Fondamenti di informatica
9.10.19
paragrafo 1.3.2 AAG

operazioni coi linguaggi = operazioni con gli insiemi (unione, intersezione, complemento [A^C], etc.)
$\Sigma = \{0,1\}$
$L = \{0, 00, 10\}$ $L^C = L^C$ in Σ^* (tutti gli elementi di Σ^* che non appartengono a L)
$L^C = \{\varepsilon, 000, 0000, 1111, 11, 110\ldots$ infiniti$\}$

Concatenazione

Dati due linguaggi L1 e L2 sottoinsiemi di Σ^*, la **concatenazione** di L1 e L2 è così definita:
$L1 \circ L2 = \{x1 \circ x2 \mid x1 \in L1, x2 \in a\ L2\}$
es.
$L1 = \{0, 00\}$
$L2 = \{10, 11\}$
$L1 \circ L2 = \{010, 011, 0010, 0011\}$
*(cioè 0 ° 10, 0 ° 11, 00 ° 10, 00 ° 11, **DIVERSO** da 10 ° 0, 11 ° 0, 10 ° 00, 11 ° 00)*

Concatenazione con ε

se $\varepsilon \in L2$ allora L1 è sottoinsieme di $L1 \circ L2$
se $L2 = \emptyset$, $L1 \circ L2 = \emptyset$ (non ci sono stringhe da concatenare)
L'elemento neutro è il linguaggio $L = \{ \varepsilon \}$ ($L1 \circ L = L \circ L1 = \mathbf{L1}$)

Concatenazione di potenze di parola

se $L1 = \{a^n \mid n >= 0\}$
$L2 = \{b^n \mid n >= 0\}$
$L1 \circ L2 = \{a^n b^m \mid n, m >= 0\}$
$L3 = \{a^n b^n \mid n >= 0\}$ è un sottoinsieme di $L1 \circ L2$

Potenza di un linguaggio (usa concatenazione)

La potenza L^h di un linguaggio è definita come $L^h = L \circ L^{h-1}$ con $h >= 1$ e premettendo che $L^0 = \{ \varepsilon \}$
$L = \{aa, ab\}$
$L^0 = \{ \varepsilon \}$
$L^1 = L \circ L^{h-1} = L \circ L^0 = L \circ \varepsilon = L$
$L^2 = L \circ L^1 = L \circ L = \{aaaa, aaab, abaa, abab\}$
$L3 = L \circ L2 = \{aaaaaa, aaabaa, abaaaa, ababaa, \ldots\}$

Star di Kleene

Dato un linguaggio L, la **star di Kleene di L** o **chiusura riflessiva del linguaggio L** rispetto alla concatenazione è **L*** ed è uguale all'unione (per n>=0) di L^n.

$$L^* = \bigcup_{n>0}^{\infty} L^n = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots L^n$$

se L = $\{Ø^0\}$, L* = { ε }
se L = $\{Ø^n\}$, L* = {Ø }
se L = { ε }, L* = { ε }

Star positiva di Kleene

Dato un linguaggio L, **la chiusura di L o star positiva di Kleene** è:

$$L+ = \bigcup_{n=1}^{\infty} L^n = L^1 \cup L^2 \cup L^3 \cup \dots L^n$$

$L^* = L^+ \cup \varepsilon$
$L^+ =/= L^*/\varepsilon$
$L^+ = L^*/\varepsilon$ solo se ε appartiene a L
es.
L = {ε, aa, ab}
L^+ = {ε, aa, aaab, ababab … }

L = {aa}
L* = {ε, aa, aaaa, aaaaaa … } = $\{(aa)^n \mid n >= 0\}$
L^+ = {aa, aaaa, aaaaaa … }

Rappresentazione dei linguaggi

Ci sono metodi riconoscitivi e generativi per rappresentare un linguaggio.
Generativi = si parte dall'assioma, si seguono le regole e si ottengono tutte le stringhe del linguaggio
Riconoscitivi = si inserisce da input una stringa e si controlla se viene individuata nel linguaggio

Metodi riconoscitivi: riconoscitori/automi (**MODELLO TEORICO DI CALCOLATORE**) (pag 58, par 2.5)

Il riconoscitore è un "dispositivo" che è in grado di ottenere una stringa da input e dire se è presente nel linguaggio o meno.
Il cuore del riconoscitore è un controllo che si può trovare in un determinato stato q.
Per elaborare l'input il riconoscitore è dotato di nastri (array che sono potenzialmente infiniti in entrambe le direzioni) divisi in celle che contengono ognuna un simbolo dell'alfabeto.
Una cella vuota si chiama **BLANK** (B o ƀ).
Ad ogni nastro corrisponde una testina di lettura o scrittura o entrambi che si può muovere a sinistra, destra o entrambe (dipende dal modello di riconoscitore), che **comunica dati di una sola cella alla volta** al controllo (es. Automi a stati finiti possono solo leggere).

Negli automi a stati finiti, il riconoscitore, in uno stato iniziale q0, inizia la computazione, fino a quando non raggiunge una **BLANK, una configurazione particolare**, dove la computazione finisce e se lo stato corrente non è finale, la stringa non viene accettata, mentre se è uno stato finale (**cioè quando non c'è più altro da computare**) la stringa è accettata.

16/10/2019

Stato iniziale q0 appartenente a Q
Stati finali F sottoinsieme di Q

Un riconoscitore è costituito da una serie di nastri infiniti in entrambe le direzioni divisi per celle che contengono un solo valore per volta

I riconoscitori ammettono un numero finito di stati diversi
Negli automi a stati finiti le testine scorrono solo verso destra e possono solo leggere e non scrivere

CONFIGURAZIONE DI UN RICONOSCITORE
La "fotografia" del riconoscitore ad un determinato momento della computazione (es. configurazione iniziale = stato q0, stringa sulle prime x celle del nastro, testina puntata sulla prima cella)
Una configurazione è composta da 3 elementi: stato del controllo, contenuto del nastro, posizione della testina.

