

Un insieme finito non vuoto $\Sigma = \{a_1, \dots, a_n\}$ 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 .

Gli elementi di Σ^* vengono detti: **parole** o **stringhe**: si noti che, mentre Σ è finito (l'insieme delle possibili stringhe di lun. 1), Σ^* è infinito.

L'elemento ε viene detto **parola vuota**. (lun. 0)

L'operazione $\circ: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ definita sul monoide è chiamata **concatenazione** e consiste nell'unire due parole di Σ^* :

$$x_1 \dots x_n \circ y_1 \dots y_k = x_1 \dots x_n y_1 \dots y_k$$

$$x_1, \dots, x_n, y_1, \dots, y_k \in \Sigma$$

Valgono le seguenti proprietà:

Σ^* chiuso rispetto a \circ : $\forall x, y \in \Sigma^*: x \circ y \in \Sigma^*$

\circ associativa: $\forall x, y \in \Sigma^*: (x \circ y) \circ z = x \circ (y \circ z)$

esiste un elemento neutro ε : $\forall x \in \Sigma^*: x \circ \varepsilon = \varepsilon \circ x = x$

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

La concatenazione non gode della proprietà commutativa:

$$x \circ y \neq y \circ x \quad x \neq y$$

Con x^h si denota la concatenazione di x con sé stessa iterata h volte. $x^0 = \varepsilon$.

Dato un alfabeto Σ , si definisce **linguaggio** un qualsivoglia insieme 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 nessuna stringa. $|\lambda|=0$, $\lambda \neq \{\varepsilon\}$.

Il linguaggio è enumerabile, cioè posso assegnare un numero di ordine alle stringhe del linguaggio.

Un insieme è definito dalla sua **descrizione**:

$$A = \{x \in L : \#_b(x) = 2k, k \geq 0, k \in \mathbb{N}\}$$

tutte le stringhe che hanno la parola pari

$P(\Sigma^*)$ è l'insieme di tutti i linguaggi di Σ^* .

Se il linguaggio ha una descrizione finita (semplice) allora è finito.

L'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 l_1 e l_2

$$L_1 \circ L_2 = \{x \in \Sigma^* \mid \exists u \in L_1, \exists v \in L_2 (x = uv)\}$$

$$L \circ \{\epsilon\} = \{\epsilon\} \circ L = L, L \circ \Lambda = \Lambda \circ L = \Lambda$$

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

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

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

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

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

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

La potenza L^h di un linguaggio è definita come

$$L^h = L \circ L^{h-1}, h \geq 1 \quad L^0 = \{\epsilon\}, \Lambda^0 = \{\epsilon\}$$

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**. $L, \epsilon \in L^*$ $L^* = \{\epsilon\}$

Si indica con L^+ la **chiusura definita da**:

$$L^+ = \bigcup_{h=1}^{\infty} L^h \quad L^* = L^+ \cup \{\epsilon\}$$

Dato un $\Sigma = \{a_0, a_1\}$, una particolare stringa $a_0a_1a_0a_0a_1\dots$ se considero la seq. degli indici posso rappresentare un intero in base 2.

In generale posso rappresentare le stringhe di ogni alfabeto come interi in base n , dove $n = |\Sigma|$.

A ogni stringa di ogni alfabeto è possibile associare un intero, perciò le posso enumerare.

Per il significato delle stringhe è necessario definirne la semantica cioè dare un'interpretazione alle stringhe es. bit che possono rappresentare un intero, un char, ... in base all'interpretazione.

L'informazione è la stessa cambia l'interpretazione.

Lezione 2

L'insieme dei linguaggi è in stretto rapporto con quello dei **problemi di decisione**.

Un problema di decisione è definito su un insieme di possibili **istanze** e associa ad ognuna di esse un valore Vero / Falso.

L'insieme delle istanze è partizionato in istanze **positive** e **negative**: il problema è, per ogni istanza, riconoscere se è una istanza positiva.

es. "La persona più alta è 1.80m" Un'istanza è un insieme di persone,

Gli insiemi di tutte le istanze sono tutti i possibili insiemi di persone e il predicato, data una particolare istanza può essere vero o falso.

I problemi di decisione rappresentano la tipologia più "semplice" di problemi.

Problemi di **ricerca**: restituzione di una soluzione, se esiste (un percorso da A a B di al più x Km).

Problemi di **enumerazione**: restituzione di tutte le soluzioni, se esistono (tutti i percorsi da A a B con al più x Km).

Problemi di **ottimizzazione**: restituzione della "migliore" soluzione possibile (il percorso più breve tra A e B).

Ci concentriamo sui problemi di decisione perché se non vale nel caso più semplice non verrà neanche nei casi più complessi.

Un problema di decisione è risolvibile da un algoritmo che opera in tempo lineare, polinomiale, esponenziale o non è risolvibile da nessun algoritmo.

È presente una **relazione** tra linguaggi e problemi di decisione.

- Dato un linguaggio L e una stringa x , determinare se $x \in L$ è un problema di decisione, le istanze sono tutte le possibili stringhe mentre L rappresenta le istanze positive.

- Dato un problema di decisione P , l'insieme delle istanze è codificato per mezzo di uno **schema di codifica**: ad ogni istanza rappresenta una stringa.

Una stringa può corrispondere a una istanza positiva, una istanza negativa o nessuna istanza.

L è l'insieme delle stringhe corrispondenti a codifiche di istanze positive.

In generale, si assume che sia possibile distinguere facilmente (efficientemente) tra le stringhe che rappresentano istanze e le altre.

Un problema di decisione è caratterizzato dall'insieme delle istanze positive, rispetto all'insieme di tutte le istanze.

Questi insiemi in generale sono infiniti, non si può definire un problema per mezzo delle istanze ma per mezzo di una **descrizione finita** sì.

Un caso particolare di descrizione: un algoritmo che decide il problema (restituisce Vero per istanze positive e Falso per le negative).

Non tutti i problemi possono essere descritti in modo infinito, equivale a dire che non per tutti i problemi di decisione esistono algoritmi che li risolvono.

I linguaggi e i problemi di decisione sono la stessa cosa.

Consideriamo i linguaggi definiti su un alfabeto dato $\{0,1\}$.

Un linguaggio è un insieme (infinito, in generale) di stringhe su $\{0,1\}$, e quindi corrisponde ad una sequenza di 0 o 1 sulla sequenza di tutte le stringhe ordinate.

Applicando la diagonalizzazione vista per i numeri reali, ne risulta che l'insieme dei linguaggi è non numerabile.

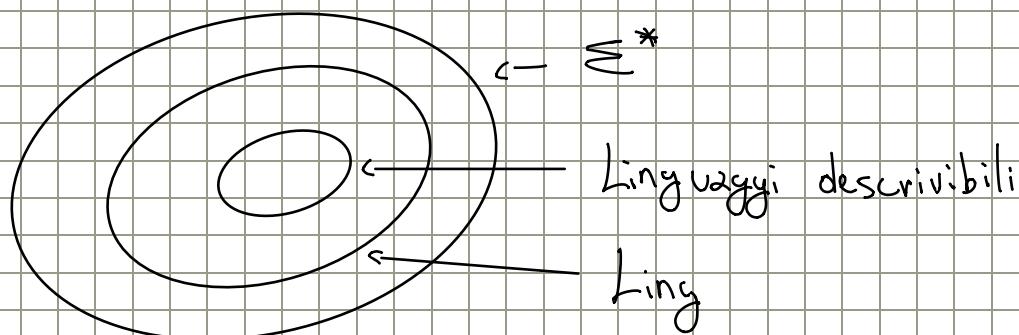
Consideriamo le descrizioni utilizzando un alfabeto dato $\{0,1\}$.

Una descrizione /algoritmo è una stringa su $\{0,1\}$

Dato un qualunque alfabeto finito, l'insieme delle stringhe corrispondenti è numerabile.

Di conseguenza: ci sono più problemi di decisione (linguaggi) che loro descrizioni finite (algoritmi).

Quindi esistono problemi /linguaggi non descrivibili in modo finito e non decidibili mediante algoritmi.



Lezione 3

Le espressioni regolari consentono di rappresentare linguaggi mediante una opportuna interpretazione dei simboli che le compongono.

Dato un alfabeto Σ e dato l'insieme $\{+, *, (,), \cdot, \circ\}$ si definisce espressione regolare sull'alfabeto Σ una stringa

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

tale che valga una delle seguenti condizioni:

$$r = \emptyset$$

$$r \in \Sigma \cup \{\epsilon\}$$

$r = (s+t)$, oppure $s = (st)$, oppure $r = s^*$, dove s e t sono espressioni regolari sull'alfabeto Σ .

Corrispondenza tra un'espressione regolare r e il linguaggio $L(r)$ che essa rappresenta.

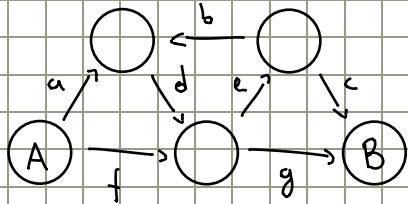
espressione regolare	linguaggi
\emptyset	Δ
a	$\{a\}$
$(s+t)$	$L(s) \cup L(t)$
(st)	$L(s) \circ L(t)$
s^*	$(L(s))^*$

es. L'espressione regolare $(a+b)^*$ è $\{x | x \in \{a,b\}^*, x \text{ termina con } a\}$.

es. Sia $c = \{0, \dots, g\}$ l'espressione regolare che rappresenta i numeri reali: $c = ((1+\dots+g)(0+\dots+g))^* + 01 \cdot (0+\dots+g)^+$

es. $(a+b)^* b (a+b) (a+b)$ rappresenta l'insieme delle stringhe $\{a,b\}^*$ il cui terz' ultimo carattere è b .

es.



$(ad+e)(ebd)^*(ec+g)$ rappresenta tutti i possibili percorsi che vanno da A a B.

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

$$a+b = b+a, a+a \equiv a, aa^* \equiv a^*a, ab \neq ba$$

$L(r) = L(s)$ se $\forall \sigma : \sigma \in L(r) \iff \sigma \in L(s)$.

$L(r) \neq L(s)$ se $\exists \sigma : ((\sigma \in L(r) \wedge \sigma \notin L(s)) \vee (\sigma \notin L(r) \wedge \sigma \in L(s)))$.

L'operatore di concatenazione \circ può essere omesso, $ab = a \cdot b$ e avrà sempre precedenza su $+$. Quindi $a+b \cdot c \equiv a+(bc)$.

\circ è commutativa ($r+s \equiv s+r$), associativa ($r+(s+t) \equiv (r+s)+t$), con elemento neutro \emptyset ($r+\emptyset \equiv r$), indempotente ($r+r \equiv r$).

\circ è associativa ($r(st) \equiv (rs)t$), con elemento neutro ϵ ($r\epsilon \equiv r$) e elemento nullo \emptyset ($r\emptyset \equiv \emptyset$).

\circ si distribuisce su $+$ ($r(s+t) \equiv rs+rt$).

\circ non si distribuisce su \circ ($r+st \neq (r+s)(r+t)$).

Lezione 4

Un altro modo per descrivere i linguaggi è tramite le grammatiche.

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

V_T è un insieme finito e non vuoto di simboli terminali.

V_N è un insieme finito e non vuoto di simboli non terminali.

P è una relazione binaria di cardinalità finita su

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

nella parte iniziale è sempre presente un accento

P è detto insieme delle produzioni.

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

$S \in V_N$ è detto assioma.

$\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$ si può scrivere come $\alpha \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$.

L'unione $V_T \cup V_N$ viene indicata con V .

es. $G = \langle \{a, b\}, \{S, B, C\}, P, S \rangle$ avente le seguenti regole di produzione

$$S \rightarrow aS | B$$

$$B \rightarrow bB | bC$$

$$C \rightarrow cC | c$$

Con questa grammatica si possono generare le stringhe del linguaggio

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

Lezione 5

Una regola $\alpha \rightarrow \epsilon$ prende il nome ϵ -produzione o ϵ -regola.

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

la coppia (ϕ, ψ) appartiene alla relazione sse esistono $\alpha \in V^*, \beta \in V_N, \gamma \in V^*$ tali che:

$$\phi = \gamma \alpha \delta \quad \psi = \gamma \beta \delta \quad \alpha \Rightarrow \beta \in P \quad \text{in questo caso scriviamo } \phi \xrightarrow{\gamma} \psi$$

es. $Ab \rightarrow bA \quad cbAbBc \Rightarrow cbbABCc \quad (\text{in un passo})$

Dato una grammatica G , una **derivazione** (in G) è una sequenza di stringhe $\phi_1, \dots, \phi_n \in V^*$ tali che $\forall i \in \{1, \dots, n-1\}: \phi_i \xrightarrow{\gamma} \phi_{i+1}$.

La relazione di derivabilità (rispetto a G) è la chiusura transitiva e riflessiva della derivazione diretta.

Scrivendo $\phi \xrightarrow{*} \psi$ indichiamo l'esistenza di (almeno) una derivazione da ϕ a ψ .

es. $bA \rightarrow \epsilon \quad abAaabAb \Rightarrow aabAb \Rightarrow aaab \quad abAaabAb \xrightarrow{*} aaab$

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

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

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

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

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

es. $S \rightarrow aSb$ e $S \rightarrow b(Ab, A \rightarrow Aula)$ sono equivalenti.

Le grammatiche di tipo 0 dette anche non limitate, definiscono la classe di linguaggi più ampio possibile. In esse le produzioni sono

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

Queste grammatiche ammettono anche derivazioni che accorciano le forme di frase.

