Palermo

Informatica teorica

Appunti

Indice degli argomenti

Indice delle date

11/10/2001

Teoria delle funzioni ricorsive: studia la combinazione di funzioni semplici al fine di arrivare a funzioni complesse.

Ognuno di questi modelli teorici definisce un campo di esistenza per delle funzioni:

1)macchina di Touring (1936);
2)macchina a registri;
3)sistemi di riscrittura/Post;

Tesi*[1] *di Touring-Church

Tutto quello che è calcolabile tramite un algoritmo lo si può fare tramite una macchina di Touring.

N.B.: la macchina di Touring è un modello generale; se si afferma che esiste un algoritmo per un certo problema, allora esiste una macchina di Touring che lo risolve.

12/10/2001

Macchina di Touring (1936)

Per la prima volta vengono messi in chiaro alcuni elementi. *Calcolare un algoritmo* ha a che fare con la manipolazione dei simboli, indipendentemente dal loro significato. Il calcolare non è quindi legato a numeri ma, in maniera più estesa, alle informazioni.

Il *modello* della macchina di Touring è astratto, ma fa riferimento a fatti reali. Esso si articola in questo mondo: quando una persona fa delle operazioni ha bisogno di supporto fisico, un foglio sul quale "memorizzare" dati, che possono essere modificati (scritti o cancellati). Da questo nasce l'idea di prendere come supporto di memorizzazione un nastro potenzialmente infinito e diviso in celle (Figura 1).

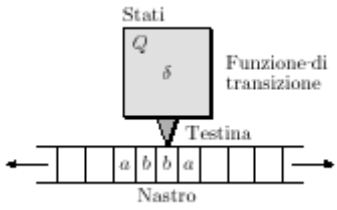


Figura 1 Schematizzazione di una macchina di Touring

Il *nastro* è considerato infinito, ma l'interazione tra corpo e nastro avviene ogni volta su singole celle. La macchina può spostare il suo occhio o testina a destra o sinistra; questo

[1] Non la si può dimostrare, ma allo stesso tempo nessuno è riuscito a smentirla.

spostamento è "deciso" da una FSM, ossia dalla "combinazione" tra stati (contenuti in celle di memoria finite) ed input (i simboli letti nelle celle).

Le *componenti interpretabili* dalla macchina sono:

- B = blank (vuoto);
- Σ = insieme finito di simboli (alfabeto);
- Q = insieme finito di stati;

In definitiva, la macchina di Touring può attuare i seguenti *atti elementari*:

1)cambiare di stato;
2)modificare il simbolo che sta leggendo dalla cella;
3)spostare l'occhio di una sola posizione a destra o sinistra;

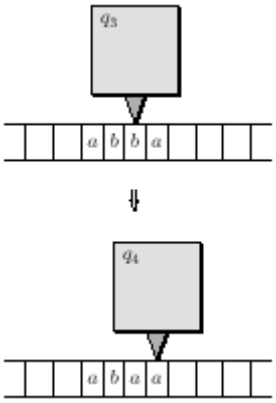


Figura 2 Cambiamento di stato di una macchina di Touring

Esempio 1

Dati:

$$\Sigma = (\mathrm{B},\ /,\ *)$$

$$Q = \{q_1,\ q_2,\ q_3,\ q_4,\ q_5,\ q_6\}$$

costruire la relativa tabella degli stati. Un esempio può essere:

	B	/	*
q1			R q_2
q2		R	R q_3
q3	L q_4	R	
q4		B L q_5	B q_6
q5		L	/ q_6
q6			

Per convenzione, alla partenza, la macchina di Touring parte da una *configurazione iniziale*, ossia:

1)stato = q_0 (il primo);
2)occhio = legge la prima casella a sinistra diversa da B;

Una *configurazione istantanea* α_i è descritta da:

1)stato della macchina;
2)nastro (tutti i simboli);
3)posizione dell'occhio;

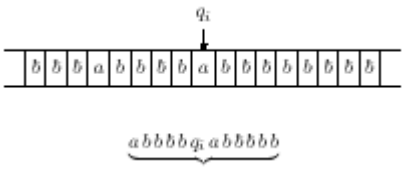


Figura 3 Esempio di configurazione istantanea

Data una configurazione istantanea, questa ne induce una successiva. Un *calcolo* si ha solo quando si perviene, dopo una successione finita di configurazioni istantanee, ad una *configurazione d'arresto*, ossia quando si arriva ad una configurazione istantanea che non porta a nessun'altra configurazione istantanea (nell'Esempio 1 potrebbe essere lo stato q_6). In questo caso si ha un output, che corrisponde al nastro stesso.

Una macchina di Touring definisce un'*applicazione parziale*[2] definita da:

$$T: \Sigma^* \to \Sigma^* \qquad \text{con } \Sigma^* = \text{tutte le parole sull'alfabeto } \Sigma^3$$

Esempio 2

Utilizzando la macchina di Touring dell'Esempio 1 con una stringa iniziale pari a:

$$* / / / * / /$$

si ha che la successione delle configurazioni istantanee della macchina è:

q_1 ↓ * / / / * / /	q_2 ↓ * / / / * / /	q_4 ↓ * / / / * / /
q_2 ↓ * / / / * / /	q_3 ↓ * / / / * / /	q_5 ↓ * / / / * /
q_2 ↓ * / / / * / /	q_3 ↓ * / / / * / /	q_5 ↓ * / / / * /
q_2 ↓ * / / / * / /	q_3 ↓ * / / / * / /	q_6 ↓ * / / / / /

I risultati sono quindi:

input: q_1 ↓ * / / / * / /

output: q_6 ↓ * / / / / /

Codificando un generico numero "n" in questo modo:

$$n = * \overbrace{/ / / \ldots}^{n}$$
$$0 = *$$

si vede che la macchina di Touring utilizzata permette di effettuare l'addizione tra due numeri,

[2] Perché per certi input può non fermarsi; il dominio non è quindi tutto Σ^*.
[3] Diventa quindi un insieme infinito di tutte le stringhe sull'alfabeto Σ.

come ad es.:

$$(3,\ 2) = */// */ / \Leftrightarrow (5) = *////$$

Esercizio 1

Creare una macchina per copiare una stringa. I dati di input sono:

- $\Sigma = \{b, c, *, B\}$
- input = bccbc
- output = bccbc**bccbc

La tabella degli stati è:

	b	**c**	*****	**B**
q_1	* L q_3	* L q_3	R q_2	
q_2	R	R	R q_6	* L q_3
q_3	L q_4	L q_5		
q_4	L	L	L	b q_2
q_5	L	L	L	c q_2
q_6	R	R q_7	b L q_1	
q_7	R q_6	R	c L q_1	

Touring-calcolabile

Data una funzione:

$$f_t :\ N^k \to N \qquad \text{con } k \geq 1$$

la funzione:

$$f :\ N^k \to N$$

è Touring-calcolabile se esiste una macchina di Touring T tale che:

$$f_t = f$$

Da calcoli si è scoperto che le funzioni Touring-calcolabili sono tutte e solo quelle ricorsive, ovvero:

f è Touring-calcolabile $\Leftrightarrow$ f è ricorsiva

15/10/2001

Macchine a registri

E' un modello elementare che permette di fare calcoli con numeri interi non negativi. E' composto da:

1)Contatore (indica l'istruzione del programma in esecuzione);
2)Programma (memorizzato all'interno della macchina);
3)Registri (contengono numeri interi non negativi);

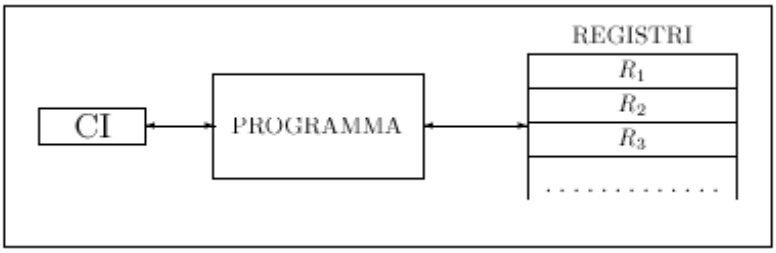


Figura 4 Esempio di macchina a registri

Le istruzioni di questa macchina sono divise in linguaggi:

1)Goto;
2)While;
3)Do Times;

Linguaggio Goto

Le *istruzioni operative*[4] sono:

1)assegnazione di variabile	X:=Y;
2)assegnazione di costante	X:=8;
3)incremento	Incr(X);
4)decremento	Decr(Y);

mentre le *istruzioni di controllo*[5] sono:

1)go to incondizionato	Goto 5
2)go to condizionato	If X=0 Goto 5

Un *programma* è quindi una successione finita di istruzioni. L'*esecuzione* di esso comporta l'inizializzazione delle variabili e il calcolo delle varie istruzioni. Ogni input diverso porta ad infiniti processi di calcoli, ossia ad infiniti output.

Le *funzioni goto-calcolabili* sono definite da funzioni che possono essere calcolate tramite una macchina a registri in linguaggio Goto.

Esempio 3

Scrivere un programma in linguaggio Goto per fare la somma tra X ed Y.

1.If Y=0 Goto 5
2.Decr(Y)
3.Incr(X)
4.Goto 1
5.Z:=X

Teorema 1

una funzione è Touring-calcolabile $\Leftrightarrow$ è Goto-calcolabile

Esercizio 2

[4] Modificano il contenuto dei registri.
[5] Modificano il contenuto del contatore.

Definiamo, a partire dal linguaggio Goto, questi nuovi linguaggi:

1)eliminare l'istruzione operativa di decremento	Goto 1
2)inserire l'istruzione di controllo "If X=Y" al posto dell'istruzione "If X=0"	Goto 2
3)Goto 1 + Goto 2	Goto 3

Hanno tutti la stessa potenza? Ossia, sono equivalenti?

18/10/2001

Basta vedere se, con le nuove istruzioni, si possono "simulare" anche le vecchie.

N.B.: il primo non è equivalente al linguaggio Goto in quanto è meno potente.

2) If X=Y in linguaggio Goto

Questo si traduce nel calcolare:

$$|x - y| = 0$$

1.If Y=0 Goto 6
2.If X=0 Goto 8
3.Decr(X)
4.Decr(Y)
5.Goto 1
6.Z:=X
7.Goto 9
8.Z:=Y
9.Stop

quindi:

1. $z = |x - y|$
2.If Z=0 Goto *n*

3) Decr(X) in linguaggio Goto 3

1.If X=0 Goto 7
2.Z:=Y
3.Incr(Y)
4.If X=Y Goto 6
5.Goto 1
6.Decr:=Z

Linguaggio While

Le istruzioni *operative* sono:

1)assegnazione di variabile	X:=Y;
2)assegnazione di costante	X:=8;
3)incremento	Incr(X);
4)decremento	Decr(Y);

In questo caso i programmi vengono divisi in *livelli* (da 0 ad *n*) o *gradi di nidificazione*. Un programma di livello "*i*" è definito da istruzioni di livello "*i−1*". Quelli di livello 0 sono

successioni finite di istruzioni operative separate da ";".

Le istruzioni di livello "*i+1*" (cioè di grado di nidificazione "*i+1*") sono:

1)If X=0 then [P];
2)If X=0 then [P]
else [Q];
3)While X>0 do [P];

con P, Q istruzioni di livello ≤ *i*

In questo caso è utile creare dei *diagrammi di flusso*.

Esempio 4

Il diagramma di flusso del linguaggio Goto dell'Esempio 5:

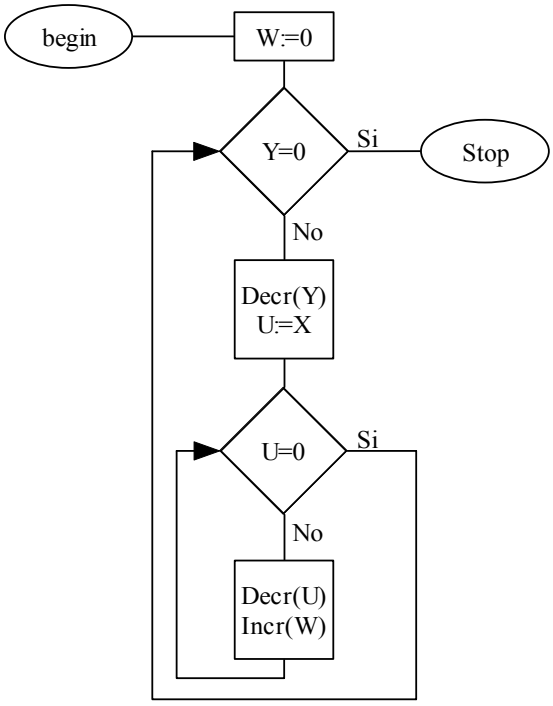


Figura 5 Diagramma di flusso dell'Esempio 5

E' importante notare come sia "meccanicamente" possibile[6] passare da un linguaggio Goto ad uno While e viceversa, ossia come sia possibile il confronto. Una difficoltà aggiuntiva si ha però quando il codice in esame non è scritto in maniera gerarchizzata, cioè quindi quando si vuole trasformare un programma da Goto a While.

Esempio 5

Ecco un esempio di "*conversione*" di un programma scritto in linguaggio Goto in uno scritto in linguaggio While (di livello 2).

Goto	***While***
1.W:=0	W:=0;
2.If Y=0 Goto 9	While Y>0 do
3.Decr(Y)	Decr(y);
4.U:=X	U:=X;
5.If U=0 Goto 2	While U>0 do
6.Decr(U)	Decr(U) ;

[6] Vedi il "*Teorema 4*".

7.Incr(W) Incr(W);
8.Goto 5
9.Stop

Linguaggio Do times

E' un linguaggio che sfrutta l'*iterazione* tramite la seguente istruzione di controllo:

1)Do *n* times [P]; con $\begin{cases} n \in \mathbb{N} \text{ numero di iterazioni} \\ \text{P istruzioni} \end{cases}$

Anche in questo caso è possibile *trasformare* un codice scritto linguaggio Do times in uno scritto in While. La cosa diventa assai difficile quando si tenta di fare il viceversa.

19/10/2001

Macchina di Touring universale

E' possibile individuare una singola macchina che, opportunamente programmata, possa risolvere tutti gli algoritmi. La minima soluzione finora trovata è quella della *macchina di Minsky* (4 simboli e 7 stati).

Tutte le possibili tabelle delle macchine di Touring possono essere un'*infinità numerabile*; quindi l'insieme di tutte queste tabelle identifica un corrispettivo insieme di macchine, e queste possono essere ordinate in base alla grandezza della tabella stessa. Si capisce immediatamente che se le macchine sono un'infinità numerabile, le rispettive funzioni che possono risolvere sono infinite[7].

Problemi di indecidibilità per gli automi (1)

Esistono dei problemi che non permettono, a nessuna macchina, di pervenire ad una soluzione dopo un *numero finito di passi*. Se definiamo con:

$$T_1, \ldots, T_n \quad \text{con } n \in \mathbb{N}$$

le *n* macchine di Touring, e con:

$$f_1(1), \ldots, f_n(n) \quad \text{con} \begin{cases} f \in F_{tc} \\ n \in \mathbb{N} \end{cases}$$

le rispettive funzioni Touring-calcolabili, la funzione:

$$F(x) = f_{n+1}(n) \quad \text{con} \begin{cases} F(x): \mathbb{N} \to \mathbb{N} \\ F(x) \notin F_{tc} \end{cases}$$

rappresenta una delle funzioni non Touring-calcolabile, ossia:

[7] Ad es. la macchina che realizza l'addizione fra due numeri non realizza soltanto una singola e specifica addizione, ma tutte le "infinite" addizioni.

$$\left.\begin{array}{l} T_1 \Rightarrow f_1(1),\ \ldots,f_1(n) \\ T_2 \Rightarrow f_2(1),\ \ldots,f_2(n) \\ \vdots\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ T_n \Rightarrow f_n(1),\ \ldots,f_n(n) \end{array}\right\} \in F_{tc}$$
$$f_{n+1}(1),\ \ldots,f_{n+1}(n) \notin F_{tc}$$

Questo fatto viene rigorosamente definito dal teorema seguente.

Teorema di Göedel (o di incompletezza)

Per ogni sistema formale di regole ed assiomi è possibile arrivare a proposizioni indecidibili, usando gli assiomi dello stesso sistema formale, ossia è impossibile dimostrare, dall'interno di un sistema di assiomi, la non contraddittorietà di questi ultimi.

Teorema della fermata di una macchina di Touring

Non esiste nessun algoritmo che riesca a stabilire se, dati una macchina di Touring *T* ed un input *W*, la macchina perverrà ad un risultato entro un numero finito di passi (ossia ad una configurazione d'arresto).

10° problema di Hilbert (1900)

Solvibilità delle *equazioni diofantee*.

Sistemi di riscrittura (1)

Sono dei sistemi che data una stringa, una successione di numeri etc. danno come output una stringa, una successione di numeri etc. appartenenti all'insieme. Si tratta quindi di vedere se da un input si può arrivare, dopo un *numero finito di passi*, all'output desiderato. La *verifica* si divide quindi in due casi:

- sì ⇔ basta un esempio;
- no ⇔ ci vuole una dimostrazione rigorosa;

Problema della parola 1

Date queste identità:

1) $ab = ba$;
2) $cb = bbc$;
3) $ca = a$;

posso affermare che:

$$abb \overset{?}{=} ab$$

A questo specifico problema si può trovare una soluzione tramite:

$$abb = bab = bba = bbca = cba = cab = ab$$

Congettura di Ulam

Data la successione di numeri:

$$x_0, x_1, \ \dots\ , x_n$$

$$x_{i+1} = \begin{cases} \dfrac{x_i}{2} & \text{se } x_i \text{ è pari} \\ 3x_i + 1 & \text{se } x_i \text{ è dispari} \end{cases}$$

si sa qual è la regola per provare i successivi numeri, ma non si può sapere a priori se si fermerà e se sì dopo quanti passi.