CONFIGURAZIONI DI ACCETTAZIONE
nel caso di automi a stati finiti
- il controllo è in uno stato Q appartenente all'insieme F degli stati finali
- la stringa x è succeduta da BLANK
- la testina si trova sulla prima posizione successiva alla stringa x.

CONFIGURAZIONE SUCCESSIVA $C_i \rightarrow C_j$
se in C_i
- il controllo è in uno stato q_i
- la stringa x è sul nastro
- la testina si trova in posizione i-esima del nastro
allora nella successiva C_j
- il controllo è in uno stato q_j dettato dalla funzione di transizione, o anche rimanere invariato
- la stringa x è sul nastro
- la testina si trova in posizione i+1 del nastro

Una computazione di un riconoscitore a partire da una configurazione iniziale C_0 si può vedere come una successione di configurazioni $C_0, C_1, C_2 \ldots$ tale che $C_i \vdash C_{i+1}$ per ogni $i >= 0$.
Una computazione termina se la successione di configurazioni è finita e non esiste alcuna configurazione successiva a C_n
Se termina in una configurazione di accettazione, allora si dirà computazione di accettazione.

Dato un riconoscitore A e una stringa in input x, se indichiamo con C0(x) la configurazione iniziale di A sull'input x, diremo che A accetta x se e solo se esiste una computazione di accettazione C tale che da $C0(x) \vdash_A C$.

Linguaggio riconosciuto: qualunque sia l'input la computazione termina
Linguaggio accettato: ci sono stringhe per cui la computazione termina, e altre per cui va avanti all'infinito

AUTOMA A STATI FINITI (pag 72)

Un automa a stati finiti deterministico (ASFD) è una quintupla $A = (\Sigma, Q, \delta$ [delta], q0, F) dove:

$\Sigma = \{a1, a2, \ldots an\}$ cioè l'alfabeto in input
$Q = \{q0, \ldots qm\}$ cioè l'insieme finito e non vuoto di stati
$q0 \in Q$ cioè lo stato iniziale
F sottoinsieme di Q è l'insieme degli stati finali
$\delta: Q \times \Sigma \rightarrow Q$ cioè la funzione di transizione

23/10/2019

$\Sigma=\{a,b\}$ $Q=\{q0, q1, q2\}$ $F=\{q1\}$
la stringa abb è accettata?

TABELLA DI TRANSIZIONE

δ	**a**	**b**
q0	q0	q1
q1	q2	q2
q2	q2	q2

a	b	b	BLANK
↑			
q0			

a	b	b	BLANK
	↑		
	q0		

a	b	b	BLANK
		↑	
		q1	

a	b	b	BLANK
			↑
			q2

Le stringhe accettate sono solo quelle stringhe formate da una sequenza qualsiasi di a (anche vuota) concatenata a b. (es. aaaaaaab, ab, aaab, aaaaab, b)

Definizione formale di computazione a stati finiti e linguaggio accettato da un A.S.F.D.

Dato un A.S.F.D. A = <Σ, Q, δ [delta], q0, F> :
una configurazione di A è una coppia (q,x) con q Є Q e x Є Σ*.
x = contenuto del nastro a partire dalla posizione della testina in poi

Data una configurazione (q,x), si dice:

- configurazione iniziale se **q = q0**;
- configurazione finale se **x = ε**;
- configurazione di accettazione/accettante se **q Є F, x = ε**.

La configurazione (q,ε) è/può essere sia iniziale, sia finale, sia accettante.

Dato un ASFD e due configurazioni (q,x) (q',y), si scrive (q,x) |---- (q',y) **se e solo se valgono le seguenti condizioni:**

1. esiste un simbolo **a** Є Σ tale che **x = ay**;
2. δ(q,a) = q'

Sequenza di configurazioni successive
esiste C0, C1, C2, … Cn con n>=0 tale che Ci = C0 |-- C1 |-- C2 |-- … |-- Cn = Cj

Una stringa appartenente a Σ* è accettata da un ASFD quando, partendo dalla configurazione (q,x), riesco a raggiungere una configurazione (q,**ε**) con q Є F:
(q0,x) |*-- (q,**ε**), q Є F

Dato un ASFD il linguaggio accettato da ***A*** (l'automa) è:
L(***A***) = {x Є Σ* | (q0,x) |*-- (q,**ε**), q Є F}

ESERCIZI
Scrivere un ASFD che accetti un linguaggio di tutte le stringhe in {a,b} che hanno b come secondo carattere.

Dimostrare che delta segnato di q,a è uguale a delta di q,a per qualsiasi coppia di q,a.

Funzione di transizione estesa

Associa a una coppia stato-**stringa** (non stato-simbolo come la funzione di transizione) uno stato.

La funzione di transizione estesa di un ASFD ***A*** = <Σ, Q, δ [delta], q0, F> è la funzione δ : Q x Σ* → Q così definita:

1. δ (q,**ε**) = q [per ogni **q Є Q**]
2. δ (q,xa) = δ(δ (q,x),a) [per ogni **x Є Σ*, a Є Σ**]

con la tabella di transizione:

δ	**a**	**b**
q0	q0	q1
q1	q2	q2

q2 q2 q2

con ab=x e b=a,
δ (q0, abb) = δ(δ (q0,ab),b)
con a=x e b=a
δ (q0,ab) = δ(δ (q0,a),b)
con **ε**=x e a=a
δ (q0,a) = δ (q0,**ε**a) = δ(δ (q0,**ε**),a) = δ(q0,a) = q0
↑
[δ (q0,**ε**) = q0]
dunque,
δ (q0,ab) = δ(δ (q0,a),b) = δ(q0,b) = **q1**
δ (q0, abb) = δ(δ (q0,ab),b) = δ(q1,b) = q2

30/10/2019

per ogni q **Є** Q e a **Є Σ**, δ(q,a) = δ(q,a)

Scrivere un ASFD che accetti un linguaggio di tutte le stringhe in {a,b} che hanno b come secondo carattere.

