

# Linguaggi e Computabilità

UniShare

Davide Cozzi  
@dlcgold

Gabriele De Rosa  
@derogab

Federica Di Lauro  
@f\_dila

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Definizioni . . . . .	2
1.1.1	Alberi Sintatici . . . . .	14
1.1.2	Grammatiche ambigue . . . . .	18
1.1.3	Grammatiche Regolari . . . . .	20
1.1.4	Espressioni Regolari (Regex) . . . . .	23
1.2	Automi . . . . .	29
1.2.1	Automi deterministici . . . . .	30
1.2.2	Automi non deterministici . . . . .	36
1.2.3	Da espressioni regolari a automi E-NFA . . . . .	50
1.2.4	Chiusura di un Linguaggio regolare . . . . .	57
1.2.5	Minimizzazione . . . . .	62
1.2.6	Pumping Lemma per i linguaggi regolari . . . . .	69
1.3	Automi a Pila . . . . .	71
1.3.1	Esercizi . . . . .	83
1.4	Macchine di Turing . . . . .	88
1.4.1	Riduzioni . . . . .	89
1.4.2	Restrizioni delle macchine di Turing . . . . .	93
1.4.3	Macchina di Turing Universale . . . . .	98

# Capitolo 1

## Introduzione

Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: <https://github.com/dlclgold/Appunti>.

Grazie mille e buono studio!

### 1.1 Definizioni

- un **linguaggio** è un insieme di stringhe che può essere generato mediante un dato meccanismo con delle date caratteristiche; un linguaggio può essere riconosciuto, ovvero dando in input una stringa un meccanismo può dirmi se appartiene o meno ad un linguaggio. I meccanismi che generano linguaggi si chiamano *grammatiche*, quelli che li riconoscono *automi*. I linguaggi formali fanno parte dell'informatica teorica (*TCS*)
- si definisce **alfabeto** come un insieme finito e non vuoto di simbolo (come per esempio il nostro alfabeto o le cifre da 0 a 9). Solitamente si indica con  $\Sigma$  o  $\Gamma$
- si definisce **stringa** come una sequenza finita di simboli (come per esempio una parola o una sequenza numerica). La stringa vuota è una sequenza di 0 simboli, e si indica con  $\varepsilon$  o  $\lambda$
- si definisce **lunghezza di una stringa** il numero di simboli che la compone (ovviamente contando ogni molteplicità). Se si ha  $w \in \Sigma^*$  è una stringa  $w$  con elementi da  $\Sigma^*$  (insieme di tutte le stringhe di tutte le lunghezze possibili fatte da  $\Sigma$ ), allora  $|w|$  è la lunghezza di  $w$ , inoltre  $|\varepsilon| = 0$ .

- si definisce **potenza di un alfabeto**  $\Sigma^k$  come l'insieme di tutte le sequenze (espressi come stringhe e non simboli) di lunghezza  $k \in \mathbb{N}$ ,  $k > 0$  ottenibili da quell'alfabeto (se  $\Sigma^2$  si avranno tutte le sequenze di 2 elementi etc...). Se ho  $k = 1$  si ha  $\Sigma^1 \neq \Sigma$  in quanto ora ho stringhe e non simboli. Se ho  $k = 0$  ho  $\Sigma^0 = \varepsilon$ . Dato  $k$  ho  $|\Sigma|$  che è la cardinalità dell'insieme  $\Sigma$  (e non la sua lunghezza come nel caso delle stringhe); sia  $w \in \Sigma^k = a_1, a_2, \dots, a_k$ ,  $a_i \in \Sigma$  e  $|\Sigma| = q$  ora:

$$|\Sigma^k| = q^k$$

- si definisce  $\Sigma^*$  come **chiusura di Kleene** che è l'unione infinita di  $\Sigma^k$  ovvero

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \dots \cup \Sigma^k$$

- si ha che  $\Sigma^+$  è l'unione per  $k \geq 1$  di  $\Sigma^k$  ovvero:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^k = \Sigma^* - \Sigma^0$$

per esempio, per l'insieme  $\{0, 1\}$  si ha:

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 100, 000, \dots\}$$

- quindi un **linguaggio**  $L$  è un insieme di stringhe e:

$$L \subseteq \Sigma^*$$

si hanno sottoinsiemi particolari, come l'insieme vuoto, che resta però un linguaggio, il **linguaggio vuoto** e  $\emptyset \in \Sigma^k$ ,  $|\emptyset| = 0$  che è diverso dal linguaggio che contiene la stringa vuota  $|\varepsilon| = 1$  (che conta come una stringa). Inoltre  $\Sigma^* \subseteq \Sigma^*$  che ha lunghezza infinita. Posso concatenare due stringhe con un punto:  $a \cdot b \cdot c = abc$  e  $a \cdot \varepsilon = a$ . Ovviamente la stringa concatenata è lunga come la somma delle lunghezze delle stringhe che la compongono. Vediamo qualche esempio di linguaggio:

- il linguaggio di tutte le stringhe che consistono in  $n$  0 seguiti da  $n$  1:

$$\{\varepsilon, 01, 0011, 000111, \dots\}$$

- l'insieme delle stringhe con un uguale numero di 0 e di 1:

$$\{\varepsilon, 01, 10, 0011, 0101, 1001, \dots\}$$

- l'insieme dei numeri binari il cui valore è un numero primo:

$$\{\varepsilon, 10, 11, 101, 111, 1011, \dots\}$$

- $\Sigma^*$  è un linguaggio per ogni alfabeto  $\Sigma$
- $\emptyset$ , il linguaggio vuoto, e  $\{\varepsilon\}$  sono un linguaggio rispetto a qualunque alfabeto

Prendiamo un alfabeto  $\Sigma = \{0, 1\}$  con la sua chiusura di Kleen  $\Sigma = \{0, 1\}^*$ . Quando si ha un input si può avere un problema di decisione,  $P$ , che dia come output "si" o "no". Posso avere un problema di decisione (o *membership*) su  $w \in \Sigma = \{0, 1\}^*$ , con  $w$  stringa, che dia in output "si" o "no". Un linguaggio  $L$  sarà:

$$L = \{w \in \{0, 1\}^* \mid P(w) = \text{si}\}$$

quindi si ha che:

$$\Sigma^* \setminus L = \{P(w) = \text{no}\}$$

Vediamo ora un esempio di *Context Free Language (CFL)*, costruito a partire da una *Context Free Grammar (CFG)*:

**Esempio 1.** Sia  $\Sigma = \{0, 1\}$  e  $L_{pal} = \text{"stringhe palindrome binarie"}$ . Quindi, per esempio,  $0110 \in L$ ,  $11011 \in L$  ma  $10010 \notin L$ . Si ha che  $\varepsilon$ , la stringa vuota, appartiene a  $L$ . Diamo una definizione ricorsiva:

- **base:**  $\varepsilon, 0, 1 \in L_{pal}$
- **passo:** se  $w$  è palindroma allora  $0w0$  è palindromo e  $1w1$  è palindromo

una variabile generica  $S$  può sottostare alle regole di produzione di una certa grammatica. In questo caso si ha uno dei seguenti:

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

Si ha che una grammatica  $G$  è una quadrupla  $G = (V, T, P, S)$  con:

- $V$  simboli variabili
- $T$  simboli terminali, ovvero i simboli con cui si scrivono le stringhe alla fine
- $P$  regole di produzione
- $S$  variabile di partenza *start*

riprendiamo l'esempio sopra:

**Esempio 2.**

$$G_{pal} = (V = \{S\}, T = \{0, 1\}, P, S)$$

con:

$$P = \{S \rightarrow \varepsilon, S \rightarrow 0, S \rightarrow 1, S \rightarrow 0S0, S \rightarrow 1S1\}$$

Si può ora costruire un algoritmo per creare una stringa palindroma a partire dalla grammatica  $G$ :

$$\underbrace{S}_{\text{start applico una regola}} \xrightarrow{\quad} 1S1 \rightarrow 01S10 \rightarrow \underbrace{01010}_{\text{sostituisco variabile}}$$

con  $S$ ,  $1S1$  e  $01S10$  che sono forme sentenziali. Posso così ottenere tutte le possibili stringhe. Esiste anche una forma abbreviata:

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

Non si fanno sostituzioni in parallelo, prima una  $S$  e poi un'altra

Si hanno 4 grammatiche formali, *gerarchia di Chomsky*:

- **tipo 0:** non si hanno restrizioni sulle regole di produzione,  $\alpha \rightarrow \beta$ . Sono linguaggi ricorsivamente numerabili e sono rappresentati dalle *macchine di Turing*, deterministiche o non deterministiche (la macchina di Turing è un automa)
- **tipo 1:** il lato destro della produzione ha lunghezza almeno uguale a quello sinistro. Sono grammatiche dipendenti dal contesto (*contestuali*) e come automa hanno *la macchina di Turing che lavora in spazio lineare*:

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 B \alpha_2$$

con  $\alpha_1$  e  $\alpha_2$  detti *contesto* e  $\alpha_1, \alpha_2, \beta \in (V \cup T)^*$

- **tipo 2:** sono quelle libere dal contesto, context free. Come regola ha  $A \rightarrow \beta$  con  $A \in V$  e  $\beta \in V \cup T)^*$  e come automa ha gli *automi a pila non deterministici*
- **tipo 3:** sono le grammatiche *regolari*. Come regole ha  $A \rightarrow \alpha B$  (o  $A \rightarrow B\alpha$ ) e  $A \rightarrow \alpha$  con  $A, B \in V$  e  $\alpha \in T$ . Come automi ha gli *automi a stato finito deterministici o non deterministici*

**Esempio 3.** Sia  $G = (V, T, O, E)$ , con  $V = \{E, I\}$  e  $T = \{a, b, 0, 1, (, ), +, *\}$  quindi ho le seguenti regole, è di tipo 3:

1.  $E \rightarrow I$
2.  $E \rightarrow E + E$
3.  $E \rightarrow E * E$
4.  $E \rightarrow (E)$
5.  $I \rightarrow a$
6.  $I \rightarrow b$
7.  $I \rightarrow Ia$
8.  $I \rightarrow Ib$
9.  $I \rightarrow I0$
10.  $I \rightarrow I1$

voglio ottenere  $a*(a+b00)$  sostituisco sempre a destra (*right most derivation*)

$$E \rightarrow E * E \rightarrow E * (E) \rightarrow E * (E + E) \rightarrow E * (E + I) \rightarrow E + (E + I0) \\ \rightarrow R + (I + b00) \rightarrow E * (a + b00) \rightarrow I * (a + b00) \rightarrow a * (a + b00)$$

usiamo ora l'inferenza ricorsiva:

passo	stringa ricorsiva	var	prod	passo stringa impiegata
1	a	I	5	\
2	b	I	6	\
3	b0	I	9	2
4	b00	I	9	3
5	a	E	1	1
6	b00	E	1	4
7	a+b00	E	2	5,6
8	(a+b00)	E	4	7
9	a*(a+b00)	E	3	5, 8

definisco formalmente la derivazione  $\rightarrow$ :

**Definizione 1.** Prendo una grammatica  $G = (V, T, P, S)$ , grammatica CFG. Se  $\alpha A \beta$  è una stringa tale che  $\alpha, \beta \in (V \cup T)^*$ , appartiene sia a variabili che terminali. Sia  $A \in V$  e sia  $a \rightarrow \gamma$  una produzione di  $G$ . Allora scriviamo:

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

con  $\gamma \in (V \cup T)^*$ .

Le sostituzioni si fanno indipendentemente da  $\alpha$  e  $\beta$ . Questa è quindi la definizione di derivazione.

**Definizione 2.** Definisco il simbolo  $\rightarrow_*$ , ovvero il simbolo di derivazioni in 0 o più passi. Può essere definito in modo ricorsivo. Per induzione sul numero di passi.

- la base dice che  $\forall \alpha \in (V \cup T)^*, \alpha \rightarrow_* \alpha$
- il passo è: se  $\alpha \rightarrow_G * \beta$  e  $\beta \rightarrow_G * \gamma$  allora  $\alpha \rightarrow_* \gamma$

Si può anche dire che  $\alpha \rightarrow_G * \beta$  sse esiste una sequenza di stringhe  $\gamma_1, \dots, \gamma_n$  con  $n \geq 1$  tale che  $\alpha = \gamma_1$ ,  $\beta = \gamma_n$  e  $\forall i, 1 < i < n - 1$  si ha che  $\gamma_i \rightarrow \gamma_{i+1}$  la derivazione in 0 o più passi è la chiusura transitiva della derivazione

**Definizione 3.** avendo ora definito questi simboli possiamo definire una forma sentenziale. Infatti è una stringa  $\alpha$  tale che:

$$\forall \alpha \in (V \cup T)^* \text{ tale che } S \rightarrow_G * \alpha$$

**Definizione 4.** data  $G = (V, T, P, S)$  si ha che  $L(G) = \{w \in T^* \mid S \rightarrow_G * w\}$  ovvero composto da stringhe terminali che sono derivabili o 0 o più passi.

**Esempio 4.** formare una grammatica CFG per il linguaggio:

$$L = \{0^n 1^n \mid n \geq 1\} = \{01, 0011, 000111, \dots\}$$

con  $x^n$  intendo una concatenazione di  $n$  volte  $x$  (che nel nostro caso sono 0 e 1).

posso scrivere:

$$0^n 1^n = 00^{n-1} 1^{n-1} 1$$

il nostro caso base sarà la stringa 01, Poi si ha:  $G = (V, T, P, S)$ ,  $T = \{0, 1\}$ ,  $V = \{S\}$ , il caso base  $S \rightarrow 01$  e  $S \rightarrow 0S1$  il caso passo è quindi: se  $w = 0^{n-1} 1^{n-1} \in L$  allora  $0w1 \in L$ .

Ora voglio dimostrare che  $000111 \in L$ , ovvero  $S \rightarrow_* 000111$ :

$$S \rightarrow 0S1 \rightarrow 00S11 \rightarrow 000S111$$



**Teorema 1.** data la grammatica  $G = \{V, T, P, S\}$  CFG e  $\alpha \in (V \cup T)^*$ . Si ha che vale  $S \rightarrow * \alpha$  sse  $S \rightarrow_{lm} * \alpha$  sse  $S \rightarrow_{rm} * \alpha$ . Con  $\rightarrow_{lm} *$  simbolo di left most derivation e  $\rightarrow_{rm} *$  simbolo di right most derivation

**Esempio 5.** formare una grammatica CFG per il linguaggio:

$$L = \{0^n 1^n | n \geq 0\} = \{\varepsilon, 01, 0011, 000111, \dots\}$$

stavolta abbiamo anche la stringa vuota. Il caso base stavolta è  $S \rightarrow \varepsilon | 0S1$

**Esempio 6.** Fornisco una CFG per  $L = \{a^n | n \geq 1\} = \{a, aa, aaa, \dots\}$ . La base è  $a$

il passo è che se  $a^{n-1} \in L$  allora  $a^{n-1}a \in L$  ( o che  $aa^{n-1} \in L$ ).

Si ha la grammatica  $G = \{V, T, P, S\}$ ,  $V = \{S\}$ ,  $T = \{a\}$  e si hanno  $S \rightarrow a | Sa$  (o  $S \rightarrow a | aS$ ). Dimostro che  $a^3 \in L$ .

$$S \rightarrow Sa \rightarrow Saa \rightarrow aaa$$

oppure

$$S \rightarrow aS \rightarrow aaS \rightarrow aaa$$

**Esempio 7.** trovo una CFG per  $L = \{(ab)^n | n \geq 1\} = \{ab, abab, ababab, \dots\}$

La base è  $ab$

il passo è che se  $(ab)^{n-1} \in L$  allora  $(ab)^{n-1}ab \in L$ .

Si ha la grammatica  $G = \{V, T, P, S\}$ ,  $V = \{S\}$ ,  $T = \{a, b\}$  (anche se in realtà  $T = \{ab\}$ ) e si hanno  $S \rightarrow ab | Aab$ . Poi dimostro come l'esempio sopra

**Esempio 8.** trovo una CFG per  $L = \{a^n cb^n | n \geq 1\} = \{acb, aacbb, aaacbbb, \dots\}$

Il caso base è  $acb$  il passo è che se  $a^{n-1}cb^{n-1} \in L$  allora  $a^{n-1}cb^{n-1}acb \in L$

Si ha la grammatica  $G = \{V, T, P, S\}$ ,  $V = \{S\}$ ,  $T = \{a, b, c\}$  e si hanno  $S \rightarrow aSb | acb$ .

dimostro che  $aaaacbbbbb \in L$ :

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaaSbbb \rightarrow aaaacbbbbb$$

provo a usare anche una grammatica regolare, con le regole  $S \rightarrow aS | c$ ,  $c \rightarrow cB$  e  $B \rightarrow bB | b$ ;

$$S \rightarrow aS \rightarrow aaS \rightarrow aaC \rightarrow aacB \rightarrow aacb \dots$$

non si può dimostrare in quanto non si può imporre una regola adatta

**Esempio 9.**  $L = \{a^n cb^{n-1} | n \geq 2\}$ , con  $a^n cb^{n-1} = a^{n-1}acb^{n-1}$ .  $S \rightarrow aSb | aacb$ . Quindi:

$$S \rightarrow aSb \rightarrow aaacbb \in L$$

**Esempio 10.** cerco CFG per  $L = \{a^n c^k b^n \mid n, k > 0\}$ .  $a$  e  $b$  devono essere uguali, uso quindi una grammatica context free, mentre  $c$  genera un linguaggio regolare.

Si ha la grammatica  $G = \{V, T, P, S\}$ ,  $V = \{S, C\}$ ,  $T = \{a, b, c\}$  e si hanno  $S \rightarrow aSb \mid aCb$  e  $C \rightarrow cC \mid c$ . dimostro che  $aaaccbbb \in L$ ,  $n = 3$ ,  $k = 2$ :

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaCbbb \rightarrow aaacCbbb \rightarrow aaaccbbb$$

**Esempio 11.** scrivere CFG per  $L = \{a^n b^n c^k b^k \mid n, k \geq 0\}$

$$= \{w \in \{a, b, c, d\}^* \mid a^n b^n c^k b^k \mid n, k \geq 0\}$$

quindi  $L$  concatena due linguaggi  $L1$  e  $L2$ ,  $X = \{a^n b^n\}$  e  $Y = \{c^k d^k\}$ :

$$X \rightarrow aXb \mid \varepsilon$$

$$Y \rightarrow cYd \mid \varepsilon$$

$$S \rightarrow XY$$

voglio derivare  $abcd$ :

$$S \rightarrow XY \rightarrow XcYd \rightarrow aXbcYd \rightarrow aXbc\varepsilon d \rightarrow a\varepsilon bc\varepsilon d \rightarrow abcd$$

voglio derivare  $cd$

$$S \rightarrow XY \rightarrow Y \rightarrow cYd \rightarrow cd$$

Quindi se ho  $w \in L1, L2$ , ovvero appartenente ad una concatenazione di linguaggi prima uso le regole di un linguaggio, poi dell'altro e infine ottengo il risultato finale.

**Esempio 12.** scrivere CFG per  $L = \{a^n b^k c^k d^n \mid n > 0, k \geq 0\}$ .

$$S \rightarrow aSd \mid aXd$$

$$X \rightarrow bXc \mid \varepsilon$$

derivo  $aabcbdd$ :

$$S \rightarrow aSd \rightarrow aaXdd \rightarrow aabXcdd \rightarrow aabcbdd$$

**Esempio 13.** scrivere CFG per  $L = \{a^n cb^n c^m ad^m \mid n > 0, m \geq 1\}$ .

$$S \rightarrow XY$$

$$X \rightarrow aXb \mid c$$

$$Y \rightarrow cUd \mid cad$$

$$S \rightarrow XY \rightarrow cY \rightarrow ccad$$

**Esempio 14.** scrivere CFG per  $L = \{a^{n+m}xc^nyd^m \mid n, m \geq 0\}$ .  $a^{n+m} = a^n a^m$  o  $a^m a^n$ . Si hanno 2 casi:

$$1. L = \{a^n a^m x c^n y d^m \mid n, m \geq 0\}$$

$$2. L = \{a^m a^n x c^n y d^m \mid n, m \geq 0\}$$

ma solo  $L = \{a^m a^n x c^n y d^m \mid n, m \geq 0\}$  può generare una CFG (dove non si possono fare incroci, solo concatenazioni e inclusioni/innesti).

$$S \rightarrow aSd \mid Y$$

$$Y \rightarrow Xy$$

$$X \rightarrow aXc \mid x$$

si può fare in 2:

$$S \rightarrow aSd \mid Xy$$

$$X \rightarrow aXc \mid x$$

derivo con  $m = n = 1$ ,  $aa x c y d$ :

$$S \rightarrow aSd \rightarrow aXyd \rightarrow aaXcyd \rightarrow aa x c y d$$

**Esempio 15.** scrivere CFG per  $L = \{a^n b^m \mid n \geq m \geq 0\}$ .

$$L = \{\varepsilon, a, ab, aa, aab, aabb, aaa, aaab, aaabb, aaabbb, \dots\}$$

Se  $n \geq m$  allora  $\exists k \geq 0 \rightarrow n = m + k$ . Quindi:

$$l = \{a^{m+k} b^m \mid m, k \geq 0\}$$

si può scrivere in 2 modi:

$$1. l = \{a^m a^k b^m \mid m, k \geq 0\} \text{ quindi con innesto}$$

$$2. l = \{a^k a^m b^m \mid m, k \geq 0\} \text{ quindi con concatenazione}$$

entrambi possibili per una CFG:

1.

$$S \rightarrow XY$$

$$X \rightarrow aX \mid \varepsilon \text{ si può anche scrivere } X \rightarrow Xa \mid \varepsilon$$

$$Y \rightarrow aYb \mid \varepsilon$$

oppure

$$S \rightarrow aS \mid X$$

$$X \rightarrow aXb \mid \varepsilon$$

2.

$$S \rightarrow aSb|\varepsilon$$

$$X \rightarrow aX|\varepsilon$$

**Esempio 16.** scrivere CFG per  $L = \{a^n b^{m+n} c^h \mid m > h \geq 0, n \geq 0\}$ .  
Se  $n > h$  allora  $\exists k \rightarrow n = h + k$ , quindi:

$$L = \{a^n b^{m+h+k} c^h \mid m > h \geq 0, n \geq 0\}$$

. ovvero:

$$L = \{a^n b^n b^k b^h c^h \mid m \geq 0, k > 0, h \geq 0\}$$

si ha:

$$S \rightarrow XYZ$$

$$X \rightarrow aXb|\varepsilon$$

$$Y \rightarrow Yb|b$$

$$Z \rightarrow bZc|\varepsilon$$

si può anche fare:

$$S \rightarrow XY$$

$$X \rightarrow aXb|\varepsilon$$

$$Y \rightarrow bYc|Z$$

$$Z \rightarrow bZ|b$$

**Esempio 17.** scrivere CFG per  $L = \{a^n b^m c^k \mid k > n + m, n, m \geq 0\}$ .  
per  $n = m = 0, k = 1$  avrò la stringa  $c$ . se  $k > n + m$  allora  $\exists l > 0 \rightarrow k = n + m + l$  quindi:

$$L = \{a^n b^m c^{n+m+l} \mid l > 0, n, m \geq 0\}$$

$$= L = \{a^n b^m c^n c^m c^l \mid l > 0, n, m \geq 0\}$$

sistemando:

$$= L = \{a^n b^m c^l c^m c^n c^l \mid l > 0, n, m \geq 0\}$$

quindi:

$$S \rightarrow aSc|X$$

$$X \rightarrow bXc|Y$$

$$Y \rightarrow cY|c$$

**Esempio 18.** scrivere CFG per  $L = \{a^n x c^{n+m} y^h z^k d^{m+h} \mid n, m, k, h \geq 0\}$ .  
ovvero:

$$L = \{a^n x c^n c^m y^h z^k d^h d^m \mid n, m, k, h \geq 0\}$$

quindi avrò:

$$\begin{aligned} S &\rightarrow XY \\ X &\rightarrow aXc \mid x \\ Y &\rightarrow cYd \mid W \\ W &\rightarrow yWd \mid X \\ Z &\rightarrow zZ \mid \varepsilon \end{aligned}$$

**Esempio 19.** vediamo un esempio di grammatica dipendente dal contesto:

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

$G = \{V, T, P, S\} = \{(S, B, C, X)\} = \{(a, b, c), P, S\}$  ecco le regole di produzione (qui posso scambiare variabili a differenza delle context free):

1.  $S \rightarrow aSBC$
2.  $S \rightarrow aBC$
3.  $CB \rightarrow XB$
4.  $XB \rightarrow XC$
5.  $XC \rightarrow BC$
6.  $aB \rightarrow ab$
7.  $bB \rightarrow bb$
8.  $bC \rightarrow bc$
9.  $cC \rightarrow cc$

vediamo un esempio di derivazione: per  $n = 1$  ho  $abc$  ovvero:

$$S \rightarrow aBC \rightarrow abC \rightarrow abc$$

con  $n = 2$  ho  $aabbcc$ :  $S \rightarrow aSBC \rightarrow aaBCBC \rightarrow aaBXBC \rightarrow aaBXCC \rightarrow aaBBCC \rightarrow aabBCC \rightarrow aabbCC \rightarrow aabbcc$

