

LINGUAGGI, GRAMMATICHE, AUTOMI

Corso di Fondamenti di Informatica - modulo I

Giorgio Gambosi

a.a. 2023-2024

Un insieme finito non vuoto Σ di simboli (detti **caratteri**) prende il nome di **alfabeto**.

Dato un alfabeto Σ , denotiamo come $\langle \Sigma^*, \circ, \varepsilon \rangle$ il monoide libero definito su Σ .

- ⊙ Σ^* chiuso rispetto a \circ : $\forall x, y \in \Sigma^* : x \circ y \in \Sigma^*$
- ⊙ \circ associativa: $\forall x, y, z \in \Sigma^* : (x \circ y) \circ z = x \circ (y \circ z)$
- ⊙ esiste un elemento neutro ε : $\forall x \in \Sigma^* : x \circ \varepsilon = \varepsilon \circ x = x$

Gli elementi di Σ^* vengono detti **parole** o **stringhe**: si noti che, mentre Σ è finito, Σ^* è infinito.

L'elemento ε viene detto **parola vuota**.

L'operazione $\circ : \Sigma^* \times \Sigma^* \mapsto \Sigma^*$ definita sul monoide è chiamata **concatenazione** e consiste nel giustapporre due parole di Σ^* :

$$x_{i_1} \dots x_{i_n} \circ y_{j_1} \dots y_{j_m} = x_{i_1} \dots x_{i_n} y_{j_1} \dots y_{j_m},$$

con $x_{i_1}, \dots, x_{i_n}, y_{j_1}, \dots, y_{j_m} \in \Sigma$.

La concatenazione di due stringhe x e y è frequentemente indicata omettendo il simbolo \circ , cioè scrivendo xy anziché $x \circ y$.

Concatenazione

Con la notazione $|x|$ indichiamo la **lunghezza** di una parola x , ovvero il numero di caratteri che la costituiscono. Chiaramente $|\varepsilon| = 0$.

La concatenazione non gode della proprietà commutativa e quindi in generale:

$$x \circ y \neq y \circ x.$$

Un caso particolare di concatenazione è quello in cui la stringa viene concatenata con sé stessa: con x^h si denota la concatenazione di x con sé stessa iterata h volte.

Per convenzione, si pone $x^0 = \varepsilon$.

Dato un alfabeto Σ , si definisce **linguaggio** un qualsivoglia sottoinsieme di Σ^* .

Poiché $\Sigma \subseteq \Sigma^*$, un alfabeto è a sua volta un linguaggio.

Si chiama **linguaggio vuoto**, e lo si indica con Λ , il linguaggio che non contiene stringa alcuna:
 $|\Lambda| = 0$.

Si noti che $\Lambda \neq \{\epsilon\}$.

L'**intersezione** di due linguaggi L_1 e L_2 è il linguaggio $L_1 \cap L_2$ costituito dalle parole di L_1 e di L_2

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

L'**unione** di due linguaggi L_1 e L_2 è il linguaggio $L_1 \cup L_2$ costituito dalle parole appartenenti ad almeno uno fra L_1 ed L_2

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

Si noti che $L_1 \cap \Lambda = \Lambda$ e $L_1 \cup \Lambda = L_1$.

Complementazione di linguaggi

Il **complemento** di un linguaggio L_1 è il linguaggio $\bar{L}_1 = \Sigma^* - L_1$ costituito dalle parole appartenenti a Σ^* ma non ad L_1

$$\bar{L}_1 = \{x \in \Sigma^* \mid x \notin L_1\}$$

Concatenazione di linguaggi

La **concatenazione** (o **prodotto**) di due linguaggi L_1 e L_2 è il linguaggio $L_1 \circ L_2$ delle parole costituite dalla concatenazione di una stringa di L_1 e di una stringa di L_2

$$L_1 \circ L_2 = \{x \in \Sigma^* \mid \exists y_1 \in L_1 \exists y_2 \in L_2 (x = y_1 \circ y_2)\}.$$

Si noti che $L \circ \{\varepsilon\} = \{\varepsilon\} \circ L = L$, e che $L \circ \Lambda = \Lambda \circ L = \Lambda$.

La **potenza** L^h di un linguaggio è definita come

$$L^h = L \circ L^{h-1}, h \geq 1$$

con la convenzione secondo cui $L^0 = \{\varepsilon\}$.

Si noti che, in base alla suddetta convenzione, $\Lambda^0 = \{\varepsilon\}$.

Chiusura di linguaggi

Il linguaggio L^* definito da

$$L^* = \bigcup_{h=0}^{\infty} L^h$$

prende il nome di **chiusura riflessiva del linguaggio** L rispetto all'operazione di concatenazione, mentre l'operatore “*” prende il nome di **iterazione** o **stella di Kleene**.

Dato un qualunque linguaggio L , $\varepsilon \in L^*$ e $\Lambda^* = \{\varepsilon\}$.

Si indica con L^+ la **chiusura (non riflessiva)** definita da

$$L^+ = \bigcup_{h=1}^{\infty} L^h$$

Risulta ovviamente $L^* = L^+ \cup \{\varepsilon\}$.

Espressioni regolari

Dato un alfabeto Σ e dato l'insieme di simboli

$$\{+, *, (,), \cdot, \emptyset\}$$

si definisce **espressione regolare** sull'alfabeto Σ una stringa

$$r \in (\Sigma \cup \{+, *, (,), \cdot, \emptyset\})^+$$

tale che valga una delle seguenti condizioni:

1. $r = \emptyset$
2. $r \in \Sigma \cup \{\varepsilon\}$
3. $r = (s + t)$, oppure $r = (s \cdot t)$, oppure $r = s^*$, dove s e t sono espressioni regolari sull'alfabeto Σ .

Espressioni regolari

Le espressioni regolari consentono di rappresentare linguaggi mediante una opportuna interpretazione dei simboli che le compongono. Nella tabella si mostra la corrispondenza tra un'espressione regolare r e il linguaggio $L(r)$ che essa rappresenta.

Espr. regolari	Linguaggi
\emptyset	Λ
a	$\{a\}$
$(s + t)$	$L(s) \cup L(t)$
$(s \cdot t)$	$L(s) \circ L(t)$
s^*	$(L(s))^*$

L'espressione regolare $(a + b)^*a$ rappresenta il linguaggio

$$\begin{aligned}L((a + b)^*a) &= L((a + b)^*) \circ L(a) \\ &= (L(a + b))^* \circ L(a) \\ &= (L(a) \cup L(b))^* \circ \{a\} \\ &= (\{a\} \cup \{b\})^* \circ \{a\} \\ &= \{a, b\}^* \circ \{a\} \\ &= \{x \mid x \in \{a, b\}^+, x \text{ termina con } a\}.\end{aligned}$$

Espressioni regolari: esempio

Sia $c = \{0, 1, \dots, 9\}$.