22/10/2001

Sistema MIU

E' un sistema il cui alfabeto è composto dalle sole tre lettere:

$$\{M,\ I,\ U\}$$

Gli oggetti sono le stringhe composte dalle lettere e seguono queste regole:

1)se si ha una stringa che termina per "*I*", si può aggiungere "*U*" (ad es. "*MI*" → "*MIU*");
2)se si ha una stringa della forma "*M**", si può ottenere la stringa "*M***" (ad es. "*MIU*" → "*MIUIU*");
3)se in una stringa compare "*III*", si possono sostituire con "*U*" (ad es. "*MIII*" → "*MU*");
4)se in una stringa compare "*UU*", si può cancellare (ad es. "*MUU*" → "*M*");

Partendo dalla stringa "*MI*", ci si chiede se si può arrivare a "*MU*".

Problema della parola 2

Questo sistema è composto dalle sole tre lettere:

$$\{a,\ b,\ c\}$$

Vale la proprietà associativa e valgono le seguenti 5 regole simmetriche:

1)$b = acc$;
2)$ca = accc$;
3)$aa = \varepsilon$[8];
4)$bb = \varepsilon$;
5)$cccc = \varepsilon$;

In questo caso ci si chiede se, date due stringhe generiche ed appartenenti all'alfabeto, esiste un algoritmo per determinare se si possono ottenere l'una dall'altra attraverso il sistema d'identità dato.

Sistemi TAG

Questo è un problema posto negli anni '30 da Post alla caffetteria dell'MIT. Data una stringa sull'alfabeto:

$$\{0,\ 1\}$$

si hanno le seguenti due regole:

[8] Indica l'identità del gruppo, in quanto se si concatena una parola con ε, questa rimane identica; indica quindi la stringa vuota.

1)se inizia per "0" allora aggiungo "00" alla sua destra e successivamente cancello i primi tre simboli;
2)se inizia per "1" allora aggiungo "1101" alla sua destra e successivamente cancello i primi tre simboli;

In questo caso, a differenza degli altri esempi, data una determinata stringa in input ed applicando le regole si ottiene un solo output ben specifico; il problema è quindi *deterministico*[9]. La domanda che ci si pone è quindi se data una stringa, si perverrà ad una fine.

Ad es.

$$0001001 \to \cancel{000}100100 \to \cancel{100}1001101 \to \cancel{100}11011101 \to$$
$$\to \cancel{110}111011101 \to \cancel{111}0111011101 \to \cancel{011}101110100 \to ...$$

Problemi di indecidibilità per gli automi (2)

Sono dei problemi che ammettono *risposta booleana*. A questo tipo di problemi appartiene ad es. quello della fermata della macchina di Touring.

Si crea una particolare *macchina di Touring decisionale* che dato un qualsiasi input, da sicuramente un output valido (sì o no).

In generale, si ha che una *macchina che riconosce una stringa* è composta da:

$Q =$ insieme degli stati con $\begin{cases} Q = Q_{sì} \cup Q_{no} \\ Q_{sì} \cap Q_{no} = \phi \end{cases}$

$Q_F =$ insieme degli stati finali di accettazione con $Q_F \subseteq Q$

e si può dire che una parola/stringa dell'alfabeto:

$$W \quad \text{con } W \in \Sigma^*$$

è *accettata*/*riconosciuta* da una macchina di Touring se si verificano entrambe le seguenti condizioni:

1)si ferma dopo un numero finito di passi;
2)si ferma in uno stato d'accettazione;

Quindi la stringa:

- non è accettata $\Leftrightarrow$ $\begin{cases} \text{se non si ferma} \\ \text{se si ferma in uno stato } \notin Q_F \end{cases}$
- è accettata $\Leftrightarrow$ se si ferma in uno stato $\in Q_F$

In questo caso si ha un insieme:

$$L(T) \subseteq \Sigma^*$$

delle stringhe riconosciute/accettate dalla macchina di Touring; quest'insieme è detto *linguaggio*.

Membership problem ed emptiness problem[10]

Preso un linguaggio, definito in un certo modo, ed una stringa:

[9] Vedi "*1-FSA*" e "*DFA*".
[10] Vedi "*Problemi di indecidibilità per le grammatiche*".

$$L(T) \subseteq \Sigma^*$$
$$w \in L$$

si possono avere i seguenti *problemi di decisione*:

1)membership problem = $w \in L(T)$?
2)emptiness problem = il linguaggio è vuoto o no? Cioè, esiste almeno una stringa accettata dalla macchina?

N.B.: il *membership problem è indecidibile* per linguaggi riconosciuti da macchine di Touring proprio per il problema della fermata.

Simbologia (1)

$\Sigma =$ alfabeto/insieme finito $\left(\text{ad es. } \{a,\ b,\ c,\ ...\}\right)$

$\Sigma^* =$ insieme infinito $\left(\text{ad es. } \{u,\ v,\ w,...\}\right)$

In Σ^* è possibile definire un'operazione di *concatenazione*, ossia date due stringhe quali ad es.:

1)$u = abaab$;
2)$v = bba$;

si ha che:

$$uv = abaabbba$$

N.B.: quest'operazione è associativa ma non commutativa, tranne che per casi particolari:

$$\left.\begin{array}{l} u = abab \\ v = ababab \end{array}\right\} \Rightarrow uv = vu = abababababab$$

25/10/2001

Sistemi di riscrittura (2)

La nozione di sistema di riscrittura cattura il *concetto di calcolo*:

$$...(a+b)(a-b)... \ \Leftrightarrow \ ...(a^2-b^2)...$$

Oppure si può applicare al *calcolo logico* dei *sistemi deduttivi*:

1)l'uomo è mortale
2)Socrate è un uomo $\Leftrightarrow$ Socrate è mortale

Sistemi di Thue

Sono particolari sistemi di riscrittura definiti da:

$$P = \{(u,\ v)...\} \qquad \text{con} \begin{cases} P = \text{insieme finito di regole di produzione} \\ P \subseteq \Sigma^* \times \Sigma^* \\ u,\ v \in \Sigma^* \end{cases}$$

Derivazione in un passo

Prese due stringhe:

$$w \underset{p}{\Rightarrow} t \quad [11] \qquad \text{con } w,\ t \in \Sigma^*$$

vale questo se:

$$w = xuy \qquad t = xvy \qquad \text{con} \begin{cases} x,\ y \in \Sigma^* \\ u \to v \in P \end{cases}$$

Graficamente si può avere che:

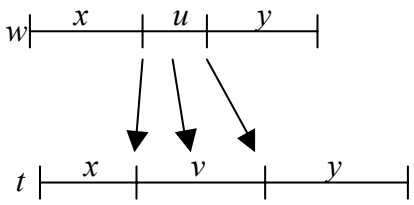


Figura 6 Esempio di derivazione in un passo.

N.B.: mentre l'insieme P è finito, la relazione $\underset{p}{\Rightarrow}$ (che è un insieme di coppie) è un insieme infinito; da un numero finito di regole si arriva ad un numero infinito di stringhe.

Derivazione

In questo caso si ha che:

$$w \overset{*}{\Rightarrow} t \quad [12] \qquad \text{con } w,\ t \in \Sigma^*$$

Da un punto di vista matematico, è la chiusura transitiva e riflessiva della relazione $\underset{p}{\Rightarrow}$, cioè:

$$w \overset{*}{\Rightarrow} t$$

se esiste una successione finita di parole:

$$v_1,\ v_2,\ \ldots,\ v_n : \begin{cases} w = v_1 \\ t = v_n \end{cases}$$

e vale che:

$$v_i \Rightarrow v_{i+1} \quad \forall i \in \{1,\ 2,\ \ldots,\ n-1\}$$

N.B.: n=1 $\Rightarrow$ w è in relazione con se stesso.

Da quanto detto si deduce che sia MIU[13] che TAG[14] non sono sistemi di Thue. E' un sistema di Thue quello definito dal Problema della parola 2. In questo caso le varie regole si possono interpretare in maniera simmetrica; ad es. la regola 1) diventa:

$$b = acc \quad \Leftrightarrow \quad \begin{cases} b \to acc \\ acc \to b \end{cases}$$

[11] Che si legge: "da w deriva in un passo t".
[12] Che di legge: "da w deriva t".
[13] Perché la regola 2) ha in se infinite regole.
[14] Perché si posso togliere ed aggiungere caratteri.

Teorema 2

dato un sistema di Thue e date due parole $u,\ v \in \Sigma^*$, è indicibile sapere se si può fare $u \overset{*}{\Rightarrow}_p v$

Dim.:

Si dimostra riportando il problema del quale si vuole verificare l'indecidibilità, ad un problema di cui si è già dimostrata l'indecidibilità. In questo caso si riconducono le regole di derivabilità ad una tabella degli stati di una macchina di Touring, riferendosi quindi al problema della fermata.

Simbologia (2)

La coppia:

$$L(F,\ P) \subseteq \Sigma^* \qquad \text{con} \begin{cases} L = \text{linguaggio} \\ F = \text{insieme finito di "assiomi"} \\ P = \text{ regole di produzione} \end{cases}$$

corrisponde alle stringhe:

$$L = \left\{ w \in \Sigma^* \ :\ u \underset{p}{\overset{*}{\Rightarrow}} w \text{ per qualche } u \in F \right\}$$

In altri casi è preferibile inserire dei simboli supplementari; in definitiva si avrà:

- Σ^* = alfabeto dei simboli terminali;
- V = alfabeto dei simboli non terminali (variabili);
- Ω = assioma ossia stringa dalla quale partire;
- P = insieme finito di regole di produzione[15], tali che:

$$P \subseteq (\Sigma uv)^* \times (\Sigma uv)^*$$
$$P : \alpha \to \beta$$

In input useremo la notazione:

- $\Sigma = a, b, c, \ldots$
- $V = A, B, C, \ldots$
- $\Sigma^* = u, v, w, \ldots$
- $(\Sigma uv)^* = \alpha, \beta, \delta, \ldots$

L'output sarà caratterizzato soltanto da simboli terminali.

Grammatiche

Per specificare una grammatica serve:

$$G = \{\Sigma,\ V,\ P,\ \Omega\} \qquad \text{con } \Omega \in V$$

mentre il linguaggio generato è:

[15] Riguardano stringhe appartenenti ad entrambi gli alfabeti.

$$L(G)=\left\{w\in\Sigma^{*}\ :\ \Omega\overset{*}{\Rightarrow}w\right\}$$

In pratica, una grammatica è un meccanismo per generare un linguaggio. A differenza della macchina di Touring, la grammatica è un sistema di calcolo non deterministico.

Linguaggio delle parentesi

Data una grammatica:

$$G:\ \Sigma\{a,\ b\}\qquad \text{con } V=\{\Omega\}$$

e le seguenti regole:

1) $\Omega\rightarrow\Omega\Omega$;
2) $\Omega\rightarrow a\Omega b$;
3) $\Omega\rightarrow ab$;

un esempio (degli infiniti possibili) di derivazione è essere:

$$\Omega\overset{1}{\Rightarrow}\Omega\Omega\overset{2}{\Rightarrow}a\Omega b\Omega\overset{1}{\Rightarrow}a\Omega\Omega b\Omega\overset{3}{\Rightarrow}aab\Omega b\Omega\overset{3}{\Rightarrow}aababb\Omega\overset{3}{\Rightarrow}aababbab$$

A questo punto ci si può fermare in quanto la stringa non contiene più simboli non terminali; è quindi in "regola" con le specifiche dell'output.

Interpretando in questo modo:

$$a\rightarrow($$
$$b\rightarrow)$$

si ha che:

$$aababbab\Rightarrow((\ )(\ ))(\ )$$

ed il linguaggio generato corrisponde ad una certa parentesizzazione.

Teorema 3

un linguaggio è riconosciuto da una macchina di Touring $\Leftrightarrow$ è definito da una grammatica

Variazioni alla macchina di Touring

1)trasformare il *nastro* unidimensionale (vettore) in uno *bidimensionale* (array):

XXX

Figura 7 Esempio di macchina di Touring con nastro bidimensionale.

2)inserire *più occhi*:

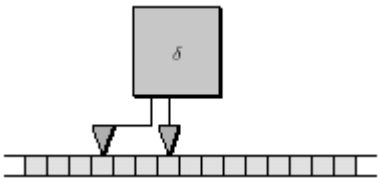


Figura 8 Macchina di Touring con più teste.

3)inserire *più nastri* (e più teste):

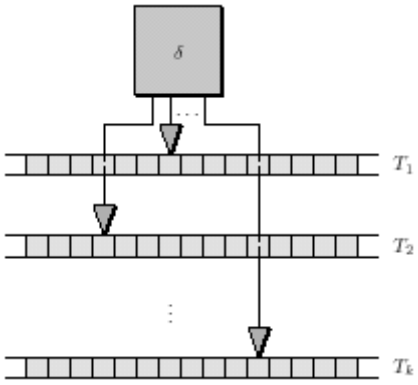


Figura 9 Esempio di macchina di Touring con più nastri.

Nel caso in cui si utilizzino due soli nastri, questi possono essere specializzati per contenere uno i dati di input (o programma) e l'altro quelli di lavoro e l'output. Si può pensare quindi che il primo si a sola lettura, mentre sul secondo si possa sia leggere che scrivere:

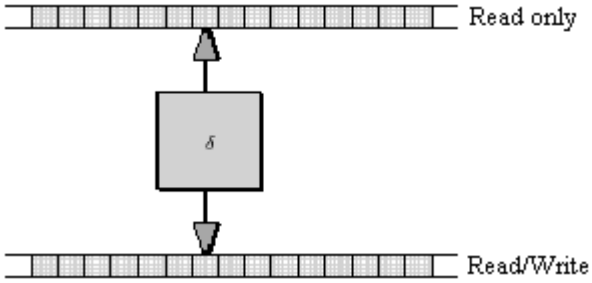


Figura 10 Macchina di Touring con due nastri.

Da questo tipo di macchina si può passare a due altri tipi tramite:

a)LBA (*Linear Bounded Automata*) = limitazione quantitativa della grandezza del nastro di lavoro tramite una funzione dipendete dalla lunghezza del nastro di input:

$$f(w) \quad \text{con} \begin{cases} f(w) = \text{ funzione della lunghezza del nastro di lavoro} \\ w = \text{ lunghezza del nastro di input} \end{cases}$$

b)limitazione sul modo di utilizzo del nastro:

- possono essere differenti gli alfabeti dei due nastri:

$$\Sigma_i = \text{alfabeto del nastro di input}$$
$$\Sigma_l = \text{alfabeto del nastro di lavoro}^{16}$$

- *PDA* (*Push Down Automata*) = lo spostamento alla casella "*n*" corrisponde a cancellare quelle precedenti; il nastro di lavoro diventa quindi una pila:

[16] Se il simbolo è uno solo, si dimostra che il modello ottenuto è più debole. Se invece si utilizzano due numeri, il modello diventa equivalente agli altri.

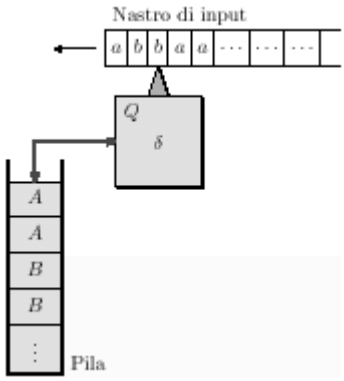


Figura 11 Esempio di PDA.

26/10/2001

N.B.: il nastro di lavoro del PDA (essendo una pila) è limitato inferiormente. Si dimostra che questo non muta la potenza del sistema rispetto ad una macchina con nastro infinito. Si può infatti pensare il nastro infinito diviso in due da un separatore. Gli stessi dati possono essere codificati in uno limitato da un lato copiando, ad es., i dati a destra del separatore nei posti dispari del secondo nastro, e i dati conservati nella parte a sinistra del separatore del primo nastro nei posti pari del secondo nastro:

XXX

Figura 12 Codifica dei dati da un nastro infinito ad uno limitato da un lato.

- *2-FSA* (*two-ways Finite State Automata*) = la testa si può spostare in entrambe le direzioni, mentre la grandezza del nastro di lavoro è costante ed indipendente dal quello di input:

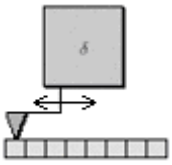


Figura 13 Esempio di un 2-FSA.

Le possibili disposizioni degli elementi nel nastro sono:

$$h^k \quad \text{con} \begin{cases} h = \text{cardinalità dell'alfabeto} \\ k = \text{grandezza del nastro di lavoro} \end{cases}$$

quindi, le possibili configurazioni di una 2-FSA sono finite ed in numero di:

$$n \cdot h^k \quad \text{con } n = \text{numero degli stati della macchina}$$

- *1-FSA* (*one-ways Finite State Automata*) = la testa si può spostare in una sola direzione e la grandezza del nastro di lavoro è costante ed indipendente dal quello di input:

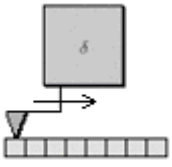


Figura 14 Esempio di un 1-FSA.