**Esempio 20.** vediamo un esempio di grammatica dipendente dal contesto:

$$L = \{a^n b^m c^n d^m \mid n, m \geq 1\}$$

Si ha:

$$G = (\{S, X, C, D, Z\}, \{a, b, c, d\}, P, S)$$

con le seguenti regole di produzione:

- $S \rightarrow aSc \mid aXc$
- $X \rightarrow bXD \mid bD$
- $DC \rightarrow CD$
- $DC \rightarrow DZ$
- $DZ \rightarrow CZ$
- $XZ \rightarrow CD$
- $bC \rightarrow bc$
- $cC \rightarrow cc$
- $cD \rightarrow cd$
- $dD \rightarrow dd$

provo a derivare  $aabbccddd$  quindi con  $n = 2, m = 3$ :

$$\begin{aligned} S &\rightarrow aSC \rightarrow aaXCC \rightarrow aabXDCC \rightarrow aabbXDDCC \rightarrow \\ &aabbbDDDCC \rightarrow aabbbCCDDD \rightarrow aabbccddd \end{aligned}$$

**Esempio 21.** Sia  $L = \{w \in \{a, b\}^* \mid w \text{ contiene lo stesso numero di } a \text{ e } b\}$ :

$$S \rightarrow aSbS \mid bSaS \mid \varepsilon$$

dimostro per induzione che è corretto:

- **caso base:**  $|w| = 0 \rightarrow w = \varepsilon$

- **caso passo:** si supponga che  $G$  produca tutte le stringhe (di lunghezza  $< n$ ) di  $\{a, b\}^*$  con lo stesso numero di  $a$  e  $b$  e dimostro che produce anche quelle di lunghezza  $n$ , sia:

$w \in \{a, b\}^* \mid |w| = n$  con  $a$  e  $b$  in egual numero,  $m(a) = m(b)$  con  $m()$  che indica il numero

quindi si ha che:

$$w = aw_1bw_2 \text{ o } w = bw_1aw_2$$

sia.

$$k_1 = m(a) \in w_1 = m(b) \in w_1$$

$$k_2 = m(a) \in w_2 = m(b) \in w_2$$

allora:

$$k_1 + k_2 + 1 = m(a) \in w = m(b) \in W$$

sapendo che  $|w_1| < n$  e  $|w_2| < n$  allora  $w_1$  e  $w_2$  sono egnerati da  $G$  per ipotesi induttiva

### 1.1.1 Alberi Sintatici

**Definizione 5.** Data una grammatica CFG,  $G = \{V, T, P, S\}$  un **albero sintattico** per  $G$  soddisfa le seguenti condizioni:

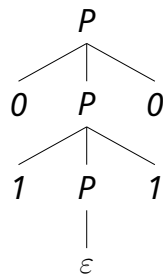
- ogni nodo interno è etichettato con una variabile
- ogni foglia è anch'essa etichettata con una variabile o col simbolo di terminale  $T$  o con la stringa vuota  $\varepsilon$  (in questo caso la foglia è l'unico figlio del padre)
- se un nodo interno è etichettato con  $A$  i suoi figli saranno etichettati con  $X_1, \dots, X_k$  e  $A \rightarrow X_1, \dots, X_k$  sarà una produzione di  $G$ . Se un  $X_i$  è  $\varepsilon$  sarà l'unica figlio e  $A \rightarrow \varepsilon$  sarà comunque una produzione di  $G$

La concatenazione in ordine delle foglie viene detto **prodotto dell'albero**

**Esempio 22.** Usiamo l'esempio delle stringhe palindrome:

$$P \rightarrow 0P0 \mid 1P1 \mid \varepsilon$$

sia il seguente albero sintattico:

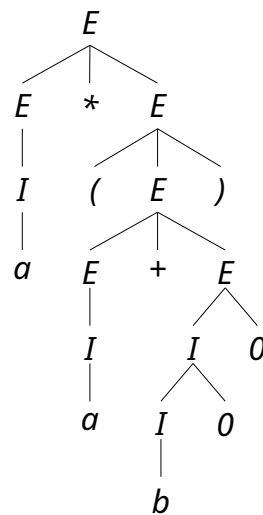


**Esempio 23.** Si ha:

$$E \rightarrow I \mid E + E \mid E * E \mid (E)$$

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

un albero sintattico per  $a * (a + b00)$  può essere:

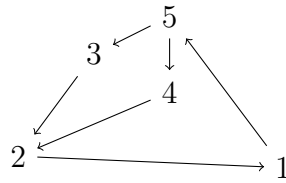




Data una CFG si ha che i seguenti cinque enunciati si equivalgono:

1. la procedura di inferenza ricorsiva stabilisce che una stringa  $w$  di simboli terminali appartiene al linguaggio  $L(A)$  con  $A$  variabile
2.  $A \rightarrow^* w$
3.  $A \rightarrow_{lm}^* w$
4.  $A \rightarrow_{rm}^* w$
5. esiste un albero sintattico con radice  $A$  e prodotto  $w$

queste 5 proposizioni si implicano l'uni l'altra:



vediamo qualche dimostrazione di implicazione tra queste proposizioni:

da 1 a 5. si procede per induzione:

- **caso base:** ho un livello solo (una sola riga),  $\exists A \rightarrow w$ :

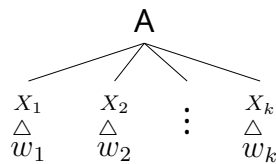
$$\begin{array}{c} A \\ \triangle \\ w \end{array}$$

- **caso passo:** suppongo vero per un numero di righe  $\leq n$ , lo dimostro per  $n + 1$  righe:

$$A \rightarrow X_1, X_2, \dots, X_k$$

$$w = w_1, w_2, \dots, w_k$$

ovvero, in meno di  $n + 1$  livelli:



□

da 5 a 3. procedo per induzione:

- **caso base (n=1):**  $\exists A \rightarrow w$  quindi  $A \rightarrow_{lm} w$ , come prima si ha un solo livello:

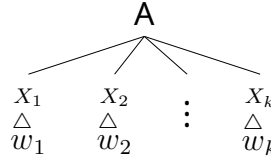
$$\begin{array}{c} A \\ \triangle \\ w \end{array}$$

- **caso passo:** suppongo che la proprietà valga per ogni albero di profondità minore uguale a  $n$ , dimostro che valga per gli alberi profondi  $n + 1$ :

$$A \rightarrow X_1, X_2, \dots, X_k$$

$$w = w_1, w_2, \dots, w_k$$

ovvero, in meno di  $n + 1$  livelli:



$$A \rightarrow_{lm} X_1, X_2, \dots, X_k$$

$x_1 \rightarrow_{lm}^* w_1$  per ipotesi induttiva si ha un albero al più di  $n$  livelli quindi:

$$A \rightarrow_{lm} X_1, \dots, X_k \rightarrow_{lm}^* w_1, X_2, \dots, X_k \rightarrow_{lm}^* \dots \rightarrow_{lm}^* w_1, \dots, w_k = w$$

**Esempio 24.**

$$E \rightarrow I \rightarrow Ib \rightarrow ab$$

$$\alpha E \beta \rightarrow \alpha I \beta \rightarrow \alpha Ib \beta \rightarrow \alpha ab \beta, \quad \alpha, \beta \in (V \cup T)^*$$

□

**Esempio 25.** Mostro l'esistenza di una derivazione sinistra dell'albero sintattico di  $a * (a + b00)$ :

$$\begin{aligned} & E \rightarrow_{lm}^* E * E \rightarrow_{lm}^* I * E \rightarrow_{lm}^* a * E \rightarrow_{lm}^* a * (E) \rightarrow_{lm}^* a * (E + E) \rightarrow_{lm}^* \\ & a * (I + E) \rightarrow_{lm}^* a * (a + E) \rightarrow_{lm}^* a * (a + I) \rightarrow_{lm}^* a + (a + I0) \rightarrow_{lm}^* a * (a + I00) \rightarrow_{lm}^* a * (a + b00) \end{aligned}$$

### 1.1.2 Grammatiche ambigue

**Definizione 6.** Una grammatica è definita ambigua se esiste una stringa  $w$  di terminali che ha più di un albero sintattico

**Esempio 26.** vediamo un esempio:

1.  $E \rightarrow E + E \rightarrow E + E * E$  ovvero:



2.  $E \rightarrow E * E \rightarrow E + E * E$  ovvero:



si arriva a due stringhe uguali ma con alberi diversi. Introduciamo delle categorie sintattiche, dei vincoli alla produzione delle regole:

1.  $E \rightarrow T \mid E + T$
2.  $T \rightarrow F \mid T * F$
3.  $F \rightarrow I \mid (E)$
4.  $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Possono esserci più derivazioni di una stringa ma l'importante è che non ci siano alberi sintattici diversi. Capire se una CFG è ambigua è un problema indecidibile

**Esempio 27.** vediamo un esempio:

$$S \rightarrow \varepsilon \mid SS \mid iS \mid iSeS$$

con  $S$ =statement,  $i$ =if e  $e$ =else. Considero due derivazioni:

1.  $S \rightarrow iSeS \rightarrow iiSeS \rightarrow iie$ :



2.  $S \rightarrow iS \rightarrow iiSeS \rightarrow iieS \rightarrow iie$ :



Si ha quindi una grammatica ambigua

**Teorema 2.** Per ogni CFG, con  $G = (V, T, P, S)$ , per ogni stringa  $w$  di terminali si ha che  $w$  ha due alberi sintattici distinti sse ha due derivazioni sinistre da  $S$  distinte.

Se la grammatica non è ambigua allora esiste un'unica derivazione sinistra da  $S$

### Linguaggi inerentemente ambigui

**Definizione 7.** Un linguaggio  $L$  è inerentemente ambiguo se tutte le grammatiche CFG per tale linguaggio sono a loro volta ambigue

**Esempio 28.** Sia  $L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b m n c^m d^n \mid n, m \geq 1\}$  si ha quindi un CFL formato dall'unione di due CFL.  $L$  è inerentemente ambiguo e generato dalla seguente grammatica:

- $S \rightarrow AB \mid C$

- $A \rightarrow aAb \mid ab$
- $B \rightarrow cBd \mid cd$
- $C \rightarrow aCd \mid aDd$
- $D \rightarrow bDc \mid bc$

si possono avere due derivazioni:

1.  $S \rightarrow_{lm} AB \rightarrow_{lm} aAbB \rightarrow_{lm} aabbB \rightarrow_{lm} aabbcBd \rightarrow_{lm} aabbccdd$
2.  $S \rightarrow_{lm} C \rightarrow_{lm} aCd \rightarrow_{lm} aaBdd \rightarrow_{lm} aabBcdd \rightarrow_{lm} aabbccdd$

a generare problemi sono le stringhe con  $n=m$  perché possono essere prodotte in due modi diversi da entrambi i sottolinguaggi. Dato che l'intersezione tra i due sottolinguaggi non è vuota si ha che  $L$  è ambiguo

### 1.1.3 Grammatiche Regolari

Sono le grammatiche che generano i linguaggi regolari (quelli del terzo tipo) che sono casi particolari dei CFL.

Si ha la solita grammatica  $G = (V, T, P, S)$  con però vincoli su  $P$ :

- $\varepsilon$  si può ottenere solo con  $S \rightarrow \varepsilon$
- le produzioni sono tutte lineari a destra ( $A \rightarrow aA$  o  $A \rightarrow a$ ) o a sinistra ( $A \rightarrow Ba$  o  $A \rightarrow a$ )

**Esempio 29.**  $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$  è una grammatica con le produzioni lineari a sinistra.

Potremmo pensarlo a destra  $I \rightarrow a \mid b \mid aI \mid bI \mid 0I \mid 1I$ .

Vediamo esempi di produzione con queste grammatiche:

- con  $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$  possiamo derivare  $ab01b0$ :

$$I \rightarrow I0 \rightarrow Ib0 \rightarrow I1b0 \rightarrow I01b0 \rightarrow Ib01b0 \rightarrow ab01b0$$

- con  $I \rightarrow a \mid b \mid aI \mid bI \mid 0I \mid 1I$  invece non riusciamo a generare nulla:

$$I \rightarrow 0I \rightarrow 0a$$

definisco quindi un'altra grammatica (con una nuova categoria sintattica):

$$I \rightarrow aJ \mid bJ$$

$$J \rightarrow a \mid b \mid aJ \mid bJ \mid 0J \mid 1J$$

che però non mi permette di terminare le stringhe con 0 e 1, la modifico ancora ottenendo:

$$I \rightarrow aJ \mid bJ$$

$$J \rightarrow a \mid b \mid aJ \mid bJ \mid 0J \mid 1J \mid 0 \mid 1$$

e questo è il modo corretto per passare da lineare sinistra a lineare destra

**Esempio 30.** Sia  $G = (\{S\}, \{0, 1\}, P, S)$  con  $S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S \mid 1S$ . Si ha quindi:

$$L(G) = \{0, 1\}^*$$

si hanno comunque due proposizioni ridondanti, riducendo trovo:

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

con solo produzioni lineari a destra. Con produzioni lineari a sinistra ottengo:

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

**Esempio 31.** Trovo una grammatica lineare destra e una sinistra per  $L = \{a^n b^m \mid n, m \geq 0\}$ :

- **lineare a destra:** si ha  $G = (\{S, B\}, \{a, b\}, P, S)$  e quindi:

$$S \rightarrow \varepsilon \mid aS \mid bB$$

$$B \rightarrow bB \mid b$$

ma non si possono generare stringhe di sole  $b$ , infatti:

$$S \rightarrow aS \rightarrow abB \rightarrow abbB \rightarrow abbb$$

ma aggiungere  $\varepsilon$  a  $B$  **non è lecito**. posso però produrre la stessa stringa da due derivazioni diverse:

$$S \rightarrow \varepsilon \mid aS \mid bB \mid b$$

$$B \rightarrow bB \mid b$$

che risulta quindi la nostra lineare a destra

- **lineare a sinistra:** si ha  $G = (\{S, A\}, \{a, b\}, P, S)$  e quindi:

$$S \rightarrow \varepsilon \mid Sb \mid Ab \mid a$$

$$A \rightarrow Aa \mid a$$

**Esempio 32.** Trovo una grammatica lineare destra e una sinistra per  $L = \{ab^n cd^m e \mid n \geq 0, m > 0\}$ :

- **lineare a destra:** si ha  $G = (\{S, A, B, E\}, \{a, b, c, d, e\}, P, S)$  e quindi:

$$S \rightarrow aA$$

$$A \rightarrow bA \mid cB$$

$$B \rightarrow dB \mid dE$$

$$E \rightarrow e$$

- **lineare a sinistra:** si ha  $G = (\{S, X, Y, Z\}, \{a, b, c, d, e\}, P, S)$  e quindi:

$$S \rightarrow Xe$$

$$A \rightarrow Xd \mid Yd$$

$$B \rightarrow Zc$$

$$E \rightarrow a \mid Zb$$

quindi se per esempio ho la stringa "ciao" si ha:

- **lineare a destra:**

$$S \rightarrow Ao$$

$$A \rightarrow Ba$$

$$B \rightarrow Ei$$

$$E \rightarrow c$$

- **lineare a sinistra:**

$$S \rightarrow cA$$

$$A \rightarrow iB$$

$$B \rightarrow aE$$

$$E \rightarrow o$$

**Esempio 33.** A partire da  $G = (\{S, T\}, \{0, 1\}, P, S)$  con:

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

$$T \rightarrow 0T \mid 1S$$

trovo come è fatto  $L(G)$ :

$$L(G) = \{w \in \{0, 1\}^* \mid w \text{ ha un numero di 1 pari}\}$$

**Esempio 34.** fornire una grammatica regolare a destra e sinistra per:

$$L = \{w \in \{0, 1\}^* \mid w \text{ ha almeno uno 0 o almeno un 1}\}$$

Si ha che tutte le stringhe tranne quella vuota contengono uno 0 o un 1 quindi  $G = (\{S\}, \{0, 1\}, P, S)$ :

- **lineare a destra:**

$$S \rightarrow 0 \mid 1 \mid 0S \mid 1S$$

- **lineare a sinistra:**

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

### 1.1.4 Espressioni Regolari (Regex)

le regex sono usate per la ricerca di un pattern in un testo o negli analizzatori lessicali. Una regex denota il linguaggio e non la grammatica. Si hanno le seguenti operazioni tra due linguaggi  $L$  e  $M$ :

- **unione:** dati  $L, M \in \Sigma^*$ , l'unione  $L \cup M$  è l'insieme delle stringhe che si trovano in entrambi i linguaggi o solo in uno dei due

**Esempio 35.**

$$L = \{001, 10, 111\}$$

$$M = \{\varepsilon, 001\}$$

$$L \cup M = \{\varepsilon, 01, 10, 111, \varepsilon\}$$

si ha che:

$$L \cup M = M \cup L$$

- **concatenazione:** dati  $L, M \in \Sigma^*$ , la concatenazione  $L \cdot M$  (o  $LM$ ) è l'insieme di tutte le stringhe ottenibili concatenandone una di  $L$  a una di  $M$



**Esempio 36.**

$$L = \{001, 10, 111\}$$

$$M = \{\varepsilon, 001\}$$

$$L \cdot M = \{001, 001001, 10, \dots\}$$

si ha che:

$$L \cdot M \neq M \cdot L$$

- si definiscono:

$$- L \cdot L = L^2, L \cdot L \cdot L = L^3 \text{ etc...}$$

$$- L^1 = L$$

$$- L^0 = \{\varepsilon\}$$

- **chiusura di Kleene:** dato  $L \subseteq \Sigma^*$  si ha che la chiusura di Kleene di  $L$  è:

$$L^* = \bigcup_{i \geq 0} L^i$$

ricordando che  $L^0 = \varepsilon$

**Esempio 37.** Sia  $L = \{0, 11\}$ , si ha:

$$L^0 = \varepsilon$$

$$L^1 = L = \{0, 11\}$$

$$L^2 = L \cdot L = \{00, 011, 110, 1111\}$$

$$L^3 = L \cdot L \cdot L = L^2 \cdot L = \{000, 0110, 1100, 11110, 0011, 01111, 11011, 111111\}$$

vediamo dei casi particolari:

- $L = \{0^n \mid n \geq 0\}$  implica  $|L| = \infty$  e quindi, essendo  $L^i = L$ ,  $i \geq 1$  e quindi  $|L^i| = \infty$ ,  $|L^*| = \infty$ . Si ha quindi:

$$L^* = L^0 \cup L^1 \cup \dots \cup L^i = L$$

- $L = \emptyset$  implica  $L^0 = \{\varepsilon\}$ ,  $L^2 = L \cdot L = \emptyset$  e così via per ogni concatenazione di  $L$ . Si ha quindi:

$$L^* = L^0 = \{\varepsilon\}$$

–  $L = \{\varepsilon\}$  implica  $L^0 = \{\varepsilon\} = L = L^1 = L^2 = \dots$ , si ha quindi:

$$L^* = \{\varepsilon\} = L$$

L'insieme vuoto e l'insieme contenente la stringa vuota hanno le uniche chiusure di Kleene finite

**Definizione 8.** *Si riporta la definizione ricorsiva di un'espressione regolare:*

• **casi base:** *si hanno tre casi base:*

1.  $\varepsilon$  e  $\emptyset$  sono espressioni regolari
2. se  $a \in \Sigma$   $a$  è un'espressione regolare,  $L(a) = \{a\}$
3. le variabili che rappresentano linguaggi regolari sono espressioni regolari,  $L(L) = L$

• **casi passo:** *si hanno i 4 casi passo:*

1. **unione:** se  $E$  e  $F$  sono espressioni regolari allora anche  $E + F = E \cup F$  è un'espressione regolare e si ha:

$$L(E + F) = L(E) \cup L(F)$$

2. **concatenazione:** se  $E$  e  $F$  sono espressioni regolari allora anche  $EF = E \cdot F$  è un'espressione regolare e si ha:

$$L(EF) = L(E) \cdot L(F)$$

3. **chiusura:** se  $E$  è un'espressione regolare allora  $E^*$  è un'espressione regolare e si ha:

$$L(E^*) = (L(E))^*$$

4. **parentesi:** se  $E$  è un'espressione regolare allora  $(E)$  è un'espressione regolare e si ha:

$$L((E)) = L(E)$$

**Esempio 38.** *trovo regex per l'insieme di stringhe in  $\{0,1\}^*$  che consistono in 0 e 1 alternati:*

$$01 \rightarrow \{01\}$$

$$(01)^* \rightarrow \{\varepsilon, 01, 0101, 010101, \dots\}$$

$$(01)^* + (10)^* \rightarrow \{\varepsilon, 01, 10, 0101, 1010, \dots\}$$

*ma posso volere diverse quantità di 0 e 1, sempre mantenendo l'alternanza, metto o uno 0 o un 1 davanti a quanto ottenuto appena sopra:*

$$(01)^* + (10)^* + 0(10)^* + 1(01)^* \rightarrow \{\varepsilon, 01, 10, 010, 101, \dots\}$$

*non è comunque l'unica soluzione, si può avere:*

$$(\varepsilon + 1)(01)^*(\varepsilon + 0) \rightarrow \{\varepsilon, 01, 10, 010, 101, \dots\}$$

*oppure ancora:*

$$(\varepsilon + 0)(10)^*(\varepsilon + 1)$$

Si ha una precedenza degli operatori, in ordine di precedenza (si valuta da sinistra a destra):

1. chiusura di Kleene  $*$
2. concatenazione  $\cdot$ , che è associativo  $((E \cdot F) \cdot G = E \cdot (F \cdot G))$  ma non è commutativo  $(E \cdot F \neq F \cdot E)$
3. unione  $+$  che è associativa  $((E+F)+G = E+(F+G))$  ed è commutativo  $(E + F = F + E)$
4. infine le parentesi

si hanno anche delle proprietà algebriche:

- due espressioni regolari sono equivalenti se denotano lo stesso linguaggio
- due espressioni regolari con variabili sono equivalenti se lo sono  $\forall$  assegnamento alle variabili
- l'unione è commutativa e associativa, la concatenazione è solo associativa
- si definiscono:

- **identità:** ovvero un valore unito all'identità è pari a se stesso (elemento neutro della somma  $0 + x = x + 0 = x$ ).  $\emptyset$  è identità per l'unione ( $\emptyset + L = L + \emptyset = L$ ),  $\{\varepsilon\}$  è identità per la concatenazione ( $\varepsilon L = L\varepsilon = L$ )
- **annichilitore:** ovvero un valore concatenato all'annichilitore da l'annichilitore (l'elemento nullo del prodotto  $0x = x0 = 0$ ).  $\emptyset$  è l'annichilitore per la concatenazione ( $\emptyset L = L\emptyset = \emptyset$ )
- **distributività:** dell'unione rispetto alla concatenazione (che non è commutativa):
  - **distributività sinistra:**  $L(M + N) = LM + LN$
  - **distributività destra:**  $(M + N)L = ML + NL$
- **idempotenza:**  $L + L = L$
- $(L^*)^* = L^*$
- $\emptyset^* = \varepsilon$  infatti  $L(\emptyset) = \{\varepsilon\} \cup L(\emptyset) \cup L(\emptyset) \cdot L(\emptyset) \cup \dots = \{\varepsilon\} \cup \emptyset \cup \emptyset \dots = \varepsilon$
- $\varepsilon^* = \varepsilon$  infatti  $L(\varepsilon) = \{\varepsilon\} \cup L(\varepsilon) \cup L(\varepsilon) = \{\varepsilon\} \cup \{\varepsilon\} \cup \dots = \{\varepsilon\} = L(\varepsilon)$
- $L^+ = L \cdot L^* = L^* \cdot L$  (quindi con almeno un elemento che non sia la stringa vuota)
- $L^* = l^+ + \varepsilon$

**Esempio 39.** *Ho  $ER = (0 + 1)^*0^*(01)^*$ :*

- *001 fa parte del linguaggio? Si:  $\varepsilon \cdot 0 \cdot 01$*
- *1001 fa parte del linguaggio? Si:  $1 \cdot 0 \cdot 01$*
- *0101 fa parte del linguaggio? Si:  $\varepsilon \cdot \varepsilon \cdot 0101$*
- *0 fa parte del linguaggio? Si:  $\varepsilon \cdot 0 \cdot \varepsilon$*
- *10 fa parte del linguaggio? Si:  $1 \cdot 0 \cdot \varepsilon$*

$L((0+1)^*) = (L(0+1))^* = (L(0)+L(1))^* = (\{0\} \cup \{1\})^* = (\{0, 1\})^* = \{0, 1\}^*$   
 ovvero tutte le combinazioni di 0 e 1

Si ricorda che:

$$(0 + 1)^* \neq 0^* + 1^*$$

**Esempio 40.** *ho*  $ER = ((01)^* \cdot 10 \cdot (0 + 1)^*)^*$

- *0101 fa parte del linguaggio? No*
- *01000 fa parte del linguaggio? No*
- *01011 fa parte del linguaggio? No*
- *10111 fa parte del linguaggio? Si,  $\varepsilon \cdot 10 \cdot 111$*
- *101010 fa parte del linguaggio? Si, prendo  $10 \cdot 1010$*
- *101101 fa parte del linguaggio? Si,  $\varepsilon \cdot 10 \cdot 1$  due volte*
- *0101100011 fa parte del linguaggio? Si,  $0101 \cdot 10 \cdot 0011$  (0011 lo posso prendere da  $(0 + 1)^*$ )*