L'espressione regolare che rappresenta i numeri reali nella forma $c_{i1}c_{i2} \dots c_{in} \cdot c_{f1}c_{f2} \dots c_{fm}$ (dove c_{ij} rappresenta la j -esima cifra prima del punto decimale, e c_{fk} la k -esima cifra dopo il punto decimale) è

$$((1 + 2 + \dots + 9)(0 + 1 + \dots + 9)^* + 0).(0 + 1 + \dots + 9)^+$$

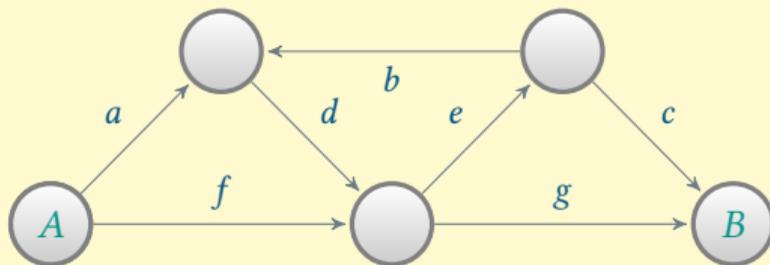
Espressioni regolari: problema

Determinare l'espressione regolare che, sull'alfabeto $\{a, b\}$, definisce l'insieme delle stringhe il cui terzultimo carattere è una b .

Determinare il linguaggio definito dall'espressione regolare $a^*((aa)^*b + (bb)^*a)b^*$.

Espressioni regolari: problema

Sia data la mappa stradale (con tratti stradali a senso unico e contrassegnati da caratteri dell'alfabeto) schematicamente indicata in figura. Fornire un'espressione regolare che definisca tutti i percorsi, anche passanti più volte per uno stesso nodo, tra A e B .



Equivalenza tra espressioni regolari

Due espressioni regolari r, s sono **equivalenti** ($r \equiv s$) se $L(r) = L(s)$.

Ad esempio, $a + b \equiv b + a$, $a + a \equiv a$, $aa^* \equiv a^*a$, $ab \not\equiv ba$.

Operatori

Assumiamo che \cdot abbia precedenza su $+$. Quindi $a + b \cdot c \equiv a + (b \cdot c)$.

Inoltre, rappresentiamo l'operatore \cdot con la concatenazione degli operandi: $ab \equiv a \cdot b$.

1. $+$ è commutativa ($r + s \equiv s + r$), associativa ($r + (s + t) \equiv (r + s) + t$), con elemento neutro \emptyset ($r + \emptyset \equiv r$), idempotente ($r + r \equiv r$)
2. \cdot è associativa ($r(st) \equiv (rs)t$), con elemento neutro ε ($r\varepsilon \equiv r$) e elemento nullo \emptyset ($r\emptyset \equiv \emptyset$)
3. \cdot si distribuisce su $+$ ($r(s + t) \equiv rs + rt$)
4. $+$ non si distribuisce su \cdot ($r + st \not\equiv (r + s)(r + t)$)

Alcune proprietà derivate di \cdot e \cdot

1. $\emptyset^* \equiv \varepsilon^* = \varepsilon$
2. $r^* \equiv r^*r^* \equiv (r^*)^* \equiv r + r^*$
3. $r^* \equiv \varepsilon + r^* \equiv \varepsilon + rr^* \equiv (\varepsilon + r)^* \equiv (\varepsilon + r)r^*$
4. $r^* \equiv (r + r^2 + \dots + r^k)^* \equiv \varepsilon + r + r^2 + \dots + r^{k-1} + r^k r^*$ per ogni $k \geq 1$
5. $r^*r \equiv rr^*$
6. $(r + s)^* \equiv (r^* + s^*)^* \equiv (r^*s^*)^* \equiv (r^*s)^*r^* \equiv r^*(sr^*)^*$
7. $r(sr)^* \equiv (rs)^*r$
8. $(r^*s)^* \equiv \varepsilon + (r + s)^*s$
9. $(rs^*)^* \equiv \varepsilon + r(r + s)^*$

Esempio di dimostrazione di equivalenza

Dimostrazione che $(a + aa)(a + b)^* \equiv a(a + b)^*$

$$\begin{aligned}(a + aa)(a + b)^* &\equiv (a + aa)a^*(ba^*)^* \text{ in quanto } (r + s)^* \equiv r^*(sr^*)^* \\ &\equiv a(\varepsilon + a)a^*(ba^*)^* \text{ in quanto } r \equiv r\varepsilon \\ &\equiv aa^*(ba^*)^* \text{ in quanto } (\varepsilon + r)r^* \equiv r^* \\ &\equiv a(a + b)^* \text{ in quanto } (r + s)^* \equiv r^*(sr^*)^*\end{aligned}$$

Esempio di dimostrazione di equivalenza

Dimostrazione che $a^*(b + ab^*) \equiv b + aa^*b^*$

$$\begin{aligned} a^*(b + ab^*) &\equiv (\varepsilon + aa^*)(b + ab^*) \text{ in quanto } r^* \equiv \varepsilon + rr^* \\ &\equiv b + ab^* + aa^*b + aa^*ab^* \text{ per distributività} \\ &\equiv b + (ab^* + aa^*ab^*) + aa^*b \text{ per associatività e commutatività di } + \\ &\equiv b + (\varepsilon + aa^*)ab^* + aa^*b \text{ in quanto } r \equiv r\varepsilon \\ &\equiv b + a^*ab^* + aa^*b \text{ in quanto } r^* \equiv \varepsilon + rr^* \\ &\equiv b + aa^*b^* + aa^*b \text{ in quanto } r^*r \equiv rr^* \\ &\equiv b + aa^*(b^* + b) \text{ per distributività} \\ &\equiv b + aa^*b^* \text{ in quanto } r^* \equiv r^* + r \end{aligned}$$

Esempio di dimostrazione di non equivalenza

Dimostrazione che $(a + b)^* \not\equiv a^* + b^*$.

Basta osservare che $ab \in L((a + b)^*) - L(a^* + b^*)$

Esempio di dimostrazione di non equivalenza

Dimostrazione che $(a + b)^* \not\equiv a^*b^*$.