Le grammatiche di tipo 1 dette anche contestuali o context sensitive, ammettono qualunque regola di produzione che non riduca la lunghezza delle stringhe, cioè:

$$\alpha \rightarrow \beta, \alpha \in V^* V_N V^*, \beta \in V^+, |\alpha| \leq |\beta|$$

Le grammatiche di tipo 2 dette anche non contestuali o context free ammettono produzioni del tipo:

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

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

Lezione 6

Le grammatiche di tipo 3 dette anche lineari destre o regolari, ammettono solo soluzioni del tipo:

$$A \rightarrow S, A \in V_N, S \in (V_T \cdot V_N) \cup V_T$$

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

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

$$A \rightarrow S, A \in V_N, S \in (V_N \cdot V_T) \cup V_T$$

In questo caso l'unico simbolo non terminale appare come primo.

I linguaggi generati da una grammatica di tipo n si dicono linguaggi di tipo n.

es.

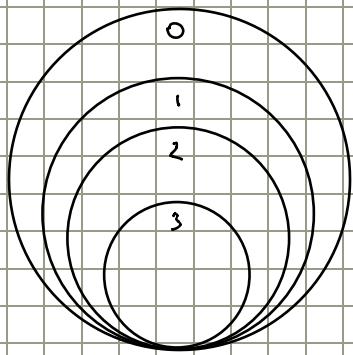
$$\begin{aligned} \text{tipo 0: } S &\rightarrow aAb \\ &aA \rightarrow aaAb \\ &A \rightarrow \epsilon \end{aligned}$$

$$\begin{aligned} \text{tipo 1: } S &\rightarrow aSa \mid aAb \mid aA \\ &aA \rightarrow aa \\ &Ab \rightarrow aab \end{aligned}$$

$$\begin{aligned} \text{tipo 2: } S &\rightarrow aSb \mid aAb \\ &A \rightarrow ab \end{aligned}$$

$$\begin{aligned} \text{tipo 3: } S &\rightarrow aS \mid b \\ &S \rightarrow c \end{aligned}$$

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.

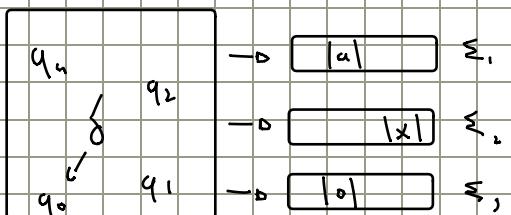
Un linguaggio L viene detto **strettamente** di tipo n se esiste una grammatica G di tipo n che genera L e non esiste alcuna grammatica G' di tipo $n > n$ che possa generarlo.

tipo produzioni dove

0	$\alpha \rightarrow \beta$	$\alpha \in V^*, \beta \in V^*$, $\beta \in V^+$
1	$\alpha \rightarrow \beta$	$ \alpha \leq \beta $, $\alpha \in V^*, \beta \in V^*$, $\beta \in V^+$
2	$A \rightarrow \beta$	$A \in V_N$, $\beta \in V^+$
3	$A \rightarrow S$	$A \in V_N$, $S \in (V_T \cup (V_i \cdot V_n))$

Un **Automa** è un dispositivo che, data una stringa x fornita gli 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. $\rightarrow A \xleftarrow{y}$

Ha un 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 che possono contenere un carattere. I caratteri vengono letti e scritti per mezzo di testine che possono muoversi lungo i nastri, posizionandosi sulle diverse celle.



automa prende
3 nastri in
input

$$\delta: Q \times \Sigma_1 \times \Sigma_2 \times \Sigma_3 \rightarrow$$

$$Q \times \Sigma_1 \times \Sigma_2 \times \Sigma_3 \setminus \{sx, d+, stay\}$$

L'insieme delle informazioni necessarie e sufficienti per determinare, in un certo istante, il comportamento futuro dell'automa: stato interno, il contenuto di tutti i nastri di memoria e la posizione di tutte le testine sui nastri.

Configurazione iniziale è quella in cui si assume si trovi inizialmente un automa, in presenza di una stringa: stato predefinito come iniziale, nastri di memoria vuoti eccetto quello di input e la testina sulla cella contenente il primo carattere.

La 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 su A .

es. Automata che riconosce se la stringa ha # pari.

$\{q_d, q_p\}$ stati

$$q_a, a \rightarrow q_d : q_p, b \rightarrow q_p$$

funz. transiz.

$$q_d, a \rightarrow q_p : q_d, b \rightarrow q_d$$

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

$$\delta(q_p, a) = q_d$$

$$\delta(q_p, b) = q_p$$

$$\delta(q_d, a) = q_p$$

$$\delta(q_d, b) = q_d$$

Lezione 7

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 su A .

Le **Configurazioni di accettazione** sono un sottoinsieme delle possibili configurazioni determinato, 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.

$$c_0, c_1, c_2, \dots \quad c_i \xrightarrow{A} c_{i+1}, \quad \forall i = 0, 1, \dots, n$$

\xrightarrow{A}^* : chiusura transitiva e riflessiva della relazione \xrightarrow{A} .

Dette due configurazioni c_i, c_j di A

$$c_i \xrightarrow{A}^* c_j$$

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

Se c_0, c_1, \dots, c_n ha lunghezza finita e non esiste nessuna c tale che $c_n \xrightarrow{A} c$ la computazione **termina**.

Ci sono tre esiti di una computazione:

1. c_n esiste ed è una configurazione di accettazione, l'automa accetta la stringa in input
2. c_n esiste ed non è una configurazione di accettazione, l'automa rifiuta la stringa in input

3. c_n non esiste, la computazione termina

Automa deterministico

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.

Un **automa a stati finiti deterministico (ASFD)** è una quintupla

$$A = \langle \Sigma, Q, \delta, q_0, F \rangle$$
, dove

$\Sigma = \{a_1, \dots, a_n\}$ è l'**alfabeto** di input

$Q = \{q_0, \dots, q_n\}$ è un insieme finito e non vuoto di **stati**.

$q_0 \in Q$ è lo **stato iniziale**

$F \subseteq Q$ è un insieme di **stati finali**

$\delta: Q \times \Sigma \rightarrow Q$ è la **funzione (totale)** di transizione che ad ogni coppia **(stato, carattere in input)** associa uno stato successivo

Dato un automa a stati finiti $A = \{\Sigma, Q, \delta, q_0, F\}$, una configurazione di A è una coppia (q, x) , con $q \in Q$ e $x \in \Sigma^*$.

Una configurazione (q, x) , $q \in Q$ ed $x \in \Sigma^*$, di A , è detta:

- **iniziale** se $q = q_0$
- **finale** se $x = \epsilon$
- **accettante** se $x = \epsilon$ e $q \in F$

Dato un ASFD $A = \{\Sigma, Q, q_0, F\}$ e due configurazioni (q, x) e (q', y) di A , avremo che $(q, x) \vdash (q', y)$ se e solo se valgono le due condizioni:

1. $x = ay$, $\exists a \in \Sigma$
2. $\delta(q, a) = q'$

Dato un automa a stati finiti deterministico $A = \langle \Sigma, Q, \delta, q_0, F \rangle$, una stringa $x \in \Sigma^*$ è accettata da A se e solo se

$$(q_0, x) \xrightarrow{*} (q, \varepsilon)$$

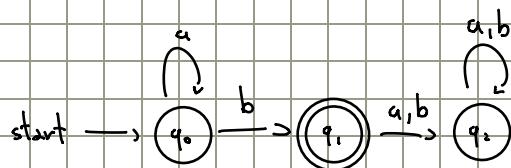
con $q \in F$

Possiamo definire il linguaggio riconosciuto da A come

$$L(A) = \{ x \in \Sigma^* \mid (q_0, x) \xrightarrow{*} (q, \varepsilon), q \in F \}$$

es.

Σ	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2



La stringa $aabb$ è accettata dall'automa a stati finiti:

$$(q_0, aabb) \xleftarrow{} (q_0, ab) \xleftarrow{} (q_0, b) \xleftarrow{} (q_1, \varepsilon)$$

Dato un automa a stati finiti deterministico $A = \langle \Sigma, Q, q_0, F \rangle$ la sua funzione di transizione estesa

$$\bar{\delta}: Q \times \Sigma^* \rightarrow Q$$

è definita come transpositiva della δ :

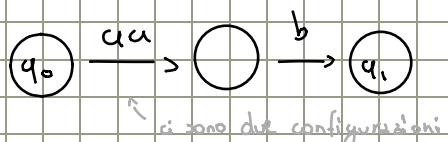
$$\bar{\delta}(q, \varepsilon) = q$$

$$\bar{\delta}(q, xa) = \delta(\bar{\delta}(q, x), a)$$

dove $a \in \Sigma$, $x \in \Sigma^*$

Una stringa $x \in \Sigma^*$ è accettata da $A = \langle \Sigma, Q, \bar{\delta}, q_0, F \rangle$ se e solo se

$$\bar{\delta}(q_0, x) \in F$$



$$\bar{\delta}(q_0, aabb) = q_1$$

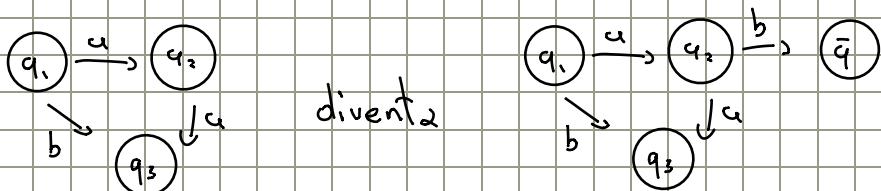
Il linguaggio riconosciuto da un automa a stati finiti deterministico A è l'insieme

$$L(A) = \{x \in \Sigma^* \mid \bar{\delta}(q_0, x) \in F\}$$

La funzione di transizione δ è stata definita come totale.

Ogni ASTFD $A = (\Sigma, Q, \delta, q_0, F)$ con funzione di transizione δ parziale può essere trasformato in ASTFD $A' = (\Sigma, Q', \delta', q_0, F)$ con funzione di transizione totale ed equivalente, ponendo $Q' = Q \cup \{\bar{q}\}$ e δ' tale che come:

1. Se $\delta(q, a) \in Q$ è definito allora $\delta'(q, a) = \delta(q, a)$
2. Se $\delta(q, a), q \in Q$ non è definito allora $\delta'(q, a) = \bar{q}$
3. $\delta'(\bar{q}, a) = \bar{q} \quad \forall a \in \Sigma$



Lezione 8

Dato un automa deterministico A, una stringa x in input ad A, $c_0(x)$ configurazione iniziale di A corrispondente alla stringa x.

A quella 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.

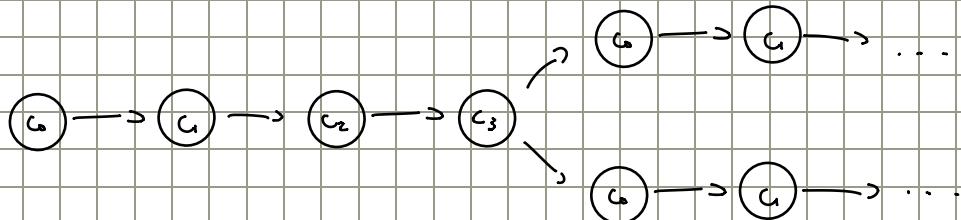
Automa non deterministico

E' una 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.



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

Deterministico: $\delta: Q \times \Sigma_1 \times \Sigma_2 \times \Sigma_3 \rightarrow Q \times \Sigma_1 \times \Sigma_2 \times \Sigma_3 \times \{dx, sx, stay\}^3$

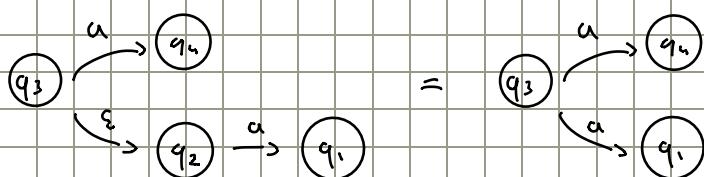
Non deterministico: $\delta: Q \times \Sigma_1 \times \Sigma_2 \times \Sigma_3 \rightarrow 2^{Q \times \Sigma_1 \times \Sigma_2 \times \Sigma_3 \times \{sx, dx, stay\}^3}$ testim
✓
L = P()

Le ε-transizioni sono transizioni che un automa può eseguire senza leggere alcun carattere in input. La presenza di ε-transizioni introduce non determinismo.

$$\delta(q_3, \epsilon) = q_2$$

$$\delta(q_3, a) = q_4$$

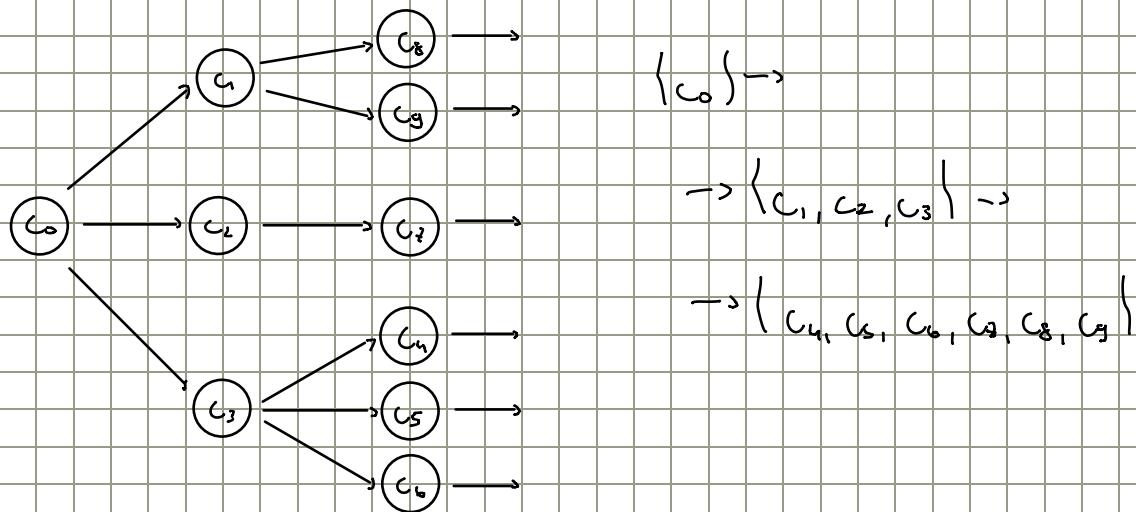
$$\delta(q_2, a) = q_1$$



L'automa non deterministico può eseguire una qualunque computazione, associata ad una sequenza di "scelta". L'automa accetta la stringa in 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).