**Esempio 41.** *ho*  $ER = ((01)^* \cdot 10 \cdot (0 + 1))^*$

- *0101 fa parte del linguaggio? No*
- *01000 fa parte del linguaggio? No*
- *01011 fa parte del linguaggio? No*
- *10111 fa parte del linguaggio? No*
- *101010 fa parte del linguaggio? No*
- *101101 fa parte del linguaggio? Si,  $\varepsilon \cdot 10 \cdot 1$  due volte*
- *0101100011 fa parte del linguaggio? No*

**Esempio 42.** *Da*  $L \subseteq \{0, 1\}^*$  *stringhe contenenti almeno una volta 01 quindi:*

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

**Esempio 43.** *ho*  $ER = (00^*1^*)^*$ , *quindi:*

$$L = \{\varepsilon, 0, 01, 000, 001, 010, 011\} = \{\varepsilon\} \cup \{w \in \{0, 1\}^* \mid w \text{ che inizia con } 0\}$$

**Esempio 44.** *ho*  $ER = a(a + b)^*b$ , *quindi:*

$$L = \{w \in \{a, b\}^* \mid w \text{ inizia con } a \text{ e termina con } b\}$$

**Esempio 45.** *ho*  $ER = (0^*1^*)^*000(0 + 1)^*$ , *quindi, sapendo che*  $\{0, 1\}^*$  *mi permette tutte le combinazioni che voglio come*  $(0 + 1)^*$ :

$$L = \{w \in \{0, 1\}^* \mid w \text{ come voglio con tre } 0 \text{ consecutivi}\}$$

**Esempio 46.** ho  $ER = a(a+b)^*c(a+b)^*c(a+b)^*b$ , quindi:

$L = \{w \in \{a, b, c\}^* \mid w \text{ inizia con } a, \text{ termina con } b \text{ e contiene almeno due } c, \\ \text{eventualmente non adiacenti}\}$

**Esempio 47.** Da  $L \subseteq \{0, 1\}^*$  ogni 1 è seguito da 0, a meno che non sia l'ultimo carattere, ovvero 11 non compare quindi:

$$(10 + 0)^*(\varepsilon + 1)^*$$

**Esempio 48.** cerco  $ER$  per  $L \subseteq \{0, 1\}^*$  stringhe contenenti un numero pari di 1:

$$(0^*10^*1)^*0^*$$

oppure:

$$(0 + 10^*1)^*$$

**Esempio 49.** cerco  $ER$  per  $L \subseteq \{0, 1\}^*$  stringhe contenenti un numero dispari di 1:

$$(0^*10^*)^*0^*10^*$$

oppure:

$$(0 + 10^*1)^*10^*$$

**Esempio 50.** cerco  $ER$  per  $L \subseteq \{0, 1\}^*$  stringhe contenenti un numero divisibile per 3 di 0:

$$(1^*01^*01^*0)^*1^*$$

**Esempio 51.** cerco  $ER$  per  $L \subseteq \{0, 1\}^*$  stringhe contenenti al più una coppia di 1 consecutivi:

$$(10 + 0)^*(11 + 1 + \varepsilon)(01 + 0)^*$$

**Esempio 52.** cerco  $ER$  per  $L \subseteq \{a, b, c\}^*$  stringhe contenenti almeno una  $a$  e almeno una  $b$ :

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

## 1.2 Automi

un automa a stati finiti ha un insieme di stati e un controllo che si muove da stato a stato in risposta a input esterni. Si ha una distinzione:

- **automi deterministici:** dove l'automa non può essere in più di uno stato per volta
- **automi non deterministici:** dove l'automa può trovarsi in più stati contemporaneamente

### 1.2.1 Automi deterministici

Un automa a stati finiti deterministico (*DFA*), un automa che dopo aver letto una qualunque sequenza di input si trova in un singolo stato. Il termine *deterministico* concerne il fatto che per ogni input esiste un solo stato verso il quale l'automa passa dal suo stato corrente. Un automa a stati finiti deterministico consiste nelle seguenti parti:

- un insieme finito di stati, spesso indicato con  $Q$
- un insieme finito di simboli di input, spesso indicato con  $\Sigma$
- una funzione di transizione, che prende come argomento uno stato e un simbolo di input e restituisce uno stato. La funzione di transizione sarà indicata comunemente con  $\delta$ . Nella rappresentazione grafica informale di automi  $\delta$  è rappresentata dagli archi tra gli stati e dalle etichette sugli archi. Se  $q$  è uno stato e  $a$  è un simbolo di input,  $\delta(q, a)$  è lo stato  $p$  tale che esiste un arco etichettato con  $a$  da  $q$  a  $p$ <sup>2</sup>
- uno stato iniziale, uno degli stati in  $Q$
- un insieme di stati finali, o accettanti,  $F$ . L'insieme  $F$  è un sottoinsieme di  $Q$

Nel complesso un DFA è rappresentato in maniera concisa con l'enumerazione dei suoi elementi, quindi con la quintupla:

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

con  $A$  nome del DFA,  $Q$  insieme degli stati,  $\Sigma$  rappresentante i simboli di input,  $\delta$  la sua funzione di transizione,  $q_0$  il suo stato iniziale e  $F$  l'insieme degli stati accettanti.

Vediamo come decidere se accettare o meno una stringa (sequenza di caratteri) in input mediante un DFA.

Ho una sequenza in input  $a_1 \dots a_n$ . Parto dallo stato iniziale  $q_0$ , consultando la funzione di transizione  $\delta$ , per esempio  $\delta(q_0, a_1) = q_1$  e trovo lo stato in cui il DFA entra dopo aver letto  $a_1$ . Poi passo a  $\delta(q_1, a_2) = q_2$  e così via,  $\delta(q_{i-1}, a_i) = q_i$  fino a ottenere  $q_n$ . Se  $q_n$  è elemento di  $F$  allora  $a_1 \dots a_n$  viene accettato, altrimenti viene rifiutato.

**Esempio 53.** *specifico DFA che accetta tutte le stringhe binarie in cui compare la sequenza 01:*

$$L = \{w \mid w \text{ è della forma } x01y, \text{ con } x \text{ e } y \text{ pari a } 0 \text{ o } 1\} = \{01, 11010, 100011, \dots\}$$

o anche:

$$L = \{x01y | x, y \in \{0, 1\}^*\}$$

abbiamo quindi:

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

ragioniamo sul fatto che  $A$ :

1. se ha "già visto" 01, accetterà qualsiasi input
2. pur non avendo ancora visto 01, l'input più recente è stato 0, cosicché se ora vede un 1 avrà visto 01
3. non ha ancora visto 01, ma l'input più recente è nullo (siamo all'inizio), in tal caso  $A$  non accetta finché non vede uno 0 e subito dopo un 1

la terza condizione rappresenta lo stato iniziale. All'inizio bisogna vedere uno 0 e poi un 1. Ma se nello stato  $q_0$  si vede per primo un 1 allora non abbiamo fatto alcun passo verso 01, e dunque dobbiamo permanere nello stato  $q_0$ ,  $\delta(q_0, 1) = q_0$ . D'altra parte se nello stato iniziale vedo 0 siamo nella seconda condizione, uso quindi  $q_2$  per questa condizione, si avrà quindi  $\delta(q_0, 0) = q_2$ . Vedo ora le transizioni di  $q_2$ , se vedo 0 ho che 0 è l'ultimo simbolo incontrato quindi uso nuovamente  $q_2$ ,  $\delta(q_2, 0) = q_2$ , in attesa di un 1. Se arriva 1 passo allo stato accertante  $q_1$  corrispondente alla prima condizione,  $\delta(q_2, 1) = q_1$ . Ora abbiamo incontrato 01 quindi può succedere qualsiasi cosa e dopo qualsiasi cosa accada potremo nuovamente aspettarci qualsiasi cosa, ovvero  $\delta(q_1, 0) = \delta(q_1, 1) = q_1$ . Si deduce quindi che:

$$Q = \{q_0, q_1, q_2\} \text{ e } F = \{q_1\}$$

quindi:

$$A = \{\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\}\}$$

con in totale le seguenti transizioni:

$$\delta(q_0, 1) = q_0$$

$$\delta(q_0, 0) = q_2$$

$$\delta(q_2, 0) = q_2$$

$$\delta(q_2, 1) = q_1$$

$$\delta(q_1, 0) = q_1$$

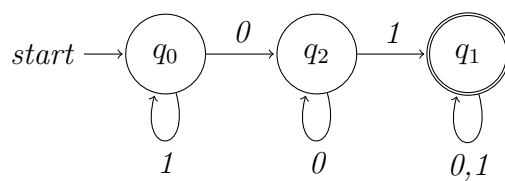
$$\delta(q_1, 1) = q_1$$

posso rappresentarle in maniera tabulare, con lo stato iniziale indicato da  $\rightarrow$  e quelli accettanti con  $*$ :



$\delta$	$0$	$1$
$\rightarrow q_0$	$q_1$	$q_0$
$* q_1$	$q_1$	$q_1$
$q_2$	$q_2$	$q_1$

o col diagramma di transizione:



**Esempio 54.** Trovo automa per:

$$L = \{w \in \{a, b\}^* \mid w \text{ che contiene un numero pari di } b\}$$



ovvero se da  $q_0$  vado a  $q_1$  sono obbligato ab generare due  $b$ , dato che il nodo accettante è  $q_0$ . In entrambi i nodi posso generare quante a voglio.

**Esempio 55.** Trovo automa per:

$$L = \{w \in \{a, b\}^* \mid w \text{ che contiene un numero dispari di } b\}$$



ovvero se da  $q_0$  vado a  $q_1$  sono obbligato ab generare una sola  $b$ , dato che il nodo accettante è  $q_1$ . In entrambi i nodi posso generare quante a voglio e posso tornare da  $q_1$  a  $q_0$  per generare altre  $b$ .

**Esempio 56.** Trovo automa per:

$$L = \{w \in \{0, 1\}^* \mid w = 0^n 1^m\}$$

vediamo i vari casi: **Si ha che  $q_E$  è lo stato pozzo dove vanno le stringhe venute male**

- $n, m \geq 0$ :



ovvero posso non generare nulla e uscire subito con  $q_0$ , generare solo un  $1$  e passare a  $q_1$  e uscire oppure generare  $0$  e  $1$  a piacere con l'ultimo stato o generare  $0$  a piacere dal primo e  $1$  a piacere dal secondo.

- $n \geq 0 \ m > 0$ :



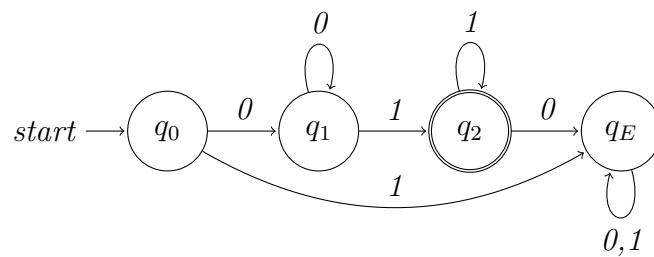
ovvero come l'esempio sopra solo che non posso uscire in  $q_0$  in quanto almeno un 1 deve essere per forza generato

- $n > 0 \ m \geq 0$ :



CHIARIRE

- $n, m > 0$ :



CHIARIRE

**Esempio 57.** Trovo automa per:

$$L = \{w \in \{a, b\}^* \mid w \text{ che contiene un numero pari di } a \text{ e dispari di } b\}$$



**Esempio 58.** Trovo automa per:

$$L = \{w \in \{a, b\}^* \mid w \text{ che contiene un numero pari di } a \text{ seguito da uno dispari di } b\}$$

$$L = \{a^{2n}b^{2k+1} \mid j, k \geq 0\}$$



ovvero in tabella:

$\delta$	$a$	$b$
$\rightarrow q_0$	$q_1$	$q_2$
$q_1$	$q_0$	$q_E$
$* q_2$	$q_E$	$q_3$
$q_3$	$q_E$	$q_2$
$q_E$	$q_E$	$q_E$

**Esempio 59.** Trovo automa per:

$$L = \{a^{2k+1}b^{2h} \mid h, k \geq 0\}$$



**Esempio 60.** Trovo automa per:

$$L = \{a^{2n+1}b^{2k+1} \mid n, k \geq 0\}$$



**Esempio 61.** Trovo automa per:

$$L = \{x010y \mid x, y \in \{0, 1\}^*\}$$



### 1.2.2 Automi non deterministici

Un automa a stati finiti non deterministici (*NFA*) può trovarsi in diversi stati contemporaneamente. Come i DFA accettano linguaggi regolari e spesso sono più semplici da trattare rispetto ai DFA.

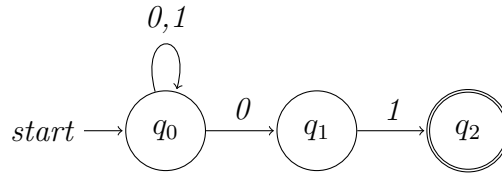
Un NFA è definito come un DFA ma si ha un diverso tipo di transizione  $\delta$ , che ha sempre come argomenti uno stato e un simbolo di input ma restituisce zero o più stati.

**Esempio 62.** Sia  $L = \{x01 \mid x \in \{0, 1\}^*\}$  ovvero il linguaggio formato da tutte le stringhe binarie che terminano in 01.

Avremo il seguente automa deterministico:



che diventa il seguente NFA:



quindi con:

$$\delta(q_0, 0) = \{q_0, q_1\}$$

$$\delta(q_0, 1) = \{q_0\}$$

$$\delta(q_1, 0) = \emptyset$$

$$\delta(q_1, 1) = \{q_2\}$$

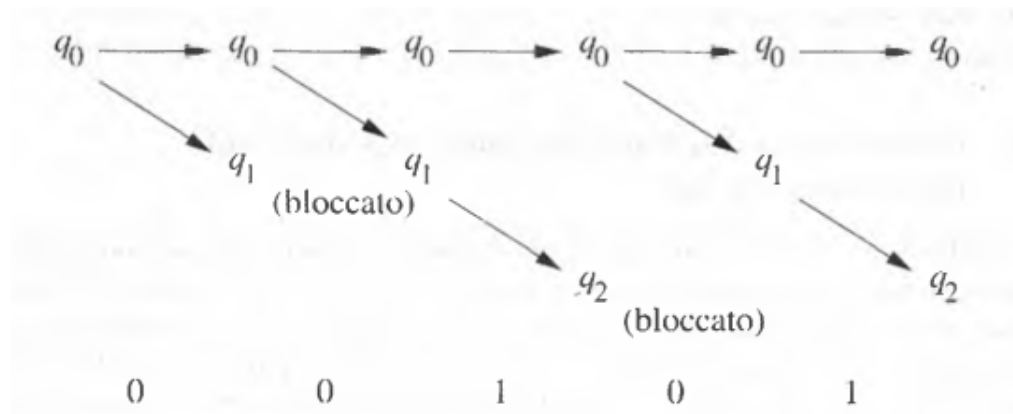
$$\delta(q_2, 0) = \emptyset$$

$$\delta(q_2, 1) = \emptyset$$

in forma tabulare:

$\delta$	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$q_1$	$\emptyset$	$\{q_2\}$
$*q_0$	$\emptyset$	$\emptyset$

vediamone la simulazione per la stringa 00101:



ovvero si parte dallo stato iniziale, quando viene letto 0 si passa a  $q_0$  e  $q_1$ , poi viene letto il secondo 0 quindi  $q_0$  va nuovamente verso  $q_0$  e  $q_1$  mentre il primo  $q_1$  muore non avendo transizioni su 0. Arriva poi l'1 quindi  $q_0$  va solon verso  $q_0$  e  $q_1$  verso  $q_2$  e sarebbe accettante ma l'input non è finito. Ora arriva 0 e  $q_2$  si blocca mentre  $q_0$  va sia in  $q_0$  che in  $q_1$ . Arriva infine un 1 che manda  $q_0$  in  $q_0$  e  $q_1$  in  $q_2$  che è accettante e non avendo altri input si è dimostrata l'appartenenza della stringa al linguaggio

definisco quindi un NFA come una quintupla:

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

con, a differenza dei DFA:

$$\delta : Q \times F \rightarrow 2^Q$$

Possiamo ora definire  $\delta$ , delta cappuccio che prende in ingresso uno stato e l'intera stringa  $w$ . Definisco ricorsivamente:

- **caso base:** se  $|w| = 0$  ovvero se  $W = \varepsilon$  si ha:

$$\hat{\delta}(q, \varepsilon) = \{q\}$$

- **caso passo:** se  $|w| > 0$ , allora  $W = xa$ ,  $a \in \Sigma$  e  $x \in \Sigma^*$ . Posto  $\hat{\delta}(q, x) = \{p_1, \dots, p_k\}$  si ha:

$$\hat{\delta}(q, w) = \cup \delta(p_i, a)$$

Per il linguaggio  $L$  accettato dall'automa si ha:

$$L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

**Esempio 63.** Automa per  $L = \{x010y \mid x, y \in \{0, 1\}^*\}$  ovvero tutte le stringhe con dentro la sequenza 010:



Troviamo ora un algoritmo che trasformi un NFA in un DFA. Dall'ultimo esempio ricavo:

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

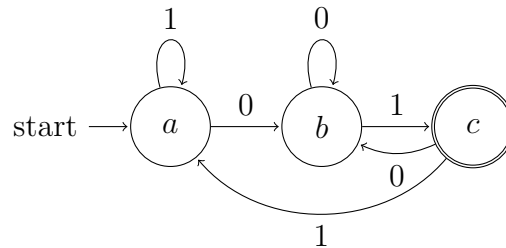
ovvero:



che è il DFA che si era anche prima ottenuto. Si hanno però dei sottoinsiemi mai raggiungibili. Si ha quindi:

	0	1
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

e definendo  $\{q_0\} = a$ ,  $\{q_0, q_1\} = b$  e  $\{q_0, q_2\} = c$  si avrà:



Definiamo questo algoritmo che avrà:

- come input un NFA  $N = (Q_n, \Sigma, \delta_N, q_0, F_N)$
- come output un DFA  $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  tale che  $L(D) = L(N)$

con:

- $Q_D = 2^{Q_N}$  (quindi se  $Q_N = n$  si ha  $|Q_D| = 2^n$ )
- $F_D = \{S \subseteq Q_n \mid S \cap F_N \neq \emptyset\}$



- $\forall S \subseteq Q_N$  e  $\forall a \in \Sigma$ :

$$\delta_D(S, a) = \cup \delta_n(p, a)$$

per esempio:

$$\delta_D(\{q_0, q_1, q_2\}, 0) = \delta_N(q_0, 0) \cup \delta_N(q_1, 0) \cup \delta_N(q_2, 0)$$

Si definisce l' $\varepsilon$ -NFA, l'automa a stati finiti non deterministici con  $\varepsilon$  transizioni. Con la transizione  $\varepsilon$  posso saltare i nodi, ovvero avanza senza aggiungere caratteri

**Esempio 64.** Si ha  $ER = a^*b^*c^*$ , che genera:

$$L = \{a^n b^m c^k \mid n, m, k \geq 0\}$$

si ha:



ovvero con  $\varepsilon$  posso per esempio generare quante a voglio da  $q_0$  e passare a  $q_2$ , uscendo senza generare altro

Si definisce la funzione  $ECLOSE : Q \rightarrow 2^Q$ , con  $ECLOSE(q)$  insieme degli stati raggiungibili da  $q$  tramite  $\varepsilon$ -mosse. Nell'esempio precedente si avrebbe:

$$ECLOSE(q_0) = \{q_0, q_1, q_2\}$$

$$ECLOSE(q_1) = \{q_1, q_2\}$$

$$ECLOSE(q_2) = \{q_2\}$$

si ha inoltre che:

- $ECLOSE 2^Q \rightarrow 2^Q \quad P \subseteq Q$
- $ECLOSE(P) = \cup ECLOSE(p)$
- $ECLOSE(\emptyset) = \emptyset$

mettendo in tabella l'esempio precedente si ha:

	a	b	c
$* \rightarrow \{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_1, q_2\}$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$	$\{q_2\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

riscrivendo:

- $a = \{q_0, q_1, q_2\}$
- $b = \{q_1, q_2\}$
- $c = \{q_2\}$
- $d = \emptyset$

ovvero:

$$\begin{aligned}
 \delta_D(\{q_0, q_1, q_2\}, a) &= ECLOSE(\delta_N(q_0, a) \cup \delta_N(q_1, a) \cup \delta_N(q_2, a)) \\
 &= ECLOSE(\{q_0\} \cup \emptyset \cup \emptyset) = ECLOSE(\{q_0\}) \\
 &= ECLOSE(q_0) = \{q_0, q_1, q_2\}
 \end{aligned}$$

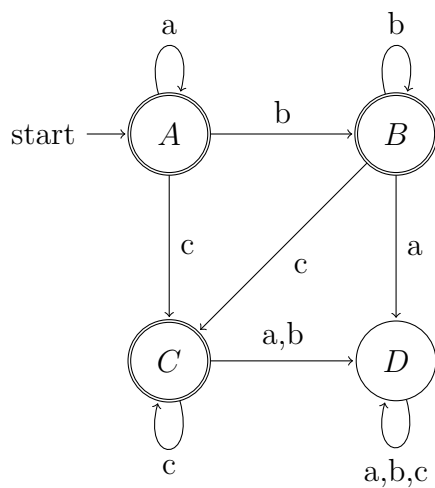
e

$$\begin{aligned}
 \delta_D(\{q_0, q_1, q_2\}, b) &= ECLOSE(\delta_N(q_0, b) \cup \delta_N(q_1, b) \cup \delta_N(q_2, b)) \\
 &= ECLOSE(\emptyset \cup \{q_1\} \cup \emptyset) = ECLOSE(\{q_1\}) \\
 &= ECLOSE(q_1) = \{q_1, q_2\}
 \end{aligned}$$

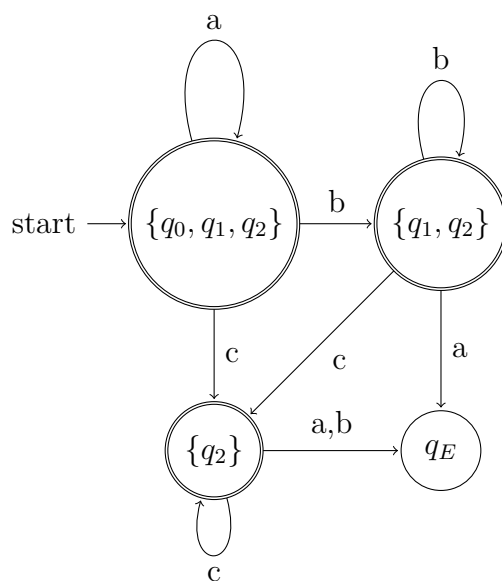
e

$$\begin{aligned}
 \delta_D(\{q_0, q_1, q_2\}, c) &= ECLOSE(\delta_N(q_0, c) \cup \delta_N(q_1, c) \cup \delta_N(q_2, c)) \\
 &= ECLOSE(\emptyset \cup \emptyset \cup \{q_2\}) = ECLOSE(\{q_2\}) \\
 &= ECLOSE(q_2) = \{q_2\}
 \end{aligned}$$

si ottiene quindi il seguente NFA:



che diventa il seguente DFA:



**esercizi**

**Esempio 65.** automa DFA per  $w = x010y$ ,  $x, y \in \{0, 1\}^*$  :  
la stringa più corta è 010



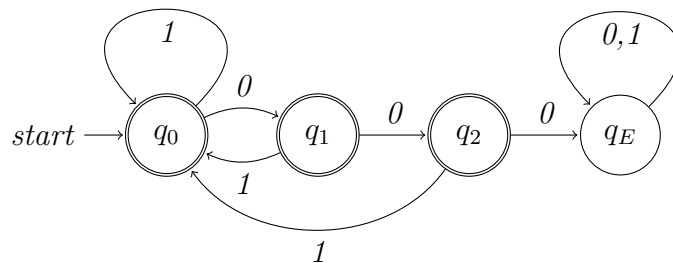
**Esempio 66.** automa DFA per  $a^{2k+1}b^{2h}$ ,  $h, k \geq 0$  :  
concatenazione di  $a$  dispari e  $b$  pari:



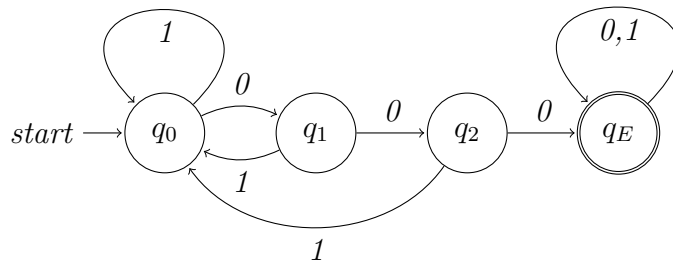
**Esempio 67.** cerco DFA per stringhe inizianti con  $a$  e finenti con  $b$ , con  
occorrenze di  $b$  singole o a coppie, nessuna regola per  $c$ .  
per esempio  $abbcb$  è nel linguaggio



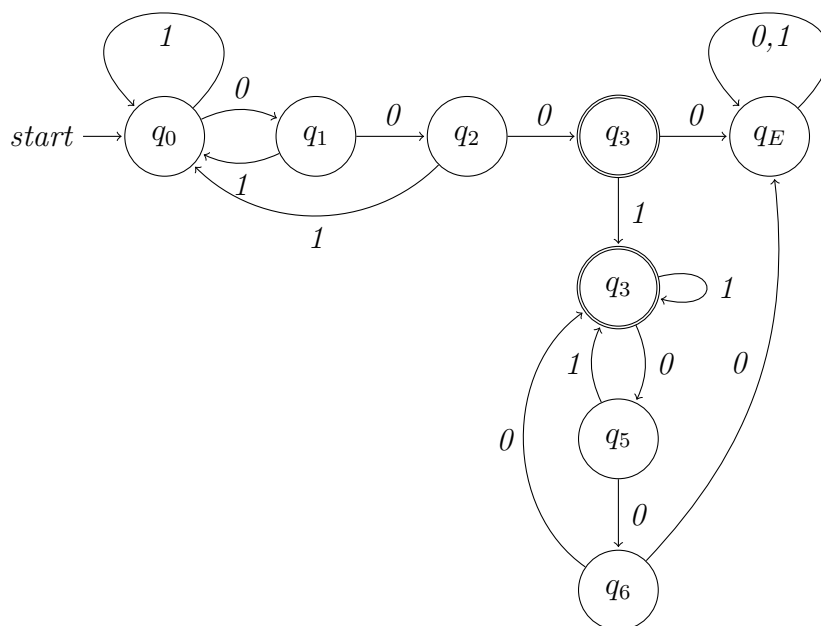
**Esempio 68.** *cerco DFA per stringhe di bit non contegano 000*



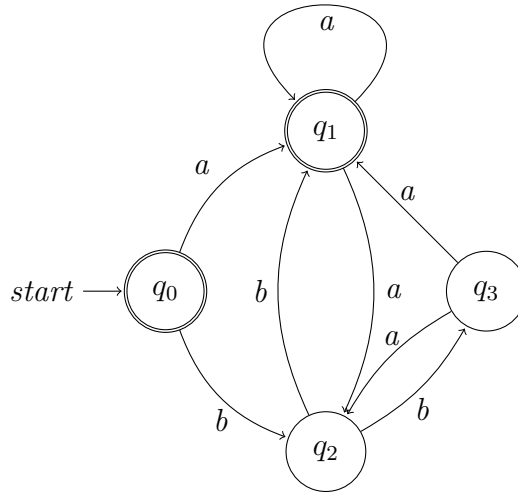
**Esempio 69.** *cerco DFA per stringhe di bit non contegano 000 almeno una volta*



**Esempio 70.** *cerco DFA per stringhe di bit che contengono 000 solo una volta*



**Esempio 71.** Trasformare il seguente NFA in un DFA:



abbiamo quindi:

$$\delta_D(\{q_0\}, a) = \delta_N(q_0, a) = \{q_1, q_2\}$$

$$\delta_D(\{q_0\}, b) = \delta_N(q_0, b) = \emptyset$$

$$\delta_D(\{q_1, q_2\}, a) = \delta_N(q_1, a) \cup \delta_N(q_2, a) = \{q_1, q_2\} \cup \emptyset = \{q_1, q_2\}$$

$$\delta_D(\{q_1, q_2\}, b) = \delta_N(q_1, b) \cup \delta_N(q_2, b) = \emptyset \cup \{q_1, q_3\} = \{q_1, q_3\}$$

...