Basta osservare che $ba \in L((a + b)^*) - L(a^*b^*)$

Esempio di semplificazione

Semplificazione dell'espressione regolare $aa(b^* + a) + a(ab^* + aa)$:

$$\begin{aligned}aa(b^* + a) + a(ab^* + aa) &\equiv aa(b^* + a) + aa(b^* + a) \text{ distributività} \\ &\equiv aa(b^* + a) \text{ in quanto } r + r \equiv r\end{aligned}$$

Una **grammatica formale** G è una quadrupla $G = \langle V_T, V_N, P, S \rangle$ in cui:

1. V_T è un insieme finito e non vuoto di simboli **terminali**
2. V_N è un insieme finito e non vuoto di simboli **non terminali**
3. P è una relazione binaria di cardinalità finita su

$$(V_T \cup V_N)^* \circ V_N \circ (V_T \cup V_N)^* \times (V_T \cup V_N)^*.$$

P è detta insieme delle **produzioni**.

Una coppia $\langle \alpha, \beta \rangle \in P$, si indica generalmente con la notazione $\alpha \longrightarrow \beta$;

4. $S \in V_N$ è detto **assioma**

Si consideri la grammatica $G = \langle \{a, b\}, \{S, B, C\}, P, S \rangle$, avente le seguenti regole di produzione

1. $S \rightarrow aS$
2. $S \rightarrow B$
3. $B \rightarrow bB$
4. $B \rightarrow bC$
5. $C \rightarrow cC$
6. $C \rightarrow c.$

Con questa grammatica si possono generare le stringhe del linguaggio

$$L(G) = \{a^n b^m c^h \mid n \geq 0, m, h \geq 1\}.$$

Un insieme di produzioni aventi stessa parte sinistra

$$\begin{aligned}\alpha &\longrightarrow \beta_1 \\ \alpha &\longrightarrow \beta_2 \\ &\dots \\ \alpha &\longrightarrow \beta_n,\end{aligned}$$

viene convenzionalmente indicato, in maniera più compatta, come

$$\alpha \longrightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n.$$

Inoltre, l'unione $V_T \cup V_N$ viene indicata con V .

Una regola del tipo $\alpha \rightarrow \varepsilon$, dove $\alpha \in V^* \circ V_N \circ V^*$, prende il nome di ε -produzione o ε -regola.

Derivazioni dirette

Data una grammatica $G = \langle V_T, V_N, P, S \rangle$, la **derivazione diretta** (rispetto a G) è una relazione su $(V^* \circ V_N \circ V^*) \times V^*$ così definita:

la coppia $\langle \phi, \psi \rangle$ appartiene alla relazione se e solo se esistono $\alpha \in V^* \circ V_N \circ V^*$ e $\beta, \gamma, \delta \in V^*$ tali che

$$\phi = \gamma\alpha\delta$$

$$\psi = \gamma\beta\delta$$

$$\alpha \rightarrow \beta \in P$$

In questo caso, scriviamo $\phi \xRightarrow{G} \psi$

Data una grammatica \mathcal{G} , una **derivazione** (in G) è una sequenza di stringhe $\phi_1, \dots, \phi_n \in V^*$ tali che

$$\forall i \in \{1, \dots, n-1\} : \phi_i \xrightarrow[G]{} \phi_{i+1}$$

La relazione di *derivabilità* (rispetto a G) è la chiusura transitiva e riflessiva della derivazione diretta: essa si rappresenta con la notazione $\xrightarrow[G]^*$.

Scrivendo $\phi \xrightarrow[G]^* \psi$ indichiamo l'esistenza di (almeno) una derivazione da ϕ a ψ .

Data una grammatica \mathcal{G} , si definisce **forma di frase** (in G) una qualunque stringa $\phi \in V^*$ tale che $S \xrightarrow[G]{*} \phi$.

Il linguaggio generato da una grammatica G è l'insieme $L(G) \subseteq \Sigma^*$ tale che

$$L(G) = \left\{ x \mid x \in V_T^* \wedge S \xrightarrow[G]{*} x \right\}.$$

$L(G)$ è l'insieme delle stringhe di caratteri terminali che si possono ottenere a partire dall'assioma mediante l'applicazione di un numero finito di passi di derivazione diretta.

Esempio

Il linguaggio

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

può essere generato dalla grammatica G in cui $V_T = \{a, b, c\}$ e $V_N = \{S, B, C, F, G\}$ e le regole di P sono le seguenti:

$$\begin{aligned} S &\longrightarrow aSBC \\ CB &\longrightarrow BC \\ SB &\longrightarrow bF \\ FB &\longrightarrow bF \\ FC &\longrightarrow cG \\ GC &\longrightarrow cG \\ G &\longrightarrow \varepsilon. \end{aligned}$$

Esempio

La stringa *aabbcc* si può generare con la derivazione:

$$\begin{aligned} S &\Rightarrow aSBC \\ &\Rightarrow aaSBCBC \\ &\Rightarrow aaSBBCC \\ &\Rightarrow aabFBCC \\ &\Rightarrow aabbFCC \\ &\Rightarrow aabbcGC \\ &\Rightarrow aabbccG \\ &\Rightarrow aabbcc. \end{aligned}$$

Quindi, $S \xRightarrow{*} aabbcc$, ed in particolare che $S \xRightarrow{i} aabbcc$ per ogni $i \geq 8$.

Come dimostrare che $L(G) = \{a^n b^n c^n \mid n \geq 1\}$?

- ⊙ Dimostrare che ogni x del tipo $a^n b^n c^n$ è derivabile in G
- ⊙ Dimostrare che ogni $z \in V_T^* = \{a, b, c\}^*$ derivabile in G ha la forma $a^n b^n c^n$

Esempio

La grammatica $G = \langle \{a, b\}, \{S, A\}, P, S \rangle$ con produzioni P

$$S \rightarrow Ab$$

$$A \rightarrow Sa$$

genera il linguaggio vuoto Λ .

Esempio

Dimostriamo che la grammatica $G = \langle \{a, b, c\}, \{S, A\}, P, S \rangle$ con produzioni P

$$\begin{aligned} S &\longrightarrow aSc \mid A \\ A &\longrightarrow bAc \mid \varepsilon, \end{aligned}$$

genera il linguaggio

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

Esempio

Proviamo che $\mathcal{L}(\mathcal{G}) \subseteq L$.

Chiaramente, tutte le forme di frase costruite da \mathcal{G} sono del tipo

⊙ $a^k S c^k$, $k \geq 0$, oppure

⊙ $a^k b^j A c^{k+j}$, $k \geq 0$ e $j \geq 0$.

Quindi, ogni parola z generata da \mathcal{G} è ottenuta tramite una derivazione del tipo

$$S \xrightarrow{k} a^k S c^k \xrightarrow{1} a^k A c^k \xrightarrow{j} a^k b^j A c^{k+j} \xrightarrow{1} a^k b^j c^{k+j} = z.$$

Ogni parola derivata dunque appartiene a L .