Anche in questo caso le possibili configurazioni sono finite.

Gerarchia di Chomsky

Chomsky era un *linguista*, ma le sue conclusioni si possono estendere agli argomenti fin qui trattati, anche per il Teorema 3.

Come gia detto, una grammatica è definita da:

$$G = \{\Sigma,\ V,\ P,\ \Omega\} \quad \text{con} \begin{cases} P \subseteq (\Sigma uv)^* \times (\Sigma uv)^* \\ \Omega = \{\alpha \to \beta\} \in V \end{cases}$$

Si possono avere delle *gerarchie*:

1)Tipo 1 (*Context sensitive CS*) = la lunghezza di α è minore o uguale di quella di β:

$$P : \alpha \to \beta \Leftrightarrow |\alpha| \le |\beta|$$

e le regole sono:

$$\begin{aligned} &\gamma_1 A \gamma_2 \Rightarrow \gamma_1 \alpha_1 \gamma_2 \\ &\beta_1 A \beta_2 \Rightarrow \beta_1 \alpha_2 \beta_2 \end{aligned} \quad \text{con } |\gamma_1 \alpha_1 \gamma_2| \ge 0$$

2)Tipo 2 (*Context free CF*)[17] = a sinistra c'è un solo simbolo non terminale, mentre a destra c'è una qualsiasi stringa composta da caratteri terminali:

$$P : A \to \beta \quad \text{con} \begin{cases} A \in V \\ |A| = 1 \end{cases}$$

Quindi la lunghezza è:

$$1 \le |\alpha|$$ [18]

3)Tipo 3 (*Lineari destre/sinistre LD/LS*) = a sinistra c'è un solo simbolo non terminale, mentre a destra c'è una particolare stringa del tipo:

$$P : \left\{ \begin{matrix} A \to a \\ A \to aB \end{matrix} \right\} \text{lineari destre}$$

oppure:

$$P : \left\{ \begin{matrix} A \to a \\ A \to Ba \end{matrix} \right\} \text{lineari sinistre}$$

Da questa teoria si capisce che c'è un *parallelismo* tra gerarchie di automi e gerarchie di grammatiche:

Automi	**Grammatiche**
Macchina di Touring	Tipo 0
LBA	Tipo 1 (CS)
PDA	Tipo 2 (CF)
2-FSA	Tipo 3 (LD/LS)
1-FSA	

Tabella 1 Parallelo tra gerarchia di automi e gerarchia di grammatiche.

[17] Si ha un simbolo non terminale che è sostituito, indipendentemente dal contesto, con una stringa.

[18] Tranne nel caso in cui α sia la stringa vuota.

Le grammatiche che si trovano in posizione più "alta" hanno algoritmi più generali ma più complessi; al contrario, via via che si scende nella graduatoria, gli algoritmi perdono di generalità e si specializzano, portando quindi ad una loro semplificazione.

Problemi di indecidibilità per le grammatiche[19]

1)membership problem (MP) = dati G e w, $w \in L(G)$?;

2)emptiness problem (EP) = data G, $\exists L : L(G) = \varnothing$?;

3)finiteness problem (FP)[20] = data G, $L(G)$ è finito?;

4)equivalence problem (EqP) = date G_1 e G_2, $L(G_1) = L(G_2)$?;

Problemi di indecidibilità per automi e grammatiche

Per gli automi, si aggiunge un quinto problema:

5)inclusion problem (IP) = dati a_1 e a_2, $L(a_1) \subseteq L(a_2)$?;

che si può verificare se:

$$L(a_1) \subseteq L(a_2) \Leftrightarrow L(a_1) \cap L(a_2)^c = \varnothing$$

Automi

	Touring	**LBA**	**PDA**	**FSA**
MP	I	D	D	D
EP	I	I	D	D
FP	I	I	D	D
EqP	I	I	I	D

Grammatiche

	Tipo 0	**Tipo 1**	**Tipo 2**	**Tipo 3**
MP	I	D	D	D
EP	I	I	D	D
FP	I	I	D	D
EqP	I	I	I	D

Tabella 2 Tabella riassuntiva con i problemi di indecidibilità per automi e grammatiche.

29/10/2001

2-FSA

Si può schematizzare in maniera diversa:

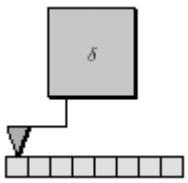


Figura 15 Nuova rappresentazione di una 2-FSA. Il nastro di lavoro è interno.

e la sua tabella degli stati può essere:

	…	**a**	**…**
…			

[19] Vedi "*Membership problem ed emptiness problem*".

[20] Si riconduce ad algoritmi su grafi classici; se non ci sono cicli *L(G)* è finito, se ce ne sono è infinito.

q		*nuovo stato* *spostamento*	
…			

Si può definire una *funzione di transizione*:

$$\delta : Q \times \Sigma \rightarrow Q \times \Delta \qquad \text{con } \Delta = \{-1,\ 0,\ +1\}$$

che fa corrispondere ad una coppia stato-lettera una coppia stato-transizione:

$$(q,\ a) \mapsto (p,\ i) \qquad \text{con } i \in \Delta$$

Questa funzione è definita su $Q \times \Sigma$, cioè per ogni cella della tabella degli stati prima definita. Analizzando la stringa intera, si possono avere tre possibilità per la *condizione d'arresto*:

1) esce a sinistra = l'occhio legge (un blank) nella posizione a sinistra dell'ultima cella $\Rightarrow$ arresto;
2) esce a destra = l'occhio legge (un blank) nella posizione a destra della prima cella $\Rightarrow$ arresto;
3) non esce mai = nessuna condizione d'arresto;

La *stringa* è però *accettata* se alla condizione 1) si aggiunge che si esce in uno stato d'accettazione. Quindi, una *stringa non è accettata* se si verifica una delle seguenti situazioni:

1) esco a sinistra;
2) esco a destra ma non in uno stato d'accettazione;
3) non esco mai;

Per *definire l'automa* devo specificare:

$$a = (\Sigma,\ Q,\ q_0,\ F,\ \delta) \qquad \text{con} \begin{cases} q_0 \in Q \\ F \subseteq Q \end{cases}$$

A questo punto posso definire il *linguaggio* riconosciuto da *a*:

$$L(a)$$

1-FSA

E' sempre un automa del tipo:

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

che ha sempre una *funzione di transizione*, questa volta definita da:

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

che fa corrispondere ad una coppia stato-lettera lo stato successivo (lo spostamento è sempre verso destra):

$$(q,\ a) \mapsto (p,\ +1) \qquad \text{cioè } \delta(q,\ a) = p$$

In questo caso, l'unica *condizione d'arresto* possibile è la 2):

2) esco a destra = l'occhio legge (un blank) nella posizione a destra dell'ultima cella $\Rightarrow$ arresto;

mentre, per verificare che la *stringa* sia *accettata*, bisogna soltanto verificare che si sia usciti in uno stato d'accettazione.

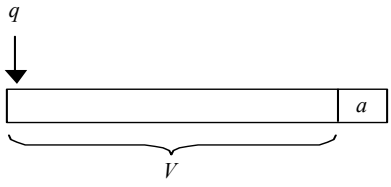
Notando che una 1-FSA è un automa deterministico[21] (dato cioè un input specifico si ottiene un solo output specifico), lo si può vedere come un sistema che ha come input dei dati e deve dare una risposta.

Si definisce:

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

ed, in maniera induttiva, si ha che:

$$\begin{cases} \delta^*(q,\ \varepsilon) = q & \text{con } n = 0 \\ \delta^*(q,\ V\ a) = \delta\left(\delta^*(q,\ V),\ a\right) & \text{con} \begin{cases} n+1 \\ \forall a \in \Sigma \\ \forall V \in \Sigma^* \end{cases} \end{cases}$$



cioè δ (azione dell'automa su un singolo carattere) si estende in un unico modo a δ^* (azione dell'automa sulla stringa); quindi:

$$L(a) = \left\{ w \in \Sigma^* : \delta^*(q_0,\ w) \in F \right\}$$

Esempio 6a

Data la tabella degli stati dell'automa:

δ	**0**	**1**
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Tabella 3 Tabella degli stati dell'automa.

si ha che q_0 è *sia stato di partenza* che *d'accettazione*.

Grafo degli stati

A questo punto si può "convertire" la tabella degli stai in un *grafo degli stati*, ossia si può trovare un altro modo di rappresentare l'automa. Si hanno però delle limitazioni:

1)non possono esistere più di due nodi di partenza;
2)non può esservi un nodo da cui escono un numero di archi maggiore rispetto ai simboli dell'alfabeto;
3)più archi uscenti dallo stesso nodo non possono avere la stessa etichetta;

Esempio 6b

Facendo riferimento all'automa dell'Esempio 6a si ha che:

[21] Vedi "*Sistemi TAG*" e "*DFA*".

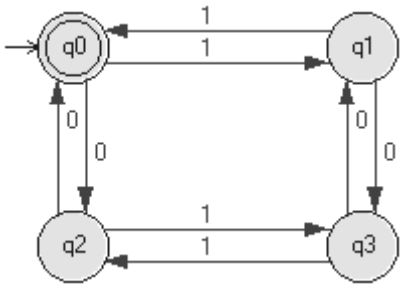


Figura 16 Esempio di grafo degli stati per la macchina dell'Esempio 6a.

Da una rapida analisi si scopre che le stringhe riconosciute da questa macchina sono quelle con un numero pari sia di "0" che di "1", in quanto una stringa del genere permette un cammino dallo stato di partenza a quello d'accettazione. Quindi, "110100010" non è accettata (si ferma in q_2); mentre "1101000100" è accettata (si ferma in q_0).

Esercizi (per le "vacanze" !!!)

Definire, se possibile, i linguaggi che contengono:

1)tutte le parole sull'alfabeto $\{a, b, c\}$ che contengono o il blocco "*cba*" oppure il blocco "*bac*";
2)tutte le parole sull'alfabeto $\{a, b\}$ che contengono un numero pari di "*a*" e un numero di "*b*" multiplo di 3;
3)tutte le parole sull'alfabeto $\{a, b\}$ che contengono lo stesso numero di "*a*" e di "*b*";
4)tutte le parole sull'alfabeto $\{a, b\}$ che simulano il linguaggio delle parentesi;
5)tutte le parole palindrome;
6)tutte le stringhe che, in notazione decimale, rappresentano numeri che o sono multipli di 1 o sono multipli di 3;
7)tutte le stringhe che, in notazione binaria (decimale), rappresentano numeri che sono potenze di 2;
8)tutte le stringhe che, in notazione binaria (decimale), rappresentano numeri primi;

Soluzioni

1)Il grafo dell'automa NFA è:

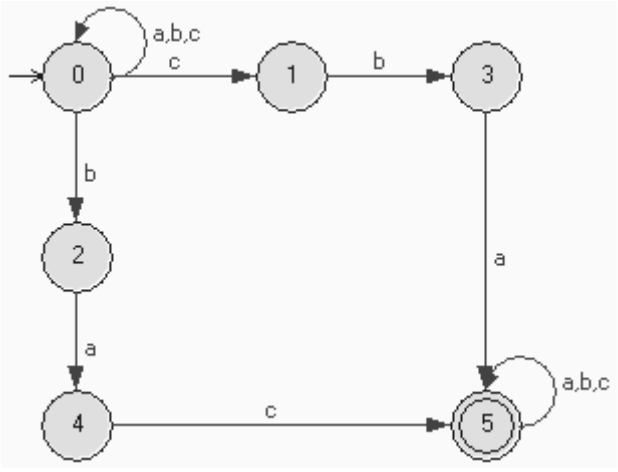


Figura 17 Grafo dell'NFA del primo esercizio.

mentre la grammatica è composta dalle regole di produzione:

1)$\Omega \rightarrow cba$;
2)$\Omega \rightarrow bac$;
3)$\Omega \rightarrow a\Omega$;
4)$\Omega \rightarrow b\Omega$;
5)$\Omega \rightarrow c\Omega$;
6)$\Omega \rightarrow \Omega a$;
7)$\Omega \rightarrow \Omega b$;
8)$\Omega \rightarrow \Omega c$;

2)Si costruiscono i due automi e poi si fa l'intersezione:

Figura 18 Automa che riconosce una stringa formata da un numero pari di "*a*".

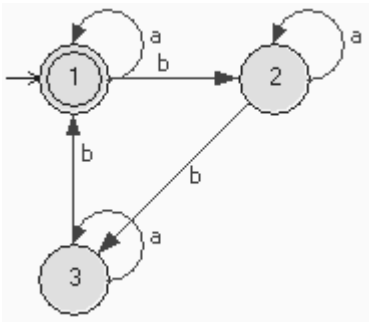


Figura 19 Automa che riconosce una stringa formata da un numero di "*b*" multiplo di 3.

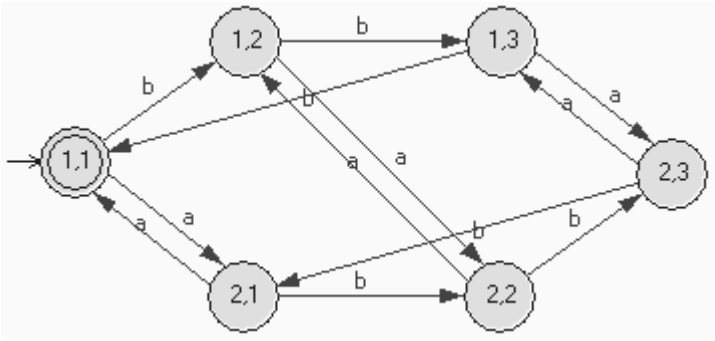


Figura 20 Automa risultante dall'intersezione degli altri due.

mentre la grammatica è composta dalle regole di produzione:

1)$\Omega \rightarrow a\Omega a$;
2)$\Omega \rightarrow b\Omega\ b\Omega\ b$;
3)$\Omega \rightarrow \varepsilon$;

oppure (visto che le regole precedenti non riconoscono la stringa "*babab*"):

1)$\Omega \rightarrow aF$;
2)$\Omega \rightarrow bB$;
3)$B \rightarrow aE$;
4)$B \rightarrow bC$;
5)$C \rightarrow aD$;
6)$C \rightarrow b\Omega$;
7)$D \rightarrow aC$;
8)$D \rightarrow bF$;
9)$E \rightarrow aB$;
10)$E \rightarrow bD$;
11)$F \rightarrow a\Omega$;
12)$F \rightarrow bE$;

13)$C \rightarrow b$;
14)$F \rightarrow a$;

3)L'automa che riconosce tutte le stringhe che contengono lo stesso numero di "*a*" e di "*b*" è:

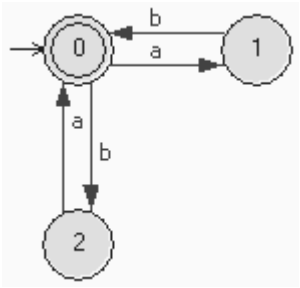


Figura 21 Grafo del DFA dell'esercizio 3.

mentre la grammatica è composta dalle regole di produzione:

1)$\Omega \rightarrow \Omega\Omega$;
2)$\Omega \rightarrow ab$;
3)$\Omega \rightarrow ba$;

4)Vedi il "*Linguaggio delle parentesi*".

5)Vedi "*Esempio 20*", "*Esempio 24*", "*Esempio 27*" ed "*Esempio 32*".

6)I due automi sono:

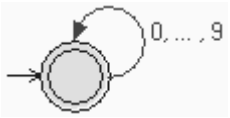


Figura 22 Automa che riconosce i numeri multipli di 1.

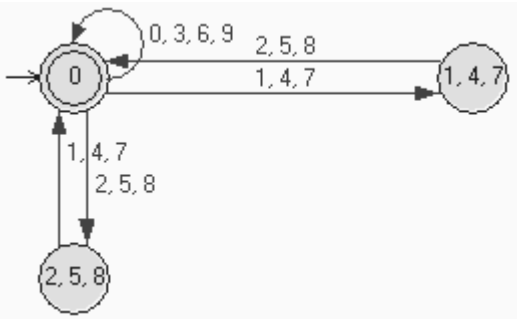


Figura 23 Automa che riconosce i numeri multipli di 3.

e l'automa richiesto dall'esercizio coincide, banalmente, con il secondo. La grammatica è composta dalle regole di produzione:

1)$\Omega \rightarrow 0$;
2)$\Omega \rightarrow 3$;
3)$\Omega \rightarrow 6$;
4)$\Omega \rightarrow 9$;
5)$\Omega \rightarrow 1A$;
6)$\Omega \rightarrow 4A$;
7)$\Omega \rightarrow 7A$;
8)$\Omega \rightarrow 2B$;
9)$\Omega \rightarrow 5B$;
10)$\Omega \rightarrow 8B$;
11)$A \rightarrow 2$;
12)$A \rightarrow 5$;
13)$A \rightarrow 8$;
14)$B \rightarrow 1$;
15)$B \rightarrow 4$;

16)$B \rightarrow 7$;

7)L'automa è:

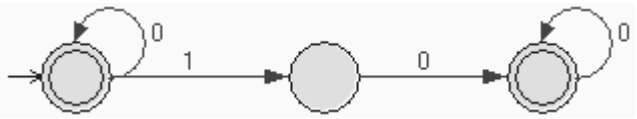


Figura 24 Automa che riconosce, in notazione binaria, numeri dati da potenze di 2.

mentre la grammatica è composta dalle regole di produzione:

1)$\Omega \rightarrow 0$;
2)$\Omega \rightarrow 1A$;
3)$\Omega \rightarrow 0A$;
4)$A \rightarrow 0$;

8)Non esiste.

