $\mathcal{L}(\text{PDA})$  e  $\text{CFL}$ , si ha che:

$$\mathcal{L}(\text{PDA}) = \text{CFL}$$

(segue dai due lemmi precedenti)

## 2.5 Pumping lemma per i linguaggi acontestuali

### Proposizione 10: Altezza delle derivazioni in una CFG in CNF

Sia  $G = (V, \Sigma, R, S)$  una CFG in CNF. Data  $x \in L(G)$  e data l'altezza  $h$  dell'albero di derivazione di  $x$ , si ha che  $|x| \leq 2^{h-1}$

*Dimostrazione.* Procediamo per induzione sull'altezza  $h$  dell'albero della derivazione  $S \xRightarrow{*} x$

*Caso base.*

- Per  $h = 1$ , la derivazione è composta da una sola produzione. Essendo  $G$  in CNF, l'unica regola applicabile è nella forma  $S \rightarrow a$ , dove  $x = a \in \Sigma$ , implicando che  $|x| = 1 \leq 2^{1-1} = 1$

*Ipotesi induttiva forte.*

- Assumiamo che data  $x \in L(G)$  tale che il suo albero di derivazione abbia altezza  $k \leq h$  si abbia che  $|x| \leq 2^{k-1}$

*Passo induttivo.*

- Consideriamo la stringa  $x$  il cui albero di derivazione ha altezza  $h+1$ . Poiché  $G$  è in CNF, la prima produzione di tale derivazione deve essere ottenuta tramite una regola nella forma  $S \rightarrow AB$ .

- Sia quindi  $x = yz$ , dove  $A \xRightarrow{*} y$  e  $B \xRightarrow{*} z$ . Poiché la derivazione  $S \Rightarrow AB \xRightarrow{*} yz = x$  ha altezza  $h + 1$ , ne segue che l'altezza dei due sottoalberi delle derivazioni  $A \xRightarrow{*} y$  e  $B \xRightarrow{*} z$  sia  $h$
- Di conseguenza, per ipotesi induttiva si ha che  $|y| \leq 2^{h-1}$  e  $|z| \leq 2^{h-1}$ , implicando che:

$$|x| = |y| + |z| \leq 2^{h-1} + 2^{h-1} = 2^h = 2^{(h+1)-1}$$

□

### Lemma 7: Pumping lemma per i linguaggi acontestuali

Dato un linguaggio  $L$ , se  $L \in \text{CFL}$  allora  $\exists p \in \mathbb{N}$ , detto **lunghezza del pumping**, tale che  $\forall w := uvxyz \in L$ , con  $|w| \geq p$  e  $u, v, x, y, z \in \Sigma^*$  (ossia sono sue sottostringhe), si ha che:

- $\forall i \in \mathbb{N} \quad uv^i xy^i z \in L$
- $|vy| > 0$ , dunque  $v \neq \varepsilon$  o  $y \neq \varepsilon$
- $|vxy| \leq p$

*Dimostrazione.*

- Dato  $L \in \text{CFL}$ , sia  $G = (V, \Sigma, R, S)$  la CFG in CNF tale che  $L = L(G)$
- Sia  $p = 2^{|V|}$ . Data una stringa  $w \in L$  tale che  $|w| \geq p$ , per la proposizione precedente l'albero di derivazione di  $w$  deve avere un'altezza  $h \geq |V| + 1$ , poiché altrimenti  $w$  non sarebbe generabile da esso
- Consideriamo quindi un cammino di lunghezza  $h$  di tale albero, dunque passante per almeno  $k \geq |V| + 2$  nodi. Trattandosi di un cammino all'interno di un albero di derivazione, solo l'ultimo nodo del cammino corrisponderà ad un terminale, implicando che in tale cammino vi siano  $k - 1 \geq |V| + 1$  variabili.
- Sia quindi  $A_1, \dots, A_{k-1}$  la sequenza di variabili del cammino (dove  $S = A_1$ ). Poiché  $k - 1 \geq |V| + 1 \geq |V|$ , ne segue necessariamente che  $\exists i, j \mid k - |V| - 2 \leq i < j \leq k - 1 \wedge A_i = A_j$ , ossia che tra le ultime  $|V| + 1$  variabili del cammino vi sia almeno una variabile ripetuta
- Consideriamo quindi le cinque sottostringhe  $u, v, x, y, z \in \Sigma^*$  tali che:
  - $w = uvxyz$
  - $S \xRightarrow{*} uA_iz$
  - $A_i \xRightarrow{*} vA_jy$
  - $A_j \xRightarrow{*} x$

- Poiché  $A_i = A_j$ , all'interno di ogni derivazione  $A_i \xRightarrow{*} vA_jy$  possiamo sostituire  $A_j$  con  $A_i$  stesso. Ripetendo tale procedimento  $i \in \mathbb{N}$  volte ricorsivamente, otteniamo che:

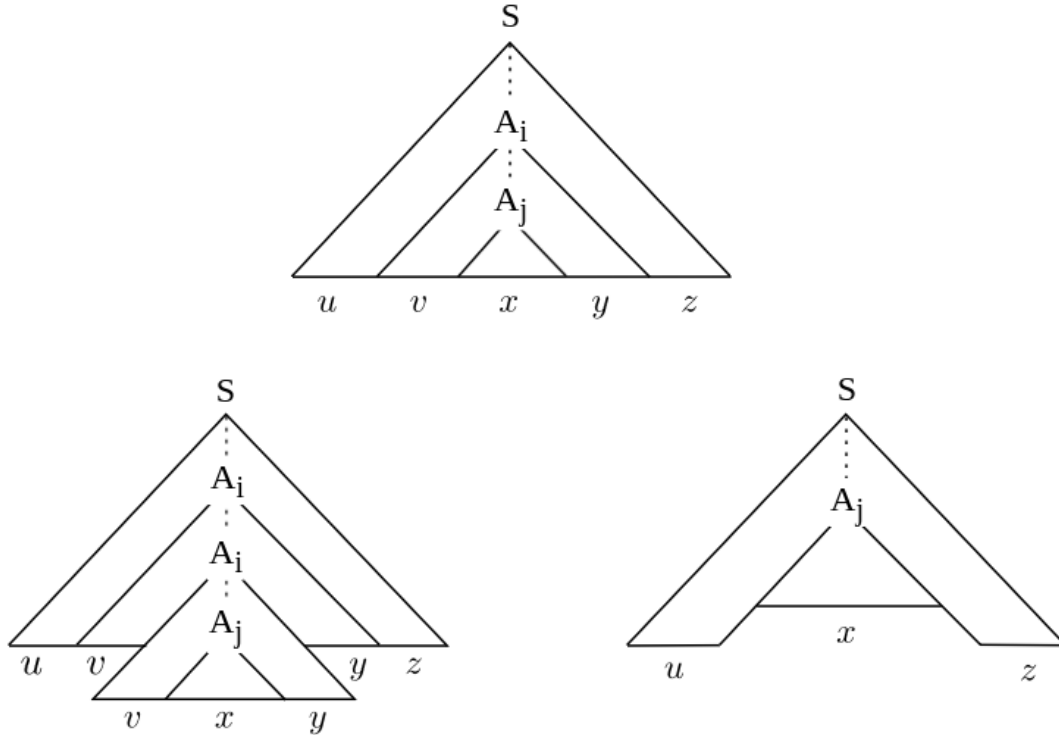
$$A_i \xRightarrow{*} vA_jy = vA_iy \xRightarrow{*} v^iA_jy^i \Rightarrow v^ixy^i$$

implicando dunque che  $\forall i \in \mathbb{N} \ S \xRightarrow{*} uv^ixy^iz$  e quindi che  $uv^ixy^iz \in L(G) = L$

- Poiché  $G$  è in CNF, dunque al suo interno non possono esserci  $\varepsilon$ -regole o regole unitarie, la derivazione  $A_i \xRightarrow{*} vA_jy$  deve necessariamente aver utilizzato una regola del tipo  $A_i \rightarrow BC$  dove  $B \xRightarrow{*} vA_j$  e  $C \xRightarrow{*} y$  oppure  $B \xRightarrow{*} v$  e  $C \xRightarrow{*} A_jy$ . Poiché non vi sono  $\varepsilon$ -regole, in entrambi i casi si ha che  $v \neq \varepsilon$  o  $y \neq \varepsilon$ , implicando che  $|vy| > 0$
- Poiché  $A_i$  si trova tra le ultime  $|V| + 1$  variabili del cammino, ne segue che il suo sottoalbero abbia altezza  $h' \leq |V| + 1$  (contando anche il terminale finale). Per la proposizione precedente, dunque, ne segue che:

$$|vxy| \leq 2^{h'-1} \leq 2^{|V|} = p$$

□



*Rappresentazione grafica della dimostrazione*

**Esempio:**

1.
  - Consideriamo il linguaggio  $L = \{0^n 1^n 2^n \mid n \in \mathbb{N}\}$
  - Supponiamo per assurdo che  $L \in \text{CFL}$ . In tal caso, ne segue che per esso debbia valere il pumping lemma, dove  $p$  è la lunghezza del pumping
  - Consideriamo quindi la stringa  $w := 0^p 1^p 2^p$ . Poiché  $|w| \geq p$ , possiamo suddividerla in cinque sottostringhe  $u, v, x, y, z \in \Sigma^*$  tali che  $w = uvxyz$ .
  - Poiché la terza condizione del pumping lemma impone che  $|vxy| \leq p$ , le uniche possibilità sono:
    - (a) Se  $vxy = 0^m$  con  $1 \leq m \leq p$ , si ha che  $u = 0^h$  e  $z = 0^{p-m-h} 1^p 2^p$ , dove  $1 \leq m+h \leq p$ . Inoltre, poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  e/o  $y$  contengono almeno uno 0
    - (b) Se  $vxy = 1^m$  con  $1 \leq m \leq p$ , si ha che  $u = 0^p 1^h$  e  $z = 1^{p-m-h} 2^p$ , dove  $1 \leq m+h \leq p$ . Inoltre, poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  e/o  $y$  contengono almeno un 1
    - (c) Se  $vxy = 2^m$  con  $1 \leq m \leq p$ , si ha che  $u = 0^p 1^p$  e  $z = 2^{p-m-h}$ , dove  $1 \leq m+h \leq p$ . Inoltre, poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  e/o  $y$  contengono almeno un 2
    - (d) Se  $vxy = 0^m 1^h$  con  $1 \leq m+h \leq p$ , si ha che  $u = 0^{p-m}$  e  $z = 1^{p-h} 2^p$ . Inoltre, poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  contiene almeno uno 0 e/o  $y$  contiene almeno un 1
    - (e) Se  $vxy = 1^m 2^h$  con  $1 \leq m+h \leq p$ , si ha che  $u = 0^p 1^{p-m}$  e  $z = 2^{p-h}$ . Inoltre, poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  contiene almeno uno 1 e/o  $y$  contiene almeno un 2
  - In tutti i casi possibili descritti, risulta automatico che

$$\nexists n \in \mathbb{N} \mid n = |uv^0xy^0z|_0 = |uv^0xy^0z|_1 = |uv^0xy^0z|_2 \implies uv^0xy^0z \notin L$$

contraddicendo quindi la prima condizione del pumping lemma

- Di conseguenza, ne segue necessariamente che  $L \notin \text{CFL}$
2.
    - Consideriamo il linguaggio  $L = \{ww \mid w \in \{0,1\}^*\}$
    - Supponiamo per assurdo che  $L \in \text{CFL}$ . In tal caso, ne segue che per esso debbia valere il pumping lemma, dove  $p$  è la lunghezza del pumping
    - Consideriamo quindi la stringa  $w := 0^p 1^p 0^p 1^p$ . Poiché  $|w| \geq p$ , possiamo suddividerla in cinque sottostringhe  $u, v, x, y, z \in \Sigma^*$  tali che  $w = uvxyz$ .