L'automa esegue una sola **computazione non deterministico**, per la quale, ad ogni passo, assume non una sola, ma un insieme di configurazioni, transitando, ad ogni passo, 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.



Un **automa a stati finiti non deterministico (ASFND)** è una quintupla:

$A = \langle \Sigma, Q, \delta_n, q_0, F \rangle$, dove

$\Sigma = \{a_1, \dots, a_n\}$ è l'**alfabeto di input**

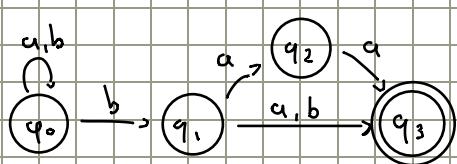
$Q = \{q_0, \dots, q_n\}$ è un insieme finito e non vuoto di **stati**.

$q_0 \in Q$ è lo **stato iniziale**

$F \subseteq Q$ è un insieme di **stati finali**

$\delta_n: Q \times \Sigma \rightarrow P(Q)$ è la **funzione (parziale) di transizione** che ad ogni coppia **(stato, carattere)** su cui è definita associa un sottoinsieme di Q (eventualmente vuoto)

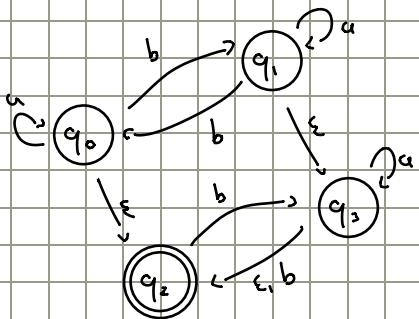
es.



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

Lezione 9

Un ϵ -ASFND è un ASFND con $\delta_n : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$, nella quale sono quindi ammesse ϵ -transizioni



L'2 presenza di ϵ rende un ASF un ASFND, perché può avere luogo o meno. Dato uno stato q dell'automa, la sua ϵ -chiusura è l'insieme $\epsilon(q)$ degli stati raggiungibili da q mediante una sequenza (anche nulla) di ϵ -transizione.

q	$\epsilon(q)$
q_0	$\{q_0, q_1\}$
q_1	$\{q_1, q_2, q_3\}$
q_2	$\{q_2\}$
q_3	$\{q_2, q_3\}$

ϵ -chiusure : $Q \rightarrow 2^Q$

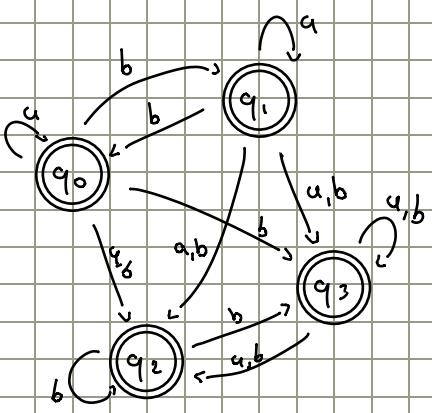
La funzione di transizione $\hat{\delta}$ è definita come:

1. $\hat{\delta}(q, \epsilon) = \epsilon(q)$
2. $\hat{\delta}(q, xu) = \bigcup \delta(q, xu) | q \in \epsilon(q)$

$$\hat{\delta}(q_0, a) = \delta(q_0, a), \delta(q_2, a)$$

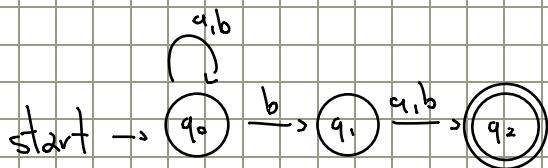
Dato un ASFND che riconosce un linguaggio L , esiste corrispondentemente un ϵ -ASFND che riconosce lo stesso linguaggio L e viceversa.

Dato un ϵ -ASFND si può passare a un ASFND togliendo le ϵ -transizioni e facendo attenzione a trasformare gli stati che si collegano tramite ϵ -transizione in stati finali in tali.

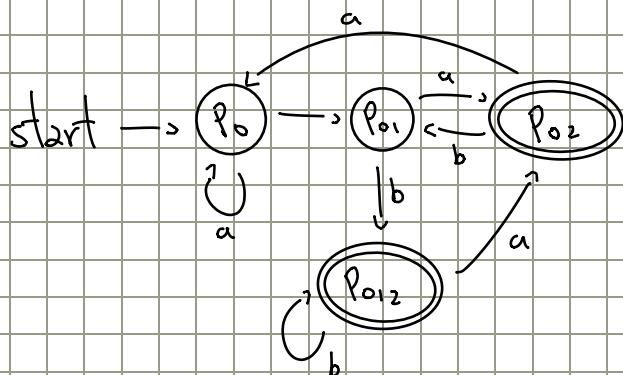


Dato un ASFND è possibile passare ad un ASFD ponendo Q' in corrispondenza biunivoca con $P(Q)$ (quindi $|Q'| = 2^{|Q|}$). Lo stato iniziale è q'_0 . Gli stati finali F' corrispondono ai sottoinsiemi di Q che contengono almeno un elemento di F :

ASFD che riconosce le stringhe in $(a+b)^* b (a+b)$



ASFD che riconosce le stringhe in $(a+b)^* b (a+b)$



Lettura 10

Esiste un equivalenza tra ASF, RG e RE.

Per ogni grammatica regolare $G = \langle V_T, V_N, P, S \rangle$, esiste un ASFND $A_N = \langle \Sigma, Q, S_N, q_0, F \rangle$ che riconosce il linguaggio che essa genera.

Viceversa per ogni ASFND A_N , esiste una grammatica regolare che genera il linguaggio che esso riconosce.

Sia $G = \langle V_T, V_N, P, S \rangle$ una grammatica di tipo 3, con al più la sola ϵ -produzione $S \rightarrow \epsilon$. Se ho G_2 con più ϵ -produzioni posso passare ad una G_3 con una sola ϵ -prod: $S \rightarrow \epsilon$ solo prod iniziale

Definiamo una procedura che a partire da G produce un AFND $A_N = \langle \Sigma, Q, \delta_N, q_0, F \rangle$ equivalente (che accetta tutte e sole stringhe prodotte da G).

Da G a A_N :

$$\Sigma = V_T$$

$$Q = \{q_I \mid I \in V_N\} \cup \{q_F\}$$

$$q_0 = q_S$$

$$F = \begin{cases} \{q_0, q_F\} & \text{se } S \rightarrow \varepsilon \in P_{\text{produzioni}} \\ \{q_F\} & \text{altrimenti} \end{cases}$$

Per ogni coppia $a \in V_T$ e $B \in V_N$

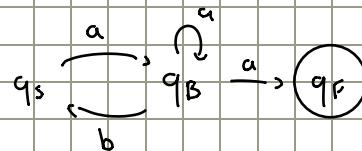
$$\delta_N(q_S, a) = \begin{cases} \{q_C \mid B \xrightarrow{a} q_C \in P\} \cup \{q_F\} & \text{se } B \xrightarrow{a} a \in P \\ \{q_C \mid B \xrightarrow{a} q_C \in P\} & \text{altrimenti} \end{cases}$$

es.

$$a(a+ba)^*$$

$$S \xrightarrow{*} aB$$

$$B \xrightarrow{*} aB \mid bS \mid a$$



Per dimostrare l'equivalenza tra G e A_N , dobbiamo mostrare che $\forall x \in \Sigma^*$ si ha che

$$S \xrightarrow{*} x \iff \delta_N(q_S, x) \cap F \neq \emptyset$$

(chiaramente vero se $x = \varepsilon$ in quanto $S \xrightarrow{*} \varepsilon \in P$ per costruzione).

Nel caso $x \in \Sigma^+$ si dimostra per induzione sulla lunghezza di x

$$S \xrightarrow{*} xz \iff q_2 \in \delta_N(q_S, x)$$

Passo base: $|x| = 1$, $x = a$, $a \in \Sigma$. Allora abbiamo che $S \xrightarrow{*} az \iff S \xrightarrow{*} a \in P$ e sse $q_2 \in \delta_N(q_S, a)$

Passo induttivo: $|x| > 1$, $x = ya$ $|y| = n \geq 1$ $a \in \Sigma$ per ipotesi induttiva

$$S \xrightarrow{*} yz \iff q_2 \in \delta_N(q_S, y)$$

Osserviamo che $S \xrightarrow{*} xz'$ sse $z \in V_N$ tale che $S \xrightarrow{*} yz \Rightarrow ya z' = xz'$
 $q_z \in \bar{\delta}_N(q_s, y)$, $z \rightarrow az' \in P$ quindi $q_z \in \delta_N(a, z)$
Quindi $q_z \in \bar{\delta}_N(q_s, ya) = \bar{\delta}_N(q_s, x)$

Abbiamo verificato che $S \xrightarrow{*} xz$ sse $q_z \in \bar{\delta}_N(q_s, x)$

$S \xrightarrow{*} x$ sse $z \in V_N$, $y \in \Sigma^*$ e $z \rightarrow a \in P$ tali che $x = ya$ e
 $S \xrightarrow{*} yz \Rightarrow ya \Rightarrow x$

Cioè è vero sse $q_z \in \bar{\delta}_N(q_s, y)$ e $q_z \in \delta_N(q_z, a)$ e quindi sse
 $q_z \in \bar{\delta}_N(q_s, ya) = \bar{\delta}_N(q_s, x)$.

In conclusione, per ogni linguaggio regolare esiste un ASFND che lo accetta.

Lezione 11

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

Definiamo una procedura che a partire da A produca una grammatica di tipo 3 $G = (V_T, V_N, P, S)$ equivalente (che genera tutte e sole stringhe accettate da A).

Da A a G :

Se $q_0 \in F$:

$$V_T = \Sigma$$

$$V_N = \{ A_i \mid \forall q_i \in Q \}$$

$$S = A_0$$

\forall regola di transizione $\delta(q_i, a) = q_j \exists A_i \rightarrow a A_j \in P$, se $q_j \in F \exists A_i \rightarrow a \in P$

Se $q_0 \in F$:

$$V_T = \Sigma$$

$$V_N = \{ A_i \mid \forall q_i \in Q \} \cup \{ A'_0 \}$$

$$S = A'_0$$

\forall regola di transizione $\delta(q_i, a) = q_j \exists A_i \rightarrow a A_j \in P$, se $q_j \in F \exists A_i \rightarrow a \in P$

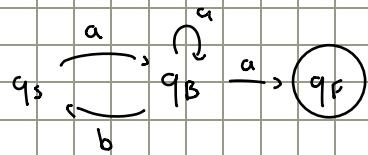
Inoltre $\forall \delta(q_0, a) = q_j \exists A'_0 \rightarrow a A_j \in P$ e se $q_j \in F \exists A'_0 \rightarrow a \in P$ e $A'_0 \rightarrow \epsilon \in P$

es.

$$a(a+ba)^*a$$

$$S \rightarrow aB$$

$$B \rightarrow aB \mid bS \mid a$$



Per dimostrare l'equivalenza tra A_n e G , dobbiamo mostrare che $\forall x \in \Sigma^*$ si ha che

$$\bar{\delta}(q_0, x) \cap F \text{ sse } S \xrightarrow{*} x$$

(chiaramente vero se $x = \varepsilon$ in quanto $q_0 \in F$ e per costruzione l'assimila di G è A'_0 e $A'_0 \rightarrow \varepsilon \in P$.

Nel caso $x \in \Sigma^+$ si dimostra per induzione sulla lunghezza di x

$$A_i \xrightarrow{*} x A_j \text{ sse } \bar{\delta}(q_i, x) = q_j$$

$$A_i \xrightarrow{*} x \text{ sse } \bar{\delta}(q_i, x) \in F$$

Passo base: $|x|=1$, $x=a$, $a \in \Sigma$. Allora abbiamo che per costruzione $A_i \rightarrow a A_j \in P$ sse $\delta(q_i, a) = q_j$ e $A_i \rightarrow a \in P$ sse $q_j \in F$

Passo induttivo: $|x|=n>1$, $x=ya$, $|y|=n-1$ per ipotesi induttiva la proprietà è valida per y , quindi

$$A_i \xrightarrow{*} y A_j \text{ sse } \bar{\delta}(q_i, y) = q_j$$

Supponiamo $A_i \xrightarrow{*} x A_j = ya A_j$: ciò è possibile sse $\exists A_k : A_i \xrightarrow{*} y A_k$ e $A_k \rightarrow a A_j \in P$.

Ne consegue che

$$A_i \xrightarrow{*} y A_k \Rightarrow ya A_j \Rightarrow x A_j \text{ sse}$$

$$q_j = \delta(q_k, a) = \delta(\bar{\delta}(q_i, y), a) = \bar{\delta}(q_i, ya) = \bar{\delta}(q_i, x)$$

Lezione 12

Tutti i linguaggi definiti da espressioni regolari sono riconosciuti da AFD

Data una grammatica G_3 esiste un'espressione regolare r tale che $L(G) = L(r)$ che descrive cioè il linguaggio generato da G .

Consideriamo una grammatica G_3 ed il linguaggio L da essa generato, che per semplicità assumiamo non contenga la stringa vuota ϵ . Se così non fosse, applichiamo le considerazioni seguenti al linguaggio $L - \{\epsilon\}$, anch'esso regolare: una volta derivata un'espressione regolare r che lo definisce, l'espressione regolare che definisce L sarà $r + \epsilon$.

Alla grammatica G possiamo far corrispondere un sistema di equazioni su espressioni regolari. Estensione del linguaggio delle espressioni regolari con variabili A, \dots, Z , associando una variabile ad ogni non terminale in G .