Esempio

Proviamo ora che $L \subseteq \mathcal{L}(\mathcal{G})$.

Basta osservare che ogni stringa $z \in L$ è del tipo $a^n b^m c^{n+m}$ e che essa viene generata da \mathcal{G} attraverso la derivazione

$$S \xrightarrow{m} a^m S c^m \xrightarrow{1} a^m A c^m \xrightarrow{n} a^m b^n A c^{m+n} \xrightarrow{1} a^m b^n c^{m+n} = z.$$

Esiste dunque in \mathcal{G} una derivazione che costruisce z .

Data la grammatica $G = \langle \{a\}, \{S, I, F, M\}, P, S \rangle$ con produzioni P

$$\begin{aligned} S &\longrightarrow a \mid aa \mid IaF \\ aF &\longrightarrow Maa \mid MaaF \\ aM &\longrightarrow Maa \\ IM &\longrightarrow Ia \mid aa \end{aligned}$$

dimostrare che G genera il linguaggio $\{a^{2^n} \mid n \geq 0\}$.

Esempi di derivazione

$S \Rightarrow a$

$S \Rightarrow aa$

$S \Rightarrow IaF \Rightarrow IMaa \Rightarrow aaaa$

$S \Rightarrow IaF \Rightarrow IMaaF \Rightarrow IaaaF \Rightarrow IaaMaa \Rightarrow IaMaaaa \Rightarrow IMaaaaaa \Rightarrow aaaaaaaaa$

Definire una grammatica che generi il linguaggio

$$L = \{a^n b^m c^p \mid n, p \geq 0, m \geq 1; n = m \vee m = p\}$$

e dimostrare la sua correttezza.

Due grammatiche G_1 e G_2 si dicono **equivalenti** se generano lo stesso linguaggio, vale a dire se $L(G_1) = L(G_2)$.

Dimostrare che la grammatica con produzioni

$$S \longrightarrow aS \mid b$$

e la grammatica con produzioni

$$\begin{aligned} S &\longrightarrow b \mid Ab \\ A &\longrightarrow Aa \mid a \end{aligned}$$

sono equivalenti.

Grammatiche di tipo 0

Dette anche **non limitate**, definiscono la classe di linguaggi più ampia possibile.

In esse le produzioni sono del tipo più generale:

$$\alpha \rightarrow \beta, \alpha \in V^* \circ V_N \circ V^*, \beta \in V^*.$$

Queste grammatiche ammettono anche derivazioni che “accorciano” le forme di frase, come ad esempio quelle che si ottengono applicando le ε -produzioni.

I linguaggi generabili da grammatiche di tipo 0 si dicono **linguaggi di tipo 0**.

Esempio

La grammatica

$$G = \langle \{a, b\}, \{S, A\}, P, S \rangle,$$

in cui P è

$$S \rightarrow aAb$$

$$aA \rightarrow aaAb$$

$$A \rightarrow \varepsilon,$$

è di tipo 0 e genera il linguaggio $L = \{a^n b^n \mid n \geq 1\}$.

Dette anche **contestuali** o **context sensitive** (CS), ammettono qualunque regola di produzione che non riduca la lunghezza delle stringhe, cioè tutte le produzioni del tipo:

$$\alpha \rightarrow \gamma, \alpha \in V^* \circ V_N \circ V^*, \gamma \in V^+, |\alpha| \leq |\gamma|.$$

I linguaggi generabili da grammatiche di tipo 1 si dicono **linguaggi di tipo 1**, **contestuali**, o **context sensitive** (CS).

Le produzioni

$$\begin{aligned} S &\longrightarrow aSa \mid aAb \mid aAa \\ aA &\longrightarrow aa \\ Ab &\longrightarrow aab \end{aligned}$$

appartengono ad una grammatica di tipo 1.

Il linguaggio $\{a^n b^n \mid n \geq 1\}$ può venir generato da una grammatica di tipo 1 avente le produzioni

$$S \longrightarrow aBS \mid ab$$

$$Ba \longrightarrow aB$$

$$Bb \longrightarrow bb.$$

Dunque il linguaggio $\{a^n b^n \mid n \geq 1\}$ è contestuale.

Definizione alternativa

Il termine “linguaggio contestuale” deriva dal fatto che sono generabili da grammatiche aventi produzioni “contestuali” del tipo

$$\beta_1 A \beta_2 \longrightarrow \beta_1 \gamma \beta_2, \quad A \in V_N, \quad \beta_1, \beta_2 \in V^*, \quad \gamma \in V^+,$$

in cui si esprime il fatto che A può essere sostituito da γ in una forma di frase solo se si trova nel contesto $\langle \beta_1, \beta_2 \rangle$.

Dette anche **non contestuali** o **context free** (CF), ammettono solo produzioni del tipo:

$$A \rightarrow \beta, \quad A \in V_N, \quad \beta \in V^+$$

cioè produzioni in cui ogni non terminale A può essere riscritto in una stringa β indipendentemente dal contesto in cui esso si trova.

Grammatica che genera espressioni aritmetiche di somme e moltiplicazioni in una variabile i :

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow i \mid (E).$$

Generare la stringa $(i + i) * i$ utilizzando la grammatica dell'esempio precedente.

Il linguaggio $\{a^n b^n \mid n \geq 1\}$ può essere generato dalla grammatica di tipo 2 con produzioni

$$S \rightarrow aSb \mid ab.$$

Dunque, il linguaggio $\{a^n b^n \mid n \geq 1\}$ è context free.

Il linguaggio delle parentesi ben bilanciate è di tipo 2. Esso può essere generato dalla grammatica $G = \langle \{(,)\}, \{S\}, P, S \rangle$, dove P è l'insieme delle produzioni

$$S \rightarrow () \mid SS \mid (S).$$

Definire una grammatica non contestuale per generare tutte le stringhe palindrome, cioè quelle che risultano uguali se lette da destra verso sinistra o da sinistra verso destra.

Grammatiche di tipo 3

Dette anche **lineari destre** o **regolari**, ammettono solo produzioni del tipo:

$$A \rightarrow \delta, \quad A \in V_N, \quad \delta \in (V_T \circ V_N) \cup V_T.$$

Il termine “regolare” deriva dal fatto che i corrispondenti linguaggi sono rappresentabili per mezzo di espressioni regolari.

I linguaggi generabili da grammatiche di tipo 3 vengono detti **linguaggi di tipo 3** o **regolari**.

Esempio

La grammatica

$$G = \langle \{a, b\}, \{S\}, P, S \rangle$$

in cui P contiene le produzioni

$$S \rightarrow aS \mid b$$

è una grammatica di tipo 3 e genera il linguaggio regolare $L = \{a^n b \mid n \geq 0\}$.