E= {a,b} Q= {q0, q1, q2, q3} F= {q3}

δ	**a**	**b**
q0	q1	q1
q1	q2	q3
q2	q2	q2
q3	q3	q3

Funzione totale: se è definita per ogni elemento dell'insieme di partenza

Si può definire linguaggio riconosciuto da un ASFD utilizzando il concetto di funzione di transizione estesa.
* = numero di passi (0, 1 … n)
L(A) = {x **Є Σ* | (q0,x) |*-- (q,e) con q Є F**}

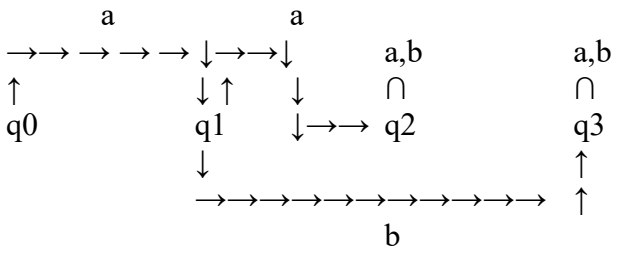
Inoltre si può dimostrare che q' = δ(q,x) se e solo se esiste y **Є Σ*** tale che (q, xy) |*-- (q', y)
L(A) = {x **Є Σ* |** δ(q0,x) **Є F}**

Scrivere δ(q,x) in funzione della funzione di transizione soprastante.

DIAGRAMMA DEGLI STATI DI UN ASFD

fare esercizio 1.8, 3.3 pag 77
E' un altro modo di rappresentare un ASFD.
Il diagramma degli stati è un grafo che rappresenta gli stati dell'automa tramite vertici (o nodi).

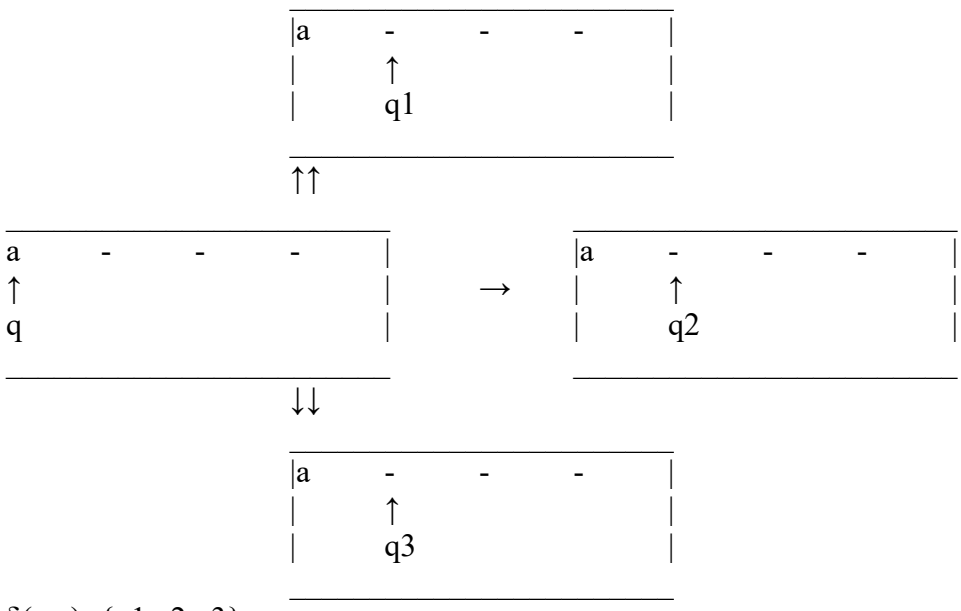
Abaa è l'etichetta del cammino che fa la computazione da q0 a q3 (a→b→ a→ a)



La classe/insieme di tutti i linguaggi riconosciuti da ASFD prende il nome di **classe/insieme dei linguaggi regolari.**

AUTOMI A STATI FINITI NON DETERMINISTICI (ASFND)

Ritorniamo al concetto generale di riconoscitore: (par. 2.5.2)
La differenza è che una configurazione può avere diverse configurazioni successive.



δ(q,a)={q1,q2,q3}

Una stringa viene accettata se esiste almeno una successione di configurazioni che termina in una configurazione di accettazione.
Tutto ciò che viene computato con un ASFND si può computare anche con un ASFD.

DEFINIZIONE DI AUTOMI A STATI FINITI NON DETERMINISTICI (ASFND)

Un automa a stati finiti non deterministico (ASFND) è una quintupla $A_N = <\Sigma, Q, \delta_N, q0, F>$ dove:

- **Σ, Q, q0, F sono definiti come per gli ASFD;**
- **$\delta_N : Q \times \Sigma \rightarrow P(Q)$ (insieme delle parti, cioè l'insieme dei sottoinsiemi di Q). È una funzione detta funzione di transizione.**

P(Q) = { {Ø}, {q0}, {q1}, {q2}, {q0, q1}, {q0, q2}, {q1, q2}, {q0, q1, q2} }

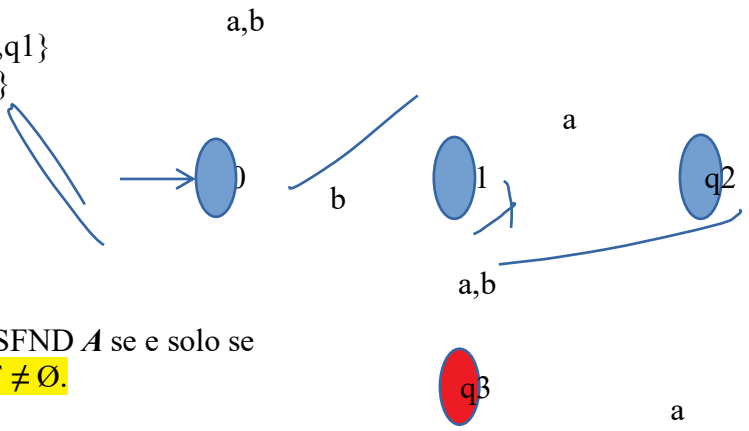
La funzione di transizione assegna ad uno stato-simbolo **un insieme di stati.**