- Poiché la terza condizione del pumping lemma impone che  $|vxy| \leq p$ , le uniche possibilità sono:

- (a) Se  $u = 0^h$ ,  $vxy = 0^m$  e  $z = 0^{p-m-h}1^p0^p1^p$ , dove  $1 \leq m+h \leq p$ , poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  e/o  $y$  contengono almeno uno 0, dunque si ha che:

$$\exists k < m \mid v^0xy^0 = 0^k \implies uv^0xy^0z = 0^h0^k0^{p-m-h}1^p0^p1^p = 0^{p-m+k}1^p0^p1^p$$

dove  $k < m \implies p-m-k < p$  e dunque che  $uv^0xy^0z \notin L$

- (b) Se  $u = 0^p1^p0^h$ ,  $vxy = 0^m$  e  $z = 0^{p-m-h}1^p$ , dove  $1 \leq m+h \leq p$ , procedendo analogamente al caso (a) otteniamo che  $uv^0xy^0z \notin L$
- (c) Se  $u = 0^p1^h$ ,  $vxy = 1^m$  e  $z = 1^{p-m-h}0^p1^p$ , dove  $1 \leq m+h \leq p$ , procedendo analogamente al caso (a) otteniamo che  $uv^0xy^0z \notin L$
- (d) Se  $u = 0^p1^p0^p1^h$ ,  $vxy = 1^m$  e  $z = 1^{p-m-h}$ , dove  $1 \leq m+h \leq p$ , procedendo analogamente al caso (a) otteniamo che  $uv^0xy^0z \notin L$
- (e) Se  $u = 0^{p-h}$ ,  $vxy = 0^h1^m$  e  $z = 1^{p-m}0^p1^p$ , dove  $1 \leq m+h \leq p$ , poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  contiene almeno uno 0 e/o  $y$  contiene almeno un 1, dunque si ha che:

$$\exists j < h, j < m \mid v^0xy^0 = 0^j1^k \implies$$

$$uv^0xy^0z = 0^{p-h}0^j1^k1^{p-m}0^p1^p = 0^{p-h+j}1^{p-m+k}0^p1^p$$

dove  $j < h, k < m \implies p-h+j, p-m+k < p$  e dunque che  $uv^0xy^0z \notin L$

- (f) Se  $u = 0^p1^p0^{p-h}$ ,  $vxy = 0^h1^m$  e  $z = 1^{p-m}$ , dove  $1 \leq m+h \leq p$ , procedendo analogamente al caso (e) otteniamo che  $uv^0xy^0z \notin L$
- (g) Se  $u = 0^p1^{p-h}$ ,  $vxy = 1^h0^m$  e  $z = 0^{p-m}1^p$ , dove  $1 \leq m+h \leq p$ , poiché la seconda condizione impone che  $|vy| > 0$ , si ha che  $v$  contiene almeno uno 1 e/o  $y$  contiene almeno un 0, dunque si ha che:

$$\exists j < h, j < m \mid v^0xy^0 = 1^j0^k \implies$$

$$uv^0xy^0z = 0^p1^{p-h}1^j0^k0^{p-m}1^p = 0^p1^{p-h+j}0^{p-m+k}1^p$$

dove  $j < h, k < m \implies p-h+j, p-m+k < p$  e dunque che  $uv^0xy^0z \notin L$

- Di conseguenza, poiché il pump down non può essere effettuato nè in un blocco di soli 0 o soli 1 (casi  $a, b, c, d$ ), nè a cavallo tra degli 0 ed 1 (casi  $e, f$ ), nè al centro della stringa (caso  $g$ ), ne segue che la prima condizione del pumping lemma venga contraddetta
- Di conseguenza, ne segue necessariamente che  $L \notin \text{CFL}$

## 2.6 Chiusure dei linguaggi acontestuali

### Teorema 12: Chiusura dell'unione in CFL

L'operatore unione è **chiuso in CFL**, ossia:

$$\forall L_1, \dots, L_n \in \text{CFL} \quad L_1 \cup \dots \cup L_n \in \text{CFL}$$

*Dimostrazione.*

- Dati  $L_1, \dots, L_n \in \text{CFL}$ , siano  $G_1, \dots, G_n$  le tali che  $\forall i \in [1, n] \quad G_i = (V_i, \Sigma_i, R_i, S_i) \wedge L_i = L(G_i)$ .
- Consideriamo quindi la CFG  $G = (V, \Sigma, R, S)$  tale che:
  - $S$  è una nuova variabile iniziale
  - $V = \left( \bigcup_{i=1}^n V_i \right) \cup \{S\}$
  - $\Sigma = \bigcup_{i=1}^n \Sigma_i$
  - $R = \left( \bigcup_{i=1}^n R_i \right) \cup \{S \rightarrow S_j \mid j \in [1, n]\}$
- Data  $w \in \bigcup_{i=1}^n L(G_i)$ , si ha che  $\exists j \in [1, n] \mid w \in L(G_j)$

Di conseguenza, poiché  $(S \rightarrow S_j) \in R$ , ne segue che

$$w \in L(G_j) \iff S_j \xRightarrow{*} w \implies S \Rightarrow S_j \xRightarrow{*} w \implies w \in L(G)$$

- Data  $w \in L(G)$ , invece, dove  $w \in L(G) \iff S \xRightarrow{*} w$ , poiché le uniche regole applicabili su  $S$  sono  $\{S \rightarrow S_j \mid j \in [1, n]\}$ , ne segue necessariamente che:

$$w \in L(G) \implies \exists j \in [1, n] \mid S \Rightarrow S_j \xRightarrow{*} w \implies w \in L(G_j) \subseteq \bigcup_{i=1}^n L(G_i)$$

- Di conseguenza, concludiamo che:

$$L_1 \cup \dots \cup L_n = L(G_1) \cup \dots \cup L(G_n) = L(G) \in \text{CFL}$$

□

**Teorema 13: Chiusura della concatenazione in CFL**

L'operatore concatenazione è **chiuso in CFL**, ossia:

$$\forall L_1, \dots, L_n \in \text{CFL} \quad L_1 \circ \dots \circ L_n \in \text{CFL}$$

*Dimostrazione.*

- Dati  $L_1, \dots, L_n \in \text{CFL}$ , siano  $G_1, \dots, G_n$  le tali che  $\forall i \in [1, n] \quad G_i = (V_i, \Sigma_i, R_i, S_i) \wedge L_i = L(G_i)$ .
- Consideriamo quindi la CFG  $G = (V, \Sigma, R, S)$  tale che:

–  $S$  è una nuova variabile iniziale

$$– V = \left( \bigcup_{i=1}^n V_i \right) \cup \{S\}$$

$$– \Sigma = \bigcup_{i=1}^n \Sigma_i$$

$$– R = \left( \bigcup_{i=1}^n R_i \right) \cup \{S \rightarrow S_1 \dots S_n\}$$

- Sia  $w := w_1 \dots w_n \in L(G_1) \circ \dots \circ L(G_n)$ , dove  $\forall j \in [1, n] \quad w_j \in L(G_j)$

Poiché  $(S \rightarrow S_1 \dots S_n) \in R$ , ne segue che

$$\forall j \in [1, n] \quad w_j \in L(G_j) \iff S_j \xRightarrow{*} w_j$$

dunque abbiamo che:

$$S \Rightarrow S_1 \dots S_n \xRightarrow{*} w_1 \dots w_n = w \implies w \in L(G)$$

- Data  $w \in L(G)$ , invece, dove  $w \in L(G) \iff S \xRightarrow{*} w$ , poiché l'unica regola applicabile su  $S$  è  $S \rightarrow S_1 \dots S_n$ , ne segue necessariamente che:

$$w \in L(G) \implies S \Rightarrow S_1 \dots S_n \xRightarrow{*} w$$

dunque  $\exists w_1 \in L(G_1), \dots, w_n \in L(G_n)$  tali che:

$$S \Rightarrow S_1 \dots S_n \xRightarrow{*} w_1 S_2 \dots S_n \xRightarrow{*} w_1 w_2 \dots w_n = w$$

implicando che:

$$w = w_1 w_2 \dots w_n \in L(G_1) \circ \dots \circ L(G_n)$$

- Di conseguenza, concludiamo che:

$$L_1 \circ \dots \circ L_n = L(G_1) \circ \dots \circ L(G_n) = L(G) \in \text{CFL}$$

□

**Esempio:**

- Consideriamo i seguenti linguaggi:

$$L_1 = \{0^n 1^n \mid n \in \mathbb{N}\} \quad L_2 = \{1^m 0^m \mid m \in \mathbb{N}\}$$

- Consideriamo quindi le due grammatiche:

$$G_1 : A \rightarrow 0A1 \mid \varepsilon$$

$$G_2 : B \rightarrow 1A0 \mid \varepsilon$$

tali che  $L_1 = L(G_1)$  e  $L_2 = L(G_2)$

- La grammatica  $G$  tale che  $L(G) = L_1 \cup L_2$ , corrisponderà a:

$$\begin{aligned} G : \quad & S \rightarrow A \mid B \\ & A \rightarrow 0A1 \mid \varepsilon \\ & B \rightarrow 0B1 \mid \varepsilon \end{aligned}$$

- La grammatica  $G'$  tale che  $L(G') = L_1 \circ L_2$ , corrisponderà a:

$$\begin{aligned} G : \quad & S \rightarrow AB \\ & A \rightarrow 0A1 \mid \varepsilon \\ & B \rightarrow 0B1 \mid \varepsilon \end{aligned}$$

**Teorema 14: Chiusura di star in CFL**

L'operatore star è **chiuso in CFL**, ossia:

$$\forall L \in \text{CFL} \quad L^* \in \text{CFL}$$

*Dimostrazione.*

- Dato  $L \in \text{CFL}$ , sia  $G = (V, \Sigma, R, S)$  la CFG tale che  $L = L(G)$ .
- Consideriamo quindi la CFG  $G' = (V, \Sigma, R', S_0)$  tale che:
  - $S_0$  è una nuova variabile iniziale
  - $R' = R \cup \{S_0 \rightarrow \varepsilon, S_0 \rightarrow S, S_0 \rightarrow S_0 S_0\}$
- Data  $w := w_1 \dots w_n \in L^*$ , abbiamo che:
  - Se  $w = \varepsilon$ , poiché  $(S_0 \rightarrow \varepsilon) \in R$ , ne segue che

$$S_0 \Rightarrow \varepsilon = w \implies w = \varepsilon \in L(G')$$



– Se  $w \neq \varepsilon$ , invece, si ha che  $\forall j \in [1, n] \ w_j \in L = L(G) \iff S \xRightarrow{*} w_j$ . Dunque si ha che:

\* Se  $n = 1$ , dunque  $w = w_1$ , tramite la regola  $(S_0 \rightarrow S) \in R$  ne segue che:

$$S_0 \Rightarrow S \xRightarrow{*} w_1 = w \implies w \in L(G')$$

\* Se invece  $n > 1$ , tramite  $(S_0 \Rightarrow S_0 S_0) \in R$  ne segue che:

$$S_0 \Rightarrow S_0 S_0 \xRightarrow{*} S_0^n \xRightarrow{*} S^n \xRightarrow{*} w_1 \dots w_n = w \implies w \in L(G')$$

• Data  $w \in L(G')$ , dove  $w \in L(G') \iff S_0 \xRightarrow{*} w$ , poiché le uniche regole applicabili su  $S_0$  sono  $\{S_0 \rightarrow \varepsilon, S_0 \rightarrow S, S_0 \rightarrow SS\}$ , ne segue necessariamente che:

- Se  $S_0 \Rightarrow \varepsilon = w$ , ne segue direttamente che  $w = \varepsilon \in L^0$
- Se  $S_0 \Rightarrow S \xRightarrow{*} w$ , ne segue direttamente che  $w \in L(G) = L^1$
- Se  $S_0 \Rightarrow S_0 S_0 \xRightarrow{*} w$ , dato  $n \geq 2$  si ha che:

$$S_0 \Rightarrow S_0 S_0 \xRightarrow{*} S_0^n \xRightarrow{*} S^n$$

Siano quindi  $w_1, \dots, w_n \in L(G) = L$ . Poiché  $\forall j \in [1, n] \ w_j \in L(G) = L \iff S \xRightarrow{*} w_j$ , ne segue automaticamente che:

$$S_0 \xRightarrow{*} S^n \xRightarrow{*} w_1 \dots w_n = w \implies w \in L^n$$

Dunque, dato  $n \geq 2$ , abbiamo che:

$$w \in L^0 \cup L^1 \cup L^n = L^*$$

• Di conseguenza, concludiamo che:

$$L^* = L(G') \in \text{CFL}$$

□

### Esempio:

• Consideriamo il seguente linguaggio e la sua grammatica generante:

$$L = \{0^n 1^n \mid n \in \mathbb{N}\} \quad G : A \rightarrow 0A1 \mid \varepsilon$$

• La grammatica  $G'$  tale che  $L(G) = L(G)^*$ , corrisponderà a:

$$\begin{aligned} G' : \quad & S \rightarrow \varepsilon \mid A \mid SS \\ & A \rightarrow 0A1 \mid \varepsilon \end{aligned}$$

**Teorema 15: Non chiusura dell'intersezione in CFL**

L'operatore intersezione **non** è chiuso in CFL, ossia:

$$\exists L_1, L_2 \in \text{CFL} \mid L_1 \cap L_2 \notin \text{CFL}$$

*Dimostrazione.*

- Consideriamo i seguenti due linguaggi:

$$L_1 = \{a^i b^i c^j \mid i, j \in \mathbb{N}\} \quad L_2 = \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$$

- Tali linguaggi sono descritti dalle seguenti due grammatiche:

$$\begin{array}{ll} G_1 : & S \rightarrow TV \\ & T \rightarrow aTb \mid \varepsilon \\ & V \rightarrow cV \mid \varepsilon \end{array} \quad \begin{array}{ll} G_2 : & S \rightarrow VT \\ & T \rightarrow bTc \mid \varepsilon \\ & V \rightarrow aV \mid \varepsilon \end{array}$$

dove  $L_1 = L(G_1)$  e  $L_2 = L(G_2)$

- L'intersezione di tali linguaggi risulta essere:

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

il quale abbiamo già dimostrato non essere un linguaggio acontestuale (sezione 2.5)

- Di conseguenza, concludiamo che  $L_1, L_2 \in \text{CFL}$  ma  $L_1 \cap L_2 \notin \text{CFL}$

□

**Teorema 16: Non chiusura del complemento in CFL**

L'operatore complemento **non** è chiuso in CFL, ossia:

$$\exists L \in \text{CFL} \mid \bar{L} \notin \text{CFL}$$

*Dimostrazione.*

- Consideriamo il seguente linguaggio:

$$L = \{a, b\}^* - \{ww \mid w \in \{a, b\}^*\}$$

- Consideriamo quindi la seguente grammatica:

$$\begin{array}{ll} G : & S \rightarrow A \mid B \mid AB \mid BA \\ & A \rightarrow a \mid aAa \mid aAb \mid bAa \mid bAb \\ & B \rightarrow b \mid aBa \mid aBb \mid bBa \mid bBb \end{array}$$

- Data  $x \in L$  tale che  $|x|$  sia dispari, notiamo che:

- Se il simbolo centrale di  $x$  è  $a$ , allora  $S \Rightarrow A \xRightarrow{*} x$
- Se il simbolo centrale di  $x$  è  $b$ , allora  $S \Rightarrow B \xRightarrow{*} x$

dunque ne segue che  $x \in L(G)$

- Viceversa, data  $x \in L(G)$  tale che  $|x|$  sia dispari, ne segue immediatamente che  $\nexists w \in \{a, b\}^* \mid x = ww \implies x \in L$
- Sia quindi  $x \in L$  tale che  $|x|$  sia pari.

Dati  $x_1, \dots, x_n \in \{a, b\}$  tali che  $x = x_1 \dots x_n$ , ne segue che:

$$x \in L \implies \exists i \in [1, n] \mid x_i \neq x_{\frac{n}{2}+i}$$

- Siano quindi  $u := x_1 \dots x_{2i-1}$  e  $v := x_{2i} \dots x_n$ . Notiamo che il simbolo centrale di  $u$  corrisponde a  $x_{\frac{1+2i-1}{2}} = x_i$ , mentre quello di  $v$  corrisponde a  $x_{\frac{2i+n}{2}} = x_{\frac{n}{2}+i}$ , da cui traiamo che:

$$x_i \neq x_{\frac{n}{2}+i} \implies x_{\frac{1+2i-1}{2}} = x_i \neq x_{\frac{n}{2}+i} = x_{\frac{2i+n}{2}} \implies u \neq v$$

- Inoltre, notiamo che  $|u|$  e  $|v|$  siano dispari, dunque si ha che  $u, v \in L(G)$ . Di conseguenza, otteniamo che:

$$S \Rightarrow AB \xRightarrow{*} uv = x \text{ oppure } S \Rightarrow BA \xRightarrow{*} uv = x$$

implicando quindi che  $x \in L(G)$

- Sia quindi  $x \in L(G)$  tale che  $|x|$  sia pari.

Poiché  $|x|$  è pari, ne segue necessariamente che:

$$S \Rightarrow AB \xRightarrow{*} x \text{ oppure } S \Rightarrow BA \xRightarrow{*} x$$

Poiché i due casi sono analoghi, senza perdita di generalità consideriamo il caso in cui  $S \Rightarrow AB \xRightarrow{*} x$

- Siano quindi  $u := x_1 \dots x_k$  e  $v := x_{k+1} \dots v_n$  tali che  $x = uv$ ,  $S \Rightarrow A \xRightarrow{*} u$  e  $S \Rightarrow B \xRightarrow{*} v$ .
- Poiché  $S \Rightarrow A \xRightarrow{*} u$  e  $S \Rightarrow B \xRightarrow{*} v$ , otteniamo che:
  - $|u| = k$  e  $|v| = n - k$  sono dispari
  - $S \Rightarrow A \xRightarrow{*} u$  implica che il simbolo centrale di  $u$  sia  $a$ , ossia che  $x_{\frac{1+k}{2}} = a$
  - $S \Rightarrow B \xRightarrow{*} v$  implica che il simbolo centrale di  $v$  sia  $b$ , ossia che  $x_{\frac{k+1+n}{2}} = b$

- Siano quindi che  $w := w_1 \dots w_h$  e  $w' := w'_1 \dots w'_h$  tali che  $|w| = |w'| = h$  e che  $u = ww'$ , implicando dunque che  $h = \frac{n}{2}$ . Per il risultato precedente, ne segue automaticamente che:

$$w_{\frac{1+h}{2}} = x_{\frac{1+k}{2}} = a \neq b = x_{\frac{k+1+n}{2}} = w'_{\frac{1+h}{2}} \implies w \neq w' \implies x = ww' \in L$$

- Dunque, abbiamo ottenuto  $L = L(G) \in \text{CFL}$
- Il complemento di tale linguaggio risulta essere:

$$\bar{L} = \{ww \mid w \in \{a, b\}^*\}$$

il quale abbiamo già dimostrato non essere un linguaggio acontestuale (sezione 2.5)

- Di conseguenza, concludiamo che  $L \in \text{CFL}$ , ma  $\bar{L} \notin \text{CFL}$

□

# 3

## Decidibilità

### 3.1 Macchine di Turing

Nel 1936, il pioniere dell'informatica Alan Turing sviluppò un modello di calcolo simile ad un automa a stati finiti ma dotato di una memoria illimitata e senza alcuna restrizione. Sebbene essa richieda una grande mole di tempo, la **macchina di Turing** è in grado di elaborare tutto ciò che un reale computer è in grado di elaborare. Per tanto, essa costituisce un perfetto modello astratto di un reale computer, implicando che ogni problema per essa **irrisolvibile** lo sarà anche per un computer.

Il modello di Turing utilizza un **nastro infinito** come memoria illimitata ed è dotata di una **testina di lettura-scrittura**. Il nastro è formato da celle, le quali, inizialmente, contengono solo una stringa data in input (tutte le altre celle sono vuote). Inoltre, il nastro viene continuamente **spostato** a sinistra e destra, in modo che la testina possa leggere o scrivere sulle varie celle. La macchina continua la sua computazione finché essa non raggiungerà lo stato di **accettazione** o lo stato di **rifiuto** della stringa in input. Se la macchina non è in grado di raggiungere nessuno dei due stati, essa rimarrà in un **loop infinito**, non terminando mai l'esecuzione.

Ad esempio, consideriamo il linguaggio  $L = \{w\#w \mid w \in \{0,1\}^*\}$ . Descriviamo in modo informale una macchina di Turing  $M$  in grado di accettare le stringhe di tale linguaggio:

$M =$  "Data la stringa  $w$  in input:

1. Muoviti a zig-zag lungo il nastro tra tutte le posizioni corrispondenti su entrambi i lati del simbolo  $\#$ . Se i due simboli combaciano, cancella entrambi sovrascrivendoli con una  $x$ . Se i due simboli non combaciano o se non viene mai trovato il simbolo  $\#$ , rifiuta la stringa.
2. Quando tutti i simboli a sinistra del simbolo  $\#$  sono stati cancellati, controlla se a destra del simbolo  $\#$  vi sono simboli diversi da  $x$ . Se vi sono, rifiuta la stringa, altrimenti accettala."