Definire una grammatica regolare per il linguaggio delle stringhe sull'alfabeto $\{a, b\}$ che contengono un numero dispari di a .

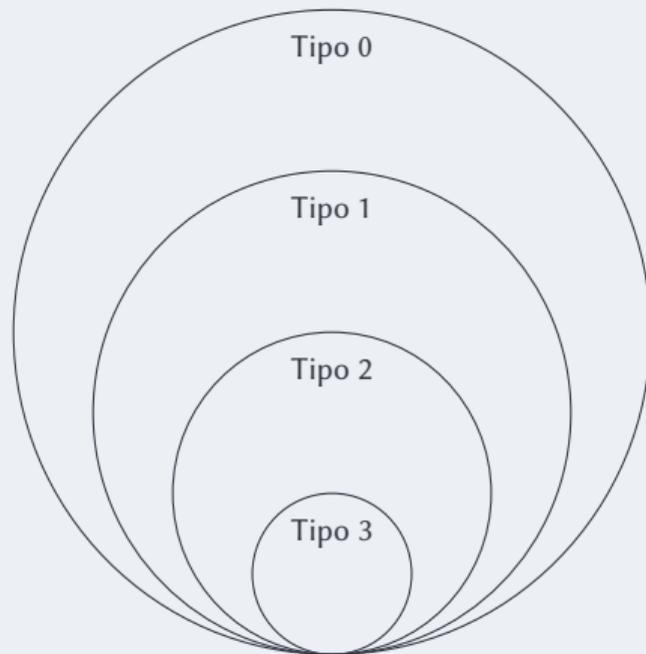
I linguaggi regolari si possono definire anche mediante grammatiche **lineari sinistre** caratterizzate da regole del tipo:

$$A \rightarrow \delta, \quad A \in V_N, \quad \delta \in (V_N \circ V_T) \cup V_T.$$

Le grammatiche lineari sinistre differiscono dalle grammatiche lineari destre in quanto l'unico simbolo non terminale che può comparire nella parte destra δ della produzione appare come primo simbolo.

Gerarchia di Chomsky

Per $0 \leq n \leq 2$, ogni grammatica di tipo $n + 1$ è anche di tipo n : pertanto l'insieme dei linguaggi di tipo n contiene tutti i linguaggi di tipo $n + 1$, formando quindi una gerarchia, detta **Gerarchia di Chomsky**



Un linguaggio L viene detto **strettamente** di tipo n se esiste una grammatica G di tipo n che genera L e non esiste alcuna grammatica G' di tipo $m > n$ che possa generarlo.

TIPO	PRODUZIONI	DOVE
TIPO 0	$\alpha \rightarrow \beta$	$\alpha \in V^* \circ V_N \circ V^*, \beta \in V^*$
TIPO 1	$\alpha \rightarrow \beta$	$ \alpha \leq \beta ,$ $\alpha \in V^* \circ V_N \circ V^*, \beta \in V^+$
TIPO 2	$A \rightarrow \beta$	$A \in V_N, \beta \in V^+$
TIPO 3	$A \rightarrow \delta$	$A \in V_N, \delta \in (V_T \cup (V_T \circ V_N))$

Dalle definizioni, non è possibile generare la stringa vuota con grammatiche di tipo 1, 2 o 3.

Linguaggi contenenti ε possono però essere generati apportando lievi modifiche (aggiunta di opportune ε -produzioni) alle grammatiche non contestuali e regolari.

Se una grammatica $G = \langle V_T, V_N, P, S \rangle$ di tipo 1, 2 o 3 genera un linguaggio L , per poter generare il linguaggio $L \cup \{\varepsilon\}$ è sufficiente utilizzare la grammatica

$$G' = \langle V_T, V_N \cup \{S'\}, P', S' \rangle,$$

dove

$$P' = P \cup \{S' \rightarrow \varepsilon\} \cup \{S' \rightarrow S\}.$$

Grammatiche con ε -produzioni

Data la grammatica CF G , con assioma S e produzioni:

$$S \longrightarrow aBSc \mid abc$$

$$Ba \longrightarrow aB$$

$$Bb \longrightarrow bb.$$

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

Il linguaggio $L(G) \cup \{\varepsilon\} = \{a^n b^n c^n \mid n \geq 0\}$ è derivato dalla la grammatica G' con assioma S' e produzioni :

$$S' \longrightarrow S \mid \varepsilon$$

$$S \longrightarrow aBSc \mid abc$$

$$Ba \longrightarrow aB$$

$$Bb \longrightarrow bb$$

Grammatiche con ε -produzioni

Aggiungere la ε -produzione direttamente come produzione dell'assioma può avere effetti indesiderati

$$\begin{aligned} S &\longrightarrow Ub \\ U &\longrightarrow ab \mid S \end{aligned}$$

genera ab^*bb . Ma

$$\begin{aligned} S &\longrightarrow Ub \mid \varepsilon \\ U &\longrightarrow ab \mid S \end{aligned}$$

genera $ab^*bb \cup \{\varepsilon\} \cup b^*b$

L'aggiunta non controllata di ε -produzioni può aumentare in modo sostanziale il potere generativo della grammatica.

Teorema

Data una grammatica $G = \langle V_T, V_N, P, S \rangle$ di tipo 0, esiste una grammatica G' equivalente a G , ottenuta estendendo una grammatica di tipo 1 con opportune ε -produzioni.

Grammatiche di tipo 1 e ε -produzioni

La grammatica $G' = \langle V'_T, V'_N, P', S' \rangle$ è caratterizzata da: $V'_T = V_T$, $V'_N = V_N \cup \{X\}$, con $X \notin V_N$, $S' = S$ e P' ottenuto da P aggiungendo la produzione $X \rightarrow \varepsilon$ e sostituendo ad ogni produzione $\phi \rightarrow \psi$ con $|\phi| > |\psi| > 0$ la produzione

$$\phi \rightarrow \psi X \dots X$$

$\boxed{\{z\}}$
 $\cancel{\{z\}}$

È semplice verificare che con la grammatica \mathcal{G}' sono derivabili tutte e sole le stringhe di V_T^* derivabili con la grammatica \mathcal{G} .

L'aggiunta indiscriminata di ε -produzioni non altera il potere generativo delle grammatiche.

Si può dimostrare che data una grammatica di tipo 2 o 3 estesa con ε -produzioni, ne possiamo sempre costruire una equivalente, dello stesso tipo, che usa ε -produzioni solo a partire dall'assioma (nel caso che ε appartenga al linguaggio da generare) o non ne usa affatto (in caso contrario).