06/11/2019

La computazione di un ASFND può essere rappresentata con uno schema ad albero: la radice è la configurazione iniziale, che si dirama in tante configurazioni successive.
L'ASFND permette di riconoscere più insiemi di linguaggi rispetto a un ASFD, quindi si dice più potente.
Più in generale, **tra due dispositivi di tipo diverso tra loro, quello più potente è quello che riconosce un numero più grande di insiemi di linguaggi.**
Tuttavia questa definizione non considera le risorse (spazio e tempo) utilizzate dai dispositivi.
Inoltre, il modello non deterministico è più semplice da progettare e richiede solitamente meno stati rispetto a un ASFD.
Tutto ciò che viene computato con un ASFND si può computare anche con un ASFD, in quanto un ASFD è un caso particolare di ASFND che ammette al più (solo) una configurazione successiva.
Esistono alcuni modelli di automi a pila non deterministico che riconoscono linguaggi che non vengono riconosciuti da uno deterministico.

es. 3.5 – input: bba (F = {q3})

δ_N	a	b
q0	{q0}	{q0,q1}
q1	{q2,q3}	{q3}
q2	{q3}	{Ø}
q3	{Ø}	{Ø}



Una stringa x è accettata da un ASFND ***A*** se e solo se
(q0,x) |*-- (P, epsilon) dove P ∩ F ≠ Ø.

({q0}, bba) |--- ({q0,q1}, ba) |--- ({q0,q1,q3}, a) |--- ({q0,q1,q3}, epsilon)

Dato un ASFND, il linguaggio accettato dall'automa L={x ϵ **Σ*** | (q0,x) |*-- (P, ε) con P sottoinsieme di Q, P∩F ≠ Ø}

Funzione di transizione estesa di un ASFND
Dato un ASFND, la funzione di transizione estesa è una funzione $\delta_N: Q \times \Sigma^* \rightarrow P(Q)$

1. $\delta_N (q, e) = \{q\}$

2. $\delta_N (q, xa) = \bigcup_{p \in \delta_N (p, x)} \delta_N (p, a)$

es. (xa)

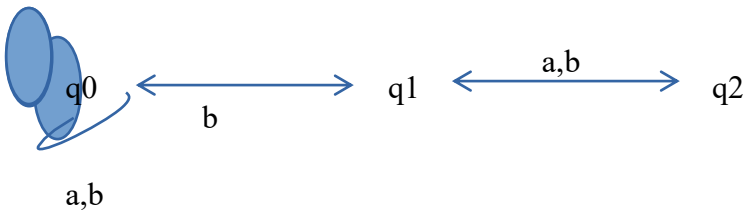
$\delta_N (q0, bba) = \bigcup_{p \in \delta_N (q0, bb)} \delta_N (p, a)$

$\delta_N (q0, bb) = \bigcup_{p \in \delta_N (q0, b)} \delta_N (p, b)$

$\delta_N (q0, \varepsilon b) = \bigcup_{p \in \delta_N (q0, \varepsilon)} \delta_N (p, b) = \bigcup_{p \in \{q0\} \text{ oppure } p = \{q0\}} \delta_N (p, b) = \delta_N (q0, b)$

Il linguaggio riconosciuto da un ASFND è:
$L(A)=\{x \in \Sigma^* \mid (\delta_N (q0, x)) \cap F \neq \emptyset\}$

Scrivere un ASFND che riconosce tutte le stringhe che contengono b come ultimo carattere.



Esercizio 3.5 (ausiello pag.79)
Trovare il linguaggio riconosciuto dall'ASFND in figura
E' il linguaggio di stringhe nell'alfabeto {a, b} che terminano con "baa", "ba" o "bb"/hanno come suffisso "baa", "ba" o "bb".

13/11/19

Definizione di suffisso e prefisso

Siano x, y, z appartenenti a E*:

- x è prefisso di xy
- y è suffisso di xy
- x è sottostringa di yxz.

Per ogni x appartenente a E*, epsilon e x sono prefissi, suffissi e sottostringhe di x.

Il prefisso/suffisso/sottostringa può essere proprio o improprio.
I prefissi/suffissi/sottostringhe propri sono tutti i prefissi/suffissi/sottostringhe diversi da epsilon e x, quelli impropri sono epsilon e x.

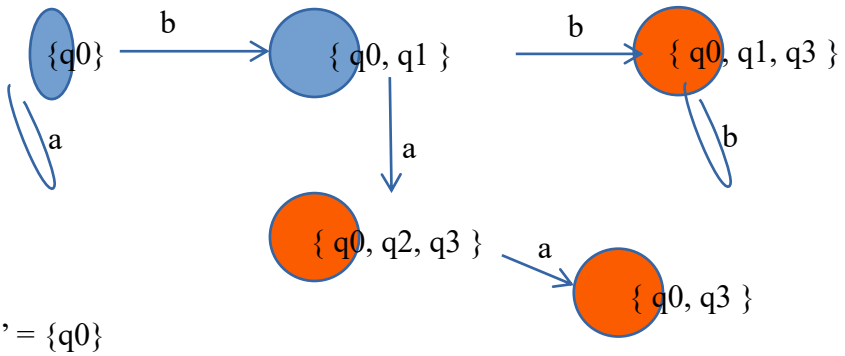
Pag.83 teorema 3.1

$A_N = <\Sigma, Q, \delta_N, q0, F>$ ASFND
$A' = <\Sigma, Q', \delta', q0', F'>$ ASFD