Data la stringa in input 011000#011000, l'esecuzione della macchina procede come:

$$\begin{array}{cccccccccccccccc}
 & \downarrow & & & & & & & & & & & & & & & & \\
 & 0 & 1 & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & \\
 & & & & & & & & & & & & & & & & & \\
 & & \downarrow & & & & & & & & & & & & & & & \\
 x & 1 & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & \downarrow & & & & & & & & & & \\
 x & 1 & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & & & & & & & & & & & & & & & & & \\
 & \downarrow & & & & & & & & & & & & & & & & \\
 x & 1 & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & & & & & & & & & & & & & & & & & \\
 & \downarrow & & & & & & & & & & & & & & & & \\
 x & x & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & \dots & & & & & & & & & & \\
 & & & & & & & & & \downarrow & & & & & & & & \\
 x & x & x & x & x & x & \# & x & x & x & x & x & x & \sqcup & \dots & & & 
 \end{array}$$

dove il simbolo  $\sqcup$  indica una **cella vuota**

### Definizione 35: Turing Machine (TM)

Una **Turing Machine (TM)** è una settupla  $(Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  dove:

- $Q$  è l'insieme finito degli stati della macchina
- $\Sigma$  è l'alfabeto della macchina, dove  $\sqcup \notin \Sigma$
- $\Gamma$  è l'alfabeto del nastro, dove  $\sqcup \in \Gamma$  e  $\Sigma \subseteq \Gamma$
- $q_{\text{start}} \in Q$  è lo **stato iniziale** dell'automa
- $q_{\text{accept}} \in Q$  è lo **stato accettante** dell'automa
- $q_{\text{reject}} \in Q$  è lo **stato rifiutante** dell'automa, dove  $q_{\text{reject}} \neq q_{\text{accept}}$
- $\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  è la **funzione di transizione** della macchina, dove se  $\delta(p, a) = (q, b, X)$  si ha che:
  - Viene letto il simbolo  $a$  dal nastro, sostituendolo con  $b$  e la macchina passa dallo stato  $p$  allo stato  $q$ . Inoltre, in nastro viene spostato a sinistra se  $X = L$  e a destra se  $X = R$
  - L'etichetta della transizione da  $p$  a  $q$  viene indicata come  $a \rightarrow b; X$

**Definizione 36: Configurazione di una TM**

Sia  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  una TM. Definiamo la stringa  $uqav$  come **configurazione di  $M$** , dove:

- $q \in Q$  è lo stato attuale della macchina
- $a \in \Gamma$  è il simbolo del nastro su cui si trova attualmente la testina della macchina
- $u \in \Gamma^*$  è composta dai simboli precedenti ad  $a$  sul nastro
- $v \in \Gamma^*$  è composta dai simboli successivi ad  $a$  sul nastro

**Definizione 37: Passo di computazione in una TM**

Data una TM  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ , si ha che:

$$uaq_i b v \text{ produce } uq_j a c v \iff \delta(q_i, b) = (q_j, c, L)$$

$$uaq_i b v \text{ produce } uacq_j v \iff \delta(q_i, b) = (q_j, c, R)$$

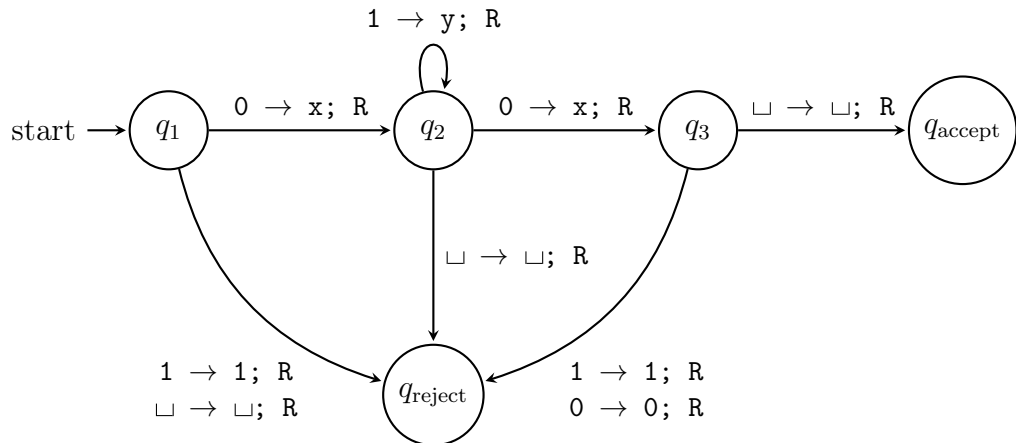
**Proposizione 11: Stringa accettata in una TM**

Sia  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  una TM. Data una stringa  $w \in \Sigma^*$ , diciamo che  $w$  è **accettata da  $M$**  se esiste una sequenza di configurazioni  $c_1, \dots, c_k$  tali che:

- $c_1 = q_{\text{start}}w$
- $\forall i \in [1, k-1] \ c_i \text{ produce } c_{i+1}$
- $q_{\text{accept}} \in c_k$

**Esempio:**

- La seguente TM riconosce il linguaggio  $L = \{01^n0 \mid n \in \mathbb{N}\}$ :

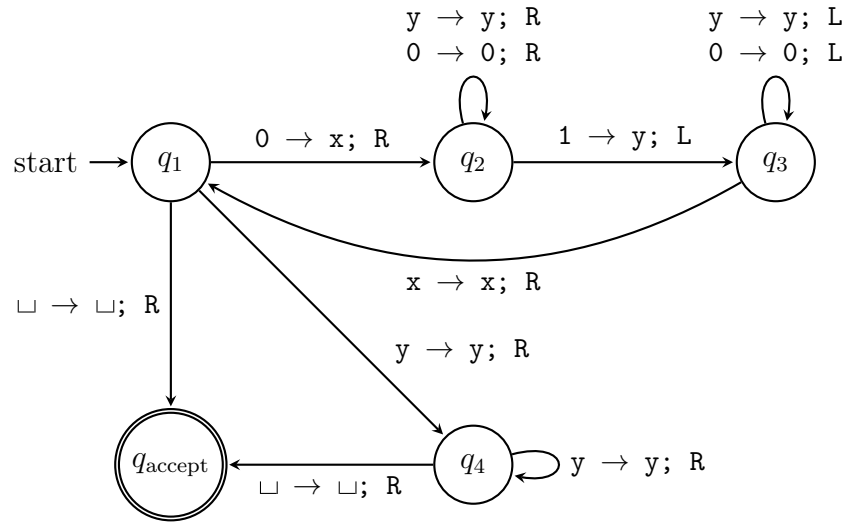


- Difatti, durante la lettura della stringa 01110, la macchina assume le seguenti configurazioni:

```

q1 0 1 1 1 0
x q2 1 1 1 0
x y q2 1 1 0
x y y q2 1 0
x y y y q2 0
x y y y x q3 □
x y y y x □ qaccept □
    
```

2. • La seguente TM riconosce il linguaggio  $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ :



(tutte le transizioni omesse vanno allo stato  $q_{\text{reject}}$ )

- Difatti, durante la lettura della stringa 000111, la macchina assume le seguenti configurazioni:

$q_1$ 0 0 0 1 1 1	$x q_1 x 0 y 1 1$	$x x q_1 0 y y 1$	
$x q_2 0 0 1 1 1$	$x x q_2 0 y 1 1$	$x x x q_2 y y 1$	$x x x q_3 y y y$
$x 0 q_2 0 1 1 1$	$x x 0 q_2 y 1 1$	$x x x y q_2 y 1$	$x x x y q_4 y y$
$x 0 0 q_2 1 1 1$	$x x 0 y q_2 1 1$	$x x x y y q_2 1$	$x x x y y q_4 y$
$x 0 q_3 0 y 1 1$	$x x 0 q_3 y y 1$	$x x x y q_3 y y$	$x x x y y y q_4 \square$
$x q_3 0 0 y 1 1$	$x x q_3 0 y y 1$	$x x x q_3 y y y$	$x x x y y y \square q_{\text{accept}} \square$
$q_3 x 0 0 y 1 1$	$x q_3 x 0 y y 1$	$x x q_3 x y y y$	



**Definizione 38: TM Decisore**

Data una TM  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ , definiamo  $M$  come **decisore** se essa termina sempre la sua esecuzione (ossia non può entrare in un loop infinito).

Inoltre, se  $M$  è un decisore diciamo che  $M$  **decide**  $L(M)$

**Definizione 39: Classe dei linguaggi Turing-riconoscibili**

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi Turing-riconoscibili di  $\Sigma$**  il seguente insieme:

$$\text{REC} = \{L \subseteq \Sigma^* \mid \exists \text{ TM } M \text{ t.c. } L = L(M)\}$$

**Definizione 40: Classe dei linguaggi Turing-decidibili**

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi Turing-decidibili di  $\Sigma$**  il seguente insieme:

$$\text{DEC} = \{L \subseteq \Sigma^* \mid \exists \text{ TM decisore } M \text{ t.c. } L = L(M)\}$$

**Esempio:**

- Entrambi i linguaggi dei due esempi precedenti sono Turing-decidibili in quanto nessuna delle due TM mostrate è in grado di entrare in un loop infinito

**Osservazione 13: Descrizione informale delle TM**

Negli esempi e dimostrazioni successive, le TM verranno descritte in modo informale, poiché la loro descrizione formale richiederebbe una grande quantità di stati e transizioni.

Ovviamente, tali descrizioni informali conterranno solo operazioni eseguibili dalle TM

**Definizione 41: Codifica di un oggetto**

Dato un oggetto  $O$ , indichiamo come  $\langle O \rangle$  la sua **codifica**, ossia una stringa che ne descriva le caratteristiche

**Esempi:**

- Dato un polinomio  $p = a_0 + a_1x_1 + \dots + a_nx_n$ , possiamo immaginare la sua codifica come una stringa composta dai suoi coefficienti, ossia  $\langle p \rangle = \#a_1, a_2, \dots, a_n\#$
- Dato un grafo  $G$ , possiamo immaginare la sua codifica  $\langle G \rangle$  come una stringa formata da una serie di coppie  $(x, y)$  rappresentanti gli archi del grafo

### 3.1.1 Varianti della macchina di Turing

#### Definizione 42: Stay-put TM

Una **Stay-put TM** è una TM  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  la cui **funzione di transizione** è definita come:

$$\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

dove il simbolo  $S$  indica che il nastro possa anche rimanere **immobile**

#### Teorema 17: Equivalenza tra TM e Stay-put TM

Dato un linguaggio  $L \subseteq \Sigma^*$  si ha che:

$$L \in \text{REC} \iff \exists \text{ Stay-put TM } M \text{ t.c. } L = L(M)$$

In altre parole, le TM e le Stay-put TM sono equivalenti tra loro

*Dimostrazione.*

*Prima implicazione.*

- Dato  $L \in \text{REC}$ , sia  $M$  la TM tale che  $L = L(M)$
- Poiché una TM è una particolare Stay-put TM le cui transizioni con non rimangono mai immobili, ne segue automaticamente che essa stessa sia la Stay-put TM in grado di riconoscere  $L = L(M)$