Esempio 7

Si vuole costruire un automa che verifichi la correttezza di una somma in notazione decimale. L'alfabeto sarà quindi composto da:

$$\{0,\ 1,\ \ldots,\ 9,\ +,\ =\}$$

Per semplicità nei ragionamenti, si utilizzerà la notazione binaria[22]. Vogliamo ad es. verificare quali delle due sequenze sia giusta:

$$\begin{array}{r} 0101 \\ \underline{0110} \\ 1011 \end{array} \qquad\qquad \begin{array}{r} 0101 \\ \underline{0110} \\ 1001 \end{array}$$

che corrisponde a fare le seguenti operazioni:

$$\begin{array}{r} 5+ \\ \underline{6=} \\ 11 \end{array} \qquad\qquad \begin{array}{r} 5+ \\ \underline{6=} \\ 9 \end{array}$$

A prima vista, si capisce subito che la prima è quella giusta, mentre la seconda è sbagliata[23]. Si procede scegliendo la seguente codifica in triple verticali:

0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**

dove le prime due righe rappresentano i due elementi da sommare, mentre la terza è il risultato della somma. Codificando le due somme in questo modo si ha che:

$$\begin{array}{r} 0101 \\ \underline{0110} \\ 1011 \end{array} \Rightarrow \text{BGDF} \qquad\qquad \begin{array}{r} 0101 \\ \underline{0110} \\ 1001 \end{array} \Rightarrow \text{BGCF}$$

Bisogna quindi costruire un linguaggio che riconosca la sequenza corretta di lettere.

[22] Tenuto conto che da una si può sempre passare all'altra in maniera semplice e veloce.
[23] Noi dobbiamo costruire un automa che "non ragioni", ma che sappia dare una risposta alla nostra domanda senza conoscere la natura degli oggetti che sta trattando.

Si noti che, leggendo la stringa da sinistra verso destra[24], dopo una G ci deve essere necessariamente[25] o una A, o una B, o una D oppure una F. La prima stringa risponde a questo criterio, mentre la seconda è errata (infatti dopo la G si ha una C).

N.B.: in questo modo la macchina ci da la risposta giusta senza calcolare effettivamente la somma tra i due numeri ma soltanto lavorando con stringhe; abbiamo quindi raggiungo il nostro scopo.

Esercizio 3

Utilizzando lo schema dell'Esempio 7, creare un automa che verifichi la correttezza di un prodotto in notazione binaria.

Si procede scegliendo la seguente codifica in triple verticali:

0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	1	0	1	0	1	0
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**

dove le prime due righe rappresentano i due elementi da sommare, mentre la terza è il risultato del prodotto.

05/11/2001

Grafi

Un *grafo orientato* (o diretto o di-grafo) G è una coppia di vertici e archi:

$$G = (V,\ E) \qquad \text{con} \begin{cases} V \text{ insieme finito} \\ E \text{ relazione binaria su } V \\ E \subseteq V \times V \end{cases}$$

L'insieme V è chiamato l'*insieme dei vertici* o nodi di G ed i suoi elementi sono chiamati vertici o nodi; l'insieme E è chiamato l'*insieme degli archi* di G ed i suoi elementi sono chiamati archi. La Figura 17 (a) è una rappresentazione di un grafo orientato con insieme dei vertici {1, 2, 3, 4, 5, 6}. Nella figura i vertici sono disegnati con cerchi e gli archi sono disegnati con frecce. Si noti che sono possibili *cappi*, ossia archi da un vertice a se stesso.

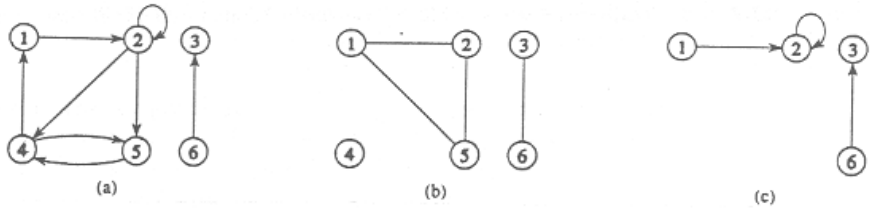


Figura 25 (a) Un grafo orientato. (b) Un grafo non orientato. (c) Il sottografo del grafo (a).

In un *grafo non orientato*:

$$G = (V,\ E)$$

l'insieme degli archi E è costituito da coppie non ordinate di vertici (piuttosto che da coppie

[24] Basta poi invertire la stringa; vedi "*Inversione*".
[25] In quanto si presuppone che la somma precedente non ha generato riporto. Seguono lo stesso ragionamento anche: A, D ed F.

ordinate); cioè, un arco è un insieme:

$$\{u,\ v\} \qquad \text{con} \begin{cases} u, v \in V \\ u \neq v \end{cases}$$

Per convenzione, si usa la notazione (u, v) per un arco, piuttosto che la notazione insiemistica {u, v}; inoltre (u, v) e (v, u) sono considerati essere lo stesso arco. In un grafo non orientato i cappi sono proibiti, per cui ogni arco consiste esattamente di due vertici distinti. La Figura 17 (b) è una rappresentazione di un grafo non orientato con insieme dei vertici {1, 2, 3, 4, 5, 6}.

Molte definizioni per grafi orientati e non orientati coincidono, benché certi termini abbiano un significato leggermente diverso. Se (u, v) è un arco di un grafo orientato e $G = (V, E)$, si dice che (u, v) è *incidente* o esce dal vertice u ed è incidente o entra nel vertice v. Per esempio, gli archi che escono dal vertice 2 nella Figura 17 (a) sono (2, 2), (2, 4) e (2, 5); mentre gli archi che entrano nel vertice 2 sono (1, 2) e (2, 2). Se (u, v) è un arco di un grafo non orientato $G = (V, E)$, si dice che (u, v) è incidente sui vertici u e v. Nella Figura 17 (b), gli archi incidenti sul vertice 2 sono (1, 2) e (2, 5).

Se (u, v) è un arco di un grafo $G = (V, E)$, si dice che il vertice v è *adiacente* al vertice u. Quando il grafo è non orientato la relazione di adiacenza è simmetrica, mentre quando è orientato la relazione non è necessariamente simmetrica. Se v è adiacente a u in un grafo orientato talvolta si scrive:

$$u \rightarrow v$$

Nelle parti (a) e (b) della Figura 17, il vertice 2 è adiacente al vertice 1, infatti l'arco (1, 2) è presente in entrambi i grafi. Il vertice 1 non è adiacente al vertice 2 nella Figura 17 (a), poiché l'arco (2, 1) non appartiene al grafo.

Il *grado* di un vertice in un grafo non orientato è il numero di archi incidenti su di esso. Per esempio, il vertice 2 nella Figura 17 (b) ha grado 2. In un grafo orientato, il grado uscente di un vertice è il numero di archi che escono da esso ed il grado entrante è il numero di archi che entrano nel vertice. Il grado di un vertice in un grafo orientato è il suo grado entrante più il suo grado uscente. Il vertice 2 nella Figura 17 (a) ha grado entrante 2, grado uscente 3 e grado 5. Un vertice di grado 0, come il vertice 4 nella Figura 17 (b), si dice isolato.

Un *cammino* di lunghezza k da un vertice u ad un vertice u' in un grafo $G = (V, E)$ è una successione:

$$\langle v_0,\ v_1,\ v_2,\ \ldots,v_k \rangle \in E$$

di vertici tale che :

$$u = v_0,\ u' = v_k \text{ et } (v_{i-1},\ v_i) \in E \qquad \forall i = 1,\ 2,\ \ldots,k$$

La *lunghezza di un cammino* è il suo numero di archi. Il cammino contiene i vertici v_o, v_1,…,v_k e gli archi (v_o , v_1,), (v_1, v_2), …, (v_{k-1} , v_k). Se vi è un cammino p da u a u', si dice che u' è raggiungibile da u tramite p; ciò talvolta, se G è orientato, si scrive come:

$$u \overset{p}{\rightarrow} u'$$

Un *cammino* è *semplice* se tutti i vertici sono distinti. Nella Figura 17 (a) il cammino (1, 2, 5, 4) e un cammino semplice di lunghezza 3. Il cammino (2, 5, 4, 5) non è semplice.

Un *sottocammino* di un cammino:

$$p = \langle v_0,\ v_1,\ \ldots,v_k \rangle$$

è una sottosequenza contigua dei suoi vertici. Cioè:

$$\forall\ 0 \leq i \leq j \leq k$$

la sottosequenza di vertici (v_i, v_{i+1}, …, v_j) è un sottocammino di p.

In un grafo orientato, un cammino (v_o , v_1, …, v_k) forma un ciclo se $v_o = v_k$ e il cammino contiene almeno un arco. Il ciclo è semplice, se v_1 , v_2, …, v_k sono distinti. Un cappio è un ciclo di lunghezza I. Due cammini (v_o , v_1, …, v_{k-1} , v_0) e (v'_o , v'_1, …, v'_{k-1} , v'_0) formano lo stesso ciclo se:

$$\exists\, j \text{ intero } :\ v_i^{'} = v_{(i+j) \bmod k} \quad \forall\ i = 0,\ 1,\ \ldots\ ,\ k-1$$

Nella Figura 17 (a) il cammino (1, 2, 4, 1) forma lo stesso ciclo dei cammini (2, 4, 1, 2) e (4, 1, 2, 4). Questo ciclo è semplice, ma il ciclo (1, 2, 4, 5, 4, 1) non lo è. Il ciclo (2, 2) formato dall'arco (2, 2)
è un cappio. Un grafo orientato senza cappi e semplice.

In un grafo non orientato, un cammino (v_o , v_1, …, v_k) forma un ciclo (semplice) se:

$$k \geq 3,\ v_0 = v_k \text{ et } v_1,\ v_2,\ \ldots\ , v_k \text{ sono distinti}$$

Per es., nella Figura 17 (b), il cammino (1, 2, 5, 1) è un ciclo. Un grafo senza cicli è aciclico.

DFA e NFA

Da ora in poi indicheremo con *DFA* (*deterministic finite automata*) un qualsiasi automa a stati finiti deterministico[26]. Ad es., dato l'alfabeto:

$$\Sigma = \{a,\ b,\ c\}$$

si può avere il seguente grafo degli stati:

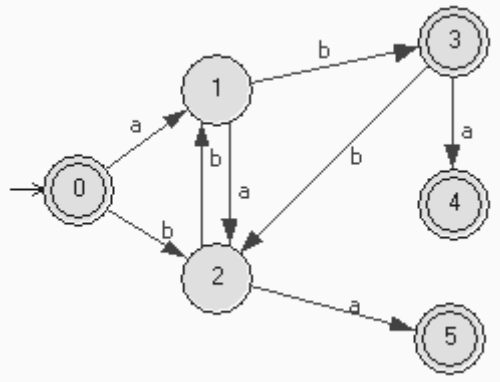


Figura 26 Esempio di grafo degli stati per la macchina dell'esempio.

Le "restrizioni" di questo grafo degli stati di una FSA sono:

1)ha un solo stato iniziale;
2)da ogni vertice escono esattamente un numero di archi pari al numero delle lettere dell'alfabeto;

Analogamente, da ora in poi indicheremo con *NFA* (*non deterministic finite automata*) un qualsiasi automa a stati finiti non deterministico. In questo caso:

1)si possono avere più stati iniziali;
2)da ogni vertice possono uscire più archi rispetto al numero delle lettere dell'alfabeto;

E' proprio quest'ultima caratteristica che determina il comportamento non deterministico, visto che vi sono più percorsi possibili.

[26] Vedi "*Sistemi TAG*" e "*1-FSA*".

Esempio 8

Dato l'alfabeto:

$$\Sigma = \{a,\ b,\ c\}$$

ed un linguaggio L che racchiude tutte le parole che contengono come blocco "*ab*" oppure "*bc*". Quindi, ad es., la stringa:

$$acbacccacbbbac \notin L$$

Costruire un NFA consiste nel disegnare un grafo in cui tutti i cammini corrispondono a tutte le stringhe, delle quali solo quelle accettate finiscono con uno stato finale:

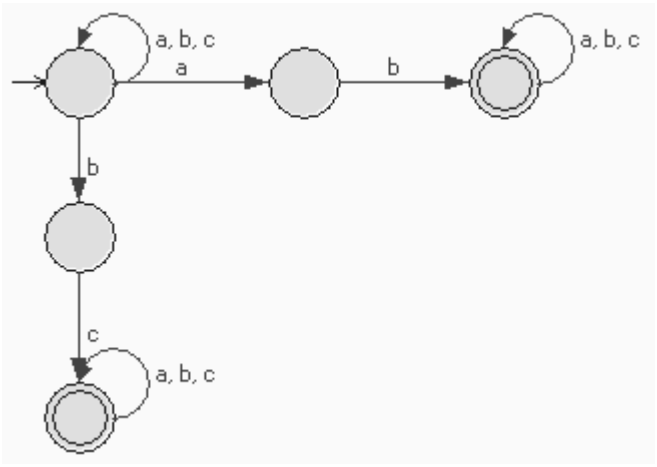


Figura 27 Grafo della NFA dell'Esempio 8.

N.B.: questo è un grafo ma non degli stati (come quello di una qualsiasi DFA) in quanto dal nodo di partenza escono ben 5 archi (e non 3 quante le lettere dell'alfabeto).

Esempio 9

Dato l'alfabeto ed il linguaggio:

$$\Sigma = \{a,\ b\},\ L_k$$

di tutte le parole che hanno nella posizione k-esima, a partire da destra, la lettera "*a*". Quindi, ad es.:

$$k = 3 \quad abbaaab\underset{\uparrow}{a}bb \in L_3$$

Il grafo del relativo NFA, per k=3, è

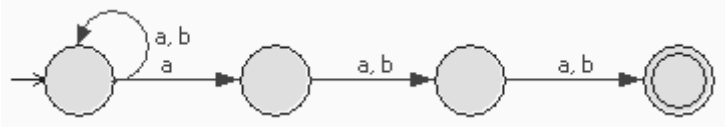


Figura 28 Grafo dell'NFA dell'Esempio 9.

DFA vs NFA

Le differenze tra una DFA ed una NFA si possono riassumere in questo modo:

NFA	***DFA***
$a_N = (\Sigma,\ Q,\ I,\ F,\ \delta)$	$a_D = (\Sigma,\ Q,\ q_0,\ F,\ \delta)$
$I \subseteq Q$	q_0 (lo stato iniziale è uno solo)

$F \subseteq Q$	$F \subseteq Q$
$\delta \subseteq Q \times \Sigma \times Q$ $\delta : Q \times \Sigma \to P(Q)$ $\delta(q, a) = \{q_1, q_2, \ldots, q_n, S\}$	δ è ad un solo valore

Per un NFA si definisce:

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

ed, in maniera induttiva, si ha che:

$$\begin{cases} \delta^*(q, \varepsilon) = q \\ \delta^*(q, V\ a) = \bigcup_{P \in \delta^*(q, V)} \delta(P, a) \end{cases}$$

cioè δ (azione dell'automa su un singolo carattere) si estende in un unico modo a δ^* (azione dell'automa sulla stringa).

Indichiamo la famiglia di linguaggi riconosciti da una DFA con:

$$L(\text{DFA})$$

Analogamente, indichiamo la famiglia di linguaggi riconosciti da una NFA con:

$$L(\text{NFA})$$

Infine, banalmente[27] si ha che:

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

08/11/2001

Teorema 4 (subset construction)

$$L(\text{DFA}) = L(\text{NFA})^{28}$$

Dim.:

N.B.: si dimostra tramite una *prova costruttiva*[29] (o *subset construction*), ossia si trovano delle regole per convertire, per far "emulare" le operazioni possibili con un automa tramite l'altro e viceversa.