Come si costruisce un A' equivalente ad A_N ?
Capiamolo a partire dall'esempio precedente (figura 3.5)

se Q' = P(Q) cioè l'insieme delle parti di cardinalità 2^n (avrà 2^n stati)

Q = {q0, q1, q2, q3}
Q' = P(Q) = { {q0}, {q1}, {q0,q1}, {q0,q1,q2}, {Ø}…}-
Trasformiamolo (non completo):



q0' = {q0}
F' = {P appartenente a Q' | P intersecato F diverso da Ø}
δ' = {

Alcuni degli insiemi (di stati) di Q' non possono essere raggiunti dall'insieme contenente il solo stato iniziale, quindi non vengono considerati al momento della rappresentazione.

Nel caso di un ASFD, qualsiasi sia l'input legge tutta la stringa, mentre nell'ASFND questo non succede sempre, quindi la testina si può fermare prima di BLANK.

TEOREMA:

Dato un ASFD che riconosce un linguaggio L esiste un ASFND corrispondente che riconosce lo stesso linguaggio. Viceversa, dato un ASFND che riconosce un linguaggio L' esiste un ASFD che riconosce lo stesso linguaggio.

DIMOSTRAZIONE:

Sia $A_N = <\Sigma, Q, \delta_N, q0, F>$ ASFND che riconosce L',
costruiamo l'ASFD $A' = <\Sigma, Q', \delta', q0', F'>$ come segue:

$$Q' = \{ [q_{i1}, \dots q_{ik}] \mid \{q_{i1}, \dots q_{ik}\} \in P(Q) \}$$
$$q0' = [q0]$$
$$F' = \{ [q_{i1}, \dots q_{ik}] \mid \{q_{i1}, \dots q_{ik}\} \cap F \neq da \ \emptyset \}$$
$$\delta' ([q_{i1}, \dots q_{ik}], a) = [q_{j1}, \dots q_{jh}]$$
$$\text{se e solo se } \{q_{j1}, \dots q_{jh}\} = \delta_N (q_{i1}, a) \cup \delta_N (q_{i2}, a) \cup \dots \cup \delta_N (q_{ih}, a)$$

In questo modo si può dimostrare che $L(A_N) = L(A')$.
La dimostrazione del teorema è costruttiva: ci permette di costruire materialmente un algoritmo che ci permette la conversione.

Pumping lemma per i linguaggi regolari (teorema 3.4 pag 92/fotocopia MM sito barbanera)

E' una condizione necessaria per i linguaggi regolari.

Condizione necessaria: vuol dire che se il linguaggio è regolare deve NECESSARIAMENTE verificare questa condizione. L regolare → L soddisfa il Pumping lemma, NO PUMPING LEMMA → NO REGOLARE
Condizione NON sufficiente: vuol dire che se il linguaggio soddisfa quella condizione, non necessariamente è un linguaggio regolare. Se L soddisfa il Pumping lemma, non vuol dire per forza che sia regolare.

La dimostrazione del Pumping lemma ci fa capire se un linguaggio non è regolare, quindi se qualsiasi automa preso in considerazione non riconosce il suddetto linguaggio.

L = {a2b2} è regolare
L = {anbn} non è regolare perché ci sarebbe un insieme di stati infinito.
Qui viene in aiuto lo strumento formale del Pumping lemma.

Per ogni linguaggio regolare L, esiste una costante n tale che se una stringa z appartiene a L e la sua lunghezza è maggiore uguale a N, essa si può dividere in u v w con v che si può ripetere.
La costante N sarebbe il numero di stati dell'automa.

ENUNCIATO FORMALE:

Per ogni linguaggio regolare L, esiste una costante n tale che se:

- **la stringa z Є L**
- **|z| >= n**

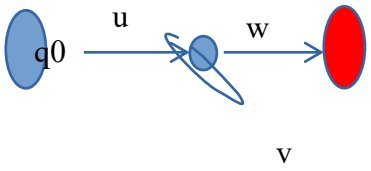
allora possiamo scrivere z = uvw con:

- **|uv| <= n (u può essere anche epsilon)**
- **|v| >= 1 e diversa da epsilon, perché altrimenti il lemma non avrebbe senso**
- **$uv^i w$ Є L per ogni i >=0**

DIMOSTRAZIONE INFORMALE:

Sia A' = <Σ, Q', δ', q0', F'> ASFD che riconosce L (cioè L(A)=L),
sia n la cardinalità di Q (|Q| = n)

Se la stringa z è molto lunga, ci sarà sicuramente almeno un ciclo durante la sua lettura.
Se ad esempio il cammino etichettato z è abbastanza lungo da essere spezzettato da (ad esempio) un solo ciclo, la parte precedente al ciclo diventa u, i indica le iterazioni del ciclo v, w è la parte successiva al ciclo.
Se L è finito, L è regolare.



DIMOSTRAZIONE FORMALE:

- **Sia A un ASFD tale che L = L (A)**
- **sia z Є L, |z| = k>=n**
- **sia cardinalità di Q (|Q|) = n**
- **sia q_{i0}, q_{i1}, q_{i2}, … q_{ik} (con qi0 = stato iniziale e qik Є F) la successione di stati raggiunti nell'automa A leggendo z [chiaramente avrò δ(q0, z) Є F)]**

Se z_h indica il prefisso di z di lunghezza h (h=0, 1, … k), allora $q_{ih} = \delta(q0, z_h)$
Poiché k >= n, k+1 > n. dato che n era il numero di stati dell'automa e k è maggiore di esso, vuol dire che almeno due stati della sequenza q_{i0}, q_{i1}, q_{i2}, … q_{ik} coincidono. Quando si legge l'n-esimo simbolo di z, troviamo questi stati coincidenti. Questi stati coincidenti si trovano nell'intervallo 0 <= r < s <= n.
Esistono quindi almeno due indici r e s diversi fra loro tali che gli stati corrispondenti coincidono ($q_{ir} = q_{is}$).