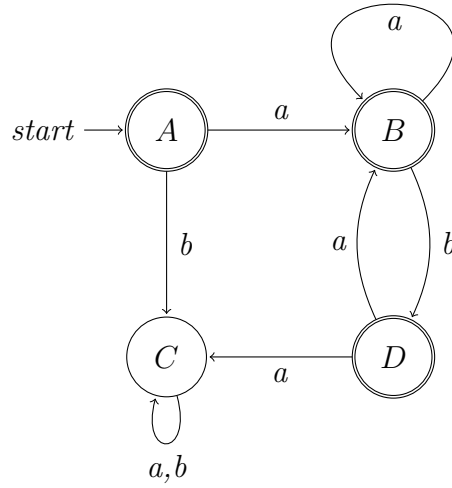
ottengo quindi:

DFA	a	b
$* \rightarrow \{q_0\}$	$\{q_1, q_2\}$	$\emptyset$
$*\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1, q_3\}$
$\emptyset$	$\emptyset$	$\emptyset$
$*\{q_1, q_3\}$	$\{q_1, q_2\}$	$\emptyset$

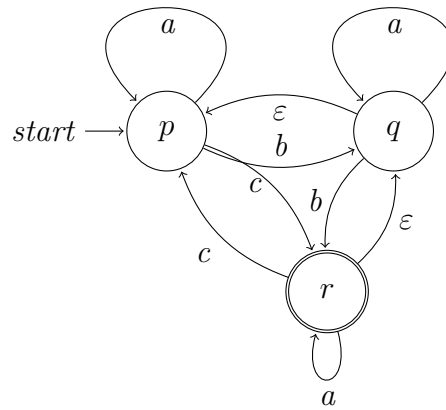
Posso ora rinominare:

- $A = \{q_0\}$
- $B = \{q_1, q_2\}$
- $C = \emptyset$
- $D = \{q_1, q_3\}$

ottengo quindi il seguente DFA:



**Esempio 72.** Trasformare il seguente  $\varepsilon$ -NFA in un DFA:



vediamo le  $ECLOSE$ :

$$ECLOSE(p) = \{p\}$$

$$ECLOSE(q) = \{p, q\}$$

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

si ottiene quindi:

	$a$	$b$	$c$
$to \{p\}$	$\{p\}$	$\{p, q\}$	$\{p, q, r\}$
$\{p, q\}$	$\{p, q\}$	$\{p, q, r\}$	$\{p, q, r\}$
$*\{p, q, r\}$	$\{p, q, r\}$	$\{p, q, r\}$	$\{p, q, r\}$

infatti, per esempio:

$$\delta_D(\{p\}, a) = ECLOSE(\delta(p, a)) = ECLOSE(\{p\}) = ECLOSE(p) = \{p\}$$

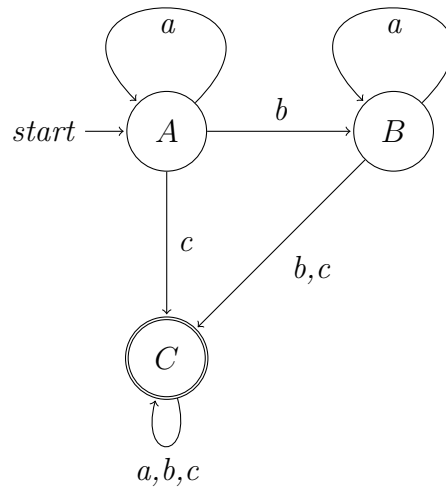
$$\begin{aligned} \delta_D(\{p, q\}, a) &= ECLOSE(\delta_N(p, a) \cup \delta_N(q, a)) = \\ ECLOSE(\{p\} \cup \{q\}) &= ECLOSE(p) \cup ECLOSE(q) = \{p\} \cup \{p, q\} = \{p, q\} \end{aligned}$$

...

si hanno quindi le seguenti rinominazioni:

- $A = \{p\}$
- $b = \{p, q\}$
- $C = \{p, q, r\}$

ovvero:





torniamo a dare bene qualche definizione per  $\delta$  in un DFA:

$$\delta: Q \times \Sigma^* \rightarrow Q$$

con:  $q \in Q, w \in \Sigma^*$  e  $\delta(p, w) = q$

- **caso base:**  $w = \varepsilon \rightarrow |w| = 0 \rightarrow \delta(q, \varepsilon) = q$   
**caso passo:**  $|w| \neq 0 \rightarrow w = ax$  con  $a \in \Sigma, x \in \Sigma^*$ :  $\delta(q, w) = \delta(q, ax) = \delta(\delta(q, a), x)$
- **caso base:**  $w = \varepsilon \rightarrow |w| = 0 \rightarrow \delta(q, w) = \delta(q, \varepsilon) = q$   
**caso passo:**  $|w| \neq 0 \rightarrow w = xa$  con  $a \in \Sigma, x \in \Sigma^*$ :  $\delta(q, w) = \delta(q, xa) = \delta(\delta(q, x), a)$

in un NFA si ha:

- **caso base:**  $\delta(q, \varepsilon) = \{q\}, \forall q \in Q$
- **caso passo:** posto  $w = ax$  e  $\delta(q, a) = \{p_1, \dots, p_n\}$  allora  $\delta(q, w) = \cup \delta(p_i, x) = \{r_1, \dots, r_n\}$ .  
 $\delta(q, a) = p$   
 $\delta(q, w) = \delta(q, xa) = \delta(p, x) = r$

oppure:

- **caso base:**  $\delta(q, \varepsilon) = \{q\}, \forall q \in Q$
- **caso passo:** posto  $w = xa$  si ha  $\delta(q, q) = \delta(q, xa) = \cup \delta(p, a)$  con  $\delta(q, x) = \{p_1, \dots, p_n\}$

Se ho un NFA  $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$  con  $\delta_N: Q \times \Sigma^* \rightarrow 2^{Q_N}$  che accetta un linguaggio  $L$  posso ottenere un DFA  $D = (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$  equivalente con  $Q_D = 2^{Q_N}$  e  $q_{0_D} = \{q_{0_N}\}$  che accetta  $L$ .

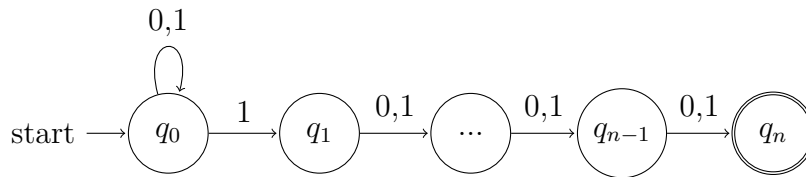
$\forall S \subseteq Q_N$  e  $\forall a \in \Sigma$  si ha:

$$F_D = \{S \subseteq Q_N \mid S \cap F_N \neq \emptyset\}$$

con  $\delta_D(S, a) = \cup \delta_N(p, a)$  Si ha che:

$$|Q_D| = |2^{Q_N}| = 2^{|Q_N|}$$

partiamo con un esempio:



che definisce un  $L \subseteq \{0, 1\}^*$  tale che  $w \in L$  sse  $n$ -simo elemento della fine è 1. Si ha:

$$|Q_N| = n + 1 \rightarrow |Q_D| = 2^n$$

Si ha che dato un  $\varepsilon$ -NFA  $E = (Q_E, \Sigma, \delta_E, q_{0_E}, F_E)$  che riconosce  $L$  in un NFA  $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$  equivalenti con  $Q_N = 2^{Q_E}$  stati in  $Q_E$   $\varepsilon$ -close:  $ECLOSE(S) = S$  e  $q_N = ECLOSE(q_E)$ :

$$F_N\{S \in Q_F, S \subseteq Q_E | S \cap F_E \neq \emptyset\}$$

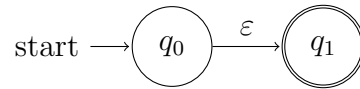
quindi:

$$\begin{aligned} \forall a \in \Sigma \text{ e } \forall S = \{p_1, \dots, p_k\}, \forall S \in Q_F \text{ e } Q_E \\ \delta_F(S, a) \text{ si ottiene } \cup \delta_E(p_i, a) = \{r_1, \dots, r_n\} \\ \delta_N(S, a) = ECLOSE(\{r_1, \dots, r_n\}) \end{aligned}$$

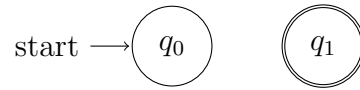
### 1.2.3 Da espressioni regolari a automi E-NFA

- **caso base:**

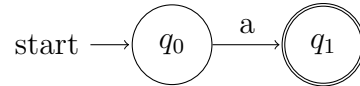
1. se  $R = \varepsilon$  ovvero  $L(R) = \{\varepsilon\}$  allora:



2. se  $R = \emptyset$  ovvero  $L(R) = \emptyset$  allora:

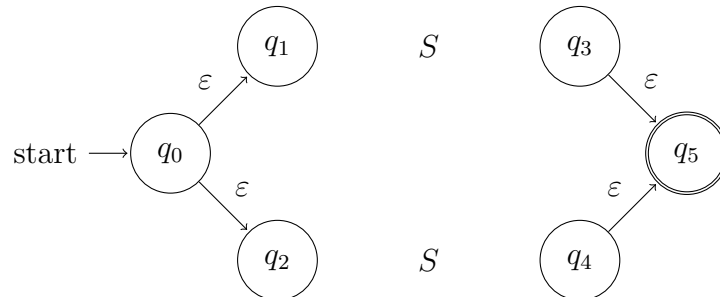


3. se  $R = a$  ovvero  $L(R) = \{a\}$  allora:

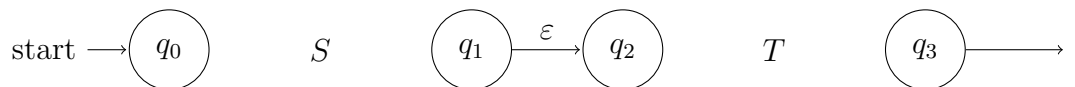


- **caso passo:**

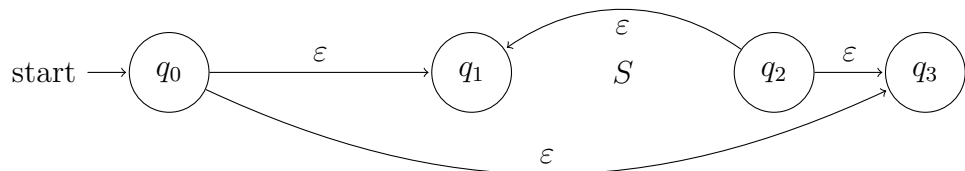
1. se  $R = S + T$  quindi  $L(R) = L(S) + L(T)$  allora:



2. se  $R = ST$  quindi  $L(R) = L(S)L(T)$ :



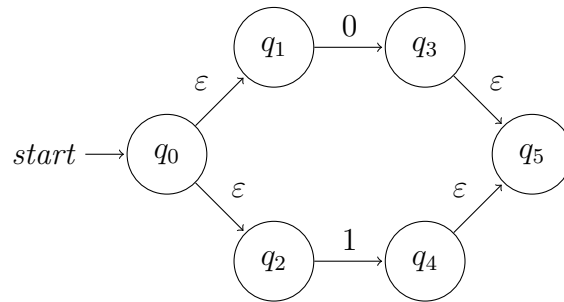
3. se  $R = S^*$  quindi  $L(R) = (L(S))^*$ :



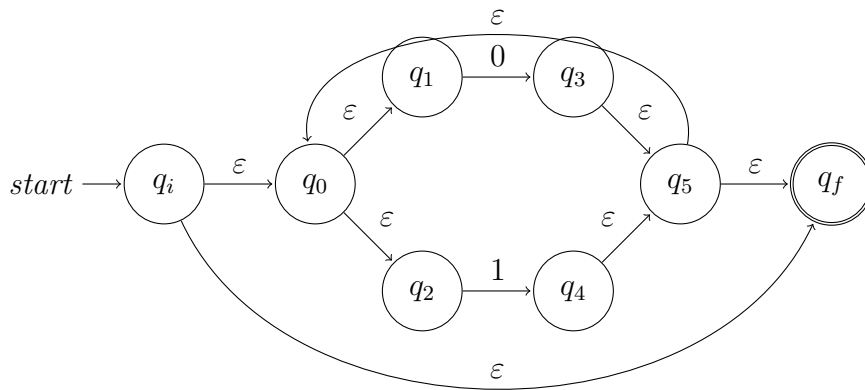
**Esempio 73.** *creo E-NFA per*

$$ER = (0 + 1)^*1(0 + 1)$$

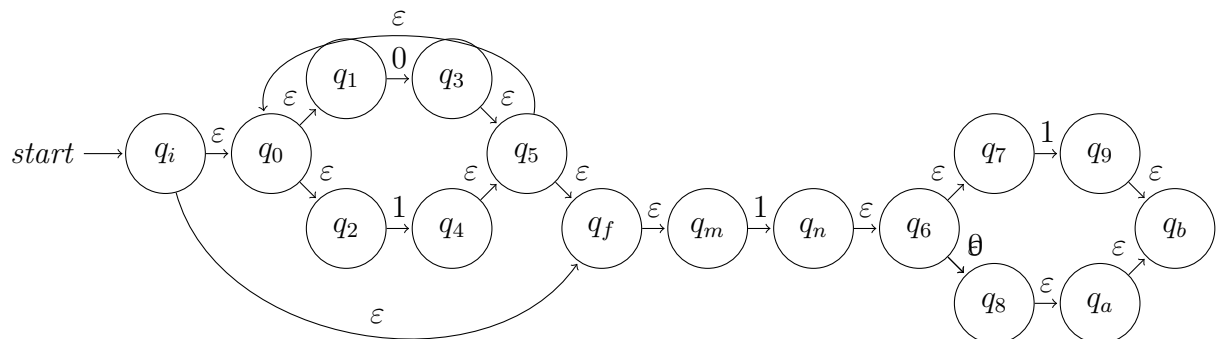
*si ha che per  $0+1$ :*



*da qui ottengo  $(0 + 1)^*$*



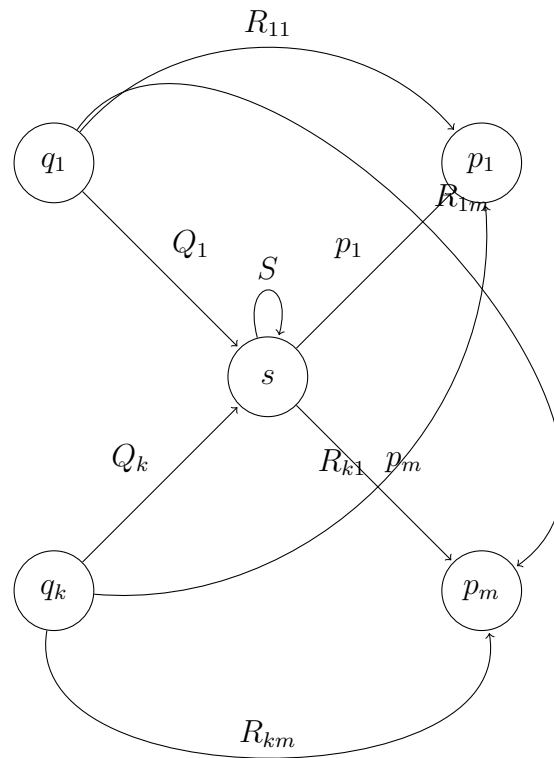
*unisco e aggiungo uno nel mezzo:*



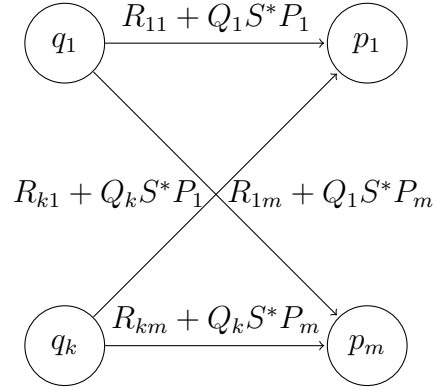
Vediamo ora l'algoritmo che trasforma DFA in una ER. Questo algoritmo permette di verificare quale linguaggio è accettato o meno dall'automa: dato DFA  $A = (Q, \Sigma, \delta, q_0, F)$  con  $Q$  e  $F$  stati non finali e  $Q, F, q_0$  che sono i primi stati da eliminare. L'algoritmo procede per eliminazioni successive. Si costruisce quindi l'automa  $B$  che ha solo  $q_0$  e  $F = \{q_1, \dots, q_k\}$ . Scrivo quindi  $K$  espressioni regolari considerando solo  $q_0$  e il  $k$ -esimo stato finale eliminando pian piano gli altri stati. Alla fine ottengo le varie espressioni regolari da unire:

$$E = E1 + E2 + \dots + Ek$$

vediamo ora come eliminare uno stato. Partiamo dal seguente automa:

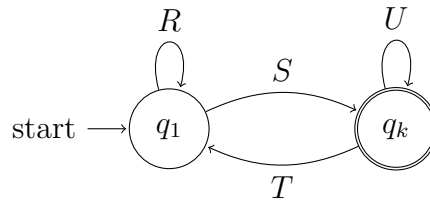


ed elimino  $s$ :



quindi quando si ha uno stato iniziale  $q_0$  e uno finale  $q_1$ :

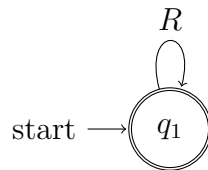
- se  $q_0 \neq q_1$ :



rappresenta:

$$E_i = (R + SU^*T)^*SU^*$$

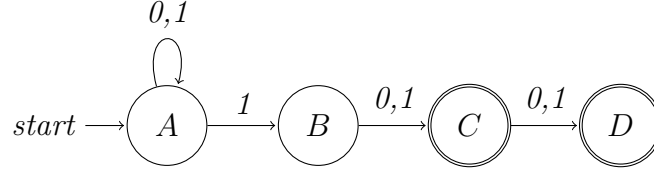
- se  $q_0 = q_1$ :



rappresenta:

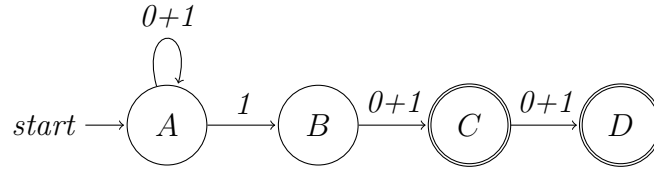
$$E_i = R^*$$

**Esempio 74.** *passo dall'automa a l'espressione regolare:*



questo automa accetta le stringhe binarie con un uno in penultima o terzultima posizione.