Tali variabili potranno assumere valori nell'insieme delle espressioni regolari.

Da G a r :

Raggruppamento di tutte le produzioni che presentano a sinistra lo stesso non terminale

$$A \rightarrow a_1 B_1 | a_2 B_2 | \dots | a_n B_n | b_1 | \dots | b_m$$

$$A = a_1 B_1 + a_2 B_2 + \dots + a_n B_n + b_1 + \dots + b_m$$

Da una grammatica regolare si ottiene un sistema di equazioni lineari destre, in cui ogni monomio contiene una variabile a destra di simboli terminali.

$$\begin{aligned} A &\rightarrow aA | bB \\ B &\rightarrow bB | c \end{aligned} \quad \text{corrisponde} \quad \left\{ \begin{array}{l} A = aA + bB \\ B = bB + c \end{array} \right.$$

Utilizzare le trasformazioni algebriche applicabili sulle operazioni di unione e concatenazione oltre alle seguenti due regole.

- sostituzione di una variabile con un'espressione regolare estesa.

es. $A = aA + bB, B = bB + c \rightarrow A = aA + bbB + bc$

- eliminazione della ricorsione: $A = \alpha_1 A + \dots + \alpha_n A + \beta_1 + \dots + \beta_m$ diventa

$$A = (\alpha_1 + \dots + \alpha_n)^* (\beta_1 + \dots + \beta_m)$$

es.

$$B = bB + c \rightarrow B = b^* c$$

es.

$$S \rightarrow aS \mid bA \mid \epsilon \mid b$$

$$A \rightarrow aA \mid bS \mid a$$

$$\left\{ \begin{array}{l} S = aS + bA + b + \epsilon \\ A = aA + bS + a \end{array} \right.$$

$$\left\{ \begin{array}{l} S = aS + bc^* (bS + a) + b + \epsilon \\ A = a^* (bS + a) \end{array} \right.$$

$$\left\{ \begin{array}{l} S = (a + ba^* b)S + ba^* + b + \epsilon \\ A = a^* (bS + a) \end{array} \right. \quad \left\{ \begin{array}{l} S = (a + ba^* b)^* (ba^* + b + \epsilon) \\ A = \epsilon \end{array} \right.$$

Dato un ASFD A esiste una espressione regolare r tale che $L(A) = L(r)$ che descrive il linguaggio riconosciuto da A .

Sia $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ un ASFD e sia L il linguaggio da esso riconosciuto.

Assumiamo $F = \{q_f\}$.

Sia $n = |Q|$ e sia $\langle q_0, \dots, q_{n-1} \rangle$ un qualunque ordinamento degli stati tale che $q_{n-1} = q_f$.

Definiamo ora come

$$R_{ij}^k \quad 0 \leq i, j \leq n-1 \quad 0 \leq k \leq n$$

l'insieme delle stringhe tali da portare A da q_i a q_j senza transitare per nessuno stato q_h con $h \geq k$.

Abbiamo cioè che $x = a_1, \dots, a_n \in R_{ij}^k$ sse:

$$\delta(q_i, x) = q_j$$

se $\delta(q_i, a_1, \dots, a_l) = q_j$ allora $i, l \leq K$ per $1 \leq l \leq m-1$.

Per $K=0$ si ha:

$$R_{ij}^0 = \left\{ \begin{array}{l} \cup \{a\} \text{ tali che } \delta(q_i, a) = q_j \text{ se ne esiste almeno uno} \\ \emptyset \text{ altrimenti} \end{array} \right.$$

Per $0 < k \leq n$, se $x \in R_{ij}^k$ è una stringa che conduce da q_i a q_j senza transitare per nessuno stato q_h con $h \geq k$, possono verificarsi due casi:

1. x conduce da q_i a q_j senza transitare neanche per q_{k-1} , dal che deriva che $x \in R_{ij}^{k-1}$

2. x conduce da q_i a q_j transitando per q_{k-1}

Nel secondo caso la sequenza degli stati attraversati può essere divisa in varie sottosequenze:

1. da q_i a q_{k-1} senza transitare per nessuno stato q_h con $h \geq k-1$, la corrispondente sottostringa di x appartiene quindi a $R_{(k-1)}^{k-1}$

2. $r \geq 0$ sequenze, ognuna delle quali inizia e termina in q_{k-1} senza transitare per nessuno stato q_h con $h \geq k-1$, le sottostringhe di x appartengono quindi ciascuna a $R_{(k-1)(k-1)}^{k-1}$

3. una sequenza finale, da q_{k-1} a q_j senza transitare per nessuno stato q_h con $h \geq k-1$
la sottostringa di x appartiene quindi a $R_{(k-1)j}^{k-1}$

Ne deriva che:

$$R_{ij}^k = R_{ij}^{k-1} \cup R_{i(k-1)}^{k-1} \circ (R_{(k-1)(k-1)}^{k-1})^* \circ R_{(k-1)j}^{k-1}$$

Dalle osservazioni precedenti deriva che è possibile costruire tutti gli insiemi R_{ij}^k a partire da $k=0$ e derivando poi man mano i successivi

$$L = R_{o(n-1)}^n$$

Ogni insieme di stringhe R_{ij}^k può essere descritto per mezzo di una espressione regolare r_{ij}^k , per $k=0$ abbiamo che:

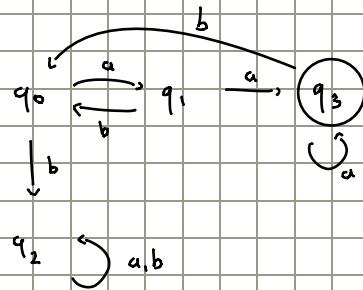
$$r_{ij}^0 = \begin{cases} a_{i1} + \dots + a_{i1} & \text{dove } \delta(q_i, a_{ik}) = q_j, k = 1, \dots, |I| \\ \emptyset & \text{se } I = \emptyset \end{cases}$$

Per $k \geq 1$ deriva che

$$r_{ij}^k = r_{ij}^{k-1} + r_{i(k-1)}^{k-1} (r_{(k-1)(k-1)}^{k-1})^* r_{(k-1)j}^{k-1}$$

L'è descritto da $r_{o(n-1)}^n$

es.



$$r_{00}^0 = \emptyset, r_{01}^0 = a, r_{02}^0 = b, r_{03}^0 = \emptyset$$

$$r_{00}' = r_{00}^0 + r_{00}^0 (r_{00}^0)^* r_{00}^0 = \emptyset + \emptyset (\emptyset)^* \emptyset = \emptyset$$

$$r_{10}^0 = b, r_{11}^0 = \emptyset, r_{12}^0 = \emptyset, r_{13}^0 = a$$

$$r_{01}' = r_{01}^0 + r_{00}^0 (r_{00}^0)^* r_{01}^0 = a + \emptyset (\emptyset)^* a = a$$

$$r_{20}^0 = \emptyset, r_{21}^0 = \emptyset, r_{22}^0 = a+b, r_{23}^0 = \emptyset$$

$$r_{02}' = r_{02}^0 + r_{00}^0 (r_{00}^0)^* r_{02}^0 = b + \emptyset (\emptyset)^* b = b$$

$$r_{30}^0 = b, r_{31}^0 = \emptyset, r_{32}^0 = \emptyset, r_{33}^0 = a$$

$$r_{03}' = r_{03}^0 + r_{00}^0 (r_{00}^0)^* r_{03}^0 = \emptyset + \emptyset (\emptyset)^* \emptyset = \emptyset$$

$$r_{10}' = r_{10}^0 + r_{10}^0 (r_{00}^0)^* r_{10}^0 = b + b (\emptyset)^* \emptyset = b$$

$$r_{11}' = r_{11}^0 + r_{10}^0 (r_{00}^0)^* r_{01}^0 = \emptyset + b (\emptyset)^* a = ba$$

$$r_{12}' = r_{12}^0 + r_{10}^0 (r_{00}^0)^* r_{02}^0 = \emptyset + b (\emptyset)^* b = bb$$

$$r_{13}' = r_{13}^0 + r_{10}^0 (r_{00}^0)^* r_{03}^0 = a + b (\emptyset)^* \emptyset = a$$

...

finisce per \emptyset
quindi annulla il
ultimo tronco

Lezione 13

Un'altra procedura è quella della **state elimination**: procedura iterativa di eliminazione degli stati su un automa non deterministico generalizzato equivalente, in cui:

- la funzione di transizione è definita su $Q \times E$, dove E è l'insieme delle espressioni regolari su Σ , per cui gli archi sono etichettati con e.r.

$$q_1 \xrightarrow{a,b} q_3, \quad q_1 \xrightarrow{a+b} q_3.$$

- lo stato iniziale non ha archi, per cui $\forall q \in Q, e \in E : q_0 \in \delta_n(q, e)$

$$q_0 \xleftarrow{\epsilon} q_1 \quad q_0 \xrightarrow{\epsilon} q_0 \xleftarrow{\epsilon} q_1$$

- esiste un solo stato finale q_f senza archi uscenti, per cui $\forall e \in E : \delta_n(q_f, e) = \emptyset$

$$q_1 \xleftarrow{\epsilon} q_f$$

$$q_1 \xleftarrow{\epsilon} q_f \xrightarrow{\epsilon} q_f$$

Dato un qualunque AFND A , un automa generalizzato \tilde{A} equivalente può essere immediatamente ottenuto:

1. mantenendo gli stati di A
2. introducendo, per ogni arco del grafo di transizione di A etichettato come l'insieme a_1, \dots, a_k , un arco nel grafo di transizione di \tilde{A} etichettato $a_1 + \dots + a_k$
3. se lo stato iniziale q_0 di A ha archi entranti, introducendo in \tilde{A} un nuovo stato iniziale \bar{q}_0 senza archi entranti, e la ϵ -transizione $\delta_{\tilde{N}}(\bar{q}_0, \epsilon) = \{q_0\}$
4. se esistono più stati finali in F , o se il solo stato finale ha archi uscenti, introducendo un ulteriore stato q_f , ponendo $F = \{q_f\}$ e introducendo la ϵ -transizione $\delta_{\tilde{N}}(q_f, \epsilon) = \{q_f\} \quad \forall q \in F$

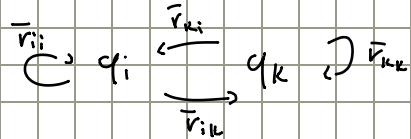
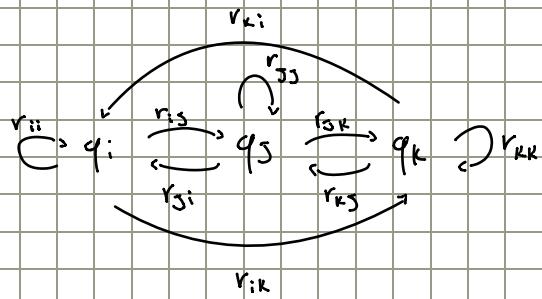
Dato un AFND (con ϵ -transizioni) A con insieme di stati Q , e dato uno stato q non iniziale né finale, è possibile ottenere un automa generalizzato equivalente \tilde{A} con stati $Q - \{q\}$ effettuando una opportuna operazione di **eliminazione dello stato**.

L'eliminazione dello stato viene effettuata considerando tutti i possibili cammini di lunghezza 3 passanti per q (sequenze q_i, q_j, q_s per le quali esistono archi da q_i a q_j e da q_j a q_s)

Per ogni cammino, le etichette degli archi interessanti vengono modificate come mostrato di seguito.

Al termine, rimangono lo stato iniziale e quello finale, collegati da un arco, la cui etichetta fornisce l'espressione regolare cercata.

es.



Lezione 16

Dato l'insieme dei linguaggi regolari dimostriamo le sue proprietà di chiusura
(Dato un insieme S e un'operazione \circ , il risultato dell'operazione \circ fornisce elementi di S)
rispetto a varie operazioni \circ :

se \circ è unaria, $L \circ L$ è un linguaggio regolare $\forall L$ regolare.

se \circ è binaria, $L_1 \circ L_2$ è un linguaggio regolare per ogni coppia L_1, L_2 di ling. regolari.
non consideriamo operatori di aritma maggiore.

Le dimostrazioni hanno lo stesso schema, a partire da un ASFD che riconoscono i linguaggi dati, viene derivato un automa che riconosce il linguaggio risultante.

Chiusura: unione

Dati $L_1, L_2, L_1 \cup L_2$ è un linguaggio regolare

Sia $A_1 = \langle \Sigma_1, Q_1, \delta_{N_1}, q_0, F_1 \rangle$ e $A_2 = \langle \Sigma_2, Q_2, \delta_{N_2}, q_0, F_2 \rangle$ due ASFD che accettano L_1 e L_2 si costruisce $A = \langle \Sigma, Q, \delta_N, q_0, F \rangle$ che riconosce $L_1 \cup L_2$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

$$Q = Q_1 \cup Q_2 \cup \{q_0\}$$

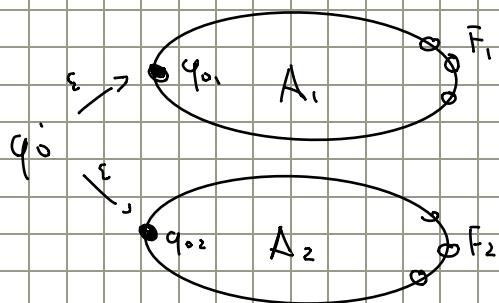
$$F = F_1 \cup F_2 \cup F_1 \cup F_2 \cup \{q_0\} \text{ se } A_1, A_2 \text{ riconoscono } \epsilon$$

$$\delta_N(q, a) = \delta_{N_1}(q, a) \text{ se } q \in Q_1, a \in \Sigma_1$$

$$\delta_N(q, a) = \delta_{N_2}(q, a) \text{ se } q \in Q_2, a \in \Sigma_2$$

$$\delta_N(q_0, a) = \delta_{N_1}(q_0, a) \cup \delta_{N_2}(q_0, a), a \in \Sigma$$

es.