*Seconda implicazione.*

- Sia  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  la Stay-put TM tale che  $L = L(M)$
- Consideriamo la TM  $M' = (Q', \Sigma, \Gamma, \delta', q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  tale che:

$$\delta(p, a) = (q, b, S) \iff \exists r \in Q \mid \forall c \in \Gamma \quad \delta'(p, a) = (r, b, R) \wedge \delta'(r, c) = (q, c, L)$$

- Per costruzione stessa di  $M'$ , si ha che:

$$x \in L = L(M) \iff x \in L(M')$$

implicando che  $L = L(M) = L(M') \in \text{REC}$

□

**Definizione 43: Multitape TM**

Una **Multitape TM a  $k$  nastri** è una TM  $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  la cui **funzione di transizione** è definita come:

$$\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}$$

dove il simbolo  $S$  indica che il nastro possa anche rimanere **immobile**

**Teorema 18: Equivalenza tra TM e Multitape TM**

Dato un linguaggio  $L \subseteq \Sigma^*$  si ha che:

$$L \in \text{REC} \iff \exists \text{ Multitape TM } M \text{ t.c. } L = L(M)$$

In altre parole, le TM e le Multitape TM sono equivalenti tra loro

*Dimostrazione.*

*Prima implicazione.*

- Dato  $L \in \text{REC}$ , sia  $M$  la TM tale che  $L = L(M)$
- Poiché una TM è una particolare Multitape TM ad 1 nastro le cui transizioni non rimangono mai immobili, ne segue automaticamente che essa stessa sia la Multitape TM in grado di riconoscere  $L = L(M)$

*Seconda implicazione.*

- Sia  $M$  la Multitape TM a  $k$  nastri tale che  $L = L(M)$
- Consideriamo la Stay-put TM  $S$  definita come:

$S =$  "Date in input le stringhe  $a_1 \dots a_n, b_1 \dots b_m, \dots, k_1 \dots k_h$  rappresentati gli input dei  $k$  nastri:

1.  $S$  pone il nastro uguale a

$$\# \overset{\bullet}{a}_1 \dots a_n \# \overset{\bullet}{b}_1 \dots b_m \# \dots \# \overset{\bullet}{k}_1 \dots k_h \#$$

dove il simbolo  $\#$  separa i vari  $k$  nastri simulati e il marcatore  $\bullet$  indica le testine virtuali di ogni nastro

2. Per simulare una mossa di  $M$ ,  $S$  scansiona il nastro dal primo  $\#$  fino al  $(k+1)$ -esimo  $\#$ , ossia dall'estremità sinistra fino all'estremità destra, determinando i simboli puntati dalle testine virtuali. Successivamente,  $S$  esegue un secondo passaggio per aggiornare i nastri simulati in base alla funzione di transizione di  $M$

3. Se in qualsiasi momento una delle testine virtuali finisce su un  $\#$  durante uno spostamento a destra,  $S$  scrive un simbolo  $\sqcup$  e sposta di una posizione a destra l'intero contenuto del nastro di  $S$  successivo al simbolo scritto, per poi riprendere la normale esecuzione"

- Per costruzione stessa di  $S$ , si ha che:

$$x \in L(M) \iff x \in L(S)$$

implicando che  $L = L(M) = L(S)$

- Infine, per l'[Equivalenza tra TM e Stay-put TM](#), se segue automaticamente che  $L = L(M) = L(S) \in \text{REC}$

□

#### Definizione 44: Non deterministic TM

Una **Non deterministic TM (NTM)** è una TM  $N = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  la cui **funzione di transizione** è definita come:

$$\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

#### Teorema 19: Equivalenza tra TM e NTM

Dato un linguaggio  $L \subseteq \Sigma^*$  si ha che:

$$L \in \text{REC} \iff \exists \text{ NTM } N \text{ t.c. } L = L(N)$$

In altre parole, le TM e le NTM sono equivalenti tra loro

*Dimostrazione.*

*Prima implicazione.*

- Dato  $L \in \text{REC}$ , sia  $M$  la TM tale che  $L = L(M)$
- Poiché una TM è una particolare NTM le cui transizioni sono tutte deterministiche, ne segue automaticamente che essa stessa sia la NTM in grado di riconoscere  $L = L(M)$

*Seconda implicazione.*

- Sia  $N = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  la NTM tale che  $L = L(N)$
- Consideriamo l'albero di computazione non deterministica di  $N$ . Ad ogni nodo di tale albero associamo un indirizzo:
  - Sia  $b$  il numero di transizioni uscenti dallo stato di  $N$  avente il maggior numero di transizioni uscenti
  - Se il nodo è la radice dell'albero, il suo indirizzo è  $\varepsilon$

- Se il nodo non è la radice, il suo indirizzo è  $xa$ , dove  $x$  è l'indirizzo del padre di tale nodo ed  $a \in \{1, \dots, b\}$  è l'identificatore associato a tale nodo tra i figli del suo padre
- Consideriamo quindi la seguente Multitape TM  $M$  a 3 nastri, dove:
  - Il nastro 1 contiene la stringa  $w$  in input ad  $N$
  - Il nastro 2 è il nastro su cui viene simulata  $N$  con  $w$  in input
  - Il nastro 3 contiene l'indirizzo del nodo dell'albero di computazione fino a cui simulare  $N$
- $M$  è definita come:
 

$M = \text{"Data la stringa } w \text{ in input:}"$

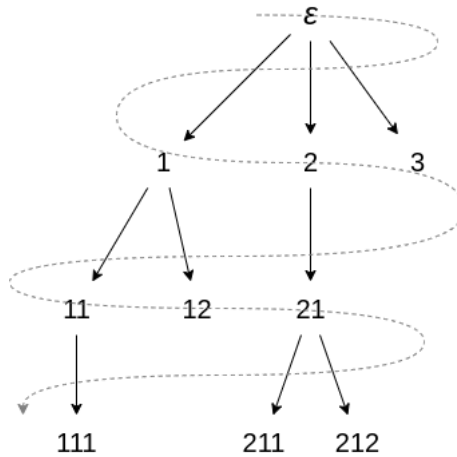
  1. Inizialmente, il nastro 1 di  $M$  contiene  $w$ , il nastro 3 contiene  $\varepsilon$  e il nastro 2 è vuoto
  2. Ripeti gli step successivi:
    3.  $M$  copia il nastro 1 sul nastro 2
    4.  $M$  simula  $N$  tramite il nastro 2 eseguendo un suo ramo di computazione. Prima di ogni passo simulato,  $M$  consulta il prossimo simbolo sul nastro 3 per poter scegliere su quale ramo proseguire.
    5. Se la simulazione accetta la stringa, anche  $M$  la *accetta*.
    6. Se invece non rimangono più simboli sul nastro 3 o se la simulazione rifiuta la stringa, sostituisci la stringa sul nastro 3 con l'indirizzo del nodo direttamente a destra del nodo precedente. Se non vi è un nodo a destra, viene scelto il nodo più a sinistra del livello successivo"
- Per costruzione stessa di  $M$ , si ha che:

$$x \in L(N) \iff x \in L(M)$$

implicando che  $L = L(N) = L(M)$

- Infine, per l'[Equivalenza tra TM e Multitape TM](#), se segue automaticamente che  $L = L(N) = L(M) \in \text{REC}$

□



*Rappresentazione grafica della dimostrazione*

#### Definizione 45: Enumeratore

Un **enumeratore** è una TM  $E = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$  connessa ad una "stampante" (ad esempio un nastro secondario), la quale stampa le stringhe di un linguaggio in ordine casuale e con eventuali ripetizioni.

Inoltre, il nastro di input dell'enumeratore è vuoto e diciamo che  $E$  **enumera**  $L(E)$

#### Teorema 20: Equivalenza tra TM e Enumeratori

Dato un linguaggio  $L \subseteq \Sigma^*$  si ha che:

$$L \in \text{REC} \iff \exists \text{ enumeratore } E \text{ t.c. } L = L(E)$$

In altre parole, le TM e gli enumeratori sono equivalenti tra loro

*Dimostrazione.*

*Prima implicazione.*

- Dato  $L \in \text{REC}$ , sia  $M$  la TM tale che  $L = L(M)$ . Siano inoltre  $w_1, w_2, \dots \in \Sigma^*$  tutte le stringhe di  $\Sigma^*$
- Consideriamo l'enumeratore  $E$  definito come:

$E$  = "Dato nulla in input:

1. Ripeti lo step seguente per  $i = 1, 2, 3, \dots$ :
2. Ripeti lo step seguente per  $j = 1, \dots, i$ :
3. Simula  $M$  per  $i$  passi con  $w_j$  in input. Se la simulazione accetta  $w_j$ , stampa  $w_j$

- Per costruzione stessa di  $E$ , si ha che:

$$x \in L(M) \iff x \in L(E)$$

implicando che  $L = L(M) = L(E)$

*Seconda implicazione.*

- Sia  $E$  l'enumeratore tale che  $L = L(E)$
- Consideriamo la TM  $M$  definita come:

$M$  = "Data la stringa  $w$  in input:

1. Simula  $E$ . Ogni volta che  $E$  stampa una stringa, comparala con  $w$ .
2. Se  $w$  appare almeno una volta nell'output di  $E$ ,  $M$  accetta

- Per costruzione stessa di  $M$ , si ha che:

$$x \in L(E) \iff x \in L(M)$$

implicando che  $L = L(E) = L(M) \in \text{REC}$

□

### 3.1.2 Tesi di Church-Turing

#### Proposizione 12: Tesi di Church-Turing

Data una funzione  $f$ , si ha che:

$$f \text{ computabile da un algoritmo} \iff f \text{ computabile da una TM}$$

In altre parole, le TM e gli algoritmi sono equivalenti tra loro, implicando che **qualsiasi tipo di computazione possa essere svolto tramite una TM**. Dunque, la tesi di Church-Turing può essere vista come una formalizzazione del concetto di algoritmo.

#### Definizione 46: TM universale

Una **TM universale** è una TM  $M$  in grado di simulare qualsiasi altra TM

#### Definizione 47: Turing-completezza

Definiamo un modello di calcolo come **Turing-completo** se esso è equivalente ad una TM universale

**Esempi:**

- Il *lambda calcolo non tipato* è un modello di calcolo Turing-completo
- Tutti i linguaggi di programmazione sono Turing-completi
- Il gioco di carte *Magic: The Gathering* è un modello di calcolo Turing-completo (più info qui: <https://arxiv.org/abs/1904.09828>)

## 3.2 Problemi decidibili

**Teorema 21: Problema dell'accettazione per DFA**

Sia  $A_{\text{DFA}}$  il linguaggio definito come:

$$A_{\text{DFA}} = \{\langle D, w \rangle \mid D \text{ DFA}, w \in L(D)\}$$

Tale linguaggio è **decidibile**, ossia  $A_{\text{DFA}} \in \text{DEC}$

*Dimostrazione.*

- Sia  $M$  la TM definita come:
 

$M = \text{"Data in input la codifica } \langle D, w \rangle, \text{ dove } D \text{ è un DFA e } w \text{ una stringa:}$

  1. Se la codifica in input è errata,  $M$  *rifiuta*
  2.  $M$  simula  $D$  con input  $w$
  3. Se la simulazione termina su uno stato accettante di  $D$ , allora  $M$  *accetta*, altrimenti *rifiuta*."
  - Per costruzione stessa di  $M$ , si ha che:

$$\langle D, w \rangle \in L(M) \iff w \in L(D) \iff \langle D, w \rangle \in A_{\text{DFA}}$$

implicando che  $L(M) = A_{\text{DFA}}$

- Inoltre, poiché un DFA termina sempre, anche la simulazione terminerà sempre, implicando che  $M$  sia un decisore, concludendo che  $A_{\text{DFA}} = L(M) \in \text{DEC}$ .