Teorema

Data una grammatica $G = \langle V_T, V_N, P, S \rangle$ il cui insieme di produzioni P comprende soltanto produzioni di tipo non contestuale e produzioni vuote, esiste una grammatica non contestuale G' tale che $L(G') = L(G) - \{\varepsilon\}$

Derivazione di G' da G .

1. determinazione dei simboli che si annullano, cioè i non terminali da cui è possibile derivare ε
2. per ogni produzione $A \rightarrow \alpha$ di P , con l'esclusione delle ε -produzioni, se nessun simbolo di α si annulla $A \rightarrow \alpha$ va in P' ; altrimenti a P' si aggiungono tutte le possibili produzioni ottenute da $A \rightarrow \alpha$ eliminando da α un sottoinsieme dei simboli che si annullano, considerati con la propria molteplicità.

Grammatiche di tipo 2 e 3 e ε -produzioni

Per provare che $L(G') = L(G) - \{\varepsilon\}$ basta mostrare, per induzione sulla lunghezza della derivazione, che $\forall A \in V_N$ e $\forall w \in V_T^+$:

$$A \xrightarrow[G']{*} w \text{ se e solo se } (w \neq \varepsilon \wedge A \xrightarrow[G]{*} w).$$

Nel caso particolare in cui la grammatica è regolare è possibile dimostrare un risultato analogo.

Le grammatiche formali sono utilizzate per la definizione di linguaggi di programmazione, caratterizzati come l'insieme di tutte le stringhe (programmi) derivabili dalla grammatica.

Per la definizione sintattica dei linguaggi di programmazione vengono adottate grammatiche non contestuali, tradizionalmente rappresentate mediante una notazione specifica, particolarmente suggestiva, denominata **Forma Normale di Backus** (**Backus Normal Form**, BNF, detta anche **Backus-Naur Form**).

Notazione per grammatiche context free resa più espressiva e succinta:

1. I simboli non terminali costituiti da stringhe (ad es.: $\langle \text{espressione} \rangle$);
2. \rightarrow sostituito da $::=$
3. Parentesi graffe $\{\dots\}$ impiegate per indicare l'iterazione illimitata. Con $\{\dots\}^n$ si indica l'iterazione per un numero di volte pari al più ad n .
4. Parentesi quadre $[\dots]$ utilizzate per indicare l'opzionalità (possibile assenza di una parte di stringa)
5. Parentesi tonde (\dots) utilizzate per indicare la fattorizzazione, vale a dire la messa in comune di una sottoespressione

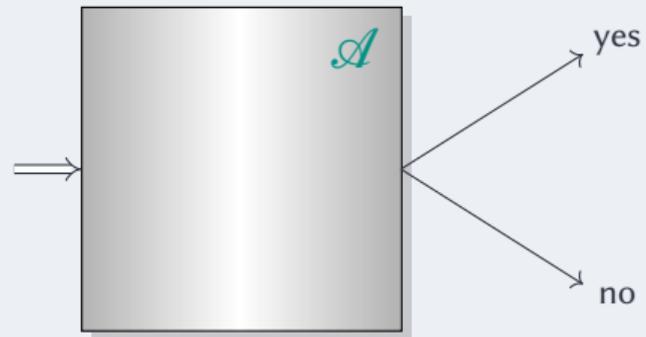
Identificatore:

$$\begin{aligned} \langle \text{id} \rangle &::= \langle \text{alfa} \rangle \{ \langle \text{alfanum} \rangle \}^{n-1} \\ \langle \text{alfanum} \rangle &::= \langle \text{alfa} \rangle \mid \langle \text{cifra} \rangle \\ \langle \text{alfa} \rangle &::= \mathbf{A} \mid \mathbf{B} \mid \dots \mid \mathbf{Z} \mid \mathbf{a} \mid \mathbf{b} \mid \dots \mid \mathbf{z} \\ \langle \text{cifra} \rangle &::= \mathbf{0} \mid \mathbf{1} \mid \dots \mid \mathbf{9} \end{aligned}$$

Dispositivo astratto che, data un stringa x fornitagli in input, esegue una **computazione**

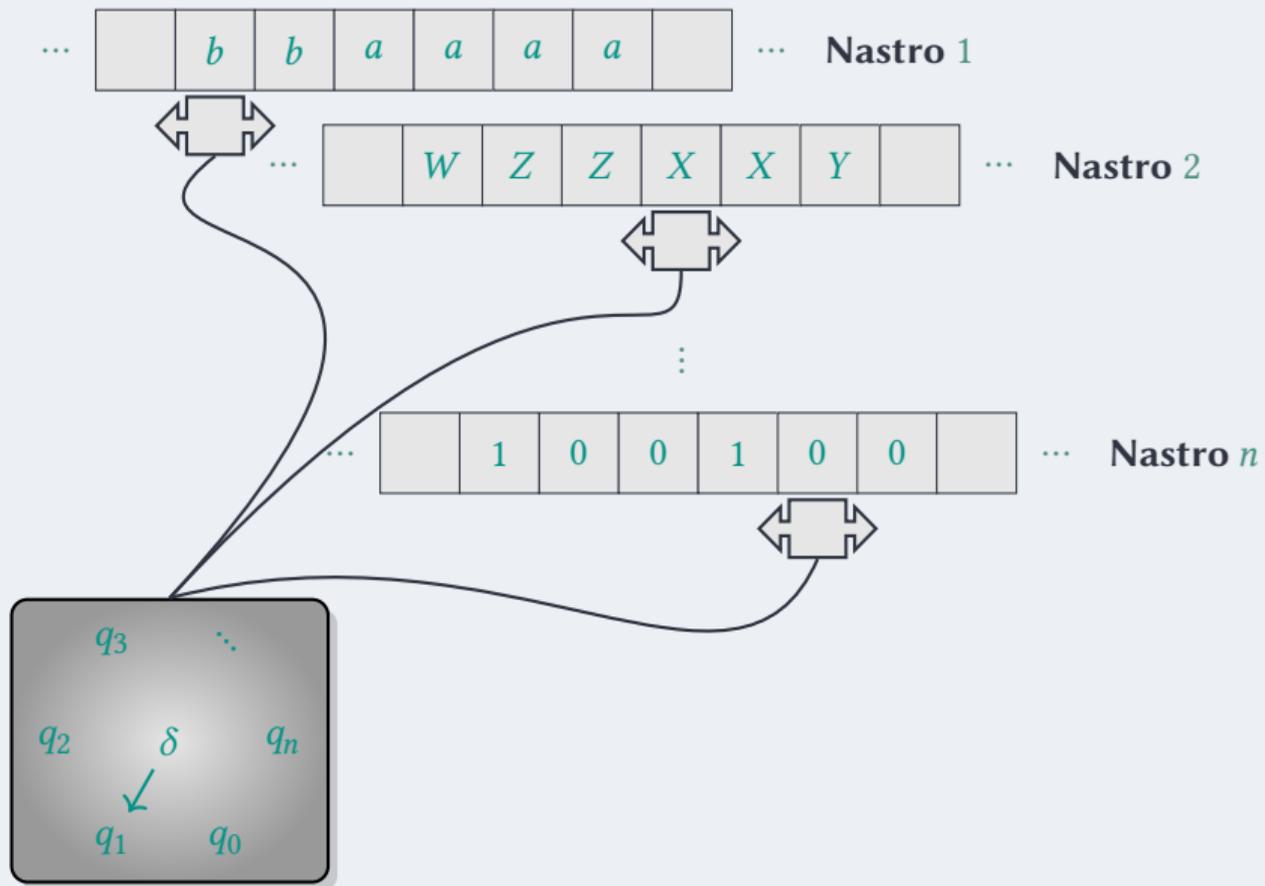
Se la computazione termina, restituisce, secondo una qualche modalità, un valore