Chiusura: complemento

Dato un linguaggio regolare L , \bar{L} è un linguaggio regolare.

Sia $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ un ASFD che riconosce L con δ funzione totale:

$\bar{A} = \langle \Sigma, Q, \delta, q_0, \{Q - F\} \rangle$ riconosce \bar{L} .

Le stringhe che portano A in uno stato finale portano \bar{A} in uno stato non finale.

Chiusura: intersezione

Dati due linguaggi regolari L_1, L_2 , $L_1 \cap L_2$ è un linguaggio regolare

$$L = L_1 \cap L_2 = \overline{\overline{L}_1} \cup \overline{\overline{L}_2}$$

Chiusura: concatenazione

Dati due linguaggi regolari L_1, L_2 , $L = L_1 \circ L_2$ è un linguaggio regolare

Sia $A_1 = \langle \Sigma_1, Q_1, \delta_{N_1}, q_{01}, F_1 \rangle$ e $A_2 = \langle \Sigma_2, Q_2, \delta_{N_2}, q_{02}, F_2 \rangle$ due ASFD che accettano

$L_1 \circ L_2$ si costruisce $A = \langle \Sigma, Q, \delta_N, q_0, F \rangle$ che riconosce $L_1 \circ L_2$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

$$Q = Q_1 \cup Q_2$$

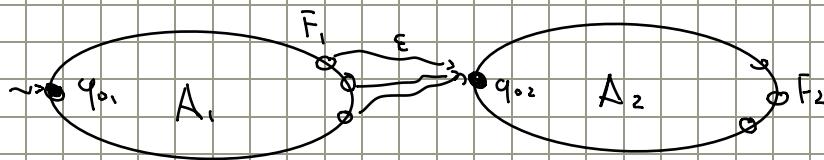
$$F = \begin{cases} F_2 \\ F_1 \cup F_2 \text{ altrimenti} \end{cases}$$

$$q_0 = q_{01}$$

$$\delta_N(q, a) = \delta_1(q, a), \forall q \in Q_1 - F_1, a \in \Sigma_1$$

$$\delta_N(q, a) = \delta_1(q, a) \cup \delta_2(q_{01}, a), \forall q \in F_1, a \in \Sigma_1$$

$$\delta_N(q_0, a) = \delta_2(q_0, a), \forall q \in Q_2, a \in \Sigma_2$$



Chiusura: iterazione

Dato un linguaggio regolare L , L^* è un linguaggio regolare.

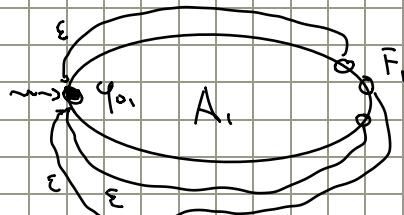
Sia $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ un ASFD che accetta L . Costruiamo un automa

$\tilde{A} = \langle \Sigma, Q \cup \{q_0\}, \tilde{\delta}, q_0, F \cup \{q_0\} \rangle$ che riconosce il linguaggio L^* :

$$\tilde{\delta}(q, a) = \delta(q, a), \forall q \in Q - F$$

$$\tilde{\delta}(q, a) = \delta(q, a) \cup \delta(q_0, a), \forall q \in F$$

$$\tilde{\delta}(q_0, a) = \delta(q_0, a)$$



Decidibilità di predici

E' possibile decidere se il linguaggio accettato da un dato automa a stati finiti è vuoto.

Oss: se un automa con n stati accetta qualche stringa, allora accetta una stringa di lunghezza inferiore a n .

Sia z la stringa più breve accettata dall'automa, se fosse $|z| \geq n$ allora, in base al pumping lemma, z potrebbe essere scritta come uvw e anche la stringa uw , più breve di z , sarebbe accettata dall'automa.

Dato un automa A con $n = |Q|$ stati e $s = |\Sigma|$ simboli di input, per decidere se $L(A) = \Delta$ basta verificare se A accetta almeno una delle stringhe di lunghezza inferiore a n .

$$\sum_{i=0}^{n-1} s^i = \frac{s^n - 1}{s - 1} = \Theta(s^{n-1})$$

Il tempo necessario per tale operazione sarà proporzionale a: $\Theta(s^n)$

Oss: dato il grafo di transizione di A , $L(A)$ è non vuoto sse esiste almeno uno stato finale raggiungibile da q_0 .

Algoritmo polinomiale: visita del grafo di transizione, avete n nodi e $m \leq ns$ archi. La visita richiede tempo $\Theta(n+m) = \Theta(n)$ lineare nel numero di stati.

E' possibile decidere se il linguaggio accettato da un dato automa a stati finiti è finito

Proprietà:

1: se A accetta una z lunga almeno n allora, per il pumping lemma, accetta infinite stringhe.

2: se $L(A)$ è infinito, allora esiste $z \in L(A)$ tale che $n \leq |z| < 2n$.

Quindi $L(A)$ è infinito sse $\exists z \in L(A): n \leq |z| < 2n$

Dato un automa A con n stati e s simboli di input, per decidere se $L(A)$ è infinito basta verificare se A accetta almeno una delle stringhe di lunghezza compresa tra n e $2n$ $\Theta(s^{2n-1})$

L'algoritmo ha quindi complessità esponenziale.

Oss: dato il grado di transizione di A, $L(A)$ è infinito se esiste almeno un ciclo a partire da uno stato su un cammino da q_0 a uno stato finale.

Il problema dell'equivalenza di due linguaggi regolari è decidibile.
Basta dimostrare che la loro intersezione coincide con l'unione.

Lezione 15

Ogni stringa sufficientemente lunga appartenente ad un linguaggio regolare presenta delle regolarità: contiene una sottostringa che può essere ripetuta quanto si vuole, ottenendo sempre stringhe del linguaggio.

Pumping Lemma

Sia L un linguaggio regolare: allora $\exists n > 0 : \forall z \in L : |z| \geq n$ si può scrivere $z = uvw$ con $|uvw| \leq n$, $|v| \geq 1$ e ottenere che $\forall i \geq 0 \quad uv^iw \in L$.

- Interpretazione

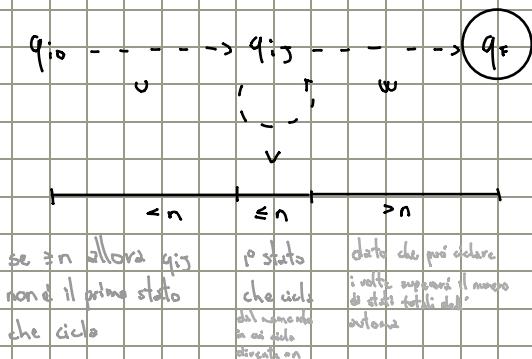
Se L è regolare, A vince sempre con B:

1. A fissa un intero $n > 0$ opportuno
2. B sceglie una stringa $z \in L$ con $|z| > n$
3. A divide z in tre parti uvw con $|uvw| \leq n$ e $|v| \geq 1$
4. B sceglie un intero $i \geq 0$
5. A mostra che $uv^iw \in L$

- Dim

Se L è regolare, sia A l'ASFD che lo decide e che ha il minimo numero n di stati.

Una stringa $z \in L$ di lunghezza $m \geq n$ in input a A gli fa eseguire m transizioni e quindi attraversare $m+1 > n$ stati, quindi esiste almeno uno stato che viene attraversato più volte.



Evidenzia il fatto che gli automi finiti non possono contare. Il numero di situazioni diverse che posso memorizzare è finito. Fornisce soltanto una condizione necessaria perché un linguaggio sia regolare: non può essere utilizzato per mostrare la regolarità di un linguaggio, ma solo per dimostrarne la non regolarità.

L regolare \Rightarrow proprietà quindi \neg proprietà $\Rightarrow \neg L$ regolare

Sia L un linguaggio e supponiamo che $\forall n > 0$ si abbia che $\exists z \in L : |z| \geq n$: comunque dividiamo z in $z = uvw$ con $|uvw| = n$, $|v| \geq 1$, $\exists i \geq 0$: $uv^iw \notin L$. Allora, L non è regolare.

- Interpretazione

Se L non è regolare, A vince sempre con B:

1. B fissa un intero $n > 0$ opportuno
2. A sceglie una stringa $z \in L$ con $|z| > n$
3. B divide z in tre parti uvw con $|uvw| \leq n$ e $|v| \geq 1$
4. A sceglie un intero $i \geq 0$ e mostra che $uv^iw \notin L$

es. Mostrare che $L = a^k b^k$, $k > 0$ non è regolare

1. B fissa un intero $n > 0$
2. A sceglie la stringa $z = a^n b^n$
3. B divide z in tre parti con $|uvw| \leq n$ e $|v| \geq 1$, $uv = a^h$ $1 \leq h \leq n$, $v = a^l$ $1 \leq l \leq h$ $w = a^{n-h} b^n$
4. A sceglie $i = 2$ $uv^2 w = a^{h+1} a^l a^{n-h} b^n = a^{n+1} b^n \notin L$

Lezione 16

L'ASFD con minimo numero di stati che riconosce un dato linguaggio L può essere derivato partizionando l'insieme Q degli stati di un automa che riconosce L in classi di equivalenza rispetto alla relazione

$$q_i \equiv q_j \iff (\forall x \in \Sigma^* \quad \delta(q_i, x) \in F \iff \delta(q_j, x) \in F)$$

Quindi $q_i \equiv q_j$ se ogni stringa che porta da q_i ad uno stato finale porta anche q_j ad uno stato finale o viceversa.

relazione di equivalenza

Se $q_i = q_j$ i due stati sono detti **indistinguibili**.

Se esiste una stringa $x \in \Sigma^*$ per cui $\bar{\delta}(q_i, x) \in F$ e $\bar{\delta}(q_j, x) \in Q - F$ (o viceversa) q_i e q_j sono **distinguibili** tramite x .

La costruzione è basata sul teorema di **Myhill-Nerode**:

Dato un linguaggio $L \subseteq \Sigma^*$ definiamo la relazione di equivalenza R_L su Σ^* come:

$$x R_L y \iff (\forall z \in \Sigma^* \quad xz \in L \iff yz \in L)$$

L è regolare sse R_L partiziona Σ^* in un numero finito di classi di equivalenza.

L regolare:

Sia $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ un automa che lo riconosce (assumiamo $F = \{q_f\}$ unico stato finale)

A permette di definire una nuova relazione R_A tra stringhe in Σ^* , in cui

$$x R_A y \iff \bar{\delta}(q_0, x) = \bar{\delta}(q_0, y)$$

R_A è una relazione di equivalenza, con un numero di classi al più pari a $|Q|$ di A quindi $i(R_A) \leq |Q|$.

Se $\bar{\delta}(q_0, x) = \bar{\delta}(q_0, y)$ allora $\forall z \in \Sigma^* \quad \bar{\delta}(q_0, xz) = \bar{\delta}(q_0, yz)$ e quindi $xz R_A yz$.

Quindi $\forall z \in \Sigma^*$ se $\bar{\delta}(q_0, xz) = q_f$ anche $\bar{\delta}(q_0, yz) = q_f$ è viceversa per cui $xz \in L \iff yz \in L$

Ne deriva che $x R_A y \iff x R_L y$ e quindi $i(R_A) \geq i(R_L)$.

Dato che $i(R_A) \leq |Q|$ è finito anche $i(R_L)$ lo è.

$i(R_L)$ finito:

Definiamo un ASF $A' = \langle \Sigma, Q', \delta', q'_0, F' \rangle$ dove:

Q' associa a ogni classe $[x]$ in R_L uno stato $q_{[x]}$

$$q'_0 = q_{[\epsilon]}$$

$$F' = \{q_{[x]} \mid x \in L\}$$

$$\delta'(q_{[x]}, a) = q_{[xa]} \quad \forall a \in \Sigma$$

Per costruzione $\forall [x]$ di R_L si ha che $\bar{\delta}'(q'_0, z) = q_{[xz]} \quad \forall z \in [x]$, quindi se $y, z \in [x]$ allora

$y R_L z$ e in generale $x R_L y \iff x R_L z$ da cui $i(R_L) \geq i(R_A)$, dalla definizione di F' , ne

consegue immediatamente che A' riconosce L , se $q_{[x]} \in F$ allora $x \in L$ e lo stesso vale $\forall z \in [x]$.

Q' è minimo.

Se così non fosse, $\exists x, y \in \Sigma^*$, non equivalenti rispetto a R_L e per cui $\bar{\delta}'(q_0, x) = \bar{\delta}'(q_0, y) = q$. Ma allora ne conseguirebbe che $\forall z \in \Sigma^* : \bar{\delta}(q_0, xz) = \bar{\delta}(q_0, yz)$ il che comporterebbe che $x R_L y$ contrariamente all'ipotesi.

Minimizzazione di un ASFD:

- Individuazione di tutte le coppie di stati indistinguibili
 - Unificazione degli stati equivalenti, eliminando quelli non raggiungibili e modificando opportunamente la funzione di transizione.

Ipotesi: Tutti gli stati di A sono raggiungibili dallo stato iniziale, altrimenti si eliminano gli stati irraggiungibili.

Per marcare le coppie di stati distinguibili si utilizza una tabella contenente una casella per ciascuna coppia (non ordinata) di stati di Q .

Le caselle vengono usate per marcire le coppie di stati distinguibili e per elencare, in una lista associata, tutte le coppie che dovranno essere marcute qualora la coppia a cui è associata la casella venga marcata.

si considera soltanto la matrice triangolare inferiore o superiore perché simmetrica, e non si considera la diagonale perché $q_i \cdot q_j = 0$ se $i \neq j$.

q_2 $\begin{pmatrix} q_1, q_1 \\ q_4, q_5 \end{pmatrix}$ se questa casella verrà marcata come distinguibile allora anche gli elementi all'interno lo saranno.