□



**Teorema 22: Problema dell'accettazione per NFA**

Sia  $A_{\text{NFA}}$  il linguaggio definito come:

$$A_{\text{NFA}} = \{\langle N, w \rangle \mid N \text{ NFA}, w \in L(N)\}$$

Tale linguaggio è **decidibile**, ossia  $A_{\text{NFA}} \in \text{DEC}$

*Dimostrazione.*

- Sia  $M_{\text{DFA}}$  la TM decisore utilizzata nel [Problema dell'accettazione per DFA](#)
- Sia  $M$  la TM definita come:

$M =$  "Data in input la codifica  $\langle N, w \rangle$ , dove  $N$  è un NFA e  $w$  una stringa:

1. Se la codifica in input è errata,  $M$  *rifiuta*
2.  $M$  converte  $N$  in un DFA  $D$  tale che  $L(N) = L(D)$
3.  $M$  esegue il programma di  $M_{\text{DFA}}$  con input  $\langle D, w \rangle$
4. Se l'esecuzione accetta, allora  $M$  *accetta*, altrimenti *rifiuta*"

- Per costruzione stessa di  $M$ , si ha che:

$$\langle N, w \rangle \in A_{\text{NFA}} \iff \langle D, w \rangle \in A_{\text{DFA}} = L(M_{\text{DFA}}) \iff \langle N, w \rangle \in L(M)$$

implicando che  $L(M) = A_{\text{NFA}}$

- Inoltre, poiché  $M_{\text{DFA}}$  è un decisore, dunque la sua esecuzione termina sempre, anche  $M$  terminerà sempre, implicando che anche esso sia un decisore, concludendo che  $A_{\text{NFA}} = L(M) \in \text{DEC}$ .

□

**Teorema 23: Problema dell'accettazione per le esp. reg.**

Sia  $A_{\text{REX}}$  il linguaggio definito come:

$$A_{\text{REX}} = \{\langle R, w \rangle \mid R \in \text{re}(\Sigma), w \in L(R)\}$$

Tale linguaggio è **decidibile**, ossia  $A_{\text{REX}} \in \text{DEC}$

*Dimostrazione.*

- Sia  $M_{\text{NFA}}$  la TM decisore utilizzata nel [Problema dell'accettazione per NFA](#)
- Sia  $M$  la TM definita come:

$M =$  "Data in input la codifica  $\langle R, w \rangle$ , dove  $R \in \text{re}(\Sigma)$  e  $w$  una stringa:

1. Se la codifica in input è errata,  $M$  *rifiuta*

2.  $M$  converte  $R$  in un NFA  $N$  tale che  $L(R) = L(N)$
  3.  $M$  esegue il programma di  $M_{\text{NFA}}$  con input  $\langle N, w \rangle$
  4. Se l'esecuzione accetta, allora  $M$  accetta, altrimenti rifiuta"
- Per costruzione stessa di  $M$ , si ha che:

$$\langle R, w \rangle \in A_{\text{REX}} \iff \langle N, w \rangle \in A_{\text{NFA}} = L(M_{\text{NFA}}) \iff \langle R, w \rangle \in L(M)$$

implicando che  $L(M) = A_{\text{REX}}$

- Inoltre, poiché  $M_{\text{NFA}}$  è un decisore, dunque la sua esecuzione termina sempre, anche  $M$  terminerà sempre, implicando che anche esso sia un decisore, concludendo che  $A_{\text{REX}} = L(M) \in \text{DEC}$ .

□

#### Teorema 24: Problema dell'accettazione per le CFG

Sia  $A_{\text{CFG}}$  il linguaggio definito come:

$$A_{\text{CFG}} = \{\langle G, w \rangle \mid G \text{ CFG}, w \in L(G)\}$$

Tale linguaggio è **decidibile**, ossia  $A_{\text{CFG}} \in \text{DEC}$

*Dimostrazione.*

- **Affermazione:** Sia  $G = (V, \Sigma, R, S)$  una CFG in CNF. Data  $w \in L(G)$ , se  $|w| \geq 1$ , la sua derivazione è composta da esattamente  $2 \cdot |w| - 1$  produzioni

*Dimostrazione.*

Procediamo per induzione sulla lunghezza  $n$  di  $w$

*Caso base.*

- Per  $n = 1$ , si ha che  $w = a$ , dove  $a \in \Sigma$ . Di conseguenza la sua derivazione è composta solo dalla regola  $S \Rightarrow a = w$ , ossia da  $2 \cdot 1 - 1 = 1$  produzioni

*Ipotesi induttiva forte.*

- Assumiamo che per ogni stringa  $w \in L(G)$  tale che  $1 \leq |w| \leq n$  sia derivabile tramite  $2|w| - 1$  produzioni

*Passo induttivo.*

- Sia  $w \in L(G)$  tale che  $|w| = n + 1$ . Essendo  $G$  in CNF, ne segue che la derivazione di  $w$  sia nella forma  $S \Rightarrow AB \xRightarrow{*} w$ .
- Siano quindi  $x, y \in \Sigma^*$  tali che  $w = xy$ , dove  $A \xRightarrow{*} x$  e  $B \xRightarrow{*} y$ .
- Poiché  $G$  è in CNF, ne segue che  $x, y \neq \varepsilon$ , implicando che  $1 \leq |x| \leq n$  e  $1 \leq |y| \leq n$

- Siano quindi  $|x| = k$  e  $|y| = n + 1 - k$ . Per ipotesi induttiva,  $x$  e  $y$  sono derivabili tramite esattamente  $2k - 1$  produzioni e  $2(n + 1 - k) - 1$  produzioni
- Di conseguenza, poiché  $S \Rightarrow AB \xRightarrow{*} xy = w$ , ne segue che il numero di produzioni della derivazione di  $w$  sia esattamente:

$$1 + 2k - 1 + 2(n + 1 - k) - 1 = 2n + 2 - 1 = 2(n + 1) - 1 = 2|w| - 1$$

- Sia  $M$  la TM definita come:

$M =$  "Data in input la codifica  $\langle G, w \rangle$ , dove  $G$  è un CFG e  $w$  una stringa:

1. Se la codifica in input è errata,  $M$  *rifiuta*
2.  $M$  converte  $G$  in una CFG  $G'$  in CNF tale che  $L(G) = L(G')$
3. Se  $|w| \neq 0$ ,  $M$  lista tutte le derivazioni di  $G$  composte da  $2n - 1$  produzioni, dove  $|w| = n$ . Altrimenti,  $M$  lista tutte le derivazioni composte da 1 produzione
4. Se almeno una delle derivazioni genera  $w$ ,  $M$  *accetta*, altrimenti *rifiuta*"

- Per costruzione stessa di  $M$ , si ha che:

$$\langle G, w \rangle \in L(M) \iff w \in L(G) \iff \langle G, w \rangle \in A_{\text{CFG}}$$

implicando che  $L(M) = A_{\text{CFG}}$

- Inoltre, poiché la lista utilizzata da  $M$  sarà sempre composta da un numero finito di derivazioni, ne segue che  $M$  terminerà sempre, concludendo che  $A_{\text{CFG}} = L(M) \in \text{DEC}$ .

□

### Teorema 25: Ling. decidibili estensione dei ling. acontestuali

Date le classi dei linguaggi CFL e DEC, si ha che:

$$\text{CFL} \subsetneq \text{DEC}$$

*Dimostrazione.*

- Sia  $M_{\text{CFG}}$  la TM decisore utilizzata nel [Problema dell'accettazione per le CFG](#)
- Dato  $L \in \text{CFL}$ , sia  $G$  la CFG tale che  $L = L(G)$
- Consideriamo quindi la TM  $M$  definita come:

$M =$  "Data la stringa  $w$  in input:

1.  $M$  esegue il programma di  $M_{\text{CFG}}$  con input  $\langle G, w \rangle$
2. Se l'esecuzione accetta,  $M$  *accetta*, altrimenti *rifiuta*"

- Per costruzione stessa di  $M$ , si ha che:

$$w \in L(M) \iff \langle G, w \rangle \in A_{\text{CFG}} \iff w \in L(G)$$

implicando che  $L(M) = L(G)$ . Inoltre, poiché  $A_{\text{CFG}}$  è un decisore, anche  $M$  è un decisore, implicando che  $\text{CFG} \subseteq \text{DEC}$

- Consideriamo quindi il linguaggio  $L = \{ww \mid w \in \{a,b\}^*\}$ . Per dimostrazione precedente (sezione 2.5), sappiamo che  $L \notin \text{CFL}$ . Tuttavia, possiamo facilmente definire una TM decisore  $M$  (simile a quella vista nella sezione 3.1) per cui  $L = L(M) \in \text{DEC}$
- Di conseguenza, concludiamo che:

$$\text{CFL} \subsetneq \text{DEC}$$

□

### Teorema 26: Problema del vuoto per DFA

Sia  $E_{\text{DFA}}$  il linguaggio definito come:

$$E_{\text{DFA}} = \{\langle D \rangle \mid D \text{ DFA}, L(D) = \emptyset\}$$

Tale linguaggio è **decidibile**, ossia  $A_{\text{DFA}} \in \text{DEC}$

*Dimostrazione.*

- Sia  $M$  la TM definita come:

$M = \text{"Data in input la codifica } \langle D \rangle, \text{ dove } D = (Q, \Sigma, \delta, q_0, F) \text{ è un DFA:}$

1. Se la codifica in input è errata,  $M$  *rifiuta*
2. Marca lo stato iniziale di  $D$
3. Ripeti lo step seguente finché vengono marcati dei nuovi stati
  4. Marca ogni stato avente una transizione entrante da uno stato già marcato
4. Se tra gli stati marcati vi è uno stato accettante di  $D$ , allora  $M$  *rifiuta*, altrimenti *accetta*"

- A questo punto, notiamo che:

$$\begin{aligned} \langle D \rangle \in E_{\text{DFA}} &\iff L(D) = \emptyset \iff \nexists w \in L(D) \iff \\ &\iff \forall w \in \Sigma^* \quad \delta^*(q_0, w) \notin F \iff \langle D \rangle \in L(M) \end{aligned}$$

implicando che  $L(M) = E_{\text{DFA}}$

- Inoltre, poiché il numero di stati marcabili da  $M$  è finito, ne segue che  $M$  termina sempre, concludendo che  $E_{\text{DFA}} = L(M) \in \text{DEC}$

□

**Teorema 27: Problema del vuoto per CFG**

Sia  $E_{\text{CFG}}$  il linguaggio definito come:

$$E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ CFG}, L(G) = \emptyset\}$$

Tale linguaggio è **decidibile**, ossia  $A_{\text{CFG}} \in \text{DEC}$

*Dimostrazione.*

- Sia  $M$  la TM definita come:

$M =$  "Data in input la codifica  $\langle G \rangle$ , dove  $G = (V, \Sigma, R, S)$  è un DFA:

1. Se la codifica in input è errata,  $M$  *rifiuta*
2. Marca tutti i terminali in  $\Sigma$
3. Ripeti lo step seguente finché vengono marcate delle nuove variabili
4. Marca ogni variabile  $A \in V$  per cui in  $R$  esiste una regola  $A \rightarrow u_1 \dots u_k$  tale che  $u_1, \dots, u_k$  sono variabili o terminali già marcati
4. Se la variabile  $S$  è marcata,  $M$  *rifiuta*, altrimenti *accetta*."