Visto che l'inclusione:

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

come si è visto, è banale, per dimostrare il teorema bisogna verificare l'inclusione inversa:

$$L(\text{DFA}) \supseteq L(\text{NFA})$$

Preso un NFA:

[27] Visto che un DFA è un caso particolare di un NFA.
[28] C'è però un "prezzo" da pagare, che è nella crescita del numero di stati (vedi il "*Teorema 5*").
[29] Vedi "*Linguaggio While*".

$$a_N = (\Sigma,\ Q,\ I,\ F,\ \delta)$$
$$I,\ F \subseteq Q$$
$$\delta:\ Q \times \Sigma \to P(Q)$$

si vuole "costruire" il corrispettivo DFA:

$$a_D = (\Sigma,\ Q_D,\ q_{0_D},\ F_D,\ \delta_D)$$
$$q_{0_D} = I \in Q_D$$
$$\delta_D:\ Q_D \times \Sigma \to Q_D$$

Essendo una prova costruttiva, utilizzeremo l'automa dell'Esempio 8, che riconosce la presenza della sottostringa "*ab*" oppure della sottostringa "*bc*". Si prende:

$$Q_D = \left\{ \underset{1}{\varnothing},\ \underset{2}{\{q_0\}},\ \underset{3}{\{q_1\}},\ \underset{\underline{4}}{\{q_2\}},\ \underset{5}{\{q_3\}},\ \underset{\underline{6}}{\{q_4\}},\ \underset{7}{\{q_0,\ q_1\}},\ \underset{\underline{8}}{\{q_0,\ q_2\}},\ \underset{9}{\{q_0,\ q_3\}},\ \underset{\underline{10}}{\{q_0,\ q_4\}},\ \ldots \right\}$$
$$Q_D = P(Q)$$
$$F_D = \{T \in P(Q)\ :\ T \cap F \neq \varnothing\}$$
$$\delta_D(T,\ a) = \bigcup_{t \in T} \delta(T,\ a)$$

in cui gli stati d'accettazione sono quelli che contengono q_2 o q_4 (o anche entrambi). Nell'esempio sono quelli sottolineati, quindi:

$$F_D = \{4,\ 6,\ 8,\ 10,\ \ldots\}$$

Si procede ricavando le funzioni δ (di transizione) del DFA a partire da quella dell'NFA. Per fare ciò, si guarda il grafo della Figura 19. Ecco alcuni esempi:

$$\delta_D(\{q_0\},\ a) = \delta(q_0,\ a) = \{q_0,\ q_1\}$$
$$\delta_D(\{q_0\},\ b) = \delta(q_0,\ b) = \{q_0,\ q_3\}$$
$$\delta_D(\{q_0\},\ c) = \delta(q_0,\ c) = \{q_0\}$$
$$\ldots$$
$$\delta_D(\{q_0,\ q_1\},\ a) = \delta(q_0,\ a) \cup \delta(q_1,\ a) = \{q_0,\ q_1\} \cup \varnothing = \{q_0,\ q_1\}$$
$$\delta_D(\{q_0,\ q_1\},\ b) = \delta(q_0,\ b) \cup \delta(q_1,\ b) = \{q_0,\ q_3\} \cup \{q_2\} = \{q_0,\ q_2,\ q_3\}$$
$$\delta_D(\{q_0,\ q_1\},\ c) = \delta(q_0,\ c) \cup \delta(q_1,\ c) = \{q_0\} \cup \varnothing = \{q_0\}$$
$$\ldots$$
$$\delta_D(\{q_0,\ q_3\},\ a) = \delta(q_0,\ a) \cup \delta(q_3,\ a) = \{q_0,\ q_1\} \cup \varnothing = \{q_0,\ q_1\}$$
$$\delta_D(\{q_0,\ q_3\},\ b) = \delta(q_0,\ b) \cup \delta(q_3,\ b) = \{q_0,\ q_3\} \cup \varnothing = \{q_0,\ q_3\}$$
$$\delta_D(\{q_0,\ q_3\},\ c) = \delta(q_0,\ c) \cup \delta(q_3,\ c) = \{q_0\} \cup \{q_4\} = \{q_0,\ q_4\}$$
$$\ldots$$

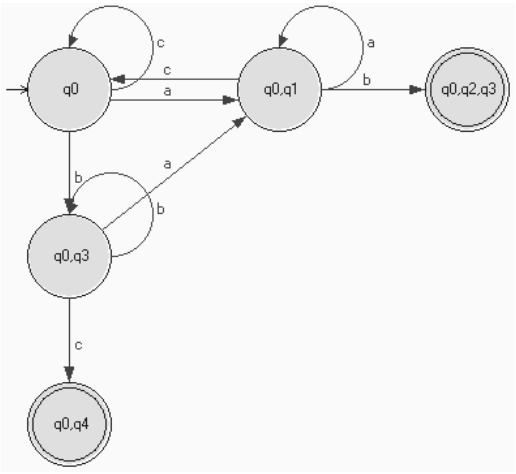


Figura 29 Grafo "parziale" degli stati del DFA corrispondente all'NFA dell'Esempio 8.

Il grafo finale degli stati è:

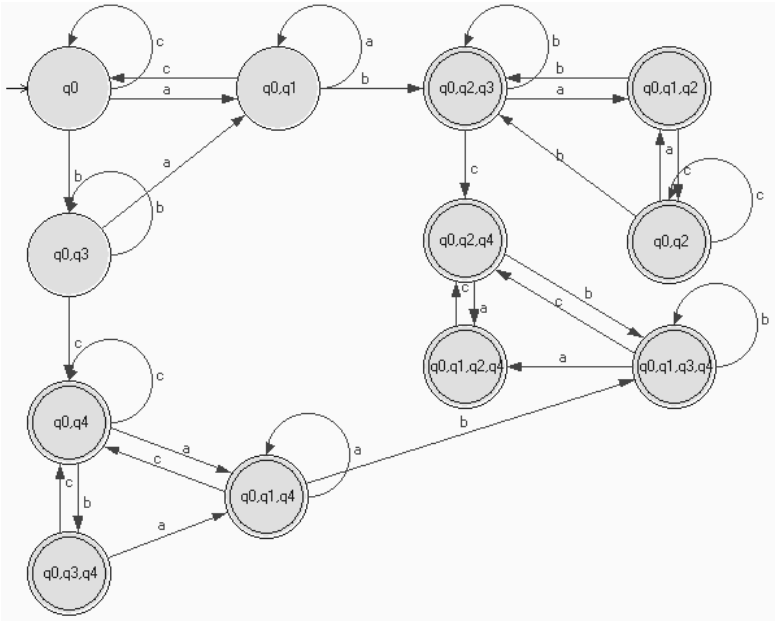


Figura 30 Grafo degli stati del DFA corrispondente all'NFA dell'Esempio 8.

N.B.: malgrado la regola direbbe che, per passare da una NFA ad una DFA, necessitano:

$$2^n \text{ stati} \quad \text{con } n = \text{numero degli stati del NFA}$$

si dimostra[30] che effettivamente necessitano:

$$\leq 2n \text{ stati} \quad \text{con } n = \text{numero degli stati del NFA}$$

Si definisce che un nodo q è *accessibile* se esiste un cammino che da uno stato iniziale porta ad un altro nodo, ossia:

$$\exists\, w \in \Sigma^* :\ \delta^*\left(q_0,\ w\right) = q$$

Mentre, si definisce *co-accessibile* se esiste un cammino che da un nodo porta ad uno stato d'accettazione, cioè:

$$\exists\, v \in \Sigma^* :\ \delta^*\left(q,\ v\right) = P \qquad \text{con } P \in F$$

N.B.: si noterà che nel grafo sono stati eliminati gli stati che non sono accessibili o co-accessibili

[30] Vedi il "*Teorema 5*".

(questo non cambia il linguaggio riconosciuto dall'automa). Si è anche scelto l'utilizzo dell'*insieme vuoto* (per altro non rappresentato nel grafo) per indicare tutte quelle transizioni non definite, come ad es.:

$$\delta_D\left(\{q_3\},\ a\right) = \delta\left(q_3,\ a\right) = \nexists \Rightarrow \varnothing$$

questo proprio perché la subset construction elimina il non determinismo e le transizioni non definite.

A questo punto si possono unificare tutti gli stati d'accettazione in uno solo (visto che una volta che si giunge in uno di questi non si può andare in uno di non accettazione) e si ottiene il seguente grafo degli stati:

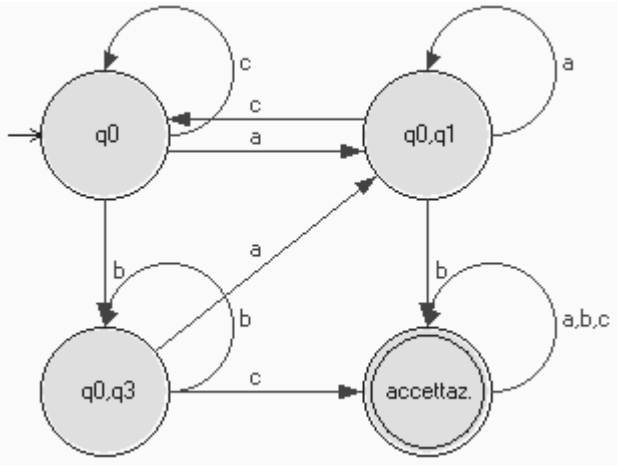


Figura 31 Grafo del DFA risultante.

Bisogna dimostrare che:

$$L\left(Q_D\right) = L\left(Q\right)$$

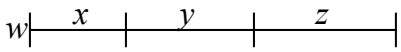
quindi:

$$L\left(Q_D\right) = \left\{w \in \Sigma^* \ :\ \delta_D^*\left(q_{0_D},\ w\right) \in F_D\right\}$$

dato che lo stato iniziale del DFA è l'insieme degli stati iniziali dell'NFA, mentre gli stati d'accettazione del DFA sono quelli che contengono almeno uno stato d'accettazione dell'NFA.

Insiemi di fattori

Sono blocchi di stringhe tali che:



con *y* fattore di *w* poiché:

$$w = xyz$$

mentre l'insieme dei fattori di *w* è:

$$F\left(w\right)$$

Esempio 10

Data la stringa:

$$w = aababababbab$$

e l'insieme di fattori:

$$F(w) = \{\varepsilon,\ a,\ b,\ aa,\ ab,\ ...\}$$

si vuole costruire un DFA che dica se è presente o meno una sottostringa nella stringa *w*. Il grafo dell'automa lineare non deterministico è:

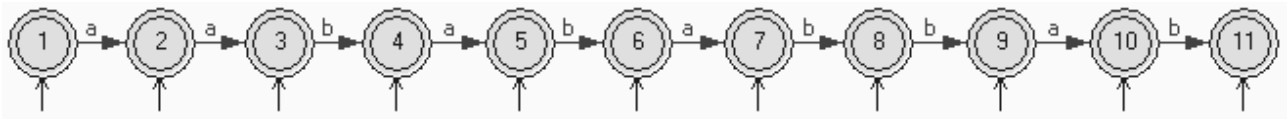


Figura 32 Grafo dell'automa NFA.

Tramite subset construction si ha il corrispettivo DFA:

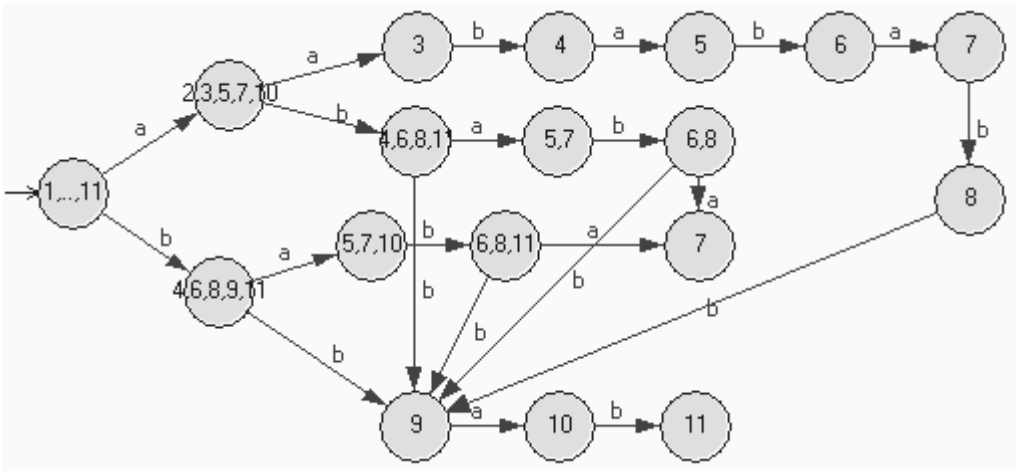


Figura 33 Grafo del DFA dell'esempio.

che rappresenta tutte e solo le parole accettate dall'automa; questi sono quindi tutti e solo gli stati d'accettazione.

N.B.: in questi casi il *tempo di ricerca* di una sottostringa dipende dalla sua taglia, e non dalla grandezza della stringa nella quale cercare. Inoltre, questo *grafo non ha cicli*, in quanto se ci fossero si avrebbero infiniti cammini.

Sottoinsiemi disgiuntivi

Da un insieme *I*, con cardinalità:

$$card(I) = n$$

prendo la famiglia di sottoinsiemi *F*. Dico che *F* è *disgiuntiva* se:

$$\forall x,\ y \text{ insiemi} \in F$$

si ha una sola delle tre possibilità:

1) $x \cap y = \varnothing$
2) $x \subseteq y$
3) $x \supseteq y$

Sorgono spontanee le seguenti domande:

1)Se applico la subset construction ad un automa lineare con tutti gli stati iniziali e finali, l'insieme che costituisco è una famiglia disgiuntiva?
2)Se ho una famiglia di sottoinsiemi disgiuntivi di una famiglia di cardinalità *n*, il numero dei

suoi insiemi non è al più superiore a $2n$?

09/11/2001

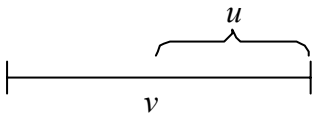
Teorema 5

NFA di taglia n $\Leftrightarrow$ DFA di taglia $\leq 2n$[31]

Dim.:

Facendo la subset construction al grafo si ottiene una sottofamiglia disgiuntiva:

$u,\ v \in F(w)$	sottostringhe
$I(u),\ I(v)$	insiemi
$I(u) \subseteq I(v)$	u è suffisso di v



ossia, o uno è disgiunto dall'altro oppure uno è sottoinsieme dell'altro. Si ha quindi una specie di "gerarchia" tra gli insiemi, una struttura ad albero. Il fatto che il numero di stati di un DFA così costruito si limiti, al massimo, a $2n$ è dato da:

$$2(n-1) \ + \ \text{insieme stesso} \ + \ \varnothing = 2n-2+1+1 = 2n$$

dove il "2" indica proprio la partizione[32] (o divisione per due) dell'insieme, che può essere iterata fino ad un massimo di "$n-1$" volte.

Equivalenza tra modelli

L'equivalenza tra tutti i modelli non è sempre vera:

$$FSA \Rightarrow DFA = NFA$$

$$LBA \Rightarrow DLBA \overset{?}{=} NLBA$$

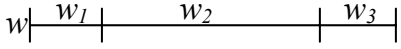
$$PDA \Rightarrow DPDA \subseteq NPDA$$

Lemma di iterazione[33]

sia L un linguaggio riconosciuto da un FSA $\Rightarrow$ $\exists\, k \in \mathbb{N}_+ \ :\ w \in L$ et $w = w_1 w_2 w_3$ $\Rightarrow \exists$ fattorizzazione $w_2 = xyz$ per cui $w_1 x y^n z w_3 \in L$ con $\begin{cases} |w_2| > k \\ y \neq \varepsilon \\ \forall n \geq 0 \end{cases}$

Dim.:

Il lemma dice che se si prende una parola $w \in L$ e un suo pezzo w_2 "abbastanza lungo":



in questo c'è un *fattore iterante* y per cui se si itera n volte (quindi anche se n=0, cioè lo si cancella), la stringa continua ad appartenere al linguaggio.

N.B.: w_1 o w_3 potrebbero non esserci, quindi w_2 si troverebbe all'inizio o alla fine di w.

[31] La crescita non è quindi esponenziale.
[32] In questo caso partizione e dividere in sottoinsiemi (non necessariamente della stessa grandezza) da lo stesso risultato.
[33] E' uno strumento, una condizione necessaria ma non sufficiente.

Dire che L è il linguaggio riconosciuto da un FSA significa dire che esiste un automa:

$$a = \{\Sigma,\ Q,\ q_0,\ F,\ \delta\}$$

che riconosce il linguaggio:

$$L(a) = L$$

Si indica la cardinalità di Q con:

$$k = card(Q)$$

Dire che $w \in L$ significa che:

$$\delta^*(q_0,\ w) \in F \qquad \text{con } w = w_1 w_2 w_3$$

Si può definire con q_1 lo stato nel quale si giunge dopo aver letto w_1:

$$q_1 = \delta^*(q_0,\ w_1)$$

Analogamente, si può definire con q_2 lo stato nel quale si giunge dopo aver letto w_1 e w_2:

$$q_2 = \delta^*(q_0,\ w_1 w_2) = \delta^*(q_1,\ w_2)$$

Si indica con:

$$w_2 = a_1 a_2 ... a_r \qquad \text{con } r > k$$

e si chiama:

$$q^{(i)} = \delta^*(q_1,\ a_1 a_2 ... a_i) \qquad \text{con } a_i \in \Sigma$$

Si definisce quindi una successione di stati rappresentanti gli stati visitati:

$$\left(q^{(1)},\ q^{(2)},\ ...\ ,\ q^{(i)},\ ...\ ,\ q^{(r)}\right)$$

e poiché $r > k$, esistono degli stati uguali:

$$\exists\ i, j\ :\ q^{(i)} = q^{(j)} \qquad \text{con } i \neq j$$

Si pone:

$$x = a_1 ... a_i$$
$$y = a_{i+1} ... a_j$$
$$z = a_{j+1} ... a_r$$

Visto che:

$$q^{(i)} = q^{(j)}$$

si ha ovviamente che:

$$\delta^*\left(q_1,\ a_1 ... a_i \left(a_{i+1} ... a_j\right)^n a_{j+1} ... a_r\right) = q_2$$

dove questo è un ciclo:

$$\delta^*\left(q^{(i)},\ a_{i+1} ... a_j\right) = q^{(i)} = q^{(j)}$$

Sostituendo si ha che:

$$\begin{aligned} &\delta^*\left(q_1,\ xy^n z\right) = q_2 \\ &\delta^*\left(q_0,\ w_1 xy^n z w_3\right) \in F \end{aligned} \qquad \forall n \geq 0$$

e questo implica che:

$$w_1 xy^n z w_3 \in L$$

L'immagine sottostante fa vedere come, presa una parola:

$$w = w_1 w_2 w_3 \qquad \text{con} \begin{cases} w \in L \\ w_2 = xy^n z \end{cases}$$

se *y* indica un ciclo ripetuto *n* volte, basta sostituire:

$$w_2 = xy^n z \Leftrightarrow w = w_1 w_2 w_3 = w_1 xy^n z w_3 \Leftrightarrow w \in L \ \Leftrightarrow \ w_1 xy^n z w_3 \in L$$

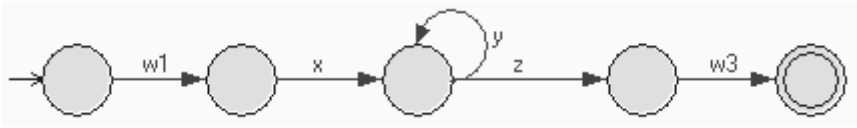


Figura 34 Visualizzazione grafica del lemma di iterazione, dove *y* indica il ciclo ripetuto *n* volte.