La procedura inizia con la marcatura delle coppie distinguibili tramite la stringa ϵ (tutte e solo le coppie costituite da uno stato finale e uno non finale)

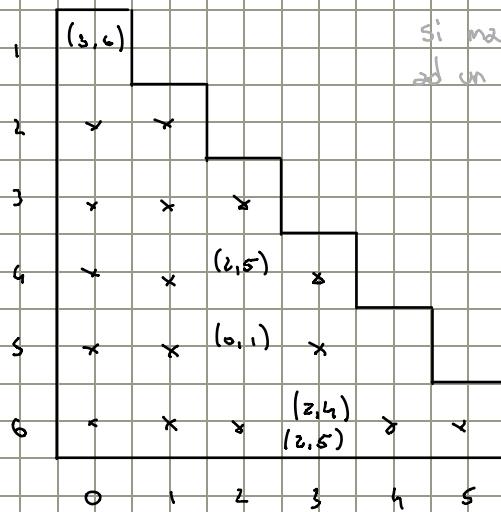
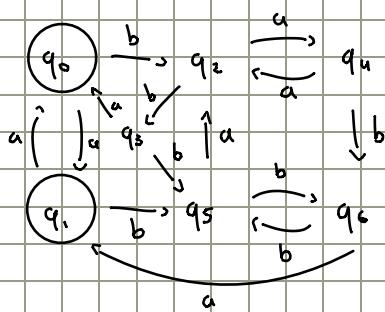
Per ogni coppia (p,q) non ancora marcata, si considerano, $\forall a \in \Sigma$, tutte le coppie (r,s) , con $r = \delta(p,a)$ e $s = \delta(q,a)$:

Se nessuna delle coppie (r,s) è marcata come distinguibile allora si inserisce (p,q) nella lista associata ad ognuna di esse.

altrimenti per quel vengono riconosciuti distinguibili e la casella viene marcata; qualora questa contenga una lista di coppie si marcano anche quelle.

Una volta identificate le coppie di stati indistinguibili, l'automa equivalente con il minimo numero di stati è dato evidentemente da $A' = \langle \Sigma, Q', \delta', q_0, F' \rangle$ in cui:
 Q' è costruito selezionando, per ogni insieme di stati indistinguibili, un solo stato di Q ;
 F' è costituito da tutti i rappresentanti appartenenti ad F ;
 δ' è ottenuta da δ mediante restrizione del dominio $Q \times \Sigma$ ed inoltre, $\forall \delta(q_i, a) = q_j$, con $q_i \in Q'$ e $q_j \in Q$, poniamo $\delta'(q_i, a) = q_k$, dove $q_k \in Q'$ è il rappresentante dell'insieme di stati indistinguibili che include q_j .

es.



si marcano tutte le coppie che si collegano ad un finale. (0, 1) non tra due stati finali!

$q_0 q_1$ a
 $q_2 q_4$ b
 $q_2 q_5$ b
 $q_3 q_6$ c
 $q_4 q_5$ b

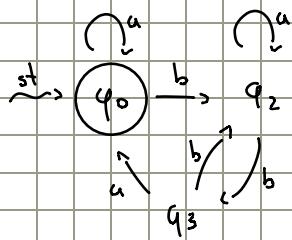
3 classi di eq:

1. $q_0 q_1$

2. $q_2 q_4 q_5$

3. $q_3 q_6$

si hanno quindi tre stati:



Lezione 17

La derivazione di una stringa generata da una grammatica di tipo 2 (**context free**) può essere rappresentata mediante una struttura ad albero. Tali alberi vengono chiamati **alberi di derivazione** o **alberi sintattici**.

In un albero sintattico, ad ogni nodo interno è associato un simbolo non-terminale e ad ogni foglia è associato un simbolo terminale. Per ogni produzione del tipo $S \rightarrow aSbA$ che viene applicata nel processo di derivazione, il nodo interno etichettato con S avrà nell'albero quattro figli etichettati con a, S, b, A .

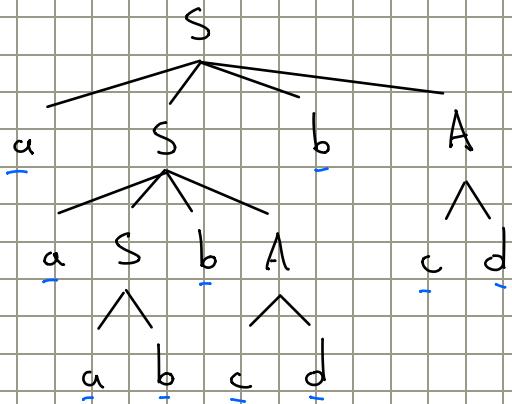
Dato la grammatica G con produzioni:

$$S \rightarrow aSbA \cup b$$

$$A \rightarrow cAd \cup cd$$

$$V_n = (V_T, V_N)^+$$

$aaabbcdabcd \in L(G)$



$$S \Rightarrow aSbA \Rightarrow aaSbAbA \Rightarrow aaabbAbA$$

$$\Rightarrow aaabbcd bA \Rightarrow aaabbcd bcd$$

(si toglie sempre il simbolo non terminale che si trova a sinistra questo tipo di derivazione si chiama derivazione sinistra)

$$S \Rightarrow aSbA \Rightarrow aSb \Rightarrow aaSbAbd \Rightarrow aaabbcdcd \Rightarrow aaabbcdcd \quad (\text{der. dx})$$

Non si mantiene traccia dell'ordine con cui le produzioni sono state applicate. Ad un unico albero possono corrispondere diverse derivazioni. Il vantaggio di un albero di derivazione è che fornisce una descrizione sintetica della struttura sintattica della stringa, indipendentemente dall'ordine con cui le produzioni sono state applicate.

Al fine di studiare alcune proprietà dei linguaggi generati da queste grammatiche, è utile considerare grammatiche "ristrette", comprendenti soltanto produzioni con struttura particolare. E' importante dimostrare che i linguaggi CF possono essere generati mediante tali tipi di grammatiche.

Una grammatica G è in forma ridotta se:

1. non contiene ϵ -produzioni (se contiene ϵ -prod., in corrispondenza all'ass. non compariranno simboli non terminali: ✓ $S \rightarrow \epsilon | ab$, ✗ $S \rightarrow \epsilon | aA$)
2. non contiene produzioni unitarie: $A \rightarrow B \quad A, B \in V_n$
3. non contiene simboli inutili, cioè simboli che non compaiono in nessuna derivazione di una stringa di simboli terminali: $S \rightarrow aSb | C, C \rightarrow aC | cc$

Trasformazione di una grammatica $G_1 = \langle V_t, V_n, P, S \rangle$ di tipo 2 in una grammatica equivalente in forma ridotta mediante sequenze di passi.

1. A partire da G_1 derivazione di G_1 di tipo 2 senza ϵ -produzioni tale che:
 $L(G_1) = L(G_1) - \{\epsilon\}$.
2. A partire da G_1 derivazione di G_2 di tipo 2 senza ϵ -produzioni e senza produzioni unitarie tale che: $L(G_2) = L(G_1)$

3. A partire da G_2 derivazione di G_3 di tipo 2 senza ϵ -produzioni e senza produzioni unitarie e senza simboli inutili tale che: $L(G_3) = L(G_2)$
4. La grammatica G_n , di tipo 2, equivalente a G coincide con G_3 se $\epsilon \in L(G_3)$; altrimenti G_n è ottenuta da G_3 introducendo un nuovo assioma ed un nuovo opportuno insieme di produzioni su tale simbolo.

Passo 1 (eliminazione ϵ -produzioni)

Dato 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 $L(G') = L(G) - \{\epsilon\}$.

Determinazione dell'insieme $N \subseteq V_N$ dei simboli che si annullano, cioè i non terminali da cui è possibile derivare ϵ in G . $A \xrightarrow{*} \epsilon$

Costruzione di una sequenza $N_0, \dots, N_k = N$ di sottoinsiemi di V_N , con

$N_0 = \{A \in V_N \mid A \rightarrow \epsilon \in P\}$ e N_{i+1} derivato da N_i :

$$A \rightarrow \epsilon \quad N_i$$

$$N_{i+1} = N_i \cup \{B \in V_N \mid (B \rightarrow \beta \in P) \wedge (\beta \in N_i^*)\}.$$

$$\begin{array}{l} A \rightarrow \epsilon \quad N_0 \\ B \rightarrow A \quad N_i \end{array}$$

La costruzione termina quando $N_{k+1} = N_k$, $k \geq 0$. $\epsilon \in L(G)$ sse $S \in N$.

Costruzione dell'insieme P' delle produzioni di G' :

Si esamina ciascuna produzione $A \rightarrow \alpha$ di P , con l'esclusione delle ϵ -produzioni. Se nessun simbolo di α è annullabile: $A \rightarrow \alpha$ è inserito in P' . Altrimenti α contiene $k > 0$ simboli che si annullano: sono inserite in P' tutte le possibili produzioni ottenute da $A \rightarrow \alpha$ eliminando da α uno dei sottoinsiemi di simboli che si annullano.

Nel caso in cui $L(G)$ contiene ϵ , si può ottenere da G' una grammatica equivalente a G tramite la semplice introduzione di una ϵ -produzione sull'assioma di G' .

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

$$S \rightarrow A \mid SS_a$$

$$- N_0 = \{A\}$$

$$- S \rightarrow A \mid SS_a \mid S_a \mid a \mid \epsilon$$

$$A \rightarrow B \mid Ab \mid \epsilon$$

$$- N_1 = \{A, S\}$$

$$A \rightarrow B \mid Ab \mid b$$

$$B \rightarrow S \mid ab \mid a \mid b$$

$$- N_2 = \{A, S, B\} = N$$

$$B \rightarrow S \mid ab \mid a \mid b$$

AlSSa|Sa|Saa

Lezione 18

Passo 2 (eliminazione produzioni unitarie)

Per ogni grammatica G di tipo 2 senza ϵ -produzioni, esiste sempre una grammatica G' di tipo 2 senza ϵ -produzioni, priva di produzioni unitarie ed equivalente a G .

Sia, $\forall A \in V_n$, $U(A)$, il sottoinsieme di $V_n - \{A\}$ comprendente tutti i non terminali derivabili da A applicando una sequenza di produzioni unitarie,

$$U(A) = \{B \in V_n - \{A\} \mid A \xrightarrow{*} B\}$$

Data la grammatica $G = \langle V_T, V_n, P, S \rangle$, P' è costruito:

inserendo dapprima in P' tutte le produzioni non unitarie in P e inserendo in P' , \forall non terminale A e $\forall B \in U(A)$, la produzione $A \rightarrow B$ sse in P esiste una produzione non unitaria $B \rightarrow B$.

$$\begin{array}{lll} V \rightarrow W & U(V) = \{W\} & V \rightarrow cWd | cd \\ W \rightarrow cWd | cd & & W \rightarrow cWd | cd \end{array}$$

Passo 3 (cancellare tutte le produzioni che non vanno in simboli terminali anche tramite $\xrightarrow{*}$)

Per ogni grammatica $G = \langle V_T, V_n, P, S \rangle$ di tipo 2 senza ϵ -produzioni e senza produzioni unitarie, esiste sempre una grammatica G' di tipo 2 senza ϵ -produzioni, priva di produzioni unitarie e simboli inutili ed equivalente a G .

Affinché un simbolo $A \in V_n$ non sia inutile, è necessario che nella grammatica G si abbia che:

- A sia un simbolo **secondo**, vale a dire che da esso siano generabili stringhe di terminali, cioè $\exists w \in V_T^*$ tale che $A \xrightarrow{*} w$;
- A sia generabile dall'assimone in produzioni che non contengono simboli non secondi, cioè $S \xrightarrow{*} \alpha A \beta$ con $\alpha, \beta \in (V_T \cup V_n)^*$ e, $\forall B \in V_n$ in $\alpha \circ \beta$, valga la proprietà precedente.

Equivalentemente, un simbolo $A \in V_n$ non è inutile se esiste una derivazione $S \xrightarrow{*} \alpha A \beta \xrightarrow{*} w \in V_T^*$.

Un non terminale A è secondo sse vale una delle due condizioni seguenti:

1. esiste $w \in V_T^+$ tale che $A \rightarrow w \in P$;
2. esiste $\alpha \in (V_N \cup V_T)^*$ tale che $A \rightarrow \alpha \in P$ e tutti i simboli non terminali in α sono secondi.

E' necessario verificare che i simboli rimasti siano generabili a partire dall'assioma.

Cio può essere effettuato in modo iterativo, osservando che A è generabile a partire da S se vale una delle due condizioni seguenti:

1. esistono $\alpha, \beta \in (F \cup V_T)^*$ tali che $S \rightarrow \alpha A \beta \in \hat{P}$;
2. esistono $\alpha, \beta \in (F \cup V_T)^*$ e $B \in F$, generabile a partire da S , tali che $B \rightarrow \alpha A \beta \in \hat{P}$.

Al fine di eliminare i simboli inutili (non secondi e non generabili da S) è necessario applicare i due algoritmi nell'ordine dato: eliminare prima i simboli non generabili e poi quelli non secondi può far sì che non tutti i simboli inutili vengano rimossi dalla grammatica.