- A questo punto, notiamo che:

$$\begin{aligned} \langle G \rangle \in E_{\text{CFG}} &\iff L(G) = \emptyset \iff \nexists w \in L(D) \iff \\ &\iff \forall w \in \Sigma^* \quad S \not\stackrel{*}{\Rightarrow} w \iff \langle G \rangle \in L(M) \end{aligned}$$

implicando che  $L(M) = E_{\text{CFG}}$

- Inoltre, poiché il numero di variabili marcabili da  $M$  è finito, ne segue che  $M$  termina sempre, concludendo che  $E_{\text{CFG}} = L(M) \in \text{DEC}$

□

**Teorema 28: Problema dell'equivalenza tra DFA**

Sia  $EQ_{\text{DFA}}$  il linguaggio definito come:

$$EQ_{\text{DFA}} = \{\langle A, B \rangle \mid A, B \text{ DFA}, L(A) = L(B)\}$$

Tale linguaggio è **decidibile**, ossia  $EQ_{\text{DFA}} \in \text{DEC}$

*Dimostrazione.*

- Consideriamo la *differenza simmetrica* tra  $L(A)$  e  $L(B)$ , definita come:

$$L(A) \Delta L(B) := (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)})$$

ossia tutti gli elementi presenti in  $L(A)$  o  $L(B)$ , ma non in  $L(A) \cap L(B)$

- Poiché le operazioni di unione, intersezione e complemento sono chiuse in REG (Teoremi 3, 4 e 5), ne segue automaticamente che:

$$L(A), L(B) \in \text{REG} \implies L(A) \Delta L(B) \in \text{REG}$$

dunque  $\exists C \text{ DFA} \mid L(C) = L(A) \Delta L(B)$

- Inoltre, mostriamo che:

$$\begin{aligned} L(A) \Delta L(B) = \emptyset &\iff \\ (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) &= \emptyset \iff \\ \nexists x \in \Sigma^* \mid (x \in L(A) \wedge x \notin L(B)) \vee (x \in L(B) \wedge x \notin L(A)) &\iff \\ \forall x \in \Sigma^* (x \in L(A) \iff x \in L(B)) &\iff \\ L(A) = L(B) \end{aligned}$$

- Sia  $M_E$  la TM decisore utilizzata nel [Problema del vuoto per DFA](#)
- Sia  $M$  la TM definita come:

$M = \text{"Data in input la codifica } \langle A, B \rangle, \text{ dove } A \text{ e } B \text{ sono due DFA:}$

1. Se la codifica in input è errata,  $M$  rifiutante
2.  $M$  costruisce il DFA  $C$  tale che  $L(C) = L(A) \Delta L(B)$  tramite le procedure dei teoremi 2, 3, 4 e 5
3.  $M$  esegue il programma di  $M_E$  con input  $\langle C \rangle$
4. Se l'esecuzione accetta,  $M$  accetta, altrimenti rifiuta."

- A questo punto, notiamo che:

$$\begin{aligned} \langle A, B \rangle \in EQ_{\text{DFA}} &\iff L(A) = L(B) \iff L(C) = L(A) \Delta L(B) = \emptyset \iff \\ \langle C \rangle \in L(M_E) &\iff \langle A, B \rangle \in L(M) \end{aligned}$$

implicando che  $L(M) = EQ_{\text{DFA}}$

□

### 3.3 Argomento diagonale di Cantor

#### Teorema 29: Insiemi con stessa cardinalità

Dati due insiemi  $A$  e  $B$  si ha che:

$$\exists f : A \rightarrow B \text{ biettiva} \implies |A| = |B|$$

(*dimostrazione omessa*)

#### Definizione 48: Insiemi infiniti numerabili

Un insieme  $A$  viene detto **numerabile** se  $|A| < +\infty$  o se  $|A| = |\mathbb{N}|$

**Esempio:**

- Dato l'insieme  $2\mathbb{N} = \{2n \mid n \in \mathbb{N}\}$ , consideriamo la seguente funzione:

$$f : \mathbb{N} \rightarrow 2\mathbb{N} : n \mapsto 2n$$

- Tale funzione risulta essere sia iniettiva:

$$f(n) = f(m) \implies 2n = 2m \implies n = m$$

sia suriettiva:

$$\forall 2n \in 2\mathbb{N} \exists n \in \mathbb{N} \mid f(n) = 2n$$

- Di conseguenza, poiché  $f$  è biettiva, concludiamo che  $|\mathbb{N}| = |2\mathbb{N}|$  nonostante  $2\mathbb{N} \subsetneq \mathbb{N}$

#### Metodo 1: Argomento diagonale di Cantor

L'**argomento diagonale di Cantor** è una tecnica dimostrativa atta a dimostrare l'**esistenza o inesistenza** di una funzione biettiva tra due insiemi  $A$  e  $B$  disponendo i loro elementi in forma tabellare, per poi concludere la tesi.

#### Teorema 30: Razionali positivi numerabili

L'insieme  $\mathbb{Q}_{\geq 0}$  dei numeri razionali non negativi è **numerabile**

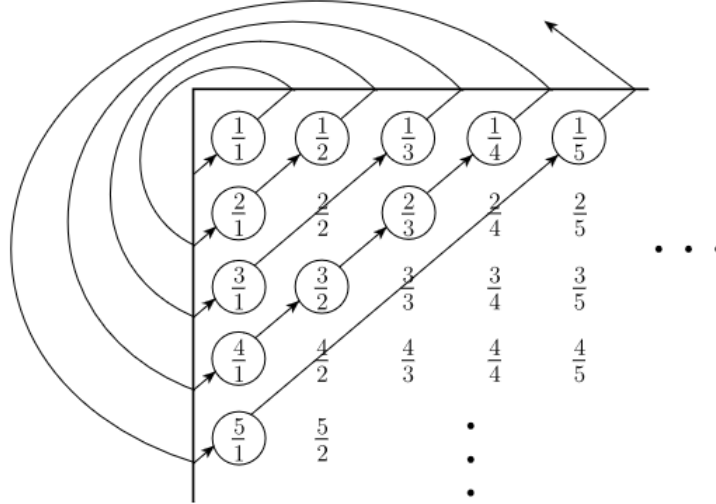
*Dimostrazione.*

- Siano  $\mathbb{N}_{>0}$  e  $\mathbb{Q}_{>0}$  gli insiemi dei numeri naturali e razionali positivi
- Consideriamo la matrice  $A$  avente righe e colonne infinite le cui entrate sono definite come:

$$a_{i,j} = \frac{i}{j}$$

dove  $i, j \in \mathbb{N}$

- Costruiamo una lista di elementi di tale matrice procedendo diagonale per diagonale, partendo dalla diagonale composta dall'entrata  $a_{1,1}$  e saltando tutti gli elementi che sono già stati inseriti nella lista (ad esempio, poiché  $a_{1,1} = \frac{1}{1} = \frac{2}{2} = a_{2,2}$ , l'entrata  $a_{2,2}$  non verrà inserita nella lista):



*Rappresentazione grafica del processo di creazione della lista*

- Procedendo all'infinito, otterremo la lista  $\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \frac{1}{3}, \dots$  contenente tutti gli elementi di  $\mathbb{Q}_{>0}$ , senza alcuna ripetizione. Inoltre, aggiungiamo all'inizio di tale lista il numero 0.
- A questo punto, consideriamo la funzione  $f : \mathbb{N} \rightarrow \mathbb{Q}_{\geq 0}$  definita come:

$f(n)$  =  $n$ -esimo elemento della lista

$n$	0	1	2	3	4	5	...
$f(n)$	0	$\frac{1}{1}$	$\frac{2}{1}$	$\frac{1}{2}$	$\frac{3}{1}$	$\frac{1}{3}$	...

- Poiché la lista contiene tutti gli elementi di  $\mathbb{Q}_{>0}$  senza alcuna ripetizione, ogni  $n$ -esimo elemento della lista sarà mappato esclusivamente dal numero  $n \in \mathbb{N}$ .
- Di conseguenza, otteniamo che  $f$  sia biettiva, concludendo che  $|\mathbb{N}| = |\mathbb{Q}_{\geq 0}|$  e quindi che  $\mathbb{Q}_{\geq 0}$  sia numerabile

□



**Teorema 31: Reali non numerabili**

L'insieme  $\mathbb{R}$  dei numeri reali **non è numerabile**

*Dimostrazione.*

- Dato  $[0, 1] \subseteq \mathbb{R}$ , supponiamo per assurdo che  $\exists f : \mathbb{N} \rightarrow [0, 1]$  biettiva
- Consideriamo il numero  $x$  definito come:

$$\forall i \geq 1 \quad i\text{-esima cifra decimale di } x \neq i\text{-esima cifra decimale di } f(i)$$

- Per definizione stessa di  $x$ , ne segue che  $\nexists n \in \mathbb{N} \mid f(n) = x$ , implicando che  $f$  non sia suriettiva, contraddicendo l'ipotesi per cui essa sia biettiva
- Di conseguenza, ne segue necessariamente che  $\nexists f : \mathbb{N} \rightarrow [0, 1]$  biettiva, implicando che  $|\mathbb{N}| < |[0, 1]| \leq |\mathbb{R}|$  e dunque che  $\mathbb{R}$  non sia numerabile

□

$n$	$f(n)$
0	0. <u>1</u> 18285101...
1	0.2 <u>1</u> 3812941...
2	0.12 <u>3</u> 124112... $\implies x = 0.67392...$
3	0.945 <u>8</u> 53164...
4	0.3924 <u>8</u> 1412...
$\vdots$	$\vdots$

*Rappresentazione grafica della dimostrazione*

**Teorema 32: Sequenze binarie infinite non numerabili**

L'insieme  $\mathcal{B}$  di tutte le stringhe binarie infinite **non è numerabile**

*Dimostrazione.*

- Supponiamo per assurdo che  $\exists f : \mathbb{N} \rightarrow \mathcal{B}$  biettiva
- Consideriamo la sequenza binaria  $x$  definita come:

$$\forall i \geq 1 \quad i\text{-esima cifra di } x \neq i\text{-esima cifra di } f(i)$$

- Per definizione stessa di  $x$ , ne segue che  $\nexists n \in \mathbb{N} \mid f(n) = x$ , implicando che  $f$  non sia suriettiva, contraddicendo l'ipotesi per cui essa sia biettiva
- Di conseguenza, ne segue necessariamente che  $\nexists f : \mathbb{N} \rightarrow \mathcal{B}$  biettiva, implicando che  $|\mathbb{N}| < |\mathcal{B}|$  e dunque che  $\mathcal{B}$  non sia numerabile

□

$n$	$f(n)$
0	<u>1</u> 0101010101...
1	1 <u>1</u> 101101011...
2	010 <u>1</u> 0100001... $\implies x = 00110...$
3	0000 <u>1</u> 010100...
4	11111 <u>1</u> 111111...
$\vdots$	$\vdots$

*Rappresentazione grafica della dimostrazione*

### 3.3.1 Esistenza di linguaggi non riconoscibili

#### Teorema 33: Esistenza di linguaggi non riconoscibili

Dato un alfabeto  $\Sigma$ , si ha che:

$$\exists L \subseteq \Sigma^* \mid L \notin \text{REC}$$

*Dimostrazione.*

- Sia  $<_\ell$  la relazione definita su  $\Sigma^*$  tale che:

$$\forall x, y \in \Sigma^* \quad x <_\ell y \iff x \text{ precede } y \text{ lessico-graficamente}$$

- Sia inoltre  $\prec$  la relazione definita su  $\Sigma^*$  tale che:

$$\forall x, y \in \Sigma^* \quad x \prec y \iff (|x| < |y|) \vee (|x| = |y| \wedge x <_\ell y)$$

ossia che ordina le stringhe di  $\Sigma^*$  in base alla loro lunghezza e, a parità di lunghezza, in base al loro ordine lessico-grafico

Dalla definizione stessa di  $\prec$ , risulta evidente che tale relazione sia un ordine totale.