15/11/2001

Proprietà di chiusura della famiglia L(FSA)

Preso:

$$L \in L(\text{FSA}) \qquad \text{con} \begin{cases} L \subseteq \Sigma^* \\ \Sigma^* \text{ monoide} \end{cases}$$

questo linguaggio s'intende chiuso rispetto alle seguenti *operazioni regolari* della famiglia dei linguaggi riconosciti da un FSA:

1)*booleane*[34] (unione, intersezione, complemento);
2)*concatenazione* (o prodotto) di linguaggi;
3)*star* (di Kleene);

Unione $\cup$

Presi due linguaggi:

$$L_1,\ L_2 \in L(\text{FSA}) \Rightarrow L_1 \cup L_2 \in L(\text{FSA})$$

cioè sono chiusi rispetto all'unione.

Dim.:

Da un punto di vista intuitivo, presi due automi DFA che riconoscono i linguaggi L_1, L_2, e presi i rispettivi grafi degli stati, si prende l'unione:

[34] Non sono operazioni indipendenti, visto che si possono esprimere tramite le leggi di De Morgan:

$$(x \cdot y)' = x' + y'$$

XXX

Figura 35 XXX

N.B.: si passa da due DFA ad un NFA (visto che i due grafi sono non connessi e quindi ci sono due stati iniziali); comunque, si può sempre passare ad un DFA tramite subset construcion.

In maniera più formale, dire che L_1 ed L_2 sono riconosciuti significa che esistono i due automi deterministici:

$$a_{1_D} = \left(\Sigma,\ Q_1,\ q_{0_1},\ F_1,\ \delta_1\right) \qquad \text{con } L\left(a_{1_D}\right) = L_1$$
$$a_{2_D} = \left(\Sigma,\ Q_2,\ q_{0_2},\ F_2,\ \delta_2\right) \qquad \text{con } L\left(a_{2_D}\right) = L_2$$

mentre l'automa NFA che rappresenta l'unione sarà:

$$a_{1\cup 2_N} = \left(\Sigma,\ Q_1 \cup Q_2,\ \left\{q_{0_1},\ q_{0_2}\right\},\ F_1 \cup F_2,\ \delta\right)$$

con:

$$\delta\left(q,\ a\right) = \begin{cases} \delta_1\left(q,\ a\right) & \text{se } q \in Q_1 \\ \delta_2\left(q,\ a\right) & \text{se } q \in Q_2 \end{cases}$$

Intersezione $\cap$

Presi due linguaggi:

$$L_1,\ L_2 \in L\left(\text{FSA}\right) \Rightarrow L_1 \cap L_2 \in L\left(\text{FSA}\right)$$

cioè sono chiusi rispetto all'intersezione.

Dim.:

Presi i due automi DFA:

$$a_{1_D} = \left(\Sigma,\ Q_1,\ q_{0_1},\ F_1,\ \delta_1\right) \qquad \text{con } L\left(a_{1_D}\right) = L_1$$
$$a_{2_D} = \left(\Sigma,\ Q_2,\ q_{0_2},\ F_2,\ \delta_2\right) \qquad \text{con } L\left(a_{2_D}\right) = L_2$$

l'automa che rappresenta l'intersezione sarà:

$$a_{1\cap 2_D} = \left(\Sigma,\ Q_1 \times Q_2,\ \left\{q_{0_1},\ q_{0_2}\right\},\ F_1 \times F_2,\ \delta\right)^{35}$$

con:

$$\delta\left(\left\{q_{0_1},\ q_{0_2}\right\},\ a\right) = \left(\delta_1\left(q,\ a\right),\ \delta_2\left(q,\ a\right)\right)$$

Attraverso un ragionamento induttivo, si può estendere il concetto ad un *percorso* (parola), in modo da passare quindi da δ a δ^*:

$$\delta^*\left(\left(q_1,\ q_2\right),\ w\right) = \left(\delta_1^*\left(q_1,\ w\right),\ \delta_2^*\left(q_2,\ w\right)\right)$$

Il linguaggio riconosciuto dall'automa diventa quindi:

[35] Con la notazione "×" s'intende l'insieme delle coppie; questo comporta che se un elemento della coppia non è definito allora l'intera coppia non è definita.

$$L(a) = \left\{ w \in \Sigma^* \ : \ \delta^* \left(\left(q_{0_1}, \ q_{0_2} \right), \ w \right) \in F_1 \times F_2 \right\} =$$
$$= \left\{ w \in \Sigma^* \ : \ \delta_1^* \left(q_{0_1}, \ w \right) \in F_1 \wedge \delta_2^* \left(q_{0_2}, \ w \right) \in F_2 \right\} =$$
$$= \left\{ w \in \Sigma^* \ : \ w \in L_1 \wedge w \in L_2 \right\}$$

che è esattamente l'intersezione.

Esempio 11

Preso il linguaggio definito dall'automa a_1:

$$L_1 = \left\{ (ab)^n \ : \ n \geq 0 \right\}$$

e ci si accorge che l'*automa* non è *completo*[36]; si aggiunge così lo stato z_1 per completare l'automa, e si ha che:

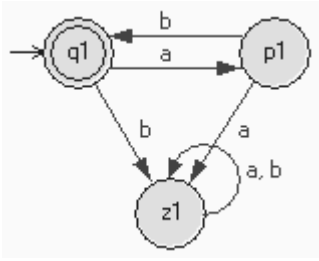


Figura 36 Grafo dell'automa a_1 (completo).

N.B.: z è accessibile ma non co-accessibile.

Lo stesso si ha con il secondo automa a_2:

$$L_2 = \left\{ a^n b \ : \ n \geq 0 \right\}$$

ed anche in questo caso si completa aggiungendo lo stato z_2:

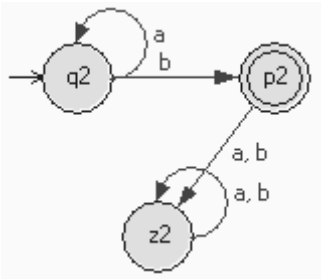


Figura 37 Automa a_2 (completo).

In maniera intuitiva, ci si accorge che l'automa risultante dall'intersezione dei due precedenti sarà:

$$L_1 \cap L_2 = \left\{ ab \right\}$$

Ad es., per gli stati q_1 e q_2 si ha che, avendo "a", si giunge rispettivamente a p_1 e p_2:

[36] Per *automa completo* s'intende quello in cui tutte le transizioni sono definite, ossia se la sua funzione di transizione δ fornisce risposta per tutti i casi possibili. Nel caso in cui non sia completo, si aggiunge un nuovo stato, accessibile ma non co-accessibile, che verrà utilizzato per gli input non definiti.

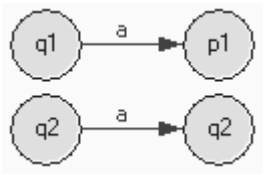


Figura 38 Esempio di input identico applicato agli automi a_1 e a_2.

e quindi si unisce il tutto nel nuovo automa:

Figura 39 Grafo degli stati dell'automa non completo risultante dall'intersezione dei primi due.

Ripetendo questo metodo, si ha che il grafo degli stati dell'automa completo è il seguente:

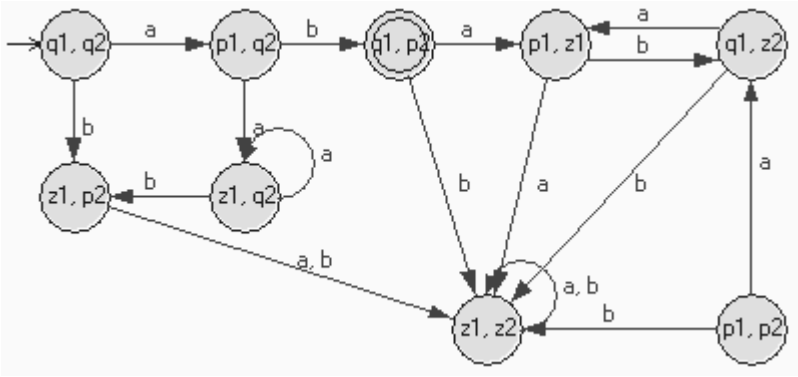


Figura 40 Grafo degli stati dell'automa completo risultante dall'intersezione dei primi due.

Complemento [c]

Preso il linguaggio:

$$L \in L(\text{FSA}) \Rightarrow L^c = \Sigma^* \setminus L \in L(\text{FSA})$$

cioè è chiuso rispetto al complemento.

Dim.:

Preso l'automa:

$$a_D = (\Sigma,\ Q,\ q_0,\ F,\ \delta) \qquad \text{con } L(a) = L$$

si ha che il suo complemento è rappresentato da:

$$a^c = (\Sigma,\ Q,\ q_0,\ F^c,\ \delta) \qquad \text{con } F^c = Q \setminus F$$

N.B.: l'automa del quale si deve fare il complemento deve essere un DFA e deve essere completo; se non lo è, si devono aggiungere gli stati mancanti per completarlo.

Esempio 12

Riprendendo l'Esempio 11, si ha che:

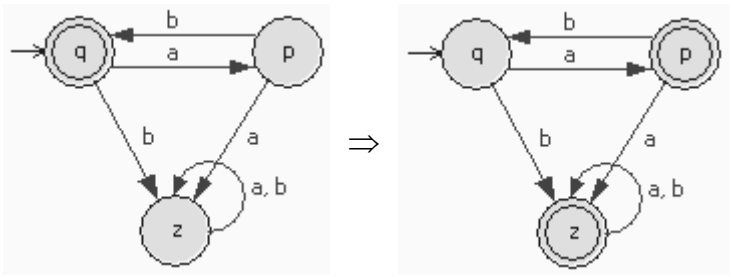


Figura 41 L'automa dell'Esempio 11 ed il suo complemento.

ossia, tutti gli stati che nel primo non erano di accettazione, nel secondo lo diventano e viceversa. Si ha quindi che, ad es., la stringa "*aaa*" (che nel primo automa non era accettata) nell'automa rappresentate il complemento è accettata.

N.B.: la classe di linguaggi di "Tipo 2" non è chiusa rispetto al complemento, visto che non si ha l'equivalenza:

$$NPDA \not\equiv DPDA$$

Linguaggi complessi

Può servire un automa che riconosca linguaggi complessi, ossia linguaggi risultanti dalla composizione di più linguaggi.

Esempio 13

Realizzare il grafo degli stati dell'automa che riconosca il linguaggio formato dalle parole che seguono queste regole:

1)le lettere "*a*" sono multiple di tre;
2)le lettere "*b*" sono pari;
3)le lettere "*b*" sono almeno cinque;

Si creano i singoli automi:

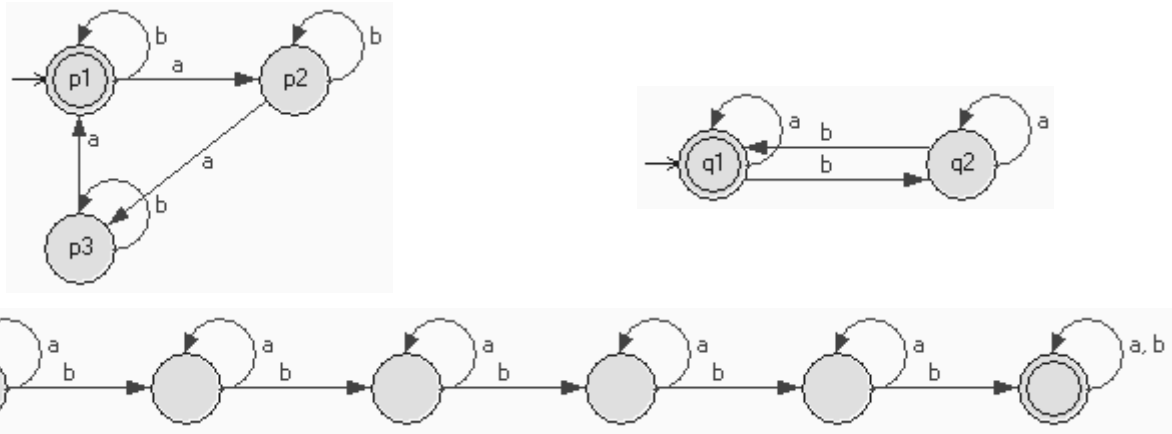


Figura 42 I grafi degli stati dei tre automi.

e poi si fa l'intersezione.

16/11/2001

Concatenazione (o prodotto) di linguaggi ·

Presi i linguaggi:

$$L_1,\ L_2 \subseteq \Sigma^* \Rightarrow L_1 \cdot L_2 = \{uv\ :\ u \in L_1,\ v \in L_2\} \qquad \text{con } \overbrace{LL\ldots}^{n} = L^n$$

cioè sono chiusi rispetto alla concatenazione.

N.B.: la concatenazione esprime il concetto di sequenzialità.

Dim.: (vedi pag. successiva)

Automa in forma speciale

E' un NFA con:

1)unico stato iniziale, nel quale non entra nessun arco;
2)unico stato d'accettazione, dal quale non esce nessun arco;
3)nel caso in cui ci sia la parola vuota, lo stato iniziale può essere d'accettazione;

XXX

Figura 43 XXX

Teorema 6

dato un generico FSA $\Rightarrow \exists$ un FSA in forma speciale *equivalente*[37]:

$$L(\text{FSA}) = L(\text{FSA}_{FS})$$

Dim.:

Dato un generico DFA:

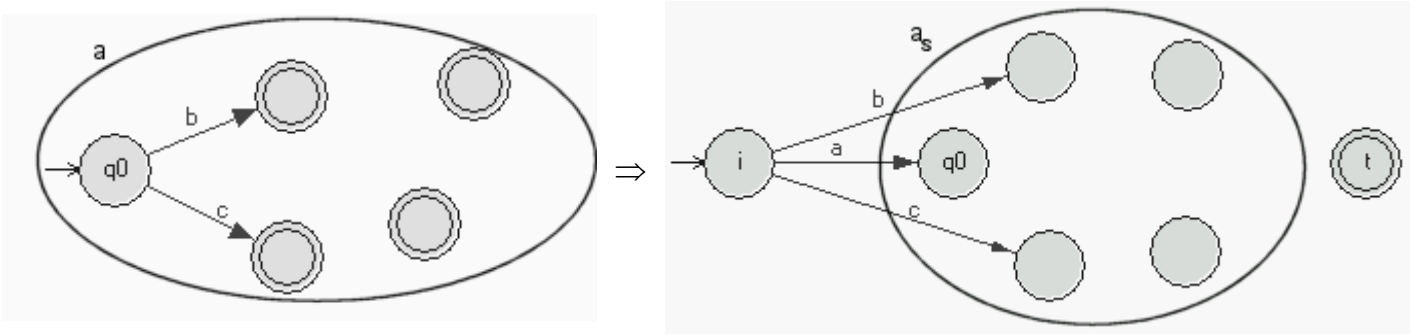


Figura 44 Un generico DFA ed il suo automa in forma speciale.

$$a_{\text{FSA}} = (\Sigma,\ Q,\ q_0,\ F,\ \delta)$$
$$a_{\text{FSA}_{FS}} = (\Sigma,\ Q \cup \{i,\ t\},\ i,\ \{t\},\ \delta_s) \qquad \text{con } i,\ t \notin Q$$

con:

$$\delta_{\text{FSA}_{FS}}(q,\ a) = \begin{cases} \delta(q,\ a) & \text{se } q \in Q \\ \delta(i,\ a) = p & \text{se } \delta(q_0,\ a) = p \\ \delta(q,\ a) = t & \text{se } \delta(q,\ a) \in F \\ \delta(i,\ a) = t & \text{se } \delta(q_0,\ a) \in F \end{cases}$$

cioè:

[37] Due *automi* si dicono *equivalenti* se riconoscono lo stesso linguaggio:

$$L(a_1) = L(a_2)$$

1)agli stati del primo automa si aggiungono XXX
2)XXX
3)si fa un discorso simmetrico, per ogni arco che entra in uno stato d'accettazione, si vede da dove viene e si crea un arco con la stessa etichetta XXX
4)se lo stato iniziale è anche d'accettazione, il nuovo stato iniziale sarà d'accettazione;
5)se si ha un arco dallo stato iniziale a quello d'accettazione, si crea un arco dal nuovo stato iniziale al nuovo stato d'accettazione;

XXX

Figura 45 XXX

N.B.: il nuovo automa non è deterministico.