Infatti, si consideri la grammatica

$$S \rightarrow AB | a \xrightarrow{\text{diventa}} S \rightarrow a \quad \text{A si cancella perché non è derivabile da nulla}$$

$$A \rightarrow a$$

Passo 4 (cancellare simboli inutili)

Una grammatica $G = \langle V_T, V_N, P, S \rangle$ può essere estesa in una grammatica $G' = \langle V_T, V'_N, P', S' \rangle$ che genera anche la stringa vuota e nel seguente modo:

1. $V'_N = V_N \cup \{T\}$, dove $T \notin V_N$;
2. $P' = P \cup \{T \sim, \varepsilon\} \cup \{T \sim, \alpha \mid S \sim, \alpha \in P\}$;
3. $S' = T$

$$S \sim aUVb|Tz$$

$$Z \sim aZ \xleftarrow{\text{cancello}}$$

$$U \sim bV|b$$

$$V \sim cWd|cd|ay$$

$$Y \sim bY|b$$

$$W \sim cWd|cd$$

$$T \sim fT|t_2$$

Simb second = {U, V, W, Y, T, S}

diventa

$$\begin{array}{lll}
 S \rightarrow aUVb \quad |T \text{ cancella} & Y \rightarrow bY|b & d_2 S \rightarrow U,V \rightarrow W \quad |d_2 V \\
 J \rightarrow bU|b & W \rightarrow cWd|cd & \\
 V \rightarrow cWd|cd \quad |uY & T \rightarrow fT|t_2 & \text{diventa} \\
 & \text{cancella} & \\
 S \rightarrow aUVb & Y \rightarrow bY|b & \\
 J \rightarrow bU|b & W \rightarrow cWd|cd & \\
 V \rightarrow cWd|cd \quad |uY & &
 \end{array}$$

Forma normale di Chomsky (\circ solo prod V_n o solo V_r)

Una grammatica di tipo 2 si dice in **forma normale di Chomsky** se tutte le sue produzioni sono del tipo $A \rightarrow BC$ o del tipo $A \rightarrow a$, con $A, B, C \in V_n$ ed $a \in V_r$.

Data una grammatica G non contestuale tale che $\epsilon \in L(G)$, esiste una grammatica equivalente in CNF.

Come mostrato, è possibile derivare una grammatica G' in forma ridotta equivalente a G : in particolare, G' non ha produzioni unitarie.

Da G' , è possibile derivare una grammatica G'' in CNF, equivalente a essa.

Sia $A \rightarrow T_1, \dots, T_m$ una produzione di G' non in CNF. Si possono verificare due casi:

- $n \geq 3$ e $T_{ij} \in V_n$, $j = 1, \dots, n$.

In tal caso, introduciamo $n-2$ nuovi simboli non terminali: Z_1, \dots, Z_{n-2} e sostituiamo la produzione $A \rightarrow T_1, \dots, T_m$ con le produzioni

$$A \rightarrow T_{ij} Z_1, \quad Z_1 \rightarrow T_{i1} Z_2, \dots, Z_{n-2} \rightarrow T_{in} T_m$$

$$\begin{aligned}
 A &\rightarrow BCBD \quad \text{diventa} \quad A \rightarrow BZ_1 \\
 Z_1 &\rightarrow CZ_2 \\
 Z_2 &\rightarrow BD
 \end{aligned}$$

- $n \geq 2$ e $T_{ij} \in V_r$ per qualche $j \in \{1, \dots, n\}$.

In tal caso per ciascun $T_{ij} \in V_r$ introduciamo un nuovo non terminale \bar{T}_{ij} , sostituendo Z_j a T_{ij} nella produzione considerata e aggiungiamo la produzione $\bar{T}_{ij} \rightarrow T_{ij}$.

Così facendo o abbiamo messo in CNF la produzione considerata (se $n=2$) o ci siamo ricondotti al caso precedente.

$$A \rightarrow aub \quad \text{diventa}$$

$$A \rightarrow X_a X_b X_b$$

$$A \rightarrow X_a Z_1$$

$$X_a \rightarrow a$$

$$Z_1 \rightarrow X_b X_b$$

$$X_b \rightarrow b$$

es:
G tipo 2 $\{a^n b^n \mid n \geq 1\}$

$S \rightarrow aSb$ già in forma ridotta
 $S \rightarrow ab$ diventa
 $S \rightarrow X_a Z_1$,
 $Z_1 \rightarrow S X_b$,
 $S \rightarrow X_a X_b$,
 $X_a \rightarrow a$,
 $X_b \rightarrow b$

Lezione 19

Pumping Lemma

Sia $L \subseteq V_T^*$ un linguaggio non contestuale. Esiste allora una costante n tale che se $z \in L$ e $|z| \geq n$ allora esistono 5 stringhe $u, v, w, x, y \in V_T^*$ tali che

- 1 $uv^iwxy = z$
- 2 $|vx| \geq 1$
- 3 $|vwx| \leq n$
- 4 $\forall i \geq 0 \quad uv^iwx^iy \in L$

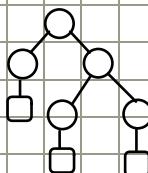
- Interpretazione

Se L è context free, Alice vince sempre questo gioco con Bob:

1. Alice fissa un intero $n > 0$ opportuno
 2. Bob sceglie una stringa $z \in L$ con $|z| > n$
 3. Alice divide z in cinque parti uv^iwxy con $|vwx| \leq n$ e $|vx| \geq 1$
 4. Bob sceglie un intero $i \geq 0$
 5. Alice mostra che $uv^iwx^iy \in L$
- Jim

Grammatica $G = \langle V_T, V_N, P, S \rangle$ in CNF che genera $L = L(G)$ e sia $K = |V_N|$ il numero di simboli non terminali in G .

Qualunque albero sintattico $A(s)$ relativo ad una stringa $s \in V_T^*$ derivata in G sarà tale da avere tutti i nodi interni (simboli non terminali) di grado 2, eccetto quelli aventi 2 figlie dell'albero come figli, che hanno grado 1. es:



Se h è l'altezza di A , il massimo numero di foglie $l(h)$ è dato dal caso in cui l'albero è completo. Si può facilmente verificare che in tal caso abbiamo $l(h) = 2^{h-1}$.

Se l'albero sintattico $A(\sigma)$ relativa alla stringa $\sigma \in L$ ha altezza $h(\sigma)$, la lunghezza di σ è allora $|\sigma| \leq 2^{h(\sigma)-1}$, e quindi $h(\sigma) \geq 1 + \log_2 |\sigma|$.

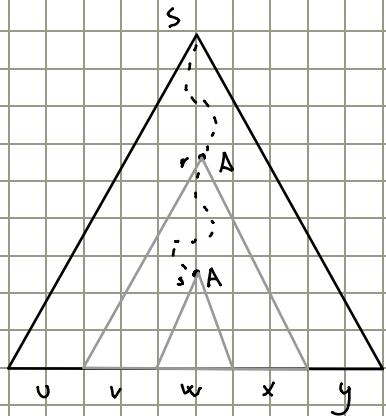
Se σ è una stringa sufficientemente lunga ($> 2^{|V_n|-1}$) ne risulta che $h(\sigma) \geq 1 + \log_2 |\sigma| > |V_n|$. Quindi, se $|\sigma| > 2^{|V_n|-1}$ esiste almeno un cammino c dalla radice ad una foglia di $A(\sigma)$ che attraversa almeno $|V_n|+1$ nodi esterni.

I nodi interni di $A(\sigma)$ sono etichettati da simboli non terminali.

Dato che i simboli non terminali sono $|V_n|$ mentre i nodi interni in c sono più di $|V_n|$, deve esistere un simbolo non terminale A che compare in due diversi nodi di c .

Di questi due nodi, indichiamo con r il nodo più vicino alla radice e con s il nodo associato ad A più vicino alla foglia.

Indichiamo con $r(\sigma)$ e $s(\sigma)$ le sottostringhe σ corrispondenti alle foglie dei due sottoalberi $R(\sigma), S(\sigma)$ di $A(\sigma)$ aventi radice r e s .



Dato che s è discendente di r , necessariamente $s(\sigma)$ è una sottostringa di $r(\sigma)$, per cui esistono due sottostringhe di v, x di σ tali che $r(\sigma) = v \cdot s(\sigma) \cdot x$

La grammatica considerata è in CNF, per cui non sono presenti produzioni unitarie, di conseguenza, non può essere $s(\sigma) = r(\sigma)$ e quindi $|v_x| \geq 1$.

Il cammino più lungo da $r(\sigma)$ ad una foglia attraversa al più $|V_n|+1$ nodi interni (r incluso).

Dalle osservazioni, ne deriva che $r(\sigma)$ ha lunghezza al più $2^{|V_n|+1-1} = 2^{|V_n|}$

Gli alberi $R(\sigma)$ e $S(\sigma)$ possono essere sostituiti l'uno all'altro all'interno di un qualunque albero sintattico.

Quindi anche la stringa $uvwy$ è generata dalla grammatica. Anche la stringa $uvvwxxy$ lo è.
 La proprietà mostrata fornisce una condizione necessaria di un linguaggio context free:
 non può essere utilizzata per mostrare la contestualità di un linguaggio, ma solo per dimostrarne la non contestualità.

$$\begin{array}{ll} L \text{ contestuale} & \Rightarrow \text{pump. lemma verificato} \\ \text{pump. lemma non verificato} & \Rightarrow L \text{ non contestuale} \end{array}$$

Se A vince allora L non è CF:

1. B fissa un intero $n > 0$ opportuno
2. A sceglie una stringa $z \in L$ con $|z| > n$
3. B divide z in cinque parti $uvwxy$ con $|vwx| \leq n$ e $|vx| \geq 1$
4. A sceglie un intero $i \geq 0$
5. B mostra che $uv^iwx^i y \notin L$

es: Mostrare che $L = \{a^k b^k c^k \mid k \geq 0\}$ non è CF

1. B sceglie un $n > 0$
2. A sceglie $a^n b^n c^n \in L$
3. B divide $uvwxy$ con $|vwx| \leq n$ e $|vx| \geq 1$
4. A sceglie $i = 2$
5. A mostra che $uv^2wx^2y \notin L$ in quanto almeno un simbolo ha aumentato il numero di occorrenze ed almeno un altro ha un numero di occorrenze invariato.

a	b	c
$\overbrace{}^n$	$\overbrace{}^n$	$\overbrace{}^n$

$|vwx| \leq n$ quindi o è composta da un solo simbolo o da due sottosequenze di stessi simboli, un simbolo verrà lasciato sempre fermo.

Proprietà dei linguaggi context free:

Chiusura intersezione:

Il linguaggio $L = \{a^n b^n c^n \mid n \geq 1\}$ non è context free.

es: $L_1 = \{a^n b^n c^m \mid n, m \geq 1\}$ e $L_2 = \{a^n b^m c^m \mid n, m \geq 1\}$ sono contestuali.

Ma $L = L_1 \cap L_2$ non è chiusa rispetto all'intersezione.

Chiusura: unione

Dati due linguaggi CF $L_1 \subseteq \Sigma^*$ e $L_2 \subseteq \Sigma^*$, siano $G_1 = \langle \Sigma_1, V_{N_1}, P_1, S_1 \rangle$ e $G_2 = \langle \Sigma_2, V_{N_2}, P_2, S_2 \rangle$ due grammatiche di tipo 2 tali che $L_1 = L(G_1)$ e $L_2 = L(G_2)$.

Il linguaggio $L = L_1 \cup L_2$ potrà allora essere generato dalla grammatica di tipo 2

$G = \langle \Sigma_1 \cup \Sigma_2, V_{N_1} \cup V_{N_2} \cup \{S\}, P, S \rangle$ dove $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 | S_2\}$.

es:

$$\begin{array}{ll} x \in L_1 & S_1 \xrightarrow{*} x \\ y \in L_2 & S_2 \xrightarrow{*} y \end{array} \quad \text{quindi} \quad S \xrightarrow{*} S_1 | S_2$$

$$S \xrightarrow{*} S_1 \xrightarrow{*} x$$

$$S \xrightarrow{*} S_2 \xrightarrow{*} y$$

Chiusura: concatenazione

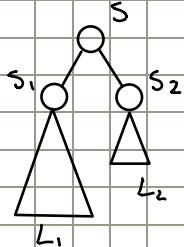
Dati due linguaggi CF $L_1 \subseteq \Sigma^*$ e $L_2 \subseteq \Sigma^*$, siano $G_1 = \langle \Sigma_1, V_{N_1}, P_1, S_1 \rangle$ e

$G_2 = \langle \Sigma_2, V_{N_2}, P_2, S_2 \rangle$ due grammatiche di tipo 2 tali che $L_1 = L(G_1)$ e $L_2 = L(G_2)$.

Mostriamo che $L = L_1 \circ L_2$ è generato dalla grammatica di tipo 2 definita come

$G = \langle \Sigma_1 \cup \Sigma_2, V_{N_1} \cup V_{N_2} \cup \{S\}, P, S \rangle$ dove $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}$.

es:



Chiusura: iterazione

Dato un linguaggio CF $L \subseteq \Sigma^*$, sia $G = \langle \Sigma, V_N, P, S \rangle$ una grammatica di tipo 2 tale che $L = L(G)$.

Il linguaggio $L' = L^*$ è allora generato dalla grammatica di tipo 2 $G' = \langle \Sigma, V_N \cup \{S'\}, P', S' \rangle$ con $P' = P \cup \{S' \rightarrow SS' | \varepsilon\}$

$$\begin{array}{ll} \text{es: } S' & S' \\ / \backslash & | \\ S & S' \\ \Delta & \Delta \\ / \backslash & / \backslash \\ S & S' \\ \vdots & \vdots \\ L & \end{array}$$

Chiusura: complemento

La classe dei linguaggi CF non è chiusa rispetto al complemento.

Se cosifosse, avremmo che dati due qualunque linguaggi CF L_1, L_2 il linguaggio

$L = \overline{L_1 \cup L_2}$ sarebbe CF anch'esso ma $L = L_1 \cap L_2$, ciò non sussiste.

Decidibilità

Dato una grammatica G di tipo 2 è decidibile stabilire se $L(G) = \emptyset$.

Assumiamo che $G = \langle V_T, V_N, P, S \rangle$ sia in CNF.

Per il pumping lemma, se esiste una stringa $z = uvwxy \in L(G)$ con $|z| \geq 2^{|V_N|}$, allora esiste una stringa $z' = uwxy \in L(G)$ con $|z'| < 2^{|V_N|}$. Quindi se il linguaggio non è vuoto, esiste una stringa in esso di lunghezza minore di $2^{|V_N|}$.

In una grammatica CNF ogni applicazione di una produzione o incremento di uno la lunghezza della forma di frase ($A \rightarrow BC$) o sostituisce un terminale a un non terminale ($A \rightarrow a$). Quindi, una stringa di lunghezza 2^k è generata da una derivazione di lunghezza $2^{k+1} - 1$.