Rietichetto con le ER:



elimino B che non è iniziale o finale. Il predecessore di B è A e il successore è C. Si ha quindi:

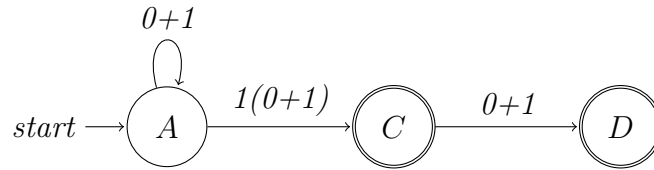
$$A \rightarrow B : 1$$

$$B \rightarrow C : 0 + 1$$

e non ho loop in B ( $\emptyset$ ) e nessun arco tra A e C ( $R_{a,c} = \emptyset$ ). Quindi:

$$R'_{A,C} = \emptyset + 1\emptyset^*(0 + 1) = 1(0 + 1)$$

ovvero:



Ho ora solo stati iniziali e finali, quindi  $E = E_1 + E_2$ . Trovo  $E_1$ , elimino quindi C. Si ha:

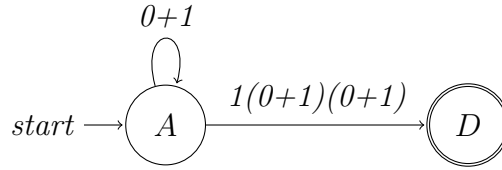
$$A \rightarrow C : 1(0 + 1)$$

$$C \rightarrow D : 0 + 1$$

e non ho loop in B ( $\emptyset$ ) e nessun arco tra A e D ( $R_{A,D} = \emptyset$ ), ovvero:

$$R_{A,D} = \emptyset + 1(0 + 1)\emptyset^*(0 + 1) = 1(0 + 1)(0 + 1)$$

che è:



quindi:

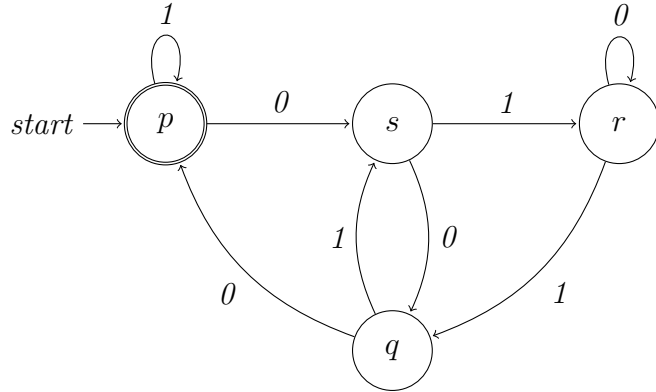
$$E_1 = ((0+1) + 1(0+1)(0+1)\emptyset^*\emptyset)^*1(0+1)(0+1)\emptyset^* = (0+1)^*1(0+1)(0+1)$$

ottengo  $E_2$  eliminando  $D$  che non ha successori, quindi:

$$E_2 = (0+1)^*1(0+1)$$

e quindi infine:

$$E = E_1 + E_2 = (0+1)^*1(0+1)(0+1) + (0+1)^*1(0+1) = (0+1)^*1(0+1)(\varepsilon + 0+1)$$



### Esempio 75.

rimuovo  $r$ :

$$s \rightarrow r : 1$$

$$r \rightarrow q : 1$$

$$\text{loop} : 0$$

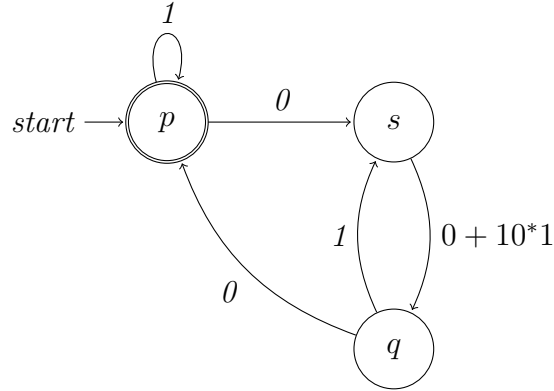
$$R_{s,q} = 0$$

Si ottiene quindi:

$$R'_{s,q} = 0 + 10^*1$$

ovvero:





elimino ora  $s$ , che ha due predecessori,  $p$  e  $q$ , quindi:

$$p \rightarrow s : 0$$

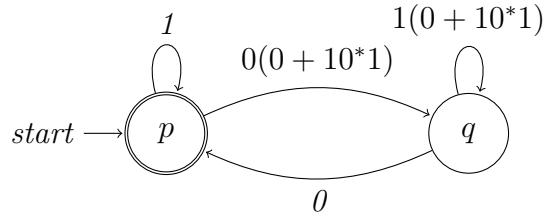
$$q \rightarrow s : 1$$

$$s \rightarrow q : 0 + 10^*1$$

non ha loop e non ha archi diretti tra  $p$  e  $q$  ( $R_{p,q} = \emptyset$ ). Si ottengono quindi:

$$R_{p,q} = \emptyset + 0\emptyset^*(0 + 10^*1) = 0(0 + 10^*1)$$

$$R_{q,q} = \emptyset + 10^*(0 + 10^*1) = 1(0 + 10^*1)$$



elimino ora  $q$  che ha  $p$  sia come predecessore che come successore:

$$p \rightarrow q : 0(0 + 10^*1)$$

$$q \rightarrow p : 0$$

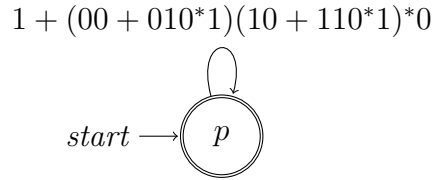
$$\text{loop} : 1(0 + 10^*1)$$

$$R_{p,p} : 1$$

ottenendo:

$$R'_{p,p} = 1 + 0(0 + 10^*1)(1(0 + 10^*1))^*0 = 1 + (00 + 010^*1)(10 + 110^*1)^*0$$

ovvero:



Essendo  $p$  sia iniziale che finale si ha:

$$E = (1 + (00 + 010^*1)(10 + 110^*1)^*0)^*$$

### 1.2.4 Chiusura di un Linguaggio regolare

Sia  $REG$  la classe dei linguaggi regolari (ovvero ogni linguaggio regolare su un alfabeto finito non vuoto). Si ha la seguente proprietà:

$$L, M \in REG \rightarrow L \cup M \in REG$$

si può dimostrare in due maniere:

- con le espressioni regolari:

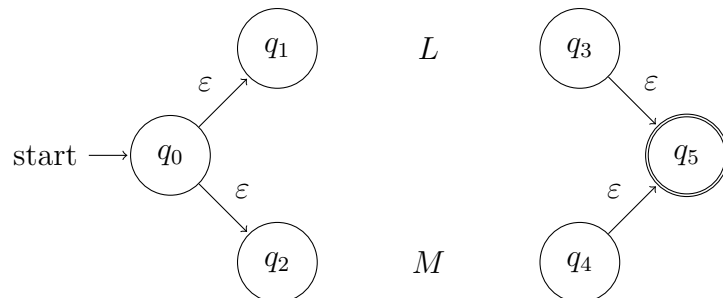
$$L, M \in REG \rightarrow \exists R, S \text{ ER tali che } L(R) = L \text{ e } L(S) = M$$

$$L \cup M = L(R + S) \text{ e quindi appartenente a } REG$$

- con gli automi:

se  $L, M \in REG \rightarrow \exists \epsilon\text{-NFA}$  che con  $L$  e  $M$  va dallo stato iniziale a quello finale

:



accetta  $L \cup M$  che quindi è  $REG$

Inoltre siano due i due alfabeti  $\Sigma \subseteq \Gamma$ , si ha che:

se  $L$  è regolare su  $\Sigma \rightarrow L$  è regolare su  $\Gamma$

inoltre:

se  $L$  è REG su  $\Sigma_1$ ,  $M$  è REG su  $\Sigma_2$ , allora  $L \cup M$  è REG su  $\Sigma_1 \cup \Sigma_2$

**Teorema 3.** *Si ha che:*

*Se  $L$  e  $M$  sono linguaggi regolari allora  $LM$ , ovvero la concatenazione è regolare.*

*Se  $L$  è un linguaggio regolare allora  $L^*$  è regolare*

**Teorema 4.** *se  $L \in REG$  su  $\Sigma$  allora  $\bar{L} = \Sigma^* - L \in REG$*

*Dimostrazione.* se  $L \in REG$  allora  $\exists$  DFA  $A = (Q, \Sigma, \delta, q_0, F)$  tale che  $L(A) = L$ . Costruisco  $B = (Q, \Sigma, \delta, q_0, Q - F)$ ,  $L(B) = \bar{L}$ .

□

si osserva che:

$$L \in REG \text{ su } \Sigma \text{ e } \Sigma \subseteq \Gamma$$

inoltre:

$$\bar{L} = \Sigma^* - L \text{ oppure } \bar{L} = \Gamma^* - L$$

*Se ho un'espressione regolare per  $L$  e voglio ottenere un'espressione regolare per  $\bar{L}$  devo ottenere un  $\varepsilon$ -NFA che devo convertire in DFA. A quel punto complemento con  $F$  e ottengo un'espressione regolare per  $\bar{L}$ .*

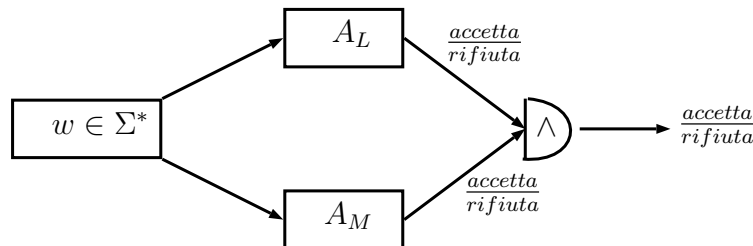
Passiamo ora all'intersezione:

**Teorema 5.** *se  $L$  e  $M$  sono regolari allora la loro intersezione è regolare*

*Dimostrazione.*

$$L, M \in REG \rightarrow \exists A_L, A_M (DFA) \text{ tali che } L(A_L) = L \text{ e } L(A_M) = M$$

e quindi:



□

Passiamo al problema della **chiusura dei linguaggi regolari rispetto all'intersezione**, ovvero che due automi che usano entrambi contemporaneamente una certa stringa in input, il ch  non   possibile. Si risolve usando l'**automa prodotto**:

$$A = (Q_L \times Q_M, \Sigma, \delta, (q_{OL}, q_{OM}), F_L \times F_M)$$

con:

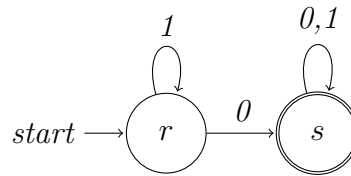
$$\delta(p, q), a) = (\delta_L(p, a), \delta_M(q, a))$$

$$A_L = (Q_L, \Sigma, \delta_L, q_{OL}, F_L)$$

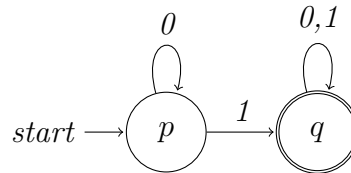
$$A_M = (Q_M, \Sigma, \delta_M, q_{OM}, F_M)$$

**Esempio 76.** Siano:

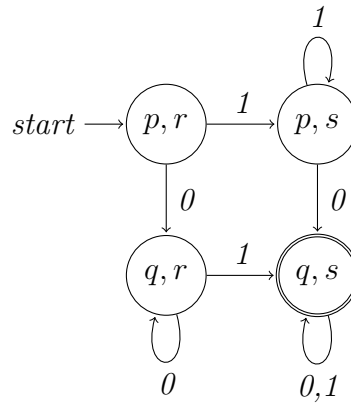
$A_L$ , stringhe binarie con almeno uno zero:



$A_M$ , stringhe binarie con almeno un uno:



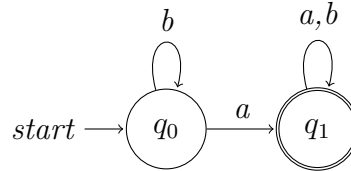
ottengo l'automa prodotto:



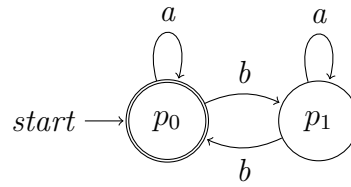
$$\delta((p, r), 0) = (\delta_L(p, o), \delta_M(r, o)) = (q, r)$$

**Esempio 77.** Siano:

$A$ , stringhe contenenti  $a$  e  $b$  contenenti almeno una  $b$ :



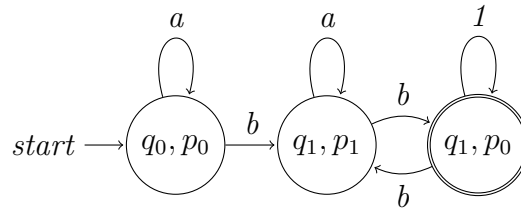
$A_M$ , stringhe contenenti  $a$  e  $b$  contenenti  $2n$ ,  $n \geq 0$   $b$ :



si ha che  $L(A \wedge B)$  è il linguaggio formato da stringhe contenenti  $a$  e  $b$  contenenti  $2n$ ,  $n \geq 1$   $b$ : faccio la tabella:

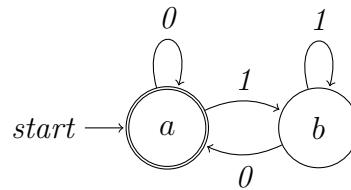
	$a$	$b$
$\rightarrow (q_0, p_0)$	$(q_0, p_0)$	$(q_1, p_1)$
$(q_1, p_1)$	$(q_1, p_1)$	$(q_1, p_0)$
$*(q_1, p_0)$	$(q_1, p_0)$	$(q_1, p_1)$

ottengo l'automa prodotto:

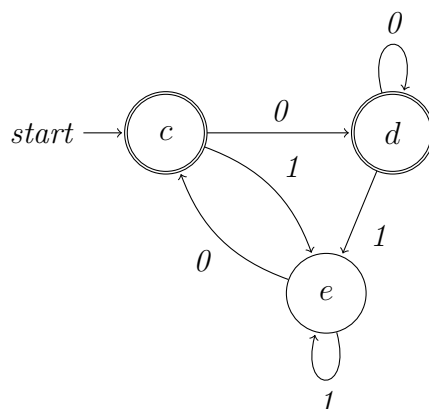


**Esempio 78.** si hanno due linguaggi  $L(A)$  e  $L(B)$  tali per cui la loro intersezione non è vuota.

Sia  $A$ :



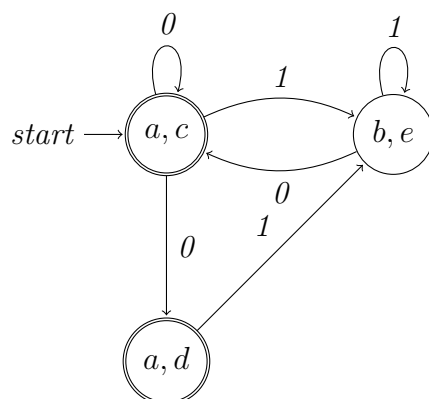
e  $B$ :



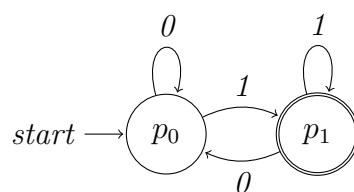
faccio la tabella:

	0	1
$\xrightarrow{*} (a, c)$	$(a, d)$	$(b, e)$
$* (a, d)$	$(a, d)$	$(b, e)$
$(b, e)$	$(a, c)$	$(b, e)$

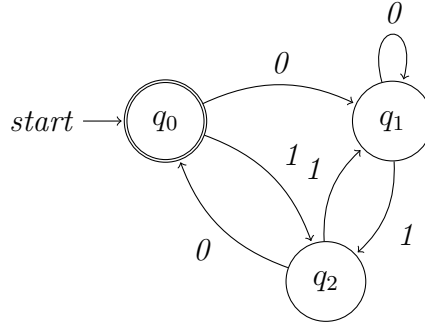
e ottengo:



**Esempio 79.** Sia  $A$ :



e  $B$ :



faccio la tabella:

	0	1
$\rightarrow (p_0, q_0)$	$(p_0, q_1)$	$(p_1, q_2)$
$(p_0, q_1)$	$(p_0, q_1)$	$(p_1, q_2)$
$(p_1, q_2)$	$(p_0, q_0)$	$(p_1, q_1)$
$(p_1, q_1)$	$(p_0, q_1)$	$(p_1, q_2)$

non si raggiungono stati finali quindi  $L(A \cap B) = \emptyset$

### 1.2.5 Minimizzazione

definiamo la **relazione di equivalenza tra stati**:

**Definizione 9.** Sia  $A$  un DFA,  $A = (Q, \Sigma, \delta, q_0, F)$ . Siano:

$$p, q \in Q \rightarrow p \approx q \text{ vale se } \forall w \in \Sigma^* \quad \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

inoltre  $p$  e  $q$  sono distinguibili se  $p \not\approx q$  ovvero:

$\exists w \in \Sigma^*$  tale che:

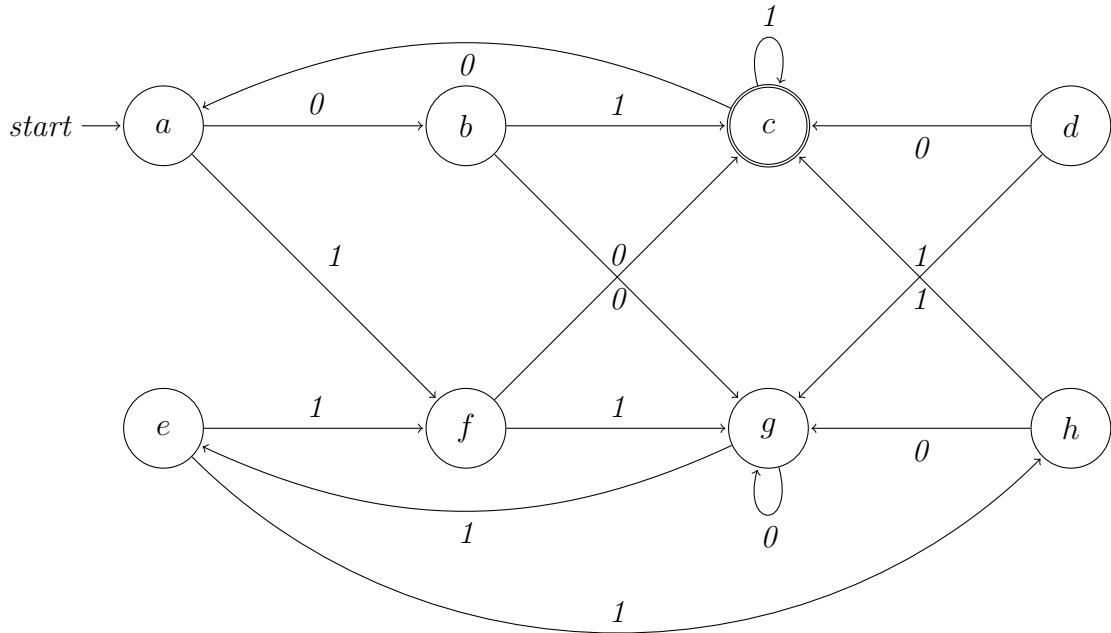
$$\hat{\delta}(p, w) \in F \text{ e } \hat{\delta}(q, w) \notin F \text{ o } \hat{\delta}(p, w) \notin F \text{ e } \hat{\delta}(q, w) \in F$$

quindi si ha una **relazione di equivalenza** e quindi si hanno le tre proprietà:

- **riflessività:**  $\forall p \in Q, p \approx p$
- **simmetricità:**  $\forall p, q \in Q, p \approx q \rightarrow q \approx p$
- **transitività:**  $\forall p, q, r \in Q, p \approx q \vee q \approx r \rightarrow p \approx r$

e quindi **in ogni classe di equivalenza ci sono stati tra loro equivalenti** due stati, uno finale e uno non finale, sono sicuramente distinguibili, basti pensare alla stringa vuota.

**Esempio 80.** Analizziamo il seguente automa:



cerco di capire se  $a$  e  $g$  sono equivalenti:

- $\varepsilon$  non li distingue perché entrambi sono non accettanti
- $0$  non li distingue perché li porta rispettivamente in  $b$  e  $g$ , che non sono accettanti
- $1$  non li distingue perché li porta rispettivamente in  $f$  e  $e$ , che non sono accettanti
- $01$  li distingue perché  $\hat{\delta}(a, 01) = c$  e  $\hat{\delta}(g, 01) = e$  e il primo è accettante mentre il secondo no, quindi i due stati non sono equivalenti

si verifica che invece  $a$  e  $e$  sono equivalenti

diamo una definizione ricorsiva dell'equivalenza tra stati:

- **caso base:** se  $p \in F$  e  $q \notin F$  o viceversa si ha che gli stati sono distinguibili
- **caso passo:** se per  $a \in \Sigma$  gli stati  $r = \delta(p, a)$  e  $s = \delta(q, a)$  sono distinguibili allora anche  $p$  e  $q$  lo sono



quindi se un certo arco  $w$  porta due stati a due diversi di cui uno è accettante essi sono distinti.

Esiste anche un algoritmo detto **riempi-tabella**:  
partiamo dalla seguente tabella

b		/	/	/	/	/	/
c			/	/	/	/	/
d				/	/	/	/
e					/	/	/
f						/	/
g							/
h							
	a	b	c	d	e	f	g

$c$  è accettante quindi segniamo tutte le caselle relative a  $c$  per il caso base:

b		/	/	/	/	/	/
c	x	x	/	/	/	/	/
d			x	/	/	/	/
e			x		/	/	/
f			x			/	/
g			x				/
h			x				
	a	b	c	d	e	f	g

poiché, per esempio  $c, h$  è distinguibile e gli stati  $e$  e  $f$  con 0 vanno in  $h$  e  $c$  si ha che anche  $e$  e  $f$  sono distinguibili, segno quindi nelle caselle appropriate. Proseguo controllando anche le altre coppie e riempio le caselle. Ottengo:

b	x	/	/	/	/	/	/
c	x	x	/	/	/	/	/
d	x	x	x	/	/	/	/
e		x	x	x	/	/	/
f	x		x	x	x	/	/
g	x	x	x	x	x	x	/
h	x		x	x	x	x	x
	a	b	c	d	e	f	g

quindi si ha solo che:

$$a \approx e \quad b \approx h \quad d \approx f$$

si ha che:

**Teorema 6.** *due stati non distinti dall'algoritmo riempi-tabella sono equivalenti. Se ne calcola la complessità:*

$$|Q| = n \rightarrow \frac{n(n-1)}{2} \text{ caselle} = O(n^2)$$

se ho una crocetta a iterazione ho il caso peggiore  $n^2n^2 = O(n^4)$

quindi per vedere se due linguaggi regolari sono equivalenti si ha che:

$$L, M \in REG$$

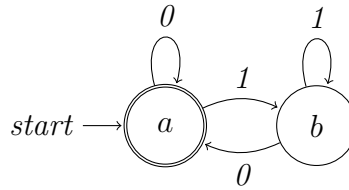
$$\exists A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$$

$$\exists A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$$

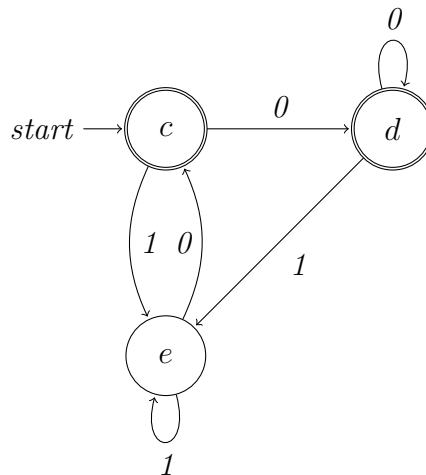
costruisco  $A = (Q_L \cup Q_M, \Sigma, \delta, q_L, F_L \cup F_M)$

$\delta$  è  $\delta_L$  per  $Q_L$  e  $\delta$  è  $\delta_M$  per  $Q_M$  per vedere se  $q_L \approx q_M$  uso il riempi tabella e se sono equivalenti si ha che  $L = M$

**Esempio 81.** *Sia A:*



e B:



entrambi i linguaggi accettano  $(\varepsilon + (0 + 1)^*0)$  applico la tabella:

$b$	$x$	$/$	$/$	$/$
$c$		$x$	$/$	$/$
$d$		$x$		$/$
$e$	$x$		$x$	$x$
	$a$	$b$	$c$	$d$

e quindi dato che  $a \approx c$  si ha che i due linguaggi sono lo stesso linguaggio.

### **CHIEDERE SPIEGAZIONI A RIGUARDO**

passiamo ora alla **minimizzazione**, che prende in input un DFA  $A$  e restituisce un DFA  $A_{min}$  tale che  $L(A) = L(A_{min})$  e che  $A_{min}$  ha il numero più piccolo possibile di stati per distinguere  $L(A)$ .

Procedo così:

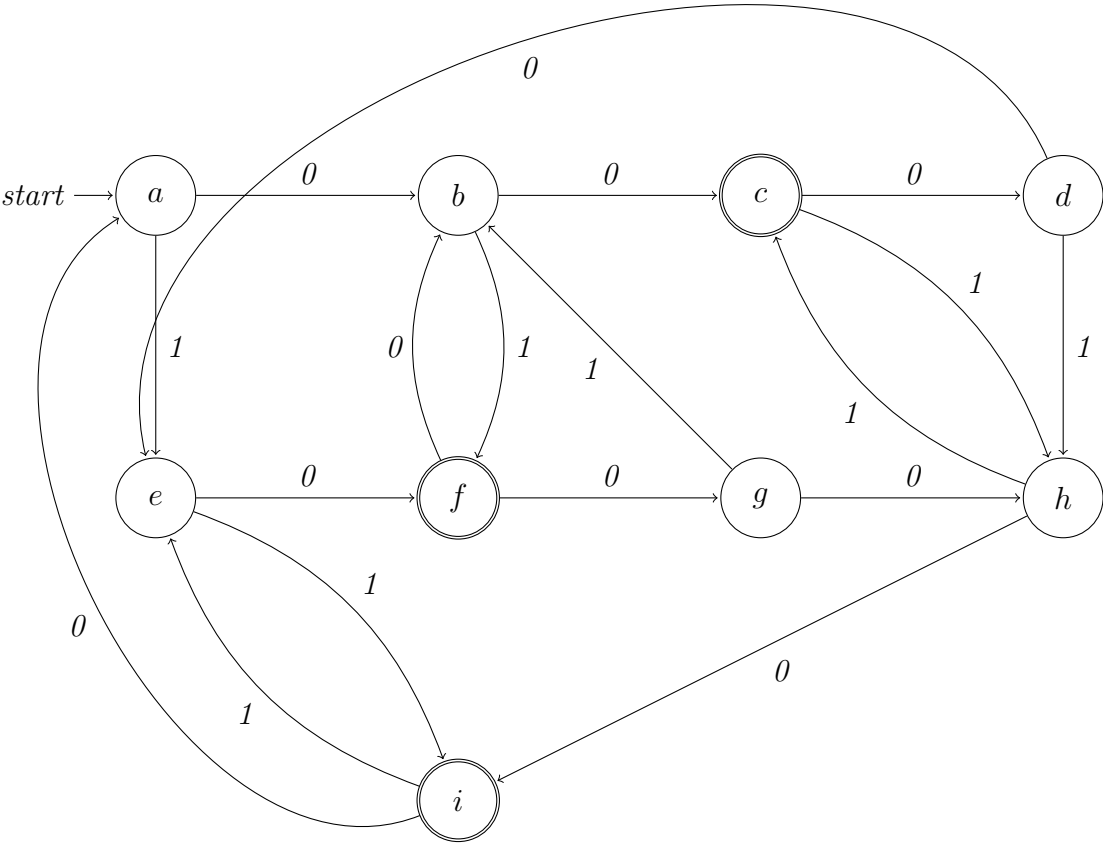
1. si rimuovono gli stati non raggiungibili
2. applico la tabella per scoprire le tabelle di equivalenza, gli stati del nuovo automa sono le classi di equivalenza

si ha che lo stato iniziale è la classe di equivalenza dello stato finale e gli stati finali sono le classi di equivalenza degli stati finali.

La minimizzazione si dimostra per assurdo:

Sia  $M$  il DFA ottenuto dalla tabella. Suppongo esista un DFA  $N$  tale che  $L(A) = L(N)$   $|Q_N| < |Q_M|$ . I due stati iniziali sono indistinguibili. Sia  $p \in M$   $q \in N$  tali  $p \approx q \forall a \in \Sigma$  quindi  $\delta(p, a) = \delta(q, a)$ , Ogni stato  $p \in Q$ , è quindi indistinguibile da almeno uno stato di  $N$ . Avendo però  $N$  meno stati si avranno almeno due stati di  $M$  indistinguibili dallo stesso di  $N$  ma non sono indistinguibili per la tabella. SI ha un assurdo.