$\delta(q0, z_r) = \delta(q0, z_s)$

Sia u = z_r, v tale che uv = z_s (quindi v = z_s – z_r), uvw = z,
dimostriamo per induzione che uv^iw Є L.
Passo base: uv0w Є L ?

δ(q0, uw) = δ(δ(q0, u), w) = δ(δ(q0, zr), w) = δ(δ(q0, zs), w) = δ(δ(q0, uv), w) = δ(q0, uvw) = δ(q0, z).
z Є L quindi δ(q0, z) Є F

Ipotesi induttiva: uv^nw Є L ?

δ(q0, uv^nw) = δ(δ(q0, uv^nw)

Tesi: $uv^{n+1}w$ Є L

δ(q0, $uv^{n+1}w$) = δ(q0, uvv^nw) = δ(δ(q0, uv), v^nw = δ(δ(q0, zs), v^nw = δ(δ(q0, zr), v^nw = δ(δ(q0, u), v^nw) = δ(q0, uv^nw) Є F

Come si dimostra che un linguaggio non soddisfa il pumping lemma?
Esempio:
L = {a^kb^k | k>=1}

Suppongo per assurdo che esso sia regolare.
Neghiamo il pumping lemma:
per ogni n costante se esiste z appartenente a L, |z| => n, per ogni u,v,w tale che z=uvw, con |uv| <= n e v diverso da epsilon, esiste i >= 0 per cui uv^iw non apppartiene a L.

Sia n un numero qualunque, sia z = a^nb^n Є L [z = |a^n| + |b^n| = 2n >= n]
uv è sempre formato solo da a, v contiene almeno una a.
Per i =/= 1, uv^iw non appartiene a L perché avrebbe un numero di a minore del numero di b

27/11/2019

ESPRESSIONI REGOLARI (pag 29, 1.3.3)

Sono formalismi per descrivere linguaggi regolari, scritture che corrispondono a determinati insiemi. Le espressioni regolari sono diverse in base all'alfabeto su cui sono definite.

Dato un alfabeto Σ e dato l'insieme di simboli S = {+, *, (,), °, Ø}, si definisce espressione regolare su Σ una stringa r appartenente a $(\Sigma \cup S)^+$ tale che valga una delle seguenti condizioni:

- r = Ø
- r appartiene a Σ
- r = s+t / r = s ° t / r = s* (dove s, t sono espressioni regolari su Σ)

Il linguaggio denotato da una espressione regolare L(r) è così definito:

- se r = Ø, allora L(r) = Λ (linguaggio vuoto)
- se r = a appartenente a Σ, allora L(r) = {a}
- se r = (s+t), allora L(r) = L(s) U L(t)
- se r = (s°t), allora L(r) = L(s) ° L(t)
- se r = s*, allora L(r) = (L(s))*

Esempio:

r = ((0+1)* ° 0)

L(r) = L(s) ° L(t)

L(s) = (L(0+1))* = (L(0) U L(1))* = ({0} U {1})* = ({0,1})*

L(t) = L(0) = {0}

quindi, L(r) = {0,1}* ° {0}

Esempio (es. 1.22, pag 30)

Espressione regolare sull'alfabeto a,b che definisce le stringhe il cui terzultimo carattere è una b.
(a+b)* ° b ° (a+b) ° (a+b)

L(r) = {a,b}* ° {b} ° {a,b} ° {a,b} = {a,b}* ° {b} ° $\{a,b\}^2$

Le espressioni regolari aiutano l'analizzatore lessicale, che mi dice se le parole del linguaggio che ho scritto sono corrette e vi appartengono.

Esempio (es. 1.23)

Determinare il linguaggio definito dall'espressione regolare a*((aa)* b + (bb)* a) b*.
$\{a^n\}°(\{a^o a^o\}°\{b\} U \{b^p b^p\}°\{a\}) ° \{b^m\}$ =
$\{a^n\}°(\{(a^2)^o b, (b^2)^p a\}) ° \{b^m\}$ oppure

E1: a*(a*)b(b*) = a*bb*
E2: a* ° b*a ° b* = (a*)(bb)*a(b*)

L(r) = a*bb* U a*(bb)*ab*
non accetta la stringa ba

GRAMMATICHE

Ci permettono di riconoscere linguaggi in modo generativo, piuttosto che riconoscitivo (come negli ASF).

V_T = alfabeto dei simboli terminali (da cui sono generate le stringhe della grammatica)
V_N = alfabeto dei simboli non terminali (variabili che ci servono per generare le giuste stringhe del linguaggio ma che non fanno parte di V_T

Le stringhe del linguaggio si generano a partire da un assioma, qualcosa di certo, che viene indicato con S (simbolo iniziale)e utilizzando un insieme delle regole di produzione P.

Considero una grammatica G in cui V_T = {a,b,c}, V_N = {S, B, C), con regole di produzione:
P1 S → aS
P2 S → B
P3 B → bB
P4 B → bC
P5 C → cC
P6 C → c
In realtà, le regole P sono definite come coppie.
Ogni coppia è definita come ($(V_T \cup V_N)^* V_N (V_T \cup V_N)^*$) x ($(V_T \cup V_N)^*$), cioè il prodotto cartesiano di questi due insiemi.
Il primo elemento della coppia deve contenere almeno un simbolo non terminale.