Nel caso del problema del riconoscimento, il valore restituito è booleano



- ⊙ dispositivo interno, che ad ogni istante assume uno **stato** in un possibile insieme finito predefinito
- ⊙ uno o più dispositivi di memoria (nastri), sui quali è possibile memorizzare delle informazioni, sotto forma di stringhe di caratteri da alfabeti predefiniti
- ⊙ nastri costituiti da celle; ogni cella può contenere un carattere
- ⊙ caratteri letti o scritti per mezzo di **testine** che possono muoversi lungo i nastri, posizionandosi sulle diverse celle

Automa



Insieme delle informazioni necessarie e sufficienti per determinare, in un certo istante, il comportamento futuro dell'automa.

1. stato interno dell'automa;
2. contenuto di tutti i nastri di memoria;
3. posizione di tutte le testine sui nastri.

Configurazione in cui si assume si trovi inizialmente un automa, in presenza di una stringa in input.

Ad esempio:

1. stato predefinito come iniziale;
2. nastri di memoria vuoti eccetto il nastro di input, contenente la stringa;
3. testina del nastro di input sulla cella contenente il primo carattere della stringa

Funzione di transizione

Induce una **relazione di transizione** tra configurazioni, che associa ad una configurazione un'altra (o più di una) **configurazione successiva**.

Definita non sull'insieme delle possibili configurazioni, ma su domini e codomini che rappresentano parti di configurazioni, quelle che effettivamente determinano e sono determinate dalla transizione.

L'applicazione della funzione di transizione ad una configurazione si dice **transizione** o **mossa** o **passo computazionale** dell'automa.

Dato un automa A e due sue configurazioni c_i, c_j , la notazione

$$c_i \xrightarrow[A]{} c_j$$

indica che c_j deriva da c_i per effetto dell'applicazione della funzione di transizione di A .

Configurazioni di accettazione

Sottoinsieme delle possibili configurazioni: determinano, se raggiunte, l'accettazione della stringa in input da parte dell'automa.

Tutte le altre configurazioni sono definite come **configurazioni di non accettazione**, o di **rifiuto**.

Un automa esegue una computazione applicando iterativamente, ad ogni istante, la propria funzione di transizione alla configurazione attuale, a partire dalla configurazione iniziale.

Sequenza di configurazioni attraversate

$$c_0, c_1, c_2, \dots$$

tale che $c_i \xrightarrow[A]{} c_{i+1}$ per $i = 0, 1, \dots$

$\overset{*}{\vdash}_A$: chiusura transitiva e riflessiva della relazione \vdash_A .

Date due configurazioni c_i, c_j di A ,

$$c_i \overset{*}{\vdash}_A c_j$$

se e solo se esiste una computazione che porta A da c_i a c_j .

Se

1. $c_0, c_1, c_2, \dots, c_n$ ha lunghezza finita
2. non esiste nessuna configurazione c tale che $c_n \xrightarrow[A]{}$ c

la computazione **termina**.

- ⊙ c_n esiste ed è una configurazione di accettazione, l'automa accetta la stringa in input
- ⊙ c_n esiste ed non è una configurazione di accettazione, l'automa rifiuta la stringa in input
- ⊙ c_n non esiste, la computazione non termina

Automati deterministici

Ad ogni stringa di input associa una sola computazione, e quindi una singola sequenza di configurazioni.



Un automa deterministico, data una stringa di input, può eseguire una sola computazione: se la computazione termina in una configurazione di accettazione, allora la stringa viene accettata.

Stringhe e linguaggio accettato

1. Automa deterministico A
2. Stringa x in input ad A
3. $c_0(x)$ configurazione iniziale di A corrispondente alla stringa x

A accetta x se e solo se esiste una configurazione di accettazione c di A per la quale

$$c_0(x) \xrightarrow[A]{*} c$$

Il linguaggio **accettato** da A è l'insieme $L(A)$ di tutte le stringhe x accettate da A .

Se la computazione eseguita dall'automa A termina per ogni stringa, e quindi se ogni stringa viene o accettata o rifiutata, allora diciamo che il linguaggio $L(A)$ è **riconosciuto** (o **deciso**) da A

Automa non deterministico

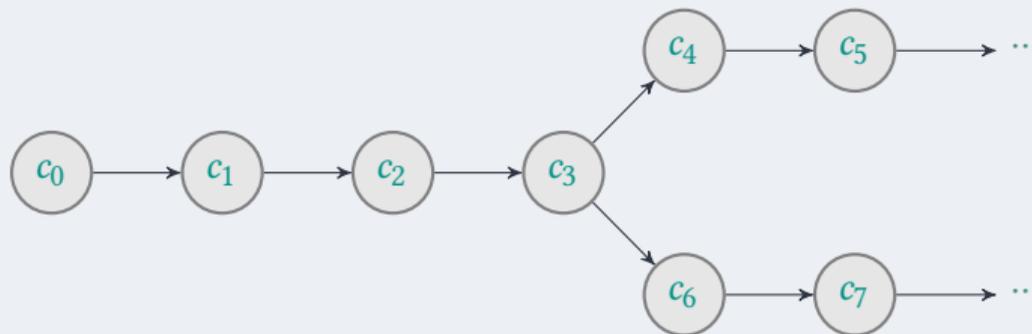
Generalizzazione dell'automa deterministico.

Associa ad ogni stringa di input un numero qualunque, in generale maggiore di uno, di computazioni.

La funzione di transizione associa ad almeno una configurazione c più di una configurazione successiva.