**Esempio 82.** Si ha il seguente automa:



faccio la tabella

<i>b</i>		/	/	/	/	/	/	/
<i>c</i>	<i>x</i>	<i>x</i>	/	/	/	/	/	/
<i>d</i>		<i>z</i>	<i>x</i>	/	/	/	/	/
<i>e</i>	<i>x</i>		<i>x</i>	<i>x</i>	/	/	/	/
<i>f</i>	<i>x</i>	<i>x</i>		<i>x</i>	<i>x</i>	/	/	/
<i>g</i>		<i>x</i>	<i>x</i>		<i>x</i>	<i>x</i>	/	/
<i>h</i>	<i>x</i>		<i>x</i>	<i>x</i>		<i>x</i>	<i>x</i>	/
<i>i</i>	<i>x</i>	<i>x</i>		<i>x</i>	<i>x</i>		<i>x</i>	<i>x</i>
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>

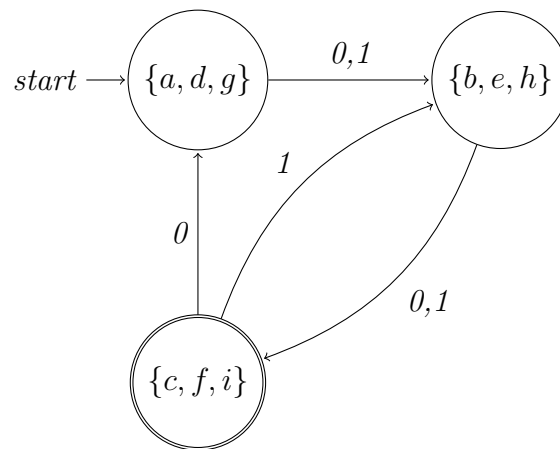
al caso base segno tutte le caselle degli stati accettanti tranne quelle che incrociano con altri stati accettanti e poi completo il resto della tabella. Ottengo:

$$a \approx d \approx g$$

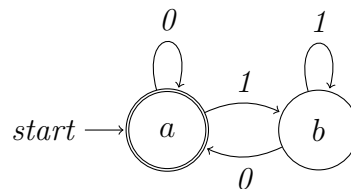
$$b \approx e \approx h$$

$$c \approx f \approx i$$

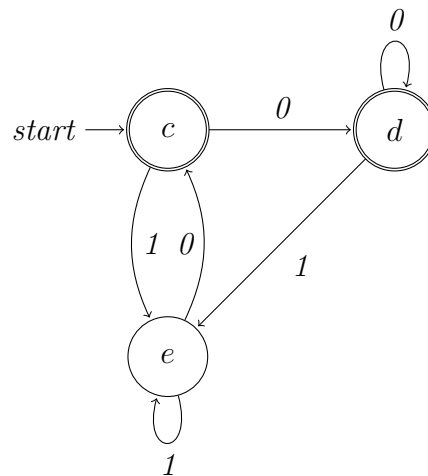
e quindi ottengo, l'automa minimizzato:



**Esempio 83.** Sia  $A$ :



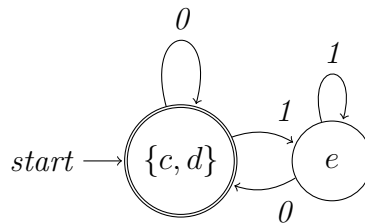
e  $B$ :



riduco  $B$ :

$d$		/
$e$	$x$	$x$
	$c$	$d$

quindi  $c \approx d$ . Ottengo quindi:



quindi  $A$  è  $B$  minimizzato, per questo riconoscevano lo stesso linguaggio

La tabella non funziona ovviamente per gli NFA

### 1.2.6 Pumping Lemma per i linguaggi regolari

Questo lemma serve a dimostrare che un linguaggio  $L$  non è regolare. Si procede per assurdo.

Partiamo da un esempio:

$$L_{01} = \{0^n 1^n \mid n \geq 1\} = \{01, 0011, \dots\}$$

supponiamo che questo linguaggio sia rappresentabile da un DFA  $A$  con  $k$  stati ( $L(A) = L_{01}$ ).

$0^k$  implica  $k + 1$  prefissi:  $\varepsilon, 0, 00, 000, \dots, 0^k$  e quindi  $\exists i, j$  tali che  $0^i$  e  $0^j$  finiscono nello stesso stato. Allora l'automa è ingannabile e accetterebbe  $0^i 0^j$  con  $i \neq j$ .

Formalmente si ha:

**Teorema 7.** *Sia  $L \in REG$  allora  $\exists n$  che dipende da  $L$  tale che  $\forall w \in L$  con  $|w| \geq n$ ,  $w$  può essere scomposta in tre stringhe  $w = xyz$  in modo che:*

$$\begin{aligned} y &\neq \varepsilon \\ |xy| &\leq n \\ \forall k \geq 0 \quad xy^kz &\in L \end{aligned}$$

in altre parole posso sempre trovare una stringa non vuota  $y$  non troppo distante dall'inizio di  $w$ , da replicare da ripetere o cancellare ( $k = 0$ ) senza uscire dal linguaggio  $L$

*Dimostrazione.* essendo  $L \in REG$  esiste un DFA  $A$  tale che  $L = L(A)$ . Suppongo  $|Q| = n$  e considero  $w = a_1 a_2 \dots a_n$  con  $m \geq n$ . Sia:

$$p_i = \hat{\delta}^\wedge(q_0, a_1, \dots, a_i) \quad \forall i = 0, 1, \dots, n$$

quindi:

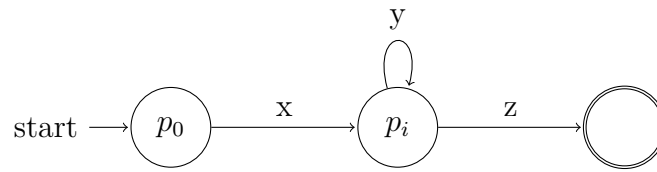
$$p_0 = q_0, p_1, \dots, p_n \rightarrow n + a \text{ stati}$$

allora  $\exists i, j$ , con  $0 \leq i < j \leq n$  tali che  $p_i = p_j$ . Scompongo ora  $w$  in  $w = xyz$  con:

$$x = a_1 a_2 \dots a_i$$

$$y = a_{i+1} a_{i+2} \dots a_j$$

$$z = a_{j+1} a_{j+2} \dots a_m$$



□

**Esempio 84.** *mostriamo che  $L_{01}$  non è regolare. Suppongo per assurdo che sia regolare, allora vale il pumping lemma. Sia  $n \in \mathbb{N}$  la costante del pumping lemma. Pongo  $w = 0^n 1^n = xyz$  tale che  $|xy| = n$ , ovvero  $|xy| = 0^n$ , quindi  $xy$  è formato da soli 0. Poniamo  $x = 0^{n-1}$   $y = 0 \neq \varepsilon$   $z = 1^n$ . Però per il pumping lemma  $\forall k \geq 0 \quad xy^kz \in L_{01}$ .*

$$xy^kz = 0^{n-1} 0^k 1^n = 0^{n+k-1} 1^n \in L_{01}$$

ma se  $k \neq 1$  ciò non è vero e quindi il linguaggio non è regolare

**Esempio 85.**  $L = \{w \in \{0,1\}^* \mid w = w^R\}$ , linguaggio delle stringhe palindrome, è regolare?

suppongo per assurdo di sì, con  $n$  costante del pumping lemma.

$w = 0^n 1 0^n$  quindi  $x = 0^{n-1}$   $y = 0$   $z = 10^n$ . Per  $k = 0$  si ha  $xy^kz = 0^{n-1} 1 0^n \notin L$ , quindi non è regolare

**Esempio 86.**  $L = \{1^p \mid p \text{ primo}\} = \{11, 111, 11111\}$  è regolare?

$w = 1^p = 1^{n-1} 1 1^{p-n}$  con:

$$x = 1^{n-1} \quad y = 1 \quad z = 1^{p-n}$$

per  $k = 0$  si ha  $1^{p-1}$  e  $p-1$  non è , quindi non è regolare

**Esempio 87.**  $L = \{0^n 1^m \mid n \leq m\}$  è regolare?

$w = 0^{n-l} 0^l 1^m$  con:

$$x = 0^{n-l} \quad y = 0^l \quad z = 1^m$$

con  $|y| = l$  e  $0 < l \leq n$

quindi  $\forall k \geq 0$  si ha  $xy^kz \in L = 0^{n-l} 0^{kl} 1^m$  con:

$$x = 0^{n-l} \quad y = 0^{kl} \quad z = 1^m$$

scelgo  $k$  tale che  $n - l + kl = n + (k-1)l > m \rightarrow k > 1 + \frac{m-n}{l}$  quindi non è regolare

**Esempio 88.**  $L = \{0^n 1^m \mid n \geq m\}$  è regolare?

$w = 0^{n-l} 0^l 1^m$  con:

$$x = 0^{n-l} \quad y = 0^l \quad z = 1^m$$

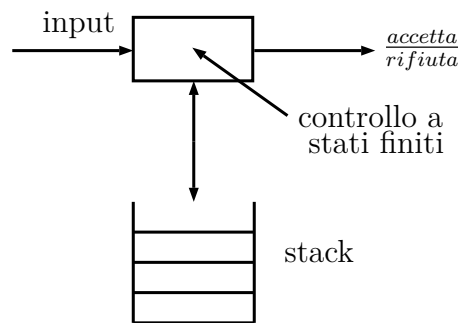
con  $|zy| \leq n$   $y \neq \varepsilon$   $|y| = l$  e  $0 < l \leq n$

scelgo  $l$  tale che  $n - l < m$  quindi non è regolare

su appunti dispense altri esempi

## 1.3 Automi a Pila

Si introduce un nuovo tipo di automa, il PDA (push down automata) che può essere pensato come un  $\varepsilon - NFA$  col supporto di una pila (stack):





e viene definito un PDA  $P$  come:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

con;

- $Q$ : insieme finito e non vuoto di stati
- $\Sigma$ : alfabeto di simboli di input
- $\Gamma$ : alfabeto di simboli di stack
- $q_0 \in Q$ : stato iniziale
- $z_0 \in \Gamma \setminus \Sigma$ : simbolo iniziale dello stack
- $F \in Q$ : insieme degli stati accettanti o finali

si ha che:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$$

quindi:

$$\delta(q_0, a, X) = \{(p_1, X_1), (p_2, X_2), \dots\} \text{ insieme finito } p_i \in Q \text{ } X_i \in \Gamma^*$$

si hanno dei casi particolari:

- lo stato  $p$  potrebbe coincidere con  $Q$  e si avrebbe un cappio
- se  $\Gamma = \varepsilon$  si ha il pop di  $X$  dallo stack
- se  $\Gamma = X$  si lascia lo stack invariato
- se  $\Gamma = Y \neq X$  si ha la sostituzione di  $X$  con  $Y$  in cima allo stack
- se  $\Gamma$  è una stringa di simboli si ha la rimozione di  $X$  dallo stack e l'aggiunta a uno a uno dei simboli nello stack

**Esempio 89.** Trovo PDA per il linguaggio delle stringhe binarie palindrome di lunghezza pari:  $L = \{ww^R | w \in \{0, 1\}^*\}$ . Con  $R$  che indica rovesciato. Si ha la CFG  $G = (\{P\}, \{0, 1\}, Prod, P)$  tale che:

$$P \rightarrow 0P0 | 1P1 | \varepsilon$$

si hanno quindi tre stati:

- $q_0$  che è quello iniziale che legge  $w$  e spinge i dati sullo stack
- $q_1$  che letta  $w$  legge i simboli di  $w^R$  e li confronta con quelli dello stack
- $q_2$  sarà la stringa accettata

descriviamo formalmente l'automa con la funzione di transizione  $\delta$ . PDA  $P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \{q_2\})$  ovvero:

$$\delta(q_0, 0, z_0) = \{(q_0, 1z_0)\}$$

$$\delta(q_0, 1, z_0) = \{(q_0, 0z_0)\}$$

$$\delta(q_0, 0, 0) = \{(q_0, 00)\}$$

$$\delta(q_0, 0, 1) = \{(q_0, 01)\}$$

$$\delta(q_0, 1, 0) = \{(q_0, 10)\}$$

$$\delta(q_0, 1, 1) = \{(q_0, 11)\}$$

$$\delta(q_0, \varepsilon, z_0) = \{(q_0, z_0)\}$$

$$\delta(q_0, \varepsilon, 0) = \{(q_0, 0)\}$$

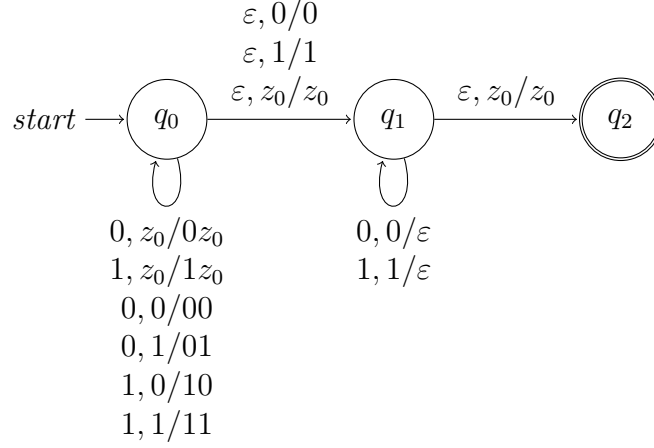
$$\delta(q_0, \varepsilon, 1) = \{(q_0, 1)\}$$

$$\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$$

$$\delta(q_2, 1, 1) = \{(q_1, \varepsilon)\}$$

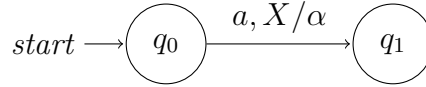
$$\delta(q_2, \varepsilon, z_0) = \{(q_2, 0z_0)\}$$

otteniamo il seguente PDA:



e si definisce questa notazione per gli archi:

$$(p, \alpha) \in \delta(q, a, X)$$



analizziamo meglio i PDA. Si ha che la **descrizione istantanea (ID)** di un PDA è una tripla:

$$ID : (q, w, \gamma)$$

con  $q \in Q$  stato attuale  $w \in \Sigma^*$  input rimanente e  $\gamma \in \Gamma^*$  contenuto attuale dello stack.

Definiamo ora il concetto di **mossa in un passo** dato  $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

la mossa è una relazione  $\overset{\vdash}{p}$ :

$$(p, \alpha) \in \delta(q, a, X) \text{ allora } \forall w \in \Sigma^* \text{ e } \forall \beta \in \Gamma^* \rightarrow (q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

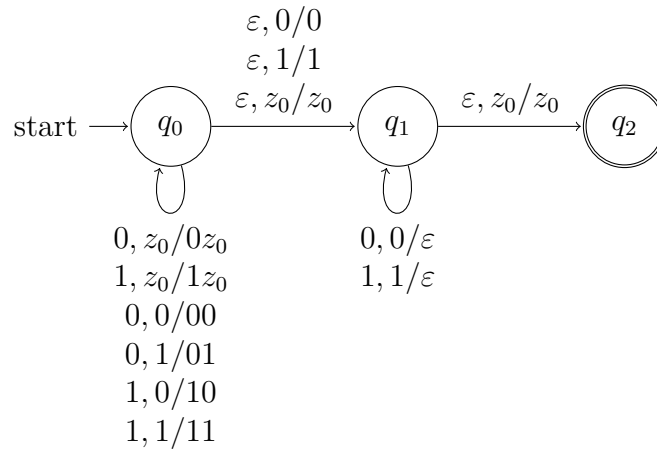
e

$$(p, \alpha) \in \delta(q, \varepsilon, X) \text{ allora } \forall w \in \Sigma^* \text{ e } \forall \beta \in \Gamma^* \rightarrow (q, w, X\beta) \vdash (p, w, \alpha\beta)$$

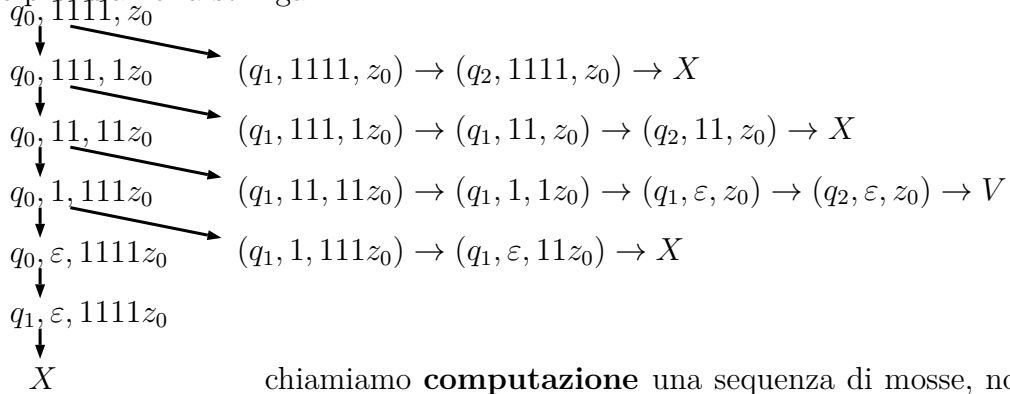
ora possiamo anche definire la relazione con 0 o più mosse in forma induttiva  $\overset{*}{\vdash}$   $p$ :

- **caso base:**  $\forall ID I, I \overset{*}{\vdash} I$
- **caso passo:**  $I \overset{*}{\vdash} J$  se  $\exists ID K$  tale che  $I \vdash K$  e  $K \overset{*}{\vdash} J$

vediamo un esempio con un PDA che accetta  $ww^R \mid w \in \{0,1\}^*$ :



e prendiamo la stringa 1111:



chiamiamo **computazione** una sequenza di mosse, non necessariamente di successo. Si hanno alcune proprietà:

- se una sequenza di ID è lecita per un PDA P allora è lecita anche la sequenza di ID ottenuta concatenando  $w \in \Sigma^*$  in ogni ID
- se una sequenza di ID è lecita per un PDA P e resta una coda di input non consumata allora posso rimuovere tale coda in ogni ID e ottenere un'altra sequenza lecita

- se una sequenza di ID è lecita per un PDA  $P$  allora è lecita la sequenza ottenuta aggiungendo  $\gamma \in \Gamma^*$  in coda alla terza sequenza di ogni ID

del resto però:

$$(q, Xw, \alpha\gamma) \stackrel{*}{\vdash} (p, Yw, \beta\gamma) \not\vdash (q, X, \alpha) \stackrel{*}{\vdash} (p, y, \beta), \quad x, w, y \in \Sigma^* \quad \alpha, \beta, \gamma \in \Gamma^*$$

per queste proprietà valgono i seguenti teoremi:

**Teorema 8.** *per la seconda:* Se  $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$  è un PDA e  $(q, Xw, \alpha) \stackrel{*}{\vdash} (p, Yw, \beta)$  allora vale anche:

$$(q, X, \alpha) \stackrel{*}{\vdash} (p, Y, \beta)$$

**Teorema 9.** *per la prima e la terza:* Se  $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$  è un PDA e  $(q, X, \alpha) \stackrel{*}{\vdash} (p, Y, \beta)$  allora:

$$\forall \gamma \in \Gamma^* \text{ vale anche } (q, Xw, \alpha\gamma) \stackrel{*}{\vdash} (p, Yw, \beta\gamma)$$

Si definiscono due modalità di accettazione per i PDA:

1. **per stato finale:** sia  $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$  si ha che:

$$L(P) = \{w \in \Sigma^* \mid (q_0, w, z_0) \stackrel{*}{\vdash} (q, \varepsilon, \alpha)\}$$

con  $q \in F$  e  $\forall \alpha \in \Gamma^*$

2. **per stack vuoto:** sia  $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$  si ha che:

$$N(P) = \{w \in \Sigma^* \mid (q_0, w, z_0) \stackrel{*}{\vdash} (q, \varepsilon, \varepsilon)\}$$

con  $q \in Q$  e in questo caso l'insieme degli stati finali  $F$  non ha alcuna influenza

In realtà si ha che la classe di linguaggi accettati dai PDA per stato finale è uguale a quella per stack vuoto, anche se passare da un tipo all'altro di PDA è complesso. Si ha il seguente teorema per la trasformazione:

**Teorema 10.** se  $L = N(P_N)$  per un PDA  $P_N = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$  allora  $\exists$  PDA  $P_F$  tale che  $L = L(P_F)$

*Dimostrazione.* Sia  $x_0 \in \Gamma$ , che indica la fine dello stack di  $P_F$ . Si ha:

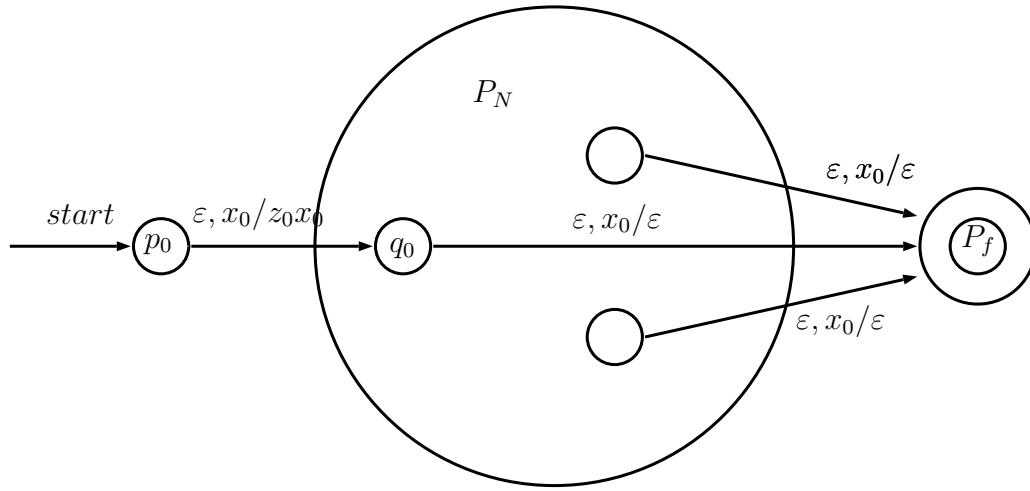
$$\delta(p_0, \varepsilon, x_0) = \{(q_0, z_0 x_0)\}$$

e:

$\forall q \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall y \in \Sigma : \delta_F(q, a, y)$  contiene tutte le coppie di  $\delta_N(q, a, y)$

$$\forall q \in Q, \delta_F(q, \varepsilon, x_0) = \{(P_F, \varepsilon)\}$$

quindi graficamente:



Bisogna dimostrare che effettivamente  $w \in L(P_F) \iff w \in N(P_N)$ .

se  $w \in N(P_N) \exists$  una sequenza di ID  $(q_0, w, z_0) \xrightarrow{*} \vdash P_N (q, \varepsilon, \varepsilon)$  per un qualche  $q \in Q$ :

$$(q_0, w, z_0 x_0) \xrightarrow{*} \vdash P_N (q, \varepsilon, x_0)$$

inoltre:

$$(q_0, w, z_0 x_0) \xrightarrow{*} \vdash P_F (q, \varepsilon, x_0)$$

e quindi:

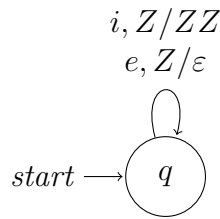
$$(p_0, w, x_0) \xrightarrow{\vdash} P_F (q_0, w, z_0 x_0) \xrightarrow{*} \vdash P_F (q, \varepsilon, x_0) \xrightarrow{\vdash} P_F (P_f, \varepsilon, \varepsilon)$$

solo se togliendo il primo e l'ultimo passo di  $P_F$  ripercorro all'indietro quanto scritto sopra.  $\square$

**Esempio 90.** trasformazione da accettante per stack vuoto a accettante per stato finale. Siano:

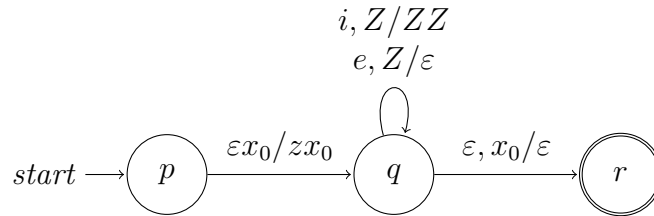
$$\begin{aligned}\Sigma &= \{i, e\} \\ P_n &= (\{q\}, \{i, e\}, \{Z\}, \delta_M, q, Z) \\ \delta_N(q, i, Z) &= \{(q, ZZ)\} \\ \delta_N(q, e, Z) &= \{(q, \varepsilon)\}\end{aligned}$$

quindi:



quindi inseriamo una  $Z$  quando leggiamo  $i$  e ne rimuoviamo una se leggiamo  $e$  e si parte con una  $Z$  nello stack.