Per verificare se esiste una stringa di lunghezza minore di $2^{|V_N|}$ generabile, è sufficiente considerare tutte le derivazioni di lunghezza minore di $2^{|V_N|+1} - 1$ che sono, al più

$$\sum_{k=1}^{2^{|V_N|+1}-2} |P|^k = \frac{|P|^2^{|V_N|+1}-1}{|P|-1} = \mathcal{O}(2^{2^{|V_N|+1}})$$

Un metodo più efficiente consiste nel portare la grammatica in forma ridotta, verificando se esistono simboli fecondi. Condizione necessaria e sufficiente affinché il linguaggio sia vuoto è che la grammatica non abbia simboli fecondi.

Dato una grammatica G di tipo 2 è decidibile stabilire se $L(G)$ è infinito.

Assumiamo che $G = \langle V_T, V_N, P, S \rangle$ sia in CNF.

Per il pumping lemma, se esiste una stringa $z = uvwxy \in L(G)$ con $|z| \geq 2^{|V_N|}$, allora esiste una stringa $z_i = u v^i w x^i y \in L(G)$ con $i \geq 0$, e almeno una di queste ha lunghezza $|z'| < 2^{|V_N|+1}$.

Quindi, se il linguaggio è infinito, esiste una stringa in esso di lunghezza $z \in [2^{|V_N|}, 2^{|V_N|+1} - 1]$, che sarà derivata (in una grammatica CNF) da una derivazione di lunghezza compresa tra $2^{|V_N|+1} - 1$ e $2^{|V_N|+2} - 3$. Di questa lunghezza ci sono

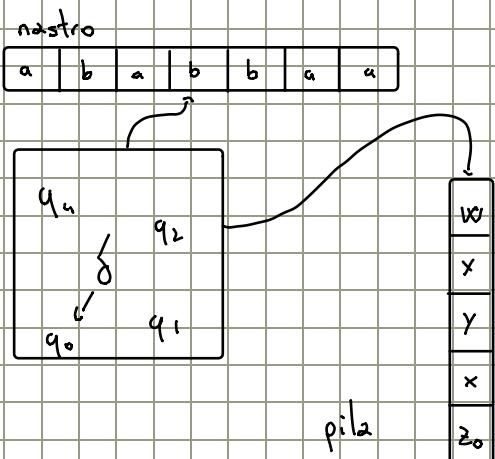
$$\sum_{k=2^{|V_N|+1}-3}^{2^{|V_N|+2}-3} |P|^k = \frac{|P|^2^{|V_N|+2}-2 - |P|^2^{|V_N|+1}-1}{|P|-1} = \mathcal{O}(2^{2^{|V_N|+2}}).$$

Metodo più efficiente: verificare la ciclicità del grafo orientato $G = (N, A)$ derivato dalla grammatica in CNF che genera L , ponendo $N = V_N$ e introducendo, per ogni produzione $B \rightarrow CD$, gli archi $\langle B, C \rangle$ e $\langle B, D \rangle$.

Automi a pila

Un automa a pila (o automa push-down) è definito come una setteupla

$M = \langle \Sigma, \Gamma, z_0, Q, q_0, F, \delta \rangle$ dove Σ è l'alfabeto di input, Γ è l'alfabeto dei simboli della pila, $z_0 \in \Gamma$ è il simbolo iniziale della pila, Q è un insieme finito e non vuoto di stati, $q_0 \in Q$ è lo stato iniziale, $F \subseteq Q$ è l'insieme degli stati finali, $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^* \rightarrow Q \times \Gamma^*$ è la funzione (parziale) di transizione.



La presenza di ϵ -transizioni può comportare che esistano continuazioni diverse di una stessa computazione anche in presenza di uno stesso carattere letto, si può dire quindi che l'automa introdotto non è deterministico. Per ottenere un comportamento deterministico dobbiamo fare l'ulteriore ipotesi che se, per una coppia $q \in Q, z \in \Gamma$, è definita $\delta(q, \epsilon, z)$ allora la funzione di transizione $\delta(q, a, z)$ non deve essere definita per nessun $a \in \Sigma$.

Ad ogni passo l'automa, a partire dallo stato attuale, dal carattere letto sul nastro di input e dal carattere affiorante sulla pila, sostituisce il simbolo affiorante nella pila con una stringa di caratteri e si porta in un nuovo stato.

Dato un automa a pila $M = \langle \Sigma, \Gamma, z_0, Q, q_0, F, \delta \rangle$, una configurazione di M è data dalla tripla $\langle q, x, \gamma \rangle$, dove $q \in Q, x \in \Sigma^*$ e $\gamma \in \Gamma^*$.

Sia $M = \langle \Sigma, \Gamma, z_0, Q, q_0, F, \delta \rangle$ un automa a pila e siano (q, x, γ) e (q', x', γ') due configurazioni di M : avremo allora che $(q, x, \gamma) \xrightarrow{M} (q', x', \gamma')$ sse valgono le tre condizioni:

1. esiste $a \in \Sigma$ tale che $x = ax'$;
 2. esistono $z \in \Gamma$ e $n, s \in \Gamma^*$ tali che $\gamma = zn$ e $\gamma' = sn$;
 3. $\delta(q, a, z) = (q', s)$;
- oppure le tre condizioni:
1. $x = x'$
 2. esistono $z \in \Gamma$ e $n, s \in \Gamma^*$ tali che $\gamma = zn$ e $\gamma' = sn$;
 3. $\delta(q, \varepsilon, z) = (q', s)$

Una computazione è definita come una sequenza c_0, \dots, c_k di configurazioni di M tale che $c_i \xrightarrow{M} c_{i+1}$.

Sia M un automa a pila. Una configurazione (q, x, γ) di M è di accettazione se $x = \gamma = \varepsilon$. Secondo tale definizione, una stringa x è quindi accettata da M sse al termine della scansione della stringa la pila è vuota. Indichiamo con $N(M)$ il linguaggio accettato per pila vuota dall'automa M .

Sia M un automa a pila. Una configurazione (q, x, γ) di M è di accettazione se $x = \varepsilon$ e $q \in F$. Secondo tale definizione, una stringa x è quindi accettata da M sse al termine della scansione della stringa l'automa si trova in uno stato finale. Indichiamo con $L(M)$ il linguaggio accettato per stato finale dell'automa.

Un automa a pila non deterministico è definito come una settuple $M = \langle \Sigma, \Gamma, z_0, Q, q_0, F, \delta \rangle$ dove Σ è l'alfabeto di input, Γ è l'alfabeto dei simboli della pila, $z_0 \in \Gamma$ è il simbolo iniziale della pila, Q è un insieme finito e non vuoto di stati, $q_0 \in Q$ è lo stato iniziale, $F \subseteq Q$ è l'insieme degli stati finali, $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$ è la funzione (parziale) di transizione.

Nel caso degli automi a pila la presenza del non determinismo comporta un aumento del potere computazionale.

Mentre gli automi a pila non deterministici riconoscono la classe dei linguaggi non contestuali, gli automi a pila deterministici riconoscono un sottoinsieme proprio di tali linguaggi, la classe dei linguaggi non contestuali deterministici.
es:

$\{w\tilde{w} \mid w \in \{a,b\}^*\}$ dove \tilde{w} indica la stringa riflessa di w abbabbba

Tabella transizione per pila vuota:

	A	B	Z_0
a	a (q_0, AA) (q_1, ϵ)	b (q_0, BA) (q_1, ϵ)	a (q_0, AB) (q_1, ϵ)
b	b (q_0, BB) (q_1, ϵ)	a (q_0, BA) (q_1, ϵ)	b (q_0, BB) (q_1, ϵ)

Tabella transizione per stato finale ($q_f = q_2$):

	A	B	Z_0	
a	a (q_0, AA) (q_1, ϵ)	b (q_0, BA) (q_1, ϵ)	a (q_0, AB) (q_1, ϵ)	b (q_0, BB) (q_1, ϵ)
b	b (q_0, BB) (q_1, ϵ)	a (q_0, BA) (q_1, ϵ)	b (q_0, BB) (q_1, ϵ)	ϵ (q_0, Z_0) (q_1, ϵ)

Teorema

Dato un automa a pila non deterministico $M = \langle \Sigma, \Gamma, Z_0, Q, q_0, F, \delta \rangle$ che accetta un linguaggio per stato finale, esiste un automa a pila non deterministico $M' = \langle \Sigma, \Gamma, Z_0, Q, q_0, F, \delta \rangle$ che accetta lo stesso linguaggio per pila vuota, vale a dire che $L(M) = N(M')$.

Vale anche il contrario.

Decidibilità e chiusura LR

E' possibile decidere se il linguaggio accettato da un automa a stati finiti è vuoto o finito:

$L(\Delta) = \emptyset$ è vuoto sse non esiste nessun stato finale raggiungibile da q_0 .

$L(\Delta) = \infty$ è infinito sse esiste almeno un ciclo a partire da uno su un cammino da q_0 a un q_f .

Chiusura:

\overline{L}_1 : L_1 è regolare

L_1 regolare $\Rightarrow \exists$ ASF A_1 che ha q_f come stati finali

$\overline{L}_1 = A_1$ con $q_f = Q \setminus q_f$

\cup : $L_1 \cup L_2$ è regolare

Δ_1 riconosce L_1

Δ_2 riconosce L_2

\cap : $L_1 \cap L_2$ è regolare

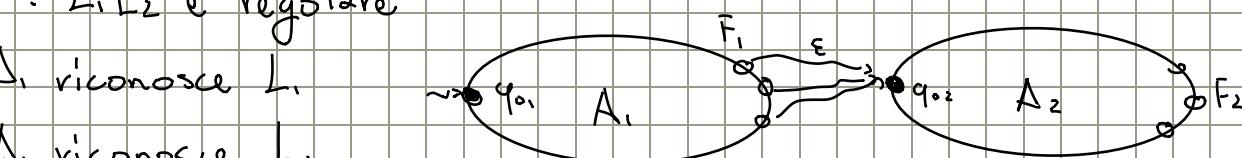
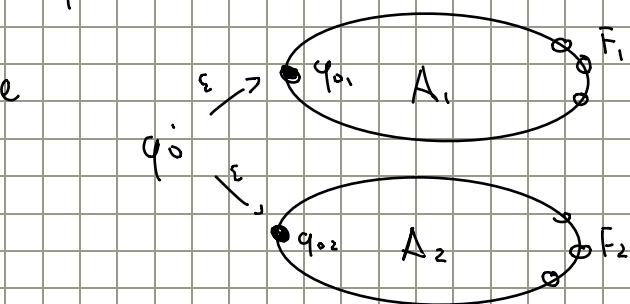
essendo chiuso rispetto a unione e complemento basta fare de morgan

$$L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$$

\circ : $L_1 L_2$ è regolare

Δ_1 riconosce L_1

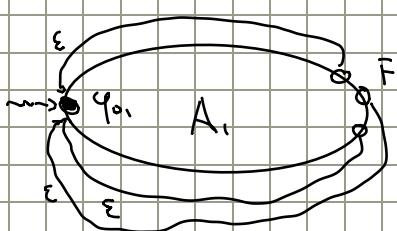
Δ_2 riconosce L_2



$*$: L_1^* è regolare

Δ_1 riconosce L_1

Δ_2 riconosce L_2



Decidibilità e chiusura CF

Dato una grammatica G di tipo 2 è decidibile stabilire se $L(G) = \emptyset$ o $L(G) = \infty$

$L(G) = \emptyset$ portare la grammatica in CNF e verificare che la grammatica non abbia simboli secondi e raggiungibili

$L(G) = \infty$ portare la grammatica in CNF e verificare la ciclicità del grafo orientato $G = (N, A)$, ponendo $N = V_N$ e introducendo, per ogni prod. $B \rightarrow C D$ gli archi $\langle B, C \rangle$ e $\langle B, D \rangle$

(chiusura:

\cap : $L_1 \cap L_2$ non è CF $L_1 = \{a^n b^n c^m \mid n, m \geq 1\}$ e $L_2 = \{a^m b^n c^n \mid n, m \geq 1\}$

L_1 e L_2 sono CF ma la loro intersezione no $L = \{a^n b^n c^n \mid n \geq 1\}$ è CF

\cup : $L_1 \cup L_2$ è CF

L_1 e L_2 sono CF allora $L_1 \cup L_2$ sarà CF con $S \rightarrow C_1 \mid C_2$

\overline{L} : non è CF

non essendo chiuso rispetto a intersezione, basta fare de Morgan e scrivere $L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$ ma ciò non è possibile

\cdot : $L_1 L_2$ è CF

L_1 e L_2 sono CF allora $L_1 L_2$ è CF con $S \rightarrow C_1 C_2$

$*$: L_1^* è CF

L_1 è CF allora anche L_1^* è CF con $S \rightarrow S' \mid a$

Ambiguità

Una grammatica è detta ambigua se $\exists w \in L$: w è derivabile da due alberi sintattici diversi.

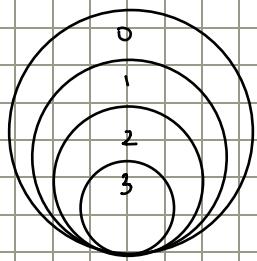
$$S \rightarrow A B / a B a B$$

$$A \rightarrow a A / a A$$

$$B \rightarrow b$$

Gerarchia Chomsky

Ogni linguaggio di tipo n contiene tutti i linguaggi di tipo $n+1$



- 0: $\alpha \rightarrow \beta \quad \alpha \in V^* V_n V^*, \beta \in V^*$
- 1: $\alpha \rightarrow \beta \quad \alpha \in V^* V_n V^*, \beta \in V^*, |\alpha| \leq |\beta|$
- 2: $A \rightarrow \beta \quad A \in V_n, \beta \in V^*$
- 3: $A \rightarrow \delta \quad A \in V_n, \delta \in (V_T \cup (V_T V_n))^*$