S => aS => aB => abB => abbC => abC => abc
oppure
S => B => bB => bbB => bbbB => bbbbB => bbbbbC => bbbbbcC => bbbbbcc
oppure
S => B => bC => bcC => bcc
etc…

Formalizziamo:

Una grammatica è una quadrupla G = <V_T, V_N, P, S> dove:

- V_T è l'insieme finito e non vuoto di simboli terminali
- V_N è l'insieme finito e non vuoto di simboli non terminali (o variabili)
- P è una relazione binaria di cardinalità finita / è sottoinsieme di (($(V_T \cup V_N)^* V_N (V_T \cup V_N)^*$) x ($(V_T \cup V_N)^*$))
- S è il simbolo iniziale

Manca qualcosa

L = $\{a^n b^n \mid n>=0\}$
G = <{a,b}, {S}, P, S>
P = S → aSb
S → epsilon

Espressione regolare su alfabeto {a,b} che denota le stringhe che contengono un numero pari di a:
(b + ab*a)*

11/12/2019

Una grammatica può generare anche un linguaggio vuoto.
Alfa → epsilon : epsilon-produzione

Due grammatiche sono equivalenti quando generano lo stesso linguaggio.
Le regole di produzione hanno la forma A→B dove A appartiene a V*VNV* e B appartiene a V*, se poniamo VT U VN = V.
Esistono tante categorie di grammatiche che dipendono dalla forma delle loro regole di produzione.

- **Tipo 0 o Unrestricted**: se le regole di produzione sono più generali possibile, del tipo A→B dove A appartiene a V*VNV* e B appartiene a V*, se poniamo VT U VN = V.
  Queste grammatiche sono riconosciute dalle macchine di Turing (TM);
  Esempio:
  Ab → Cc
  Bb → epsilon
- **Tipo 1 o Context-sensitive**: tutte le regole di produzione sono del tipo Alfa → Gamma dove Alfa appartiene a V*VNV*, Gamma appartiene a **V+**, e la lunghezza di Alfa è minore o uguale alla lunghezza di Gamma, **quindi non esistono epsilon-produzioni;**
  Queste grammatiche sono riconosciute dagli automi limitati linearmente, particolari macchine di Turing;
  Esempio:
  Ab → Cc
  ~~Bb → epsilon~~
- **Tipo 2 o Context-free**: tutte le regole di produzione sono del tipo Alfa → Beta dove Alfa appartiene a VN e Beta appartiene a V+.
  Queste grammatiche sono riconosciute dagli automi a pila.
  Le grammatiche context-free sono **fondamentali per la costruzione dei parser (analizzatori lessicali) dei compilatori.**
  Le grammatiche context-free sono le uniche per le quali si definiscono gli alberi di derivazione.
  Esempio:
  A → Cbb
  ~~A → epsilon~~
- **Tipo 3 o lineari a destra/sinistra**: tutte le regole di produzione sono del tipo Alfa → Delta dove Alfa appartiene a VN e Delta appartiene a (VTVN) U VT.
  Queste grammatiche sono riconosciute dagli automi a stati finiti e producono solo linguaggi regolari.
  Esempio:
  A → Bb
  A → b
  ~~A → epsilon~~

Si dice context-sensitive se si può scrivere una grammatica equivalente dove le regole di produzione hanno tipo Alfa A Beta → Alfa Gamma Beta dove Alfa, Gamma, Beta appartiene a V*, Gamma appartiene a V+, A appartiene a VN. Qui si può sostituire A con Gamma solamente se A è in mezzo ad Alfa e Beta, cioè è vincolata da Alfa e Beta.

L = {anbn | n>=1} NON REGOLARE (quindi non è di tipo 3)
L è di tipo 2? Sì

S → aSb
S → ab

Una grammatica di tipo 3 è anche di tipo 2, 1 e 0 ma non vale sempre lo stesso in modo inverso.
Il tipo 3 è sottoinsieme del tipo 2, che a sua volta è sottoinsieme del tipo 1, che a sua volta è sottoinsieme del tipo 0. Questa "gerarchia" sulle classi di linguaggi viene detta gerarchia di Chomsky.

Un linguaggio è strettamente di tipo n se esiste una grammatica di tipo n che lo genera e non esiste una grammatica di tipo n+1 che lo genera.

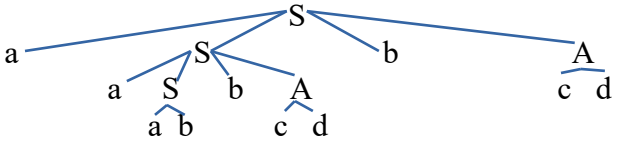
ACCORGIMENTI VARI:

- I simboli non terminali a volte si esprimono tra parentesi uncinate: <espressione>, <identificatore> etc.;
- Le regole di produzione A → a1 e A → a2 si possono anche scrivere come A → a1 | a2;
- A → x | xy | xyy | xyyy si può scrivere anche A → x{y}^{3};
- A ::= [x]y si può scrivere come A ::= xy | y, e viceversa;
- A ::= xu | xy | xz si può scrivere come A → x (u | y | z).

Le forme normali di Bakus (BNF), Chomsky (CNF) e Greibach (GNF) per le grammatiche context-free sono importanti perché possono aiutare a capire meglio quando c'è equivalenza tra le grammatiche.
La CNF è:
A → BC oppure
A → a
dove A,B,C appartengono a VN e a appartiene a VT.
La GNF è:
A → a beta
dove beta appartiene a VN*

La derivazione in una grammatica si rappresenta con gli alberi di derivazione.
Consideriamo la grammatica G con le seguenti produzioni:
S → aSbA | ab
A → cAd | cd

S → aSbA → aaSbAbA → aaabbAbA → aaabbcdbcd



13/01/2020

S → b | Ab
A → Aa | a è equivalente a S → aS | b perché generano lo stesso linguaggio.