Costruisco ora il PDA  $P_F$  che accetta lo stesso linguaggio ma per stato finale, introduco lo stato iniziale  $p$  e quello accettante  $r$ , uso  $x_0$  come segnale della fine dello stack:



e si ha formalmente:

$$P_F = (\{p, q, r\}, \{i, e\}, \{Z, x_0\}, \delta_F, p, x_0, \{r\})$$

con  $\delta_F$  che rappresenta le seguenti quattro regole:

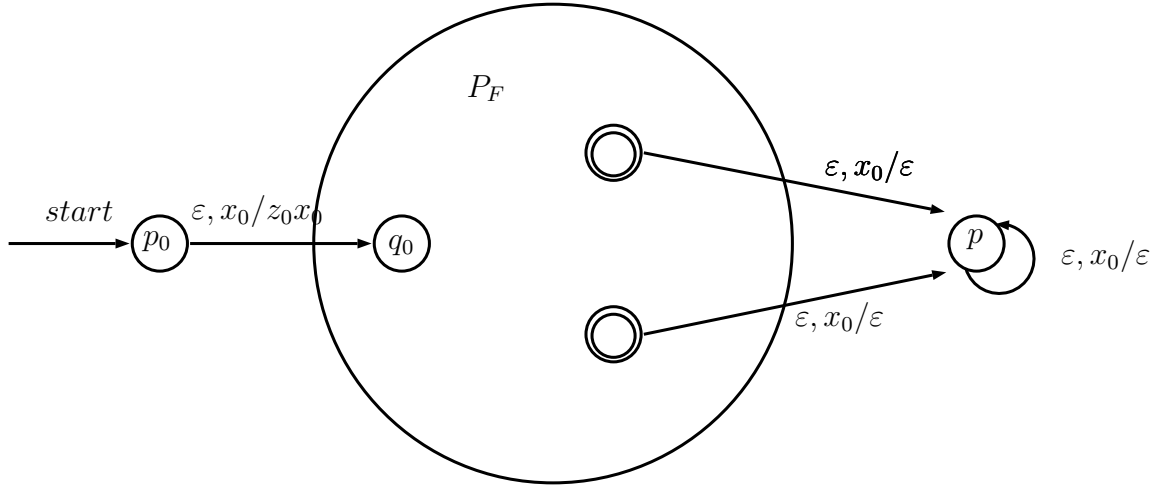
1.  $\delta_F(p, \varepsilon, x_0) = \{(q, Zx_0)\}$  regola che fa partire  $P_F$  con  $x_0$  come segnalatore dello stack
2.  $\delta_F(p, i, Z) = \{(q, ZZ)\}$  regola che inserisce  $Z$  quando si ha  $i$  simulando  $P_N$
3.  $\delta_F(p, e, Z) = \{(q, Z\varepsilon)\}$  regola che rimuove  $Z$  quando si ha  $e$  simulando  $P_N$

4.  $\delta_F(p, e, x_0) = \{(r, \varepsilon)\}$  regola che permette a  $P_F$  di accettare quando  $P_N$  esaurisce lo stack

Si può anche effettuare la trasformazione inversa:

**Teorema 11.** Sia  $P_F = (Q, \Sigma, \Gamma, \delta_f, q_0, Z_0, F)$ .

Si aggiunge una transizione  $\varepsilon$  a un nuovo stato  $p$  da ogni accettante di  $P_F$ . quando si ha  $p$   $P_N$  svuota lo stack senza consumare input. Quindi se  $P : F$  entra in uno stato accettante dopo aver consumato l'input  $w$ ,  $P_N$  svuota lo stack dopo aver consumato  $w$ . Per evitare che si svuoti lo stack per una stringa non accettata uso  $x_0$  per indicare il fondo dello stack. Il nuovo  $P_N$  parte da  $p_0$  che ha il solo scopo di inserire il simbolo iniziale di  $P_F$  e passare al suo stato iniziale. Si ottiene quindi:



e si ha formalmente:

$$P_F = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{x_0\}, \delta_N, p_0, x_0)$$

dove  $\delta_N$  è così definita:

1.  $\delta_N(p_0, \varepsilon, x_0) = \{(q_0, Z_0 x_0)\}$  inserisce il simbolo iniziale di  $P_F$  nello stack e va allo stato iniziale di  $P_F$
2.  $\forall q \in Q$  ogni simbolo di input  $a \in \Sigma$ , compreso l'input vuoto, e  $\forall y \in \Gamma$ ,  $\delta_N(q, a, y)$  contiene tutte le coppie di  $\delta_F(q, a, y)$ . Quindi  $P_N$  simula  $P_F$
3. per tutti gli stati accettanti  $q \in F$  e i simboli di stack  $y \in \Gamma$ , compreso  $x_0$ , si ha che  $\delta_N(q, \varepsilon, y)$  contiene  $(p, \varepsilon)$ , quindi ogni volta che  $P_F$  accetta  $P_N$  inizia scaricare lo stack senza consumare ulteriori input



4. per tutti i simboli di stack  $y \in \Gamma$ , compreso  $x_0$ , si ha che  $\delta_N(q, \varepsilon, y) = \{(p, \varepsilon)\}$ , quindi giunti allo stato  $p$ , ovvero quando  $P_F$  ha accettato,  $P_N$  elimina ogni simbolo nel suo stack fino a svuotarlo

inoltre formalmente voglio dimostrare che:

$$w \in L(P_F) \rightarrow w \in N(P_N)$$

e quindi ho le seguenti mosse:

$$(q_0, w, z_0) \overset{*}{\vdash} P_F (q, \varepsilon, \alpha) \quad q \in F, \quad \alpha \in \Gamma^*$$

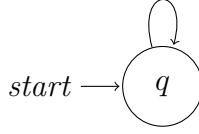
$$(p_o, w, x_0) \vdash (q_0, w, z_0 x_0) \overset{*}{\vdash} P_N (q, \varepsilon, \alpha, x_0) \overset{*}{\vdash} P_N (p, \varepsilon, \varepsilon)$$

**Esempio 91.** si ha una CFG  $G = (\{i, e\}, \{a, b, 0, 1, *, +, (, )\}, P, E)$  con:

$$P : I \rightarrow a|b|Ia|Ib|I0|I1$$

$$E \rightarrow E + E | E * E | (E)$$

si ha il PDA  $P_G = (\{q\}, \Sigma, \Sigma \cup \{i, e\}, \delta, q, E)$ :



si ha quindi:

$$\delta(q, \varepsilon, i) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$$

$$\delta(q, \varepsilon, E) = \{(q, I), (q, E + E), (q, E * E), (q, I(E))\}$$

$$\delta(q, a, a) = \{(q, \varepsilon)\}$$

$$\delta(q, b, b) = \{(q, \varepsilon)\}$$

$$\dots = \{(q, \varepsilon)\}$$

quindi si ha:

$$E \rightarrow E + E \rightarrow i + E \rightarrow a + (E) \rightarrow a + (i) \rightarrow a + (i0) \rightarrow a + (b0)$$

$$(q, a + (b0), E) \vdash (q, a + (b0), E + E) \vdash (q, a + (b0), i + E) \vdash (q, a + (b0), a + E)$$

$$\vdash (q, +(b0), +E) \vdash (q, (b0), E) \vdash (q, (b0), (E)) \vdash (q, b0), E))$$

$$\vdash (q, b0), i)) \vdash (q, b0), i0)) \vdash (q, b0), b0) \vdash (q, 0), 0))$$

$$\vdash (q, ), )) \vdash (q, \varepsilon, \varepsilon)$$

questo esempio è generalizzabile ad ogni CFG:

$$\begin{aligned} \exists PDA Q = (\{q\}, T, V \cup T, \delta, q, S) \text{ tale che } N(Q) &= L(G) \\ \forall A \in V \quad \delta(q, \varepsilon, A) &= \{(q, \beta) \mid A \rightarrow B \text{ e' una produzione di } G\} \\ \forall a \in T \quad \delta(q, a, a) &= \{(a, \varepsilon)\} \end{aligned}$$

**Teorema 12.** *sia  $G = (V, T, P, S)$  una CFG:*

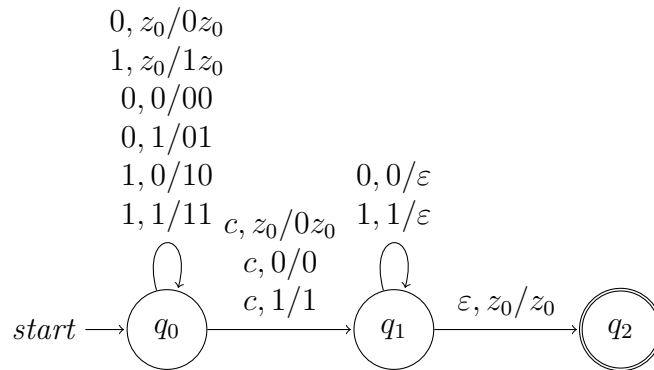
Questo dimostra che ogni CFL può essere accettato da un PDA accettante per stack vuoto. Per il teorema visto in precedenza, posso sempre costruire un altro PDA accettante per stati finale. I PDA accettano tutti e soli i linguaggi CF. Mostrare che accettano solo linguaggi di tipo 2 è complicato. Un tipo di PDA interessante, soprattutto per i parse, è il PDA deterministico, il **DPDA**.

Un PDA  $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$  è deterministico se:

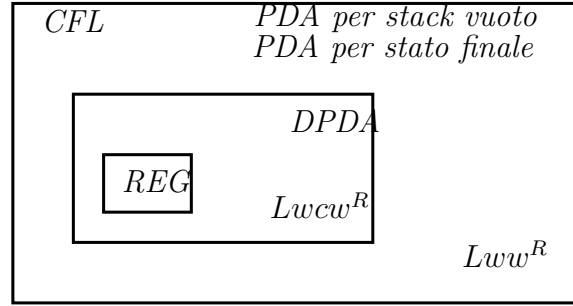
1.  $|\delta(q, a, x)| \leq 1 \quad \forall q \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall x \in \Gamma$
2. se  $|\delta(q, a, x)| \neq 0$  per qualche  $a \in \Sigma$  allora  $|\delta(q, \varepsilon, x)| = 0$

**Esempio 92.** *abbiamo il linguaggio  $L_{wcw^R} = \{wcw^R \mid w \in \{0, 1\}^*\}$*

*Gli automi a pila deterministici non riconoscono tutti i CFL, ma solo una classe strettamente più piccola. Ad esempio non potrebbero riconoscere il linguaggio delle palindrome senza "il segnalibro"  $c$ . SI ha quindi:*



quindi:



si ha infatti il seguente teorema:

**Teorema 13.**  $L \in REG \rightarrow \exists PDA P$  tale che  $L = L(P)$

*Dimostrazione.*

$$L \in REG \rightarrow \exists DFA A = (Q, \Sigma, \delta_A, q_0, F) \text{ tale che } L = L(A)$$

costruisco il DPDA  $P = (Q, \Sigma, \{z_0\}, \delta_p, q_0, z_0, F)$  con:

$$\delta_p(q, a, z_0) = \{p, z_0\} \forall p, q \in Q \text{ tali che } \delta_A(q, a) = 0$$

vale:

$$(q_0, w, z_0) \xrightarrow{A} P (p, \varepsilon, z_0) \longleftrightarrow \delta_A^\wedge(q_0, w) = p$$

□

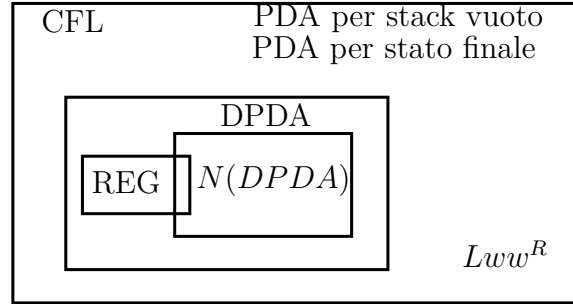
si ha inoltre il seguente teorema:

**Teorema 14.**  $L \in N(P)$  per un DPDA  $P$  sse  $L \in L(P')$  per un DPDA  $P'$  e  $L$  ha le proprietà di prefisso **prefix-free**

definiamo così la proprietà di prefisso:

$$\nexists x, y \in L \text{ tali che } x \neq y \text{ e } x \text{ è prefisso di } y$$

per esempio  $L = \{0\}^0 = \{\varepsilon, 0, 00, 000, \dots\}$  non ha la proprietà di prefisso. Osserviamo che se la stringa vuota appartiene al linguaggio, tale stringa è prefissa di tutte le altre e quindi il linguaggio non può avere la proprietà di prefisso. Affermiamo che  $L$  è regolare, quindi è accettato da un DPDA per stati finali ma non da uno per stack vuoto. Completiamo il diagramma precedente sulle classi di linguaggi:



SI ha che  $L_{wcw^R}$  gode della proprietà di prefisso:

$$y = wcw^R \in L \text{ Se } x \neq y, \text{ prefisso di } y, x \notin L$$

tornando alle grammatiche si hanno ora due teoremi:

**Teorema 15.** *se  $L = N(P)$  per un DPDA  $P$ , allora  $L$  ha una CFG non ambigua*

**Teorema 16.** *se  $L = L(P)$  per un DPDA  $P$ , allora  $L$  ha una CFG non ambigua*

dimostriamo il secondo:

*Dimostrazione.*  $L = L(P)$  per un DPDA  $P$ , costruiamo  $L' = L$ , quindi  $L'$  ha la proprietà di prefisso. Esiste quindi un DPDA  $P'$  tale che  $L'N(P)$ , esiste quindi per il teorema sopra una CFG  $G'$  tale che  $L(G') = L'$  che non è ambigua.

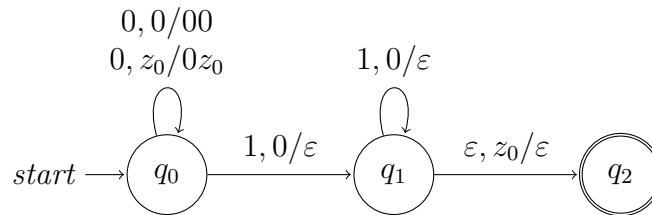
Costruiamo  $G$  per  $L$  con le stesse produzioni di  $G'$  più  $\$ \rightarrow \varepsilon$ , applicata solo all'ultimo passo.  $\square$

Vogliamo scoprire se è vero il viceversa: per ogni  $L$  che ha una CFG non ambigua è vero che  $L$  è accettato da un DPDA? No, mostriamo infatti un controesempio:

$S \rightarrow 0S0|1S1|\varepsilon$  produce  $L_{ww^R}$  che non è accettato da alcun PDA

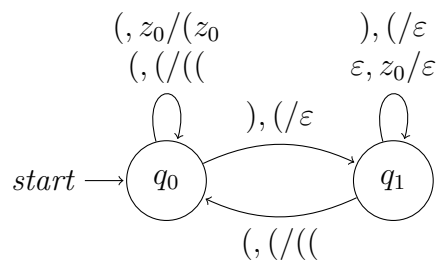
### 1.3.1 Esercizi

**Esercizio 1.** *costruire un PDA per  $L = \{0^n n | n \geq 1\}$ :*

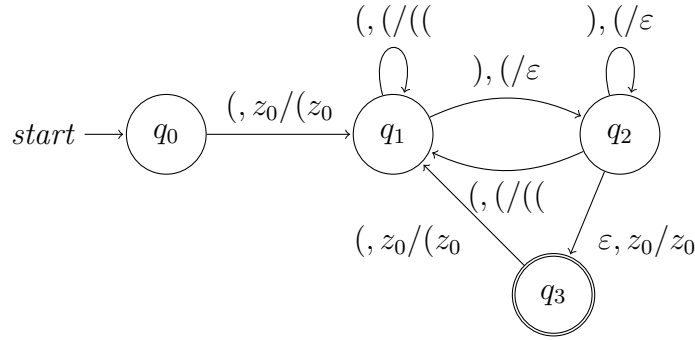


$$y = 0^n 1^n \in L, x \neq y$$

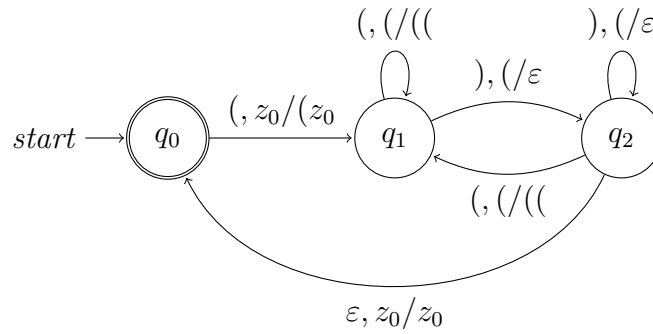
**Esercizio 2.** costruire un PDA per  $L = \{0^n n \mid n \geq 0\}$ :



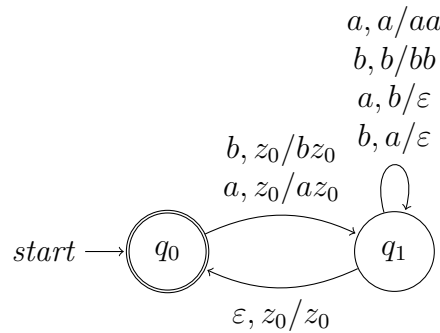
**Esercizio 4.** Si ha  $B \rightarrow BB|(B)|()$ . La stringa vuota non appartiene al linguaggio ma  $L$  non ha la proprietà del prefisso (come abbiamo mostrato nell'esercizio precedente). Possiamo quindi realizzare solo un DPDA per stato finale:



**Esercizio 5.** Si ha  $B \rightarrow BB|(B)|\varepsilon$  ho un DPDA per stato finale perché  $L$  non ha la proprietà del prefisso:



**Esercizio 6.** sia  $L = \{q \in \{a, b\}^* | \text{numero uguale di } a \text{ e } b\}$  DPDA per stato finale ( $L$  non ha la proprietà di prefisso):

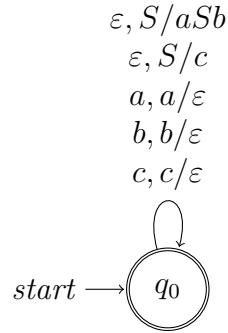


**Esercizio 7.** sia dato il CFL  $L = \{a^n cb^n | n \geq 0\}$   
 è generato da :

$$G = (\{S\}, \{a, b, c\}, P, S)$$

$$S \rightarrow aSb | c$$

si ha il seguente automa non deterministico:



con:

$$\delta(q, \varepsilon, S) = \{(q, aSb), (q, c)\}$$

$$\delta(q, a, a) = \{(q, \varepsilon)\}$$

$$\delta(q, b, b) = \{(q, \varepsilon)\}$$

$$\delta(q, c, c) = \{(q, \varepsilon)\}$$

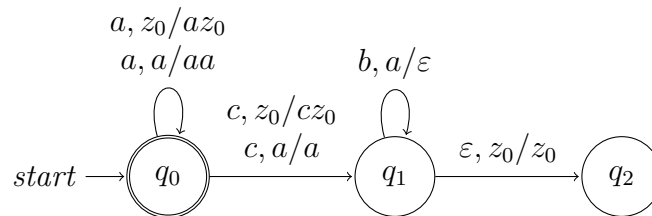
mostro la derivazione per  $n=3$ ,  $aaacbbb$  e il comportamento dell'automa:

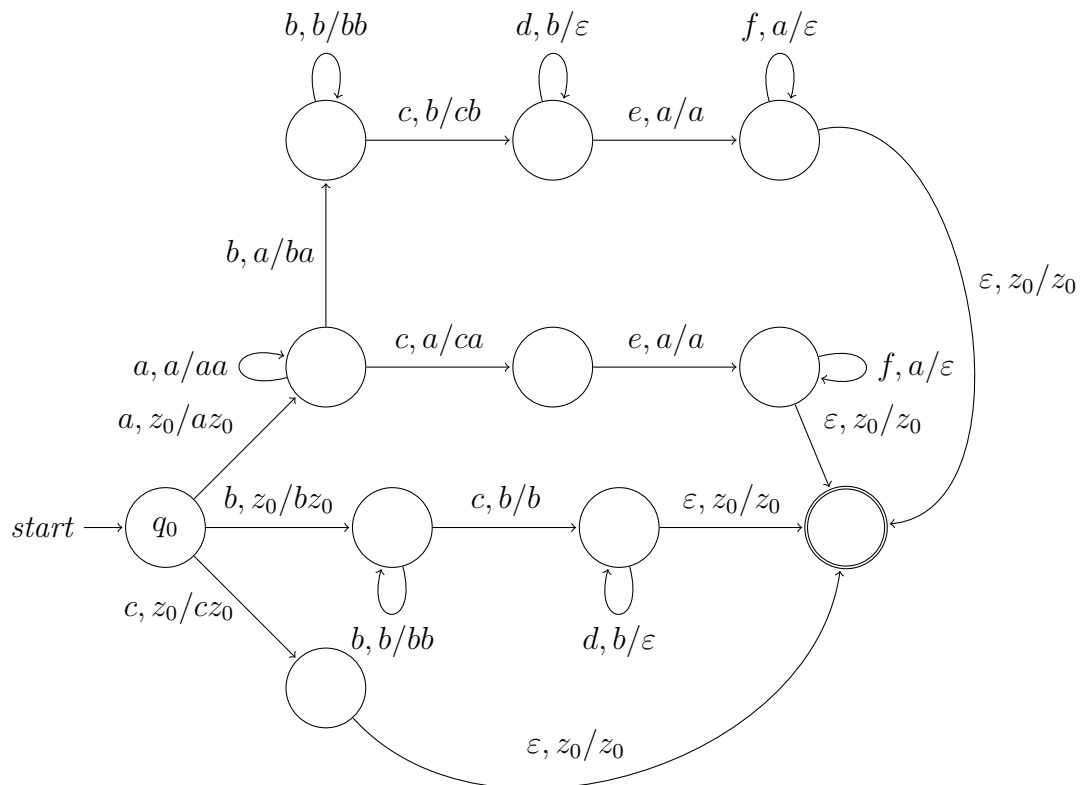
$$(q, aaacbbb, S) \vdash (q, aaacbbb, aSb) \vdash (q, aacbb, Sb) \vdash (q, aacbb, aSb)$$

$$\vdash (q, acbbb, Sbb) \vdash (q, acbbb, aSbbb) \vdash (q, cbbb, Sbbbb) \vdash (q, cbbb, cbbb)$$

$$\vdash (q, bbb, bbb) \vdash (q, bb, bb) \vdash (q, b, b) \vdash (q, \varepsilon, \varepsilon) \rightarrow \text{accetta}$$

si ha che vale la proprietà del prefisso e si ha il seguente DPDA:



$$ce \ n = 0 = m$$




## 1.4 Macchine di Turing

Nascono in risposta al problema di Hilbert, che si chiedeva se esiste un algoritmo per dimostrare teoremi. Non ci sono comunque funzioni non calcolabili con le macchine di Turing:

$$f : \mathbb{N} \rightarrow \{0, 1\} \quad s^{|\mathbb{N}|} > \mathbb{N}$$

Siamo interessati a studiare la calcolabilità (computabilità). Esiste un programma che calcola una certa funzione? Se non esiste,  $f$  è indecidibile. Se sì, in quanto tempo è calcolabile (complessità computazionale), quante mosse e quante celle del nastro sono necessarie?

vediamo due esempi:

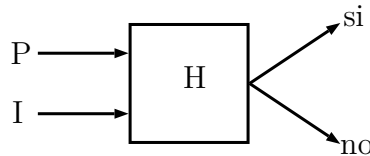
**Esempio 93.** *data una CFG  $G = (V, T, P, S)$  stabilire se è ambigua. Questo problema è indecidibile, cioè non esiste nessun algoritmo che dia una risposta.*

**Esempio 94.** *Problema "ciao mondo": dato un sorgente di un programma in C/Java/..., i primi undici caratteri stampati dal programma sono "ciao, mondo"? Consideriamo il seguente algoritmo, che prende in input un numero intero  $N$  e stampa "ciao mondo" sse  $\exists x, y, z \in \mathbb{N}$  tali che  $x^n + y^n = z^n$*

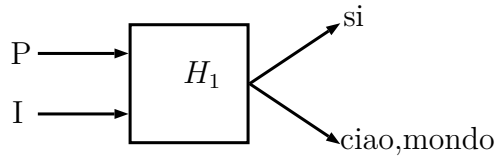
quest'ultimo problema è indecidibile:

*Dimostrazione.* procediamo per assurdo.