Per ogni computazione che conduce a c sono definite continuazioni diverse, che definiscono diverse computazioni.

Automa non deterministico

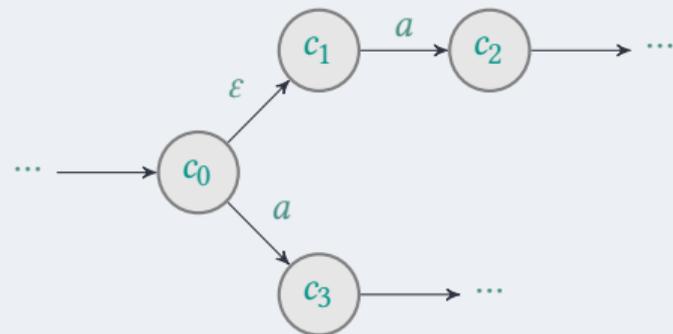


Il **grado di non determinismo** di un automa è il massimo numero di configurazioni che la funzione di transizione associa ad una configurazione.

Transizioni che un automa può eseguire senza leggere alcun carattere in input.

La presenza di ϵ -transizioni introduce non determinismo.

Se da una configurazione c un automa può passare o meno in un'altra configurazione c' senza leggere caratteri in input, sono possibili due diverse continuazioni della computazione attuale, l'una da c e l'altra da c' .



⊙ Computazione 1: $\dots c_0 c_1 c_2 \dots$

⊙ Computazione 2: $\dots c_0 c_3 \dots$

Computazioni non deterministiche

Automa deterministico: associa alla stringa di input una sequenza di configurazioni (computazione lineare), con la prima uguale alla configurazione iniziale della stringa

Automa non deterministico: associa alla stringa di input un albero di configurazioni (**albero di computazione**), con la radice uguale alla configurazione iniziale della stringa. Ogni computazione corrisponde ad un cammino avente origine dalla radice stessa.

Computazioni non deterministiche: interpretazione 1

L'automa non deterministico può eseguire una qualunque computazione, associata ad una sequenza di “scelte” su quale transizione applicare in corrispondenza ad ogni applicazione della funzione di transizione.

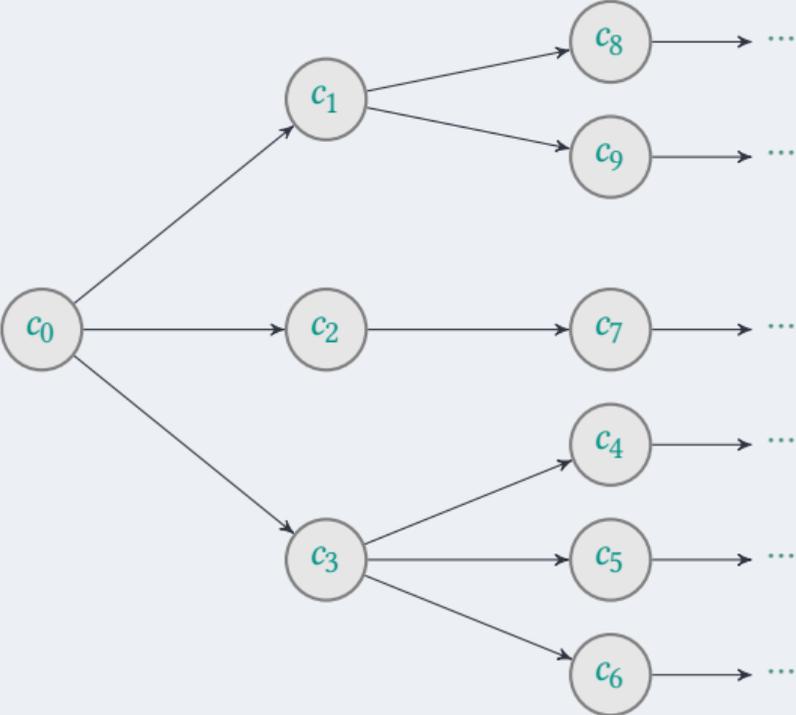
L'automa accetta la stringa di input se, tra tutte le computazioni possibili, ne esiste almeno una di accettazione (se esiste, nell'albero di computazione, un cammino dalla radice ad un nodo rappresentante una configurazione di accettazione).

Computazioni non deterministiche: interpretazione 2

L'automata esegue una sola **computazione non deterministica**, per la quale, ad ogni passo, assume non una sola, ma un insieme di configurazioni, transitando, ad ogni passo, non da configurazione a configurazione ma da un insieme di configurazioni ad un insieme di configurazioni.

Un insieme di configurazioni è di accettazione se include almeno una configurazione di accettazione.

Albero di computazione



Asimmetria tra accettazione e rifiuto di una stringa, introdotta dal non determinismo.

- ⦿ Nel caso deterministico la stringa viene accettata se la singola computazione definita è di accettazione, e rifiutata se essa termina in una configurazione non di accettazione
- ⦿ Nel caso non deterministico la stringa viene accettata se **una qualunque** delle computazioni definite è di accettazione, mentre non lo viene se **tutte** le possibili computazioni che terminano non sono di accettazione.

Dato un modello di calcolo, un problema decisionale (o un linguaggio) è detto **decidibile** nel modello di calcolo se esiste un algoritmo (in particolare, un automa) che per ogni istanza del problema risponde correttamente VERO oppure FALSO. In caso contrario il problema è detto **indecidibile**.

Dato un modello di calcolo, un problema decisionale (o un linguaggio) è detto **semidecidibile** nel modello di calcolo se esiste un algoritmo (in particolare, un automa) che per tutte e sole le istanze positive del problema risponde correttamente VERO.

I linguaggi di tipo 1, 2, 3 sono decidibili all'interno di modelli di calcolo opportuni.

Non esistono modelli di calcolo al cui interno i linguaggi di tipo 0 siano decidibili.

Esiste un modello di calcolo all'interno del quale i linguaggi di tipo 0 sono semidecidibili.

Linguaggi di tipo 3: decidibili per automi con memoria di dimensione costante, dipendente dal linguaggio, ma non dalla particolare stringa (**Automi a stati finiti**)

Linguaggi di tipo 2: decidibili per automi non deterministici con memoria di dimensione lineare e modalità di accesso limitate, di tipo *last-in-first-out* (**Automi a pila non deterministici**)

Linguaggi di tipo 1: decidibili per automi non deterministici con memoria di dimensione lineare e accesso casuale (**Automi lineari non deterministici**)

Linguaggi di tipo 0: semidecidibili per automi con memoria di dimensione potenzialmente illimitata e accesso casuale (**Macchine di Turing**)