Adesso si deve controllare se entrambi riconoscono le stesse stringhe; per fare questo si controlla la loro lunghezza:

- ✓lunghezza=0 $\Rightarrow$ sono riconosciute da entrambi;
- ✓lunghezza=1 $\Rightarrow$ se una parola è accettata dall'automa di partenza sarà accettata dal nuovo automa e viceversa. La parole di lunghezza unitaria corrispondono agli archi diretti da uno stato d'accettazione ad uno finale; gli unici che si aggiungono sono quelli dati dall'utilizzo della regola 5);
- ✓lunghezza>1 $\Rightarrow$ se una parola è accettata dall'automa di partenza, nel nuovo automa sarà:

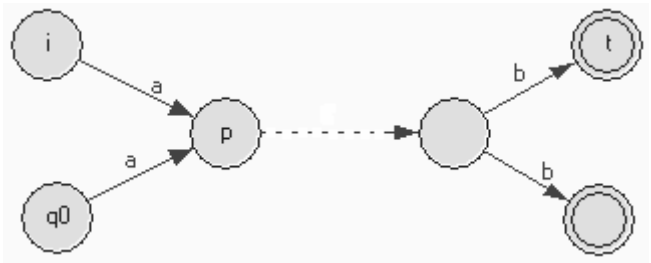


Figura 46 Grafo del nuovo automa.

Dim.:

Con questa costruzione, dimostriamo che:

$$L_1,\ L_2 \in L(\mathrm{FSA}) \Rightarrow L_1 \cdot L_2 \in L(\mathrm{FSA})$$

Procediamo costruttivamente prendendo due automi in forma speciale:

$L(a_1) = L_1$ $\qquad$ $L(a_2) = L_2$

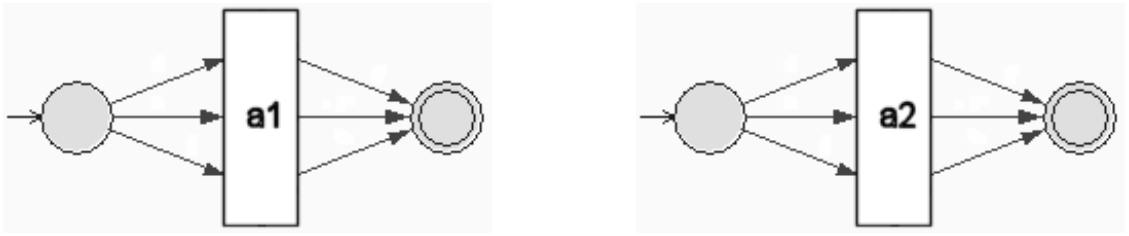


Figura 47 Due automi in forma speciale.

e, fondando i due stati, l'automa risultante sarà:

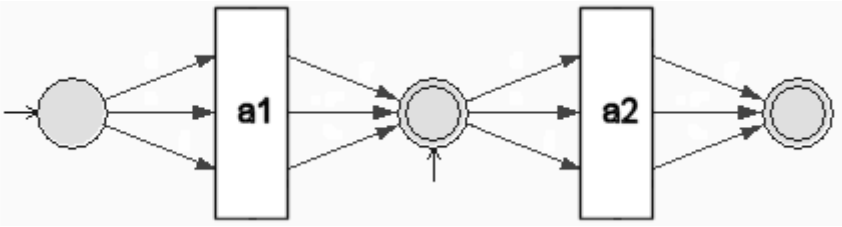


Figura 48 L'automa risultante dalla fusione dei due stati.

N.B.: se in a_1 lo stato iniziale è pure d'accettazione[38], in quello risultante quello "intermedio" sarà iniziale e di accettazione (per riconoscere la parola vuota ε).

Star[39] (di Kleene) *

Preso il linguaggio:

$$L \in L(\text{FSA}) \Rightarrow L^* \in L(\text{FSA})$$

cioè è chiuso rispetto alla star.

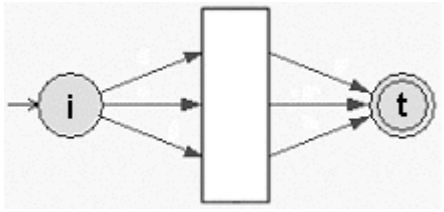
N.B.: quest'operazione non si può ottenere da un numero finito di combinazioni delle altre, dato che esprime il concetto di iterazione:

$$L^* = \{v_1 v_2 ... v_k \ : \ k \geq 0, \ v_i \in L\} \qquad \text{con } i = 1, \ ... \ , k$$
$$= \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup ...$$

Dim.:

Preso un generico automa:

$$L = L(a)$$



si costruisce l'automa L^* fondendo i due stati, ossia ponendo:

$$i = t$$

Linguaggi elementari

I linguaggi più semplici sono quelli formati da una sola parola (e per giunta) molto piccola:

$$\phi, \ \{\varepsilon\}, \ \{a\} \quad \text{con } a \in \Sigma$$

Dato un alfabeto, formato da *k* simboli, si avranno *k+2* linguaggi.

Linguaggi regolari generali GREG

Questi possono essere composti tramite le operazioni regolari. Questa famiglia è la più piccola famiglia che contiene i linguaggi elementari ed è chiusa rispetto alle operazioni regolari, cioè si può ottenere applicando un numero finito di volte operazioni regolari ai linguaggi

[38] L'automa riconosce quindi la parola vuota.
[39] Caratterizza i linguaggi riconosciuti da un FSA.

elementari:

$$L\left(\cup,\ \cap,\ \cdot,\ ^{c},\ *\right) = GREG$$

Espressioni regolari

Sono espressioni che coinvolgono simboli dell'alfabeto, operazioni regolari, parentesi ed, eventualmente, l'insieme vuoto:

$$L\left(\cup,\ \cdot,\ *\right) = REG$$

quale ad es.:

$$\left(a^{*}b\right)^{*} a^{*}$$

che rappresenta l'espressione finita di un linguaggio infinito.

Esempio 14

Ad es., quelli che hanno la seguente struttura:

$$\left(\{a\} \cup \{b\}\{c\}\right)^{*} \{b\}\{a\}^{*} = \{a,\ bc\}^{*}\, b\{a\}^{*}$$

N.B.: solitamente si eliminano le parentesi graffe:

$$\left(a,\ (bc)\right)^{*} b(a)^{*}$$

Oppure, presi:

$$L\left(\cup,\ \cap,\ ^{c}\right) = BOOLEANI \qquad \text{con } \Sigma = \{a,\ b\}$$

si ha l'insieme finito:

$$\phi,\ \{\varepsilon\},\ \{a\},\ \{b\},\ \{a,\ b\},\ \Sigma^{*} \setminus \{a\},\ \Sigma^{*} \setminus \{b\},\ \Sigma^{*} \setminus \{a,\ b\},\ \Sigma^{*},\ \Sigma^{*} \setminus \{\varepsilon\}$$

Linguaggi finiti FIN

Sono del tipo:

$$L\left(\cup,\ \cdot\right) = L\left(\cup,\ \cap,\ \cdot\right) = FIN$$

si hanno tutti i linguaggi finiti, perché applicando la concatenazione tra linguaggi finiti elementari si anno linguaggi finiti:

$$\{a\} \cdot \{b\} \Rightarrow \{aabbabaabab\}$$

Linguaggi star-free SF

Sono del tipo:

$$L\left(\cup,\ \cdot,\ ^{c}\right) = L\left(\cup,\ \cap,\ \cdot,\ ^{c}\right) = SF$$

Linguaggi regolari REG

Sono del tipo:

$$L(\cup,\ \cdot,\ ^*) = REG$$

Gerarchia dei linguaggi 1

Utilizzando un reticolo si ha:

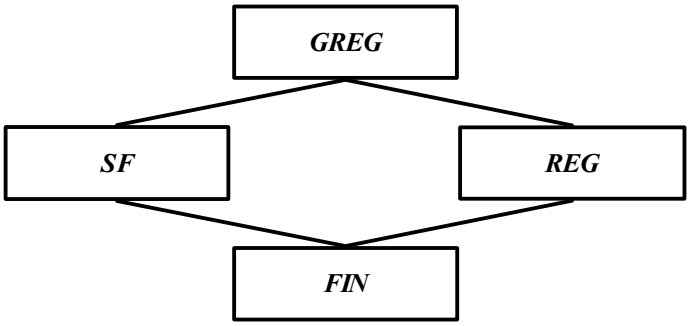


Figura 49 Rappresentazione del rapporto tra i vari linguaggi.

cioè valgono le seguenti inclusioni:

$$FIN \subset SF \subset GREG$$
$$FIN \subset REG \subseteq GREG$$

visto che *FIN* è un linguaggio finito mentre *SF*, *REG* e *GREG* sono infiniti.

19/11/2001

Esempio 15 (Eliminazione della *)[40]

Da:

$$(ab)^*$$

si usa il ^{c} al posto della * e si converte in un linguaggio finito:

$$(ab)^* = \{\varepsilon,\ ab,\ abab,\ ababab,\ ...\}$$

cioè tutte le parole che:

- iniziano per *a*;
- terminano per *b*;
- non contengono *aa*;
- non contengono *bb*;

Si può quindi fare l'intersezione tra i vari linguaggi:

$$\varepsilon \cup \left(\underbrace{a(a \cup b)^*}_{\text{inizia per } a} \cap \underbrace{(a \cup b)^* b}_{\text{finisce per } b} \cap \underbrace{\left((a \cup b)^* aa(a \cup b)^*\right)^c}_{\text{non contiene } aa} \cap \underbrace{\left((a \cup b)^* bb(a \cup b)^*\right)^c}_{\text{non contiene } bb} \right) =$$

e dato che:

$$(a \cup b)^* = \varnothing^c$$

si ha:

[40] In applicazioni reali potrebbe essere comodo non usare la *.

$$= \varepsilon \cup \left(a\varnothing^c \cap \varnothing^c b \cap \left(\varnothing^c aa\varnothing^c \right)^c \cap \left(\varnothing^c bb\varnothing^c \right)^c \right)$$

questa è un'espressione star-free che rappresenta:

$$(ab)^*$$

Quindi:

$$REG \supseteq SF$$

N.B.: non si può fare il viceversa, altrimenti si avrebbe:

$$SF = REG$$

Problema 1

Date due stringhe, ad es.:

$$\left(a^* b\right)^* a^* \qquad\qquad (a \cup b)^*$$

sono equivalenti? Più in generale, dati due automi, riconoscono lo stesso linguaggio?

Teorema 7

$$GREG \subseteq L(\text{FSA})$$

Dim.:

Si è dimostrato che i linguaggi elementari sono riconosciuti da un FSA, che è chiuso rispetto alle operazioni.

Teorema di Kleene[41]

$$L(\text{FSA}) = REG = GREG$$

Dim.:

Si deve dimostrare, in modo costruttivo, che:

$$L(\text{FSA}) \subseteq REG$$

Si ha un linguaggio:

$$L \in L(\text{FSA}) \Rightarrow \exists a : L = L(a)$$

l'automa costituito da:

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

e gli stati:

$$Q = \{q_0,\ q_1,\ q_2,\ \ldots,\ q_n\}$$

è possibile risalire all'espressione regolare E da esso riconosciuta, e da lì giungere all'espressione regolare generale E' anch'essa riconosciuta (e viceversa):

[41] Kleene era un logico che nel 1954 scrisse un articolato "*Rappresentazione degli eventi della rete nervosa*".

$$a \Rightarrow E \Rightarrow E' \quad \text{et} \quad E' \Rightarrow E \Rightarrow a$$

Si prendono i tre indici:

$$0 \le i,\ j,\ k\ \le n$$

e l'insieme di stringhe:

$$L_{i,j}^{(k)}$$

che sono tutte le parole che sono etichette dei percorsi dallo stato q_i allo stato q_j e che hanno come stati intermedi quelli con indice minore di k; ad es.:

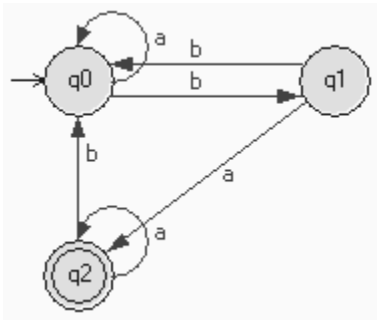


Figura 50 Esempio di percorso.

$$\text{con} \begin{cases} i = j = 2 \\ k = 1 \end{cases} \Rightarrow \text{ si ha il singolo arco } a \text{ di } q_2$$

$$\text{con} \begin{cases} i = j = 2 \\ k = 2 \end{cases} \Rightarrow \text{ si ha XXX}$$

Si fa la dimostrazione per *induzione* su k, $\forall\ i, j$ e si prova che tali linguaggi sono rappresentati da linguaggi regolari:

- $k = 0 \Rightarrow L_{i,j}^{(0)} \underbrace{a_1 \cup a_2 \cup ... \cup a_n}_{\text{espressione regolare}} \Rightarrow$ linguaggio regolare;
- $k \neq 0 \Rightarrow$ supponiamo che, per ipotesi, sia vero per $k-1 \Rightarrow L_{i,j}^{(k-1)}$ è regolare, allora:

$$L_{i,j}^{(k)} = L_{i,j}^{(k-1)} \cup L_{i,k-1}^{(k-1)} \left(L_{k-1,k-1}^{(k-1)} \right)^* L_{k-1,j}^{(k-1)}$$

cioè si utilizzano:

$$\cup,\ \cdot,\ *$$

e se sono espressioni regolari, anche $L_{i,j}^{(k)}$ è regolare $\forall\ i,\ j,\ k$.

Il linguaggio riconosciuto dall'automa è quello composto da tutte le etichette da q_0 a q_t che passano da $n+1$.

$$L(a_D) = \bigcup_{t:q_t \in F} L_{0,t}^{(n+1)}$$

Gerarchia dei linguaggi 2

Rivisitando il reticolo si ha:

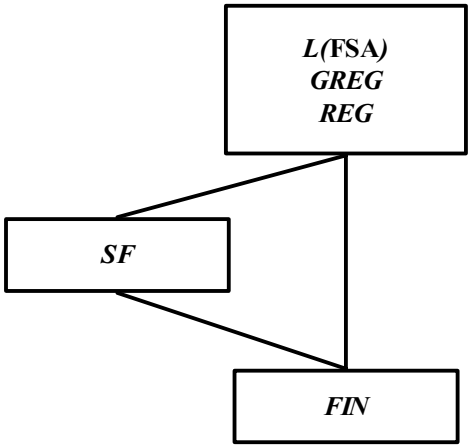


Figura 51 Nuova rappresentazione del rapporto tra i vari linguaggi.

26/11/2001

Monoide finitamente presentato

Un monoide[42] finitamente presentato è:

$$M = (G,\ R) \quad \text{con} \begin{cases} G \text{ insieme dei generatori} \\ R \text{ insieme delle relazioni} \end{cases}$$

1)finitamente generato;
2)l'insieme delle relazioni è deducibile da un insieme finito di identità;

Il caso estremale è $R = \varnothing \Rightarrow G = \Sigma$, cioè rappresenta un *monoide libero* da ogni relazione:

$$M = \Sigma$$

Approccio algebrico

Si sfrutta il fatto che Σ^* ha una struttura algebrica, essendo un *monoide libero*. In campo informatico viene usato per:

- prodotto diretto di due monoidi liberi: $\Sigma_1^* \times \Sigma_2^*$, ossia le coppie di parole:

$$(u,\ v)\cdot(u',\ v') = (uu',\ vv')$$

- monoide parzialmente commutativo libero: è una struttura nella quale non vale sempre la proprietà commutativa, ossia in una stringa alcune lettere possono commutare, mentre altre no. Ad es.:

$$abb\underline{ac}b\underline{a}cb = abb\underline{ca}b\underline{a}cb$$

L'approccio algebrico sfrutta il fatto che Σ^* è un *semigruppo con identità* (ossia un monoide libero).

Relazione ρ di equivalenza in un insieme I[43]

[42] Si chiama *monoide* una struttura algebrica che gode della proprietà associativa e di esistenza dell'elemento neutro.
[43] Definizioni puramente insiemistiche.

E' collegata al concetto di partizione dell'insieme (se si ha l'una si ha l'altro e viceversa).

Se si hanno due relazioni di *equivalenza* ρ_1 e ρ_2, si dice che ρ_1 è *meno fine* di ρ_2, e si indica con $\rho_1 \leq \rho_2$, se:

$$x\rho_2 y \Rightarrow x\rho_1 y \qquad \forall x,\ y \in I$$

e si può anche dire che le classi di ρ_1 sono unione di classi di ρ_2.

$X \subseteq I$, e si dice che ρ è *compatibile* con X se:

$$\underbrace{x\rho y}_{x\sim y} \Rightarrow \left(x \in X \Leftrightarrow y \in X\right) \qquad \forall x,\ y \in I$$

oppure

X è unione di classi di ρ

Problema 2

Dato $X \subseteq I$, qual è l'equivalenza meno fine compatibile con X?

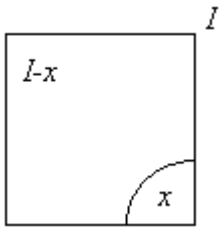


Figura 52 Equivalenza corrispondente alla partizione $\{x,\ I \setminus x\}$.

Definendo un'operazione associativa, si ha che I è un monoide, e ρ è equivalente su I.