- Sia quindi  $f : \mathbb{N} \rightarrow \Sigma^*$  la funzione definita come:

$$f(i) = i\text{-esima stringa di } \Sigma^* \text{ secondo } \prec$$

Tale funzione risulta intuitivamente essere biettiva, implicando che  $|\mathbb{N}| = |\Sigma^*|$ , dunque che  $\Sigma^*$  sia numerabile

- Consideriamo quindi il linguaggio  $\mathcal{M} \subseteq \Sigma^*$  definito come:

$$\mathcal{M} = \{\langle M \rangle \mid M \text{ è una TM}\}$$

Poiché  $\mathcal{M} \subseteq \Sigma^*$  e  $\Sigma^*$  è numerabile, ne segue automaticamente che anche  $\mathcal{M}$  sia numerabile

- Consideriamo inoltre l'insieme  $\mathcal{L} = \mathcal{P}(\Sigma^*)$ , corrispondente alla classe di tutti i linguaggi definiti su  $\Sigma$
- Dato un linguaggio  $L \in \mathcal{L}$ , definiamo la sequenza binaria  $\chi_L = b_1b_2\dots$  come *sequenza caratteristica di  $L$* , definita come:

$$b_i = \begin{cases} 1 & \text{se } s_i \in L \\ 0 & \text{se } s_i \notin L \end{cases}$$

dove  $s_1, s_2, \dots$  sono tutte le stringhe di  $\Sigma^*$

- Consideriamo quindi la seguente funzione:

$$g : \mathcal{L} \rightarrow \mathcal{B} : L \mapsto \chi_L$$

Tale funzione risulta intuitivamente essere biettiva, implicando che  $|\mathcal{L}| = |\mathcal{B}|$ . Di conseguenza, poiché  $\mathcal{B}$  non è numerabile, ne segue che anche  $\mathcal{L}$  non sia numerabile

- A questo punto, poiché  $\mathcal{M}$  è numerabile e  $\mathcal{L}$  no, concludiamo che la seguente funzione:

$$h : \mathcal{M} \rightarrow \mathcal{L} : M \mapsto L(M)$$

non sia biettiva, implicando che  $\exists L \in \mathcal{L} \mid \nexists M \in \mathcal{M} \text{ t.c. } L = L(M)$

□

## 3.4 Problemi indecidibili

### Teorema 34: Problema dell'accettazione per le TM

Sia  $A_{\text{TM}}$  il linguaggio definito come:

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ TM}, w \in L(M)\}$$

Tale linguaggio è **indecidibile**, ossia  $A_{\text{TM}} \in \text{REC} - \text{DEC}$

*Dimostrazione riconoscibilità.*

- Sia  $U$  una TM universale a 2 nastri definita come:

$U = \text{"Data in input la codifica } \langle M, w \rangle, \text{ dove } M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}}) \text{ è una TM e } w \text{ una stringa:}$

1. Se la codifica in input è errata,  $M$  *rifiuta*
2.  $M$  scrive  $\langle M, w \rangle$  sul nastro 1
3.  $M$  scrive  $\langle q_{\text{start}}, w \rangle$  sul nastro 2

4. Ripeti lo step seguente:

5. Sia  $\langle(x, q, y)\rangle$  la stringa attuale sul nastro 2, dove  $x, y \in \Sigma^*$ .

$M$  scansiona il nastro 1 in cerca di  $\langle\delta\rangle$ . Una volta trovato,  $M$  cerca una stringa  $\langle(q, a), (r, b, Z)\rangle$ , dove  $\delta(q, a) = (r, b, Z)$  e  $Z \in \{L, R\}$

6. Se  $a \neq y[i]$ ,  $M$  cerca la prossima regola valida

7. Se  $a = y[i]$ ,  $M$  scrive sul nastro due la configurazione prodotta dalla configurazione  $xqy$  passando per la transizione  $\delta(q, a) = (r, b, Z)$

8. Se nel nastro 2 è scritto  $\langle q_{\text{accept}} \rangle$ ,  $M$  accetta. Se è scritto  $\langle q_{\text{reject}} \rangle$ ,  $M$  rifiuta"

- Per costruzione stessa di  $U$ , si ha che:

$$\langle M, w \rangle \in L(U) \iff w \in L(M) \iff \langle M, w \rangle \in A_{\text{TM}}$$

implicando che  $A_{\text{TM}} = L(U) \in \text{REC}$ .

**Nota:** poiché  $M$  potrebbe andare in loop, anche  $U$  può andare in loop, implicando che essa non sia un decisore.

□

*Dimostrazione indecidibilità.*

- Supponiamo per assurdo che  $A_{\text{TM}} \in \text{DEC}$ . Sia quindi  $H$  la TM decisore tale che  $L(H) = A_{\text{TM}}$
- Sia  $D$  la TM definita come:

$D =$  "Data in input la codifica  $\langle M, w \rangle$ , dove  $M$  è una TM e  $w$  una stringa:

1. Esegui il programma di  $H$  con input  $\langle M, w \rangle$
2. Se l'esecuzione accetta,  $D$  rifiuta, altrimenti accetta

- Per costruzione stessa di  $D$ , si ha che:

$$\langle M, w \rangle \in L(D) \iff \langle M, w \rangle \notin L(H) = A_{\text{TM}} \iff w \notin L(M)$$

Inoltre, poiché  $H$  è un decisore, ne segue che anche  $D$  sia un decisore, implicando che essa possa solo accettare o rifiutare, senza altre opzioni

- Consideriamo quindi la codifica  $\langle D, \langle D \rangle \rangle$ . Notiamo che:

$$\begin{aligned} \langle D, \langle D \rangle \rangle \in L(D) &\iff \langle D, \langle D \rangle \rangle \notin L(H) = A_{\text{TM}} \\ &\iff \langle D \rangle \notin L(D) \iff \langle D, \langle D \rangle \rangle \notin L(D) \end{aligned}$$

ottenendo quindi una contrazione in quanto  $D$  possa solo accettare o rifiutare

- Di conseguenza, ne segue necessariamente che  $A_{\text{TM}} \notin \text{DEC}$

□

**Corollario 5: Gerarchia dei linguaggi**

Dato un alfabeto  $\Sigma$ , si ha che:

$$\text{REG} \subsetneq \text{CFL} \subsetneq \text{DEC} \subsetneq \text{REC} \subsetneq \mathcal{P}(\Sigma^*)$$

(segue dai teoremi 9, 25, 33 e 34)

**Definizione 49: Classe dei linguaggi coTuring-riconoscibili**

Dato un alfabeto  $\Sigma$ , definiamo come **classe dei linguaggi coTuring-riconoscibili di  $\Sigma$**  il seguente insieme:

$$\text{COREC} = \{L \subseteq \Sigma^* \mid \bar{L} \in \text{REC}\}$$

**Nota:**  $\text{COREC} \neq \mathcal{P}(\Sigma^*) - \text{REC}$

**Teorema 35: Decidibilità, riconoscibilità e co-riconoscibilità**

Un linguaggio  $L$  è **decidibile** se e solo se è **riconoscibile** e **co-riconoscibile**.

In altre parole, si ha che:

$$\text{DEC} = \text{REC} \cap \text{COREC}$$

*Dimostrazione.*

*Prima implicazione.*

- Dato  $L \in \text{DEC}$ , sia  $M$  la TM decisore tale che  $L = L(M)$
- Sia  $\bar{M}$  la TM definita come:

$\bar{M} =$  "Data in input la stringa  $w$ :

1. Esegui il programma di  $M$  con input  $w$
2. Se l'esecuzione accetta,  $\bar{M}$  rifiuta, altrimenti accetta

- Per costruzione stessa di  $\bar{M}$ , si ha che:

$$w \in L(\bar{M}) \iff w \notin L(M)$$

implicando che  $\bar{L} = \overline{L(M)} = L(\bar{M}) \in \text{REC}$

- Dunque, poiché  $L \in \text{DEC} \subseteq \text{REC}$  e  $\bar{L} \in \text{REC}$ , ne segue che  $L \in \text{REC} \cap \text{COREC}$

*Seconda implicazione.*

- Dato  $L \in \text{REC} \cap \text{COREC}$ , siano  $M$  e  $\overline{M}$  le TM tali che  $L = L(M)$  e  $\overline{L} = L(\overline{M})$
- Sia  $D$  la TM definita come:

$D =$  "Data in input la stringa  $w$ :

1. Esegui in parallelo, ossia alternando ad ogni istruzione le loro esecuzioni, i programmi di  $M$  e  $\overline{M}$  con input  $w$
2. Se l'esecuzione di  $M$  accetta,  $D$  accetta. Se l'esecuzione di  $\overline{M}$  accetta,  $D$  rifiuta

- Per costruzione stessa di  $D$ , si ha che:

$$w \in L(D) \iff w \in L(M)$$

implicando che  $L(D) = L(M) = L$

- Inoltre, per definizione stessa si ha che:

$$w \in L = L(M) \iff w \notin \overline{L} = L(\overline{M})$$

Di conseguenza, una delle due esecuzioni parallele accetterà qualsiasi stringa in input, implicando che  $D$  non vada mai in loop e quindi che  $L = L(D) \in \text{DEC}$

□

### Corollario 6

Dato  $L \subseteq \Sigma^*$ , si ha che:

$$L \in \text{REC} - \text{DEC} \implies \overline{L} \notin \text{REC}$$

(segue dal teorema 35)

**Esempio:**

- Il linguaggio  $\overline{A_{\text{TM}}}$  **non è riconoscibile**