GRAMMATICHE AMBIGUE

La grammatica è ambigua quando una stringa ammette due diversi alberi di derivazione.
Consideriamo la seguente grammatica:
Vn = {E} Vt = {+, *, (,), /, a}
E → E+E | E*E | (E) | E/E | a

Esempio di produzione:
E → E+E → a+E → a +E*E →a+a*E → a+a*a

Questa grammatica è ambigua perché la stringa a+a*a ammette due diversi alberi di derivazione:
E → E*E → E+E*E → a+E*E → a+a*E → a+a*a

Cambiando l'ordine di applicazione delle regole di produzione, l'albero di derivazione rimane invariato.

Data una grammatica ambigua, ci sono dei metodi che permettono di disambiguare la grammatica creandone una equivalente alla precedente, ma priva di ambiguità.
Tuttavia l'ambiguità dipende dal linguaggio generato dalla grammatica. Se il linguaggio ci obbliga ad includere ambiguità, si chiama linguaggio inerentemente ambiguo e la sua grammatica non può essere disambiguata.

CARDINALITA' DEGLI INSIEMI

Dato un insieme finito A, la cardinalità dell'insieme è pari a n se esiste una corrispondenza biunivoca fra l'insieme A e l'insieme dei numeri naturali da 0 a n-1 (oppure da 1 a n).

Se esiste una corrispondenza biunivoca tra A e l'insieme dei naturali, si parla di insieme numerabile.
Un insieme si dice contabile se finito o numerabile.

L'insieme delle parti di **N** (naturali) non è numerabile.
Supponiamo per assurdo che sia numerabile, pertanto i sottoinsiemi di **N** saranno P1, P2, P3… Pn.

	0	1	2	…	n
P0	No				
P1		Si			
P2			No		
…				…	
Pn					

L'insieme Σ* è numerabile.

Si può parlare anche di cardinalità dei linguaggi.

PROGRAMMI CHE RICONOSCONO LINGUAGGI

Alcuni linguaggi non possono essere riconosciuti perché l'insieme delle parti di Σ* ha cardinalità maggiore di quello dei programmi possibili. Quindi esistono linguaggi che non hanno programmi che li **decida**, all'interno di qualsiasi riconoscitore di ogni tipo.

Si distingue quindi tra linguaggi decidibili, linguaggi semi-decidibili (per cui esistono dispositivi che si fermano se e solo se l'input appartiene al linguaggio) e linguaggi non decidibili (quelli sopracitati).

MACCHINE DI TURING

Sono dispositivi riconoscitivi (come gli ASFD) dotati di un nastro infinito in entrambe le direzioni, diviso in celle ognuna contenente un solo simbolo di un alfabeto, fornito di testina che **legge o scrive** una cella alla volta muovendosi in entrambe le direzioni o anche rimanere ferma.

Una macchina di Turing deterministica (MTD) è un sistema **M = < Γ, ƀ, Q, q0, F, δ >** dove:

- Γ è l'alfabeto finito e non vuoto dei simboli che posso trovare sul nastro (Σ è un sottoinsieme di Γ)
- **ƀ** è un carattere particolare che non appartiene a Γ (corrisponde alla cella vuota BLANK)
- Q è l'insieme finito e non vuoto degli stati
- Q0 appartiene a Q ed è lo stato iniziale
- F sottoinsieme di Q è l'insieme degli stati finali
- δ è la funzione di transizione definita **solo per gli stati non finali** (la computazione si ferma su uno stato finale) che associa ad una coppia stato-simbolo/**ƀ** una terna <stato, simbolo che posso scrivere, movimento> .
  δ : (Q-F) x (Γ U {ƀ}) → Q x (Γ U {ƀ}) x ({d, s, i})
- Γ segnato = Γ U {**ƀ**}

Esiste solamente un numero finito di celle che contengono un simbolo diverso da **ƀ,** perché abbiamo a che fare sempre con stringhe di lunghezza **finita**.
La macchina di Turing può anche eseguire una computazione all'infinito, senza raggiungere uno stato finale.

Una configurazione istantanea di una macchina di Turing si compone di:

- Posizione della testina
- Carattere in lettura
- Stringa intera sul nastro
- Stato del controllo

e viene rappresentata da una stringa $\alpha_1 q \alpha_2$, dove α_1 è la stringa che precede la testina, q lo stato attuale del controllo, α_2 la stringa che succede la testina a partire dal simbolo in lettura.
Se dopo q ci sono solo caratteri blank, la stringa $\alpha_1 q \alpha_2$ termina con **ƀ.**

Una configurazione si dice iniziale se α_1 = epsilon e q = q0.
Una configurazione si dice finale se q appartiene a F.

20/01/2020

Configurazione istantanea si esprime con una stringa c = xqy dove:

- X appartiene a ΓΓ* U {epsilon}
- Q è uno stato
- Y appartiene a Γ*Γ U {blank}

Quando si parla di configurazione d'accettazione?
Quando in c = xqy:

- x = epsilon
- q0 (stato iniziale)
- y appartiene a Γ^+ U {B}

Per convenzione, la stringa in input va scritta in celle successive senza blank in mezzo.
Se c'è un blank in mezzo ad una stringa, non sarò in una configurazione iniziale perché la macchina avrà già lavorato in scrittura, scrivendo dei blank tra i caratteri.

Data una MT, la stringa x appartenente a sigma*:

- è accettata se da q0 si arriva ad α1qα2 in uno o più passi, dove q è uno stato finale.
- È rifiutata se da (q0,x) la computazione termina in uno stato non finale.
- Può far andare avanti all'infinito la computazione, rendendoci impossibile decidere se possa essere accettata o rifiutata.

La MT riconosce un linguaggio L sottoinsieme di sigma* se la computazione termina su ogni input, su una configurazione di accettazione se l'input appartiene al linguaggio, di rifiuto se non vi appartiene.

La MT accetta un linguaggio L se per ogni x di L la computazione termina in uno stato finale e per ogni x non appartenente ad L non termina o termina in uno stato non finale.

CODICI – CRITTOGRAFIA