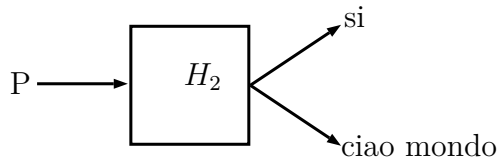
Supponiamo che esista un algoritmo che risolva il problema. Assumiamo che  $P$  sia corretto (e quindi compilabile) e che stampi solo stringhe sulla console:



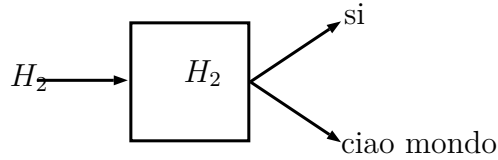
quindi:



suppongo di dare in input solo programmi, sse  $H_1$  stampa con  $P = I$ :



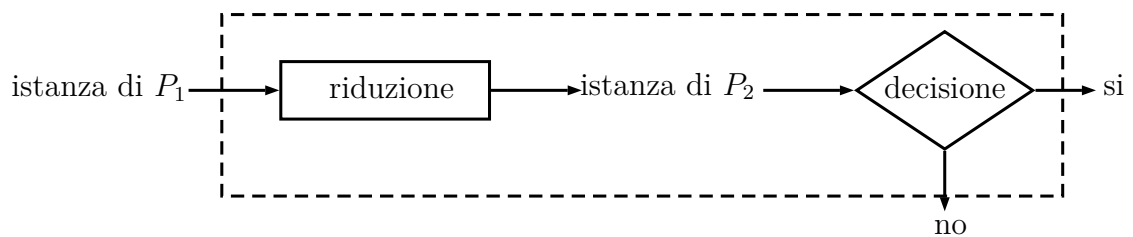
Allora posso pensare di dare ad  $H_2$  il programma  $H_2$  stesso:



Ma questo è assurdo: perché  $H_2$  dice sì quando in realtà stampa ciao mondo e viceversa, quindi non funziona. Ma allora in conclusione  $H$  non può esistere (perché i passaggi da  $H$  a  $H_1$  e da  $H_1$  a  $H_2$  sono leciti) e quindi il problema non è decidibile.  $\square$

### 1.4.1 Riduzioni

Supponiamo  $P_2$  indecidibile. Considerato un nuovo problema  $P_2$ , vorrei stabilire se anch'esso è indecidibile. Vorremmo quindi fare una "riduzione" da  $P_1$  a  $P_2$ :



Supponiamo per assurdo che  $P_2$  sia decidibile. Allora esiste l'algoritmo di decisione. Esistendo il processo di riduzione, allora l'intero rettangolo tratteggiato sarebbe un algoritmo di decisione per  $P_1$ , assurdo perché  $P_1$  è indecidibile per ipotesi.

**Esempio 95.**  $P_2$  (Problema della chiamata): dato un programma  $Q$ , esso chiama il metodo  $m()$ ?

Dobbiamo quindi trovare un algoritmo di riduzione. Procediamo:

- rinominiamo tutte le istanze di  $m$  in qualcosa d'altro
- aggiungiamo un metodo  $m$  che non verrà quindi mai chiamato
- modifichiamo questo programma in modo che salvi in un array i primi 11 caratteri che stampa a video
- modifichiamo questo programma in modo che se sfora gli 11 caratteri e controlla tali caratteri. Se sono esattamente "ciao mondo", chiama  $m$

quindi anche questo problema è indecidibile

È importante che la riduzione sia fatta nel verso corretto. Se facessimo la riduzione da  $P_2$  a  $P_1$ , staremmo mostrando che se  $P_1$  è decidibile allora  $P_2$  è decidibile. Ma sappiamo che  $P_1$  è indecidibile, quindi non staremmo dimostrando niente. Nel caso contrario stiamo invece dicendo che se  $P_2$  è decidibile allora  $P_1$  è decidibile. Ma essendo  $P_1$  indecidibile, allora l'antecedente è falsa e quindi  $P_2$  non può essere decidibile, quindi è indecidibile.

Diamo ora la definizione formale della Macchina di Turing:

**Definizione 10.** Si definisce *MdT*:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

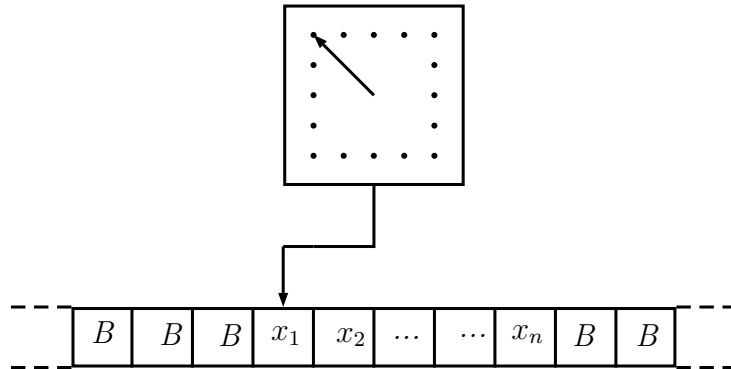
con:

- $Q$  insieme finito non vuoto di stati
- $\Sigma$  insieme finito non vuoto di simboli di input
- $\Gamma$  insieme finito non vuoto di simboli sul nastro
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$  stato iniziale
- $B \in \Gamma \setminus \Sigma$  simboli di blank
- $F \subseteq Q$  insieme di stati finali

inoltre si ha che:

$$\Sigma \subseteq \Gamma \text{ e } \Gamma \cap \Sigma \neq \emptyset$$

quindi si ha per esempio questa rappresentazione, per l'input  $x_1, \dots, x_n \in \Sigma^*$ :



Questa era una definizione deterministica, passiamo ora ad una definizione *istantanea*:

**Definizione 11.** *suppongo di avere lo stato  $q$ , il nastro  $x_1, \dots, x_n$  con la testina  $x_i$ . SI ha:*

$$ID : x_1x_2, \dots qx_ix_{i+1} \dots x_n$$

e suppongo  $Q \cap \Gamma = \emptyset$  senza perdere generalità e uso la simbologia dei  $\vdash$  introdotta coi PDA.

Se  $\delta(q, x_i) = (p, y, L)$  allora  $\vdash x_1x_2 \dots x_{i-2}px_{i-1}yx_{i+1} \dots x_n$ .

Si hanno dei casi particolari:

- se  $i = 1$  si ha  $qx_1x_2 \dots x_n \vdash pByx_2 \dots x_n$
- se  $i = n$   $y = B$  si ha  $qx_1x_2 \dots qx_n \vdash x_1x_2 \dots x_{n-2}px_{n-1}$

a destra abbiamo invece: Se  $\delta(q, x_i) = (p, y, R)$  allora  $\vdash x_1x_2 \dots x_{i-1}Ypx_{i+1} \dots x_n$ .

Si hanno dei casi particolari:

- se  $i = 1$  si ha  $q_1x_2 \dots x_{n-1} \vdash ypB$
- se  $i = n$   $y = B$  si ha  $qx_1x_2 \dots qx_n \vdash px_2 \dots x_n$

**Esempio 96.** *sia  $L = \{0^n1^n \mid n \geq 1\}$  un CFL. vediamo una tabella per capire l'azione della macchina:*

	$0$	$1$	$x$	$y$	$B$
$q_0$	$(q_1, x, R)$	—	—	$(q_3, y, R)$	—
$q_1$	$(q_1, 0, R)$	$(q_2, y, L)$	—	$(q_1, y, R)$	—
$q_2$	$(q_2, 0, L)$	—	$(q_0, x, R)$	$(q_2, y, L)$	—
$q_3$	—	—	—	$(q_2, y, L)$	$(q_4, B, R)$
$q_4$	—	—	—	—	—

ovvero, per esempio:

000111

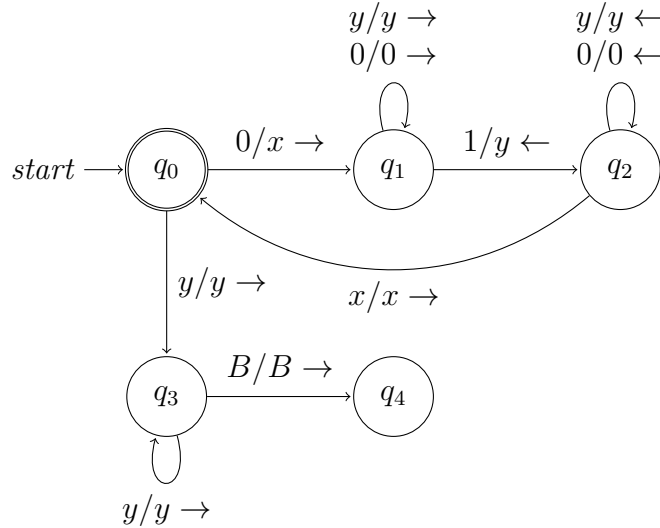
$x$ 00111

$x$ 00 $y$ 11

$xx$ 0 $yy$ 1

$xxx$  $yyy$

ma la tabella non è comodissima, usiamo quindi i diagrammi di transizione:



e per l'input 0010 si hanno i seguenti passi:

$$\begin{aligned}
 q_0 0 1 0 &\vdash x q_1 0 1 0 \vdash x 0 q_1 1 0 \vdash x q_2 0 y 0 \vdash q_2 x 0 y 0 \\
 &\vdash x x q_2 y 0 \vdash x x y q_1 0 \vdash x x y 0 q_1 B
 \end{aligned}$$

definiamo quindi il linguaggio accettato da una macchina di Turing:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

$$L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* \alpha p \beta \mid p \in F \mid \alpha, \beta \in \Gamma^*\}$$

La classe dei linguaggi accettati dalle MdT sono i ricorsivamente enumerabili (RE). Le MdT possono anche calcolare le funzioni:

**Esempio 97.**

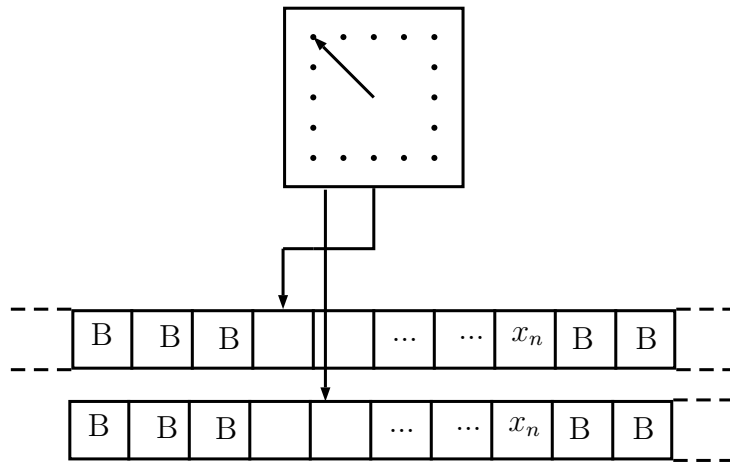
$$f(n, m) = m - n = \max(m - n, 0) = \begin{cases} m - n & m \geq n \\ 0 & m < n \end{cases}$$

quindi se l'input fosse  $0^m 10^n$  si avrebbe in output  $0^{m-n}$

si ha che se la MdT accetta, allora si ferma e che se la MdT non accetta, allora non si può dire se si ferma oppure no. Se non si ferma: linguaggi ricorsivamente enumerabili. Se si ferma sempre, otteniamo una sottoclasse che sono i linguaggi ricorsivi. Inoltre il problema dell'arresto, **halting problem**, è indecidibile e quindi i linguaggi ricorsivamente enumerabili sono quindi semidecidibili.

Si possono avere delle estensioni della macchina di Turing:

- macchina non deterministica; anziché una sequenza di ID avremmo un albero. Non si aggiunge comunque potenza di calcolo, perché posso simulare una nondet con una det che esplora questo albero
- MdT multinastro Hanno un numero finito di nastri. In una mossa guarda lo stato del controllo e il simbolo sotto ciascuna delle testine, si muove in un nuovo stato, scrive un nuovo simbolo per ogni nastro e si muove indipendentemente su ogni nastro:



si ha il seguente teorema:

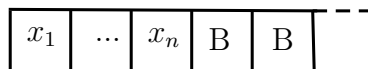
**Teorema 17.** *ogni linguaggio accettato da una MdT multinastro è Ricorsivamente Enumerabile*

Ora possiamo definire in maniera più precisa cosa si intendeva per simulazione di una macchina nondet: uso due nastri, il primo con l'input e il secondo gestito come una coda di ID da elaborare

**Teorema 18.** *Se  $M_n$  è una NTM (Nondeterministic Turing Machine) allora esiste una DTM (Deterministic Turing Machine) tale che il linguaggio accettato dalla NTM è uguale a quello accettato dalla DTM*

### 1.4.2 Restrizioni delle macchine di Turing

Consideriamo una MdT con nastro semiinfinito (ovvero è infinito solo da un lato):



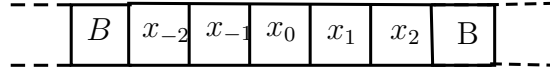
si ha il seguente teorema:

**Teorema 19.** *Ogni linguaggio accettato da una DTM  $M_2$  è anche accettato da una DTM  $M_1$  tale che:*

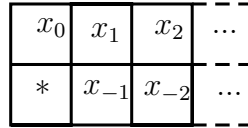
- *La testina di  $M_1$  non va mai a sinistra della posizione iniziale*
- *$M_1$  non scrive mai un Blank:*

$$B' \notin \Gamma \text{ e scrive } B' \text{ al posto di } B$$

Consideriamo ora una MdT tradizionale con il nastro:

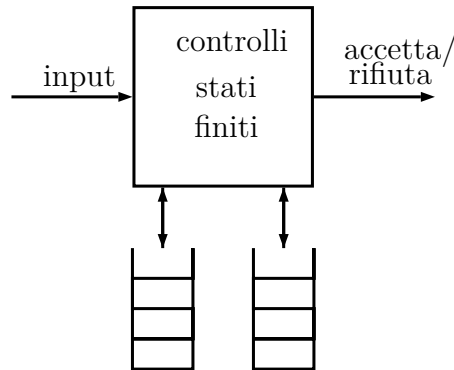


e con un nastro semiinfinito:



dove si ha l'alfabeto  $\Gamma \times (\Gamma \cup \{*\})$

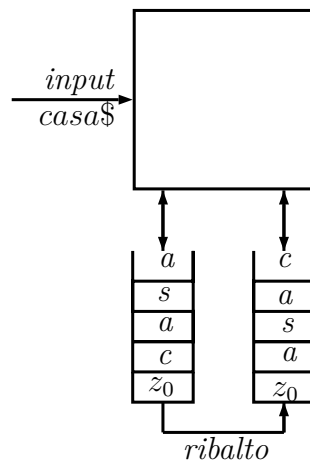
Macchine multi stack, che sono DPDA con un controllo a stati finiti e un numero finito di pile:



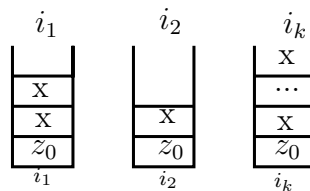
e si ha:

$$\delta(q, a, x_1, x_2, \dots, x_k) = (p, \gamma_1, \gamma_2, \dots, \gamma_k)$$

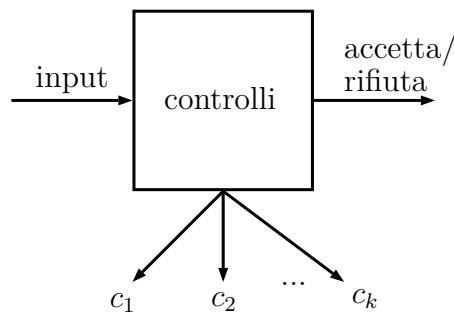
**Esempio 98.** *Legge l'input e lo spinge tutto sul primo stack (sfruttando un simbolo ausiliario di fine input):*



si ha un'ulteriore restrizione: Supponiamo che i simboli sulle pile possano essere solo  $x$  o  $z_0$ . Si riescono ancora a simulare le macchine di Turing



e il modello diventa:



dove possiamo modificare i contatori, che sono numeri naturali, a nostro piacimento "aggiungendo o togliendo X". Vediamo perché questo modello simula una MdT. Supponiamo che l'alfabeto del nastro contenga  $R-1$  simboli. Allora:

$$\begin{array}{|c|c|c|c|} \hline x_1 & x_2 & \dots & x_k \\ \hline \end{array} \longrightarrow x_1 + x_2 r + x_3 r^2 + \dots$$

Così come codifico il nastro posso codificare una pila. Per togliere  $x_1$  in cima alla pila devo dividere il numero per  $R$  e prendere il resto. Per aggiungere un nuovo simbolo, moltiplico per  $R$  e aggiungo il nuovo simbolo. Per modificare



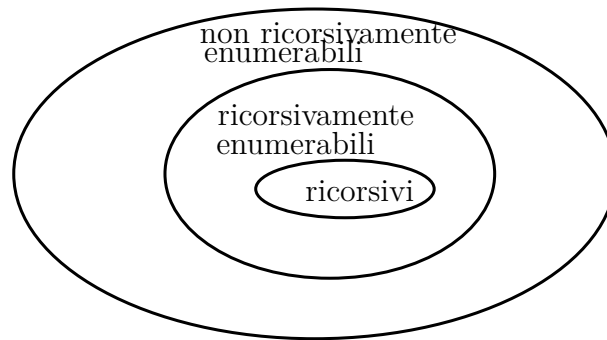
il simbolo, basta aggiungere la differenza tra il nuovo e la cima precedente. È immediato farlo con tre contatori (usandone uno di appoggio). Per fare tutto ciò bastano due contatori (e quindi due pile) dove nel primo è codificato come:

$$2^{c_1} 3^{c_2} 5^{c_3}$$

Per i linguaggi si ha quindi:

- linguaggi ricorsivi  $\rightarrow$  decidibili e MdT si ferma sempre
- linguaggi ricorsivamente enumerabili  $\rightarrow$  semidecidibili (ma comunque indecidibili) e MdT si ferma se accetta ma potrebbe non fermarsi
- linguaggi non ricorsivamente enumerabili  $\rightarrow$  indecidibili e  $\nexists$  MdT

quindi, insiemisticamente:



**Esempio 99.** vediamo un linguaggio non ricorsivamente enumerabile.

$w \in \{0,1\}^*$  biiezione tra stringhe e numeri:

$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	...
1	2	3	4	5	6	...
$\varepsilon$	0	1	00	01	10	..

Consideriamo però le due stringe 00101 e 0101: se le leggessi come numeri binari sono entrambi 5, quindi abbiamo perso la biiezione. Forziamo quindi l'interpretazione delle stringhe mettendoci davanti un 1. Codifichiamo anche una MdT in binario:

$$M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, \{q_2\})$$

$$Q = \{q_1, \dots, q_r\}$$

$$\Gamma = \{x_1, \dots, x_s\} = \{0, 1, B, \dots\}$$

$$\text{direzioni} : L \rightarrow D_1 \quad R \rightarrow D_2$$

$$\underbrace{\delta(q_i, x_j) = (q_k, x_1, D_m)}_{0^i 10^j 10^k 10^l 10^m}$$

$$Cod_1 11 Cod_2 11 Cod_3 \dots = Cpd(\delta)$$

**Esempio 100.** sia  $M = (\{q_1, q_2, q_3\}, \{o, 1\}, \{0, 1, B\}, \delta, q_1, B, q_2)$  con:

$$\begin{array}{cccccc} \overbrace{0}^{q_1} & 1 & \overbrace{00}^{x_2} & 1 & \overbrace{000}^{q_3} & 1 & \overbrace{0}^{x_1} & 1 & \overbrace{00}^{D_2} \end{array}$$

$$\begin{aligned} \delta(q_1, 1) &= (q_3, 0, R) \\ \delta(q_3, 0) &= (q_1, 1, R) \\ \delta(q_3, 1) &= (q_2, 0, R) \\ \delta(q_3, B) &= (q_3, 1, L) \end{aligned}$$

Quindi ora possiamo dire che  $M$  è  $M_i$ , ovvero l' $i$ -esima macchina di Turing. si ha però un problema: io posso permutare le delta, ottenendo numeri diversi, pur descrivendo la stessa macchina. Quindi la stessa macchina compare più volte nella sequenza infinita. Inoltre, data una stringa, non è detto che rappresenti una MdT. Inoltre:

$$Cod((M, W)) = Cod(M)111Cod(W)$$

definisco quindi  $L_d$

$$L_d = \{w_i \in \{0, 1\}^* \mid w_L \in L(M_i)\}$$

e considero la tabella infinita:

	1	2	3	$j$ 4	5	6
1	0	1	1	0	...	...
2	1	1	0	0	...	...
3	0	0	1	1	...	...
$(M_i) \ i$	4	0	1	0	1	...
5	:	:	:	:	...	...
6	:	:	:	:	...	...

Osserviamo la diagonale della tabella. Fanno parte di  $L_d$  solo le stringhe che hanno 0 sulla diagonale. Prendo in sostanza il complemento della diagonale. Se esistesse una MdT accettante, allora esisterebbe una riga uguale a tale complemento. Però tale complemento non è uguale a nessuna delle righe, perché differisce per l' $i$ -esimo valore dall' $i$ -esima riga (essendo il negato di tale valore!).

**Teorema 20.** *Se  $L$  è ricorsivo, allora il complementare di  $L$  è ricorsivo.*

*Dimostrazione.* Se  $L$  è ricorsivo, allora esiste una MdT  $M$  tale che  $L = L(M)$ . Costruiamo  $M'$  tale che  $L(M') = \text{complementare di } L$ .  $\square$

**Teorema 21.**

$$L, \bar{L} \in RE \rightarrow L, \bar{L} \in RIC$$

*se un linguaggio e il suo complementare sono ricorsivamente enumerabili allora quel linguaggio è ricorsivo*

*Dimostrazione.*

$$L \in RE \rightarrow \exists M_L \rightarrow L = L(M_L)$$

$$\widehat{\bar{L}} \in RE \rightarrow \exists M_{\bar{L}} \rightarrow \bar{L} = L(M_{\bar{L}})$$

$\square$

Poiché ogni stringa o appartiene a  $L$  o al complemento di  $L$ , costruiamo una nuova macchina  $M$  che simula  $M_L$  e  $M_{\bar{L}}$ .

Prima o poi una delle due macchine deve fermarsi. Allora possiamo accettare o rifiutare.

Si ha quindi la seguente tabella per i vari casi:

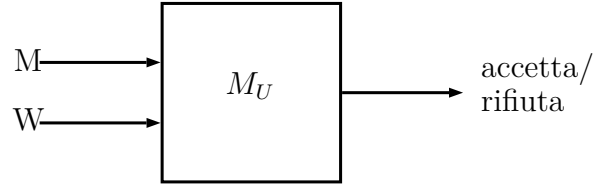
accettabile	Ric	RE	non RE
si	$L, \bar{L}$		
no	$L$	$\bar{L}$	
no	$L$		$\bar{L}$
no	$\bar{L}$	$L$	
no		$L, \bar{L}$	
si		$L$	$\bar{L}$
no	$\bar{L}$		$L$
si		$\bar{L}$	$L$
si			$L, \bar{L}$

### 1.4.3 Macchina di Turing Universale

Abbiamo già visto che si può codificare in binario una MdT, possiamo codificare anche la coppia MdT con il suo input:

$$Cod(M)111Cod(M) [M111W]$$

Possiamo quindi pensare a una MdT universale che sappia simulare qualsiasi altra MdT specificata come da codifica:

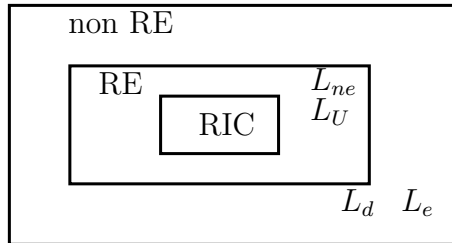


$$L_U\{(M, W) | W \in L(M)\}$$

La macchina universale corrisponde alla nostra idea di computer programmabile. Descriviamo questa macchina universale, che ha quattro nastri:

- **primo nastro:**  $M111W$
- **secondo nastro:** nastro/codifica di  $M$
- **terzo nastro:** stato/codifica di  $M$
- **quarto nastro:** ausiliario

Il linguaggio universale è ricorsivamente enumerabile ma non ricorsivo (non potrebbe esserlo, visto che la macchina che simula potrebbe non fermarsi). Il diagramma delle classi dei linguaggi è quindi diventato il seguente:

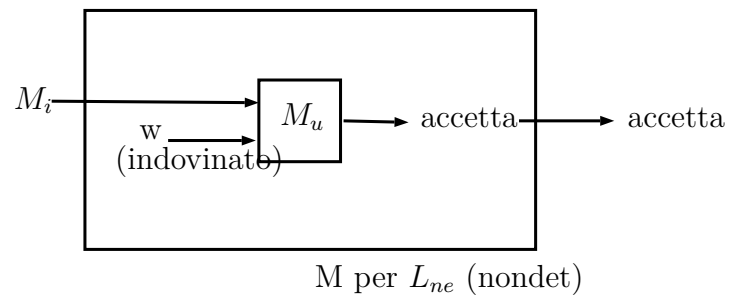


Valgono ancora le riduzioni. Sapendo che  $P_1$  è indecidibile (non RE o RE), posso fare una riduzione dalle istanze di  $P_1$  alle istanze di  $P_2$  mostrando che  $P_2$  è indecidibile. Con la riduzione stiamo dicendo che  $P_2$  è almeno difficile quanto  $P_1$  (e non viceversa!). Notiamo che non è nemmeno necessario che tutte le istanze di  $P_2$  siano "coperte" dal processo di riduzione.

**Teorema 22.**

$$L_{ne} \in RE$$

*Dimostrazione.* Dobbiamo fare una riduzione da  $L_u$  a  $L_{ne}$  per mostrare che  $L_{ne}$  è ricorsivamente enumerabile. La riduzione è descritta da questo schema:



□

**Teorema 23.**

$$L_{ne} \notin RIC$$

*Dimostrazione.* infatti essendo  $l_e$  il complementare di  $l_{ne}$ , per i teoremi visti in precedenza  $l_e$  non può essere ricorsivo né può essere ricorsivamente enumerabile. Segue che  $l_e$  è non RE □

**Esempio 101.** *considero:*

$$L_e = \{M \mid L(M) = \emptyset\}$$

$$L_{ne} = \{M \mid L(M) = \emptyset\}$$