Relazione di equivalenza invariante a destra/sinistra, congruenza

Si danno le due definizioni:

1) ρ è *invariante a destra* se $\left(x\rho y \Rightarrow xz\rho yz\right) \quad \forall x,\ y,\ z \in M$;

2) ρ è *invariante a sinistra* se $\left(x\rho y \Rightarrow zx\rho zy\right) \quad \forall x,\ y,\ z \in M$;

N.B.: se c'è uno non è detto che ci sia l'altro.

3) γ è una *congruenza* se è invariante sia a destra che a sinistra[44], ossia:

$$\left.\begin{matrix} x\rho y \\ z\rho t \end{matrix}\right\} \Rightarrow xz\rho yt \qquad \forall x,\ y,\ z,\ t \in M$$

Problema 3

Dato $X \subseteq M$, qual è l'equivalenza invariante a destra (ovvero, qual è la congruenza) meno fine compatibile con X?

[44] E' importante perché si può fare il quoziente.

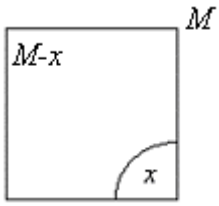


Figura 53 Equivalenza corrispondente alla partizione $\{x,\ \mathrm{M} \setminus x\}$.

Equivalenza ρ_x associata a X

Dato un insieme $X \subseteq M$, definisco l'equivalenza ρ_x associata ad X; due elementi sono equivalenti se:

$$x \rho_x y \Rightarrow \left(xz \in X \Leftrightarrow yz \in X\right) \quad \text{con} \begin{cases} \forall z \in M \\ x,\ y \in M \end{cases}$$

cioè o entrambi appartengono ad X oppure no.

Teorema 8 (invariante a destra)

ρ_x è invariante a destra

Dim.:

Si deve dimostrare che:

$$xt \rho_x yt \quad \forall t \in M$$

cioè significa dire che:

$$\left((xt)z' \in X \Leftrightarrow (yt)z' \in X\right) \quad \forall z' \in M$$

Utilizzando la proprietà associativa si ha:

$$\left(x(tz') \in X \Leftrightarrow y(tz') \in X\right) \quad \forall z' \in M$$

che è un caso particolare della definizione.

Teorema 9 (compatibile con X)

ρ_x è compatibile con X

Dim.:

Partendo dalla definizione di ρ_x:

$$x \rho_x y \Rightarrow \left(xz \in X \Leftrightarrow yz \in X\right) \quad \text{con} \begin{cases} \forall z \in M \\ x,\ y \in M \end{cases}$$

questo implica che, prendendo z pari all'unità, si ha che:

$$\left(x \in X \Leftrightarrow y \in X\right)$$

Teorema 10 (proposizione)

ρ_x è l'equivalenza invariante a destra meno fine compatibile con X

Dim.:[45]

Sia ρ un'altra arbitraria equivalenza invariante a destra e compatibile con X:

$$x\rho y \Rightarrow xz\rho yz \Rightarrow \quad \text{con} \begin{cases} x,\ y \in M \\ \rho \text{ è invariante a destra} \\ \forall z \in M \end{cases}$$

usando il fatto che è compatibile con X:

$$\Rightarrow \left(xz \in X \Leftrightarrow yz \in X\right) \quad \forall z \in M$$

ma questa è la definizione di ρ_x. In più ρ_x è meno fine di una qualunque ρ scelta inizialmente perché:

$$x\rho y \Rightarrow x\rho_x y \Rightarrow \quad \text{con} \begin{cases} \rho \le \rho_x \\ \forall x,\ y \in M \end{cases}$$

Teorema 11 (congruenza γ_x associata ad X)

$$x\gamma_x y \Rightarrow \left(z_1 x z_2 \in X \Leftrightarrow z_1 y z_2 \in X\right) \quad \text{con} \begin{cases} x,\ y \in M \\ \forall z_1,\ z_2 \in M \end{cases}$$

Dim.:

La dimostrazione si articola nei punti:

1) γ_x è una congruenza;
2) γ_x è compatibile con X;
3) γ_x è la meno fine;

Teorema 12 (congruenza γ meno fine compatibile con X)

γ_x è la congruenza meno fine compatibile con X

Dim.:

Sia γ un'arbitraria congruenza:

$$x\gamma y \Rightarrow \left(\left(z_1 x z_2\right)\gamma\left(z_1 y z_2\right)\right) \Rightarrow \quad \text{con} \begin{cases} x,\ y \in M \\ \forall z_1,\ z_2 \in M \end{cases}$$

si ha poi che è compatibile con X:

$$\Rightarrow \left(z_1 x z_2 \in X \Leftrightarrow z_1 y z_2 \in X\right) \quad \text{con} \begin{cases} x,\ y \in M \\ \forall z_1,\ z_2 \in M \end{cases}$$

cioè:

$$x\gamma_x y$$

Teorema 13

Preso un monoide M infinito:

[45] Se ne prende un'altra e si vede che la prima è meno fine di tutte le altre.

$$X \subseteq M,\ X \in REG(M) \Rightarrow \begin{cases} i_\gamma(\gamma_x) < \infty \\ i_\rho(\rho_x) < \infty \end{cases} \Rightarrow i_\gamma(\gamma_x) < \infty \Leftrightarrow i_\rho(\rho_x) < \infty$$ [46]

Teorema di Myhill-Nerode

Sia $L \subseteq \Sigma^*$; le seguenti affermazioni sono equivalenti:

1) $L \in L(\text{FSA})$;

2) γ_L ha indice finito $\Leftrightarrow$ $i(\gamma_L) < \infty$;

3) ρ_L ha indice finito $\Leftrightarrow$ $i(\rho_L) < \infty$;

N.B.: da un punto di vista linguistico si ha:

$$(z_1 x z_2 \in L \Leftrightarrow z_1 y z_2 \in L) \quad \forall z_1,\ z_2 \in \Sigma^*$$

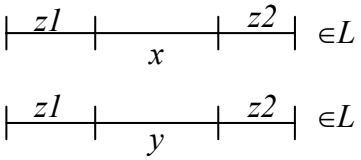


Figura 54 Rappresentazione linguistica del Teorema di Myhill-Nerode.

29/11/2001

Dim.:

1⇒2) Dato un automa DFA:

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

per ogni automa si associa l'equivalenza ρ_a e si dimostra che questa è invariante a destra.

Prese due parole, queste sono due *parole equivalenti* se:

$$u \rho_a v \Rightarrow \delta^*(q_0,\ u) = \delta^*(q_0,\ v) \quad \text{con } u,\ v \in L(a)$$

ossia *u* e *v* portano allo stesso stato:

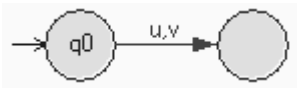


Figura 55 Due parole equivalenti.

Si ha che:

$$i(\rho_a) = Card(Q) \quad \text{con } i \text{ finito}$$

Facciamo vedere che l'equivalenza ρ_a è invariante a destra; si deve quindi dimostrare che:

$$u \rho_a v \Rightarrow (uw)\, \rho_a\, (vw)$$

$$\delta^*(q_0,\ uw) = \delta^*(\delta^*(q_0,\ u),\ w) = \delta^*(\delta^*(q_0,\ v),\ w) \Rightarrow \delta^*(q_0,\ vw)$$

[46] *Indice* di ρ (indicato con *i(ρ)*) è il numero della classi di ρ, cioè il numero degli stati.

Dimostriamo che ρ_a è compatibile con L:

$$\delta^*(q_0,\ u) \in F$$
$$\delta^*(q_0,\ v) \in F$$

si ha quindi che:

1) $i(\rho_a) = Card(Q)$;

2) ρ_a è invariante a destra in $\Sigma^* \Leftrightarrow\ u\rho_a v \Rightarrow (u \in L \Leftrightarrow v \in L)$;

3) ρ_a è compatibile con L;

2⇒1) Data l'equivalenza invariante a destra su Σ^*, di indice finito e compatibile con L, e preso l'automa:

$$a_\rho = (\Sigma,\ Q_\rho,\ q_{0_\rho},\ F_\rho,\ \delta_\rho)$$

con:

$$Q_\rho = \text{ insieme delle classi di equivalenza } \subseteq \Sigma^*$$
$$q_{0_\rho} = [\varepsilon]$$
$$F_\rho = \{[u] : u \in L\}\ ^{47}$$
$$\delta_\rho([u],\ a) = [u\ a]^{48}$$

che riconosce L, visto che:

$$L(a_\rho) = \{w \in \Sigma^* : \delta_\rho^*(q_{0_\rho},\ w) \in F_\rho\} = \{w \in \Sigma^* : \delta_\rho^*([\varepsilon],\ w) \in F_\rho\} =$$
$$= \{w \in \Sigma^* : [w] \in F_\rho\} \underset{\text{dalla def.}}{=} L$$

Esempio 16 (1⇒2)

Dato l'automa:

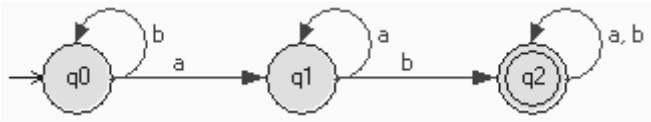


Figura 56 Automa dell'Esempio 16.

Le classi sono:

1°) in q_0 ci sono: b^*;
2°) in q_1 ci sono: b^*aa^*;
3°) in q_2 ci sono tutte le parole che contengono ab;

N.B.: queste classi sono disgiunte, sono la partizione di Σ^*.

L'indice dell'equivalenza è 3, L è la 3° classe ed è invariante a destra[49].

Esempio 17 (2⇒1)

[47] Si prendono le classi di cui L è unione.
[48] Se si cambia u con v, si ha la stessa classe visto che $u \sim v$.
[49] Preso b^n e moltiplicato a destra, si rimane in uno delle tre classi.

Data l'equivalenza, si hanno le classi:

1°) in q_0 ci sono: b^*;
2°) in q_1 ci sono: b^*aa^*;
3°) in q_2 ci sono tutte le parole che contengono ab;

e si sa che L è la classe 3.

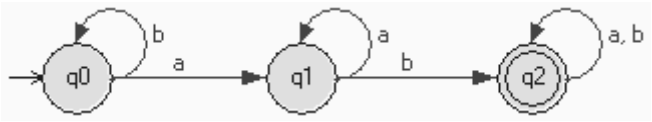


Figura 57 Automa dell'Esempio 17.

La funzione di transizione si calcola:

$$\delta_\rho(q_0,\ a) = \delta_\rho([\varepsilon],\ a) = [a] = q_1$$
$$\delta_\rho(q_0,\ b) = \delta_\rho([\varepsilon],\ b) = [b] = q_0$$
$$\delta_\rho(q_1,\ a) = \delta_\rho([a],\ a) = [a\ a] = q_1$$
$$\delta_\rho(q_1,\ b) = \delta_\rho([a],\ b) = [a\ b] = q_2$$

Si ha una relazione d'ordine nella quale si ha un unico[50] *automa minimale*, cioè tutte le altre equivalenze invarianti a destra e compatibili con L hanno indice maggiore:

$$\rho_1 \le \rho_2 \Rightarrow i(\rho_1) \le i(\rho_2)$$

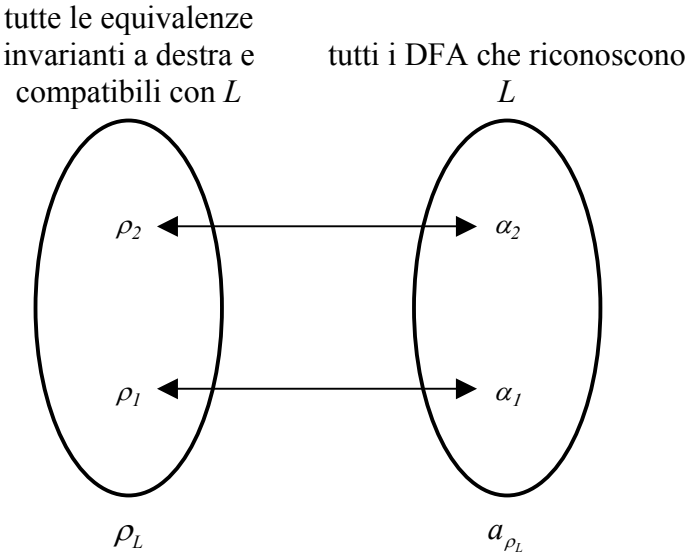


Figura 58 Corrispondenza diretta tra equivalenza ed automi.

Corollario 1

l'automa DFA minimale è unico

N.B.: utilizzando questo corollario si riesce a costruire l'automa minimale. A questo punto si è in grado di dire se due *automi* a_1, a_2 sono *equivalenti* o meno, ossia:

$$L(a_1) = L(a_2)$$

in quanto si riducono agli automi minimali e, se questi coincidono, quelli di partenza sono

[50] Visto che è quello con il minor numero di stati possibili.

equivalenti.

Teorema di Robin-Shepherdson

$$L(\text{1-DFA}) = L(\text{2-DFA})$$

30/11/2001

Dim.:

Visto che l'inclusione:

$$L(\text{1-DFA}) \subseteq L(\text{2-DFA})$$

è ovvia, si deve dimostrare solo la seconda:

$\supseteq$) Preso un 2-DFA:

$$a = (\Sigma,\ Q,\ q_0,\ F,\ \delta)$$
$$\delta : Q \times \Sigma \to Q \times \Delta \quad \text{con } \Delta = \{-1,\ 0,\ +1\}$$

dobbiamo dimostrare che si ha un automa 1-DFA che riconosce il linguaggio *L(a)*. Per far questo si dimostra che esiste un'equivalenza di indice finito, compatibile con *L* ed invariante a destra. Per il teorema di Myhill-Nerode, il linguaggio è riconosciuto dal 1-DFA:

$$L \in L(\text{1-DFA})$$

Costruiamo l'equivalenza definendo le due funzioni:

$$f_w : Q \to Q \cup \{\overline{q}\} \quad \text{con} \begin{cases} w \in \Sigma^* \\ \overline{q} \notin Q \end{cases}$$
$$g_w : Q \to Q \cup \{\overline{q}\} \quad \text{con} \begin{cases} w \in \Sigma^* \\ \overline{q} \notin Q \end{cases}$$

e, per entrambi, ci possono essere due casi:

$$f_w(q) = \begin{cases} \overline{q} & \text{non esce mai/esce a sinistra} \\ P & \text{esce a destra nello stato } P \end{cases}$$
$$g_w(q) = \begin{cases} \overline{q} & \text{non esce mai/esce a sinistra} \\ P & \text{esce a destra nello stato } P \end{cases}$$

Si definisce l'equivalenza E_a:

$$u\ \mathrm{E}_a\ v \Leftrightarrow (f_u,\ g_u) = (f_v,\ g_v) \Rightarrow \begin{cases} f_u = f_v \\ g_u = g_v \end{cases} \quad \text{con } u,\ v \in \Sigma^*$$

cioè le due parole definiscono le stesse due funzioni. Si deve dimostrare che $u\ \mathrm{E}_a\ v$ è:

- <u>di indice finito</u>:

$$Card(Q) = n$$
$$Card(Q \cup \{\overline{q}\}) = n+1$$

quindi, il numero di possibili applicazioni f_w distinte da Q in $Q \cup \{\overline{q}\}$ è:

$$(n+1)^n$$

Discorso analogo si può fare per g_w ; le possibili coppie sono, in totale:

$$(n+1)^n \cdot (n+1)^n = (n+1)^{2n}$$

e l'equivalenza ha quindi indice finito:

$$i(\mathrm{E}_a) \le (n+1)^{2n}$$

➢ compatibile con L:

$$u \; \mathrm{E}_a \; v \Leftrightarrow (u \in L \Leftrightarrow v \in L)$$

e per definizione di E_a, sappiamo che:

$$f_u = f_v$$

Se $u \in L$ significa:

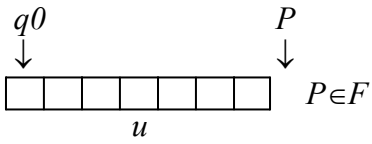


Figura 59 Uscita a destra.

cioè:

$$f_v(q_0) = f_u(q_0) \Rightarrow f_u(q_0) = p \in F \quad \text{con } u,\ v \in L$$

➢ invariante a destra:

$$u \; \mathrm{E}_a \; v \Rightarrow uw \; \mathrm{E}_a \; vw \quad \forall w \in L$$

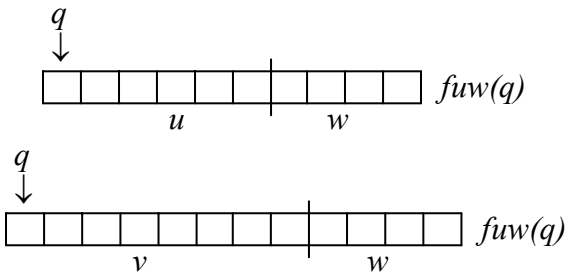
che significa dimostrare:

$$\left.\begin{aligned} f_u &= f_v \\ g_u &= g_v \end{aligned}\right\} \Rightarrow \begin{cases} f_{uw} = f_{vw} \\ g_{uw} = g_{vw} \end{cases}$$

e questo significa verificare le due implicazioni:

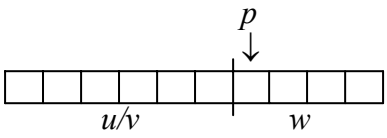
$$H \Rightarrow f_{uw} = f_{vw}$$
$$H \Rightarrow g_{uw} = g_{vw}$$

Si fa una prova per induzione sul numero n di volte che si attraversa la linea che separa u e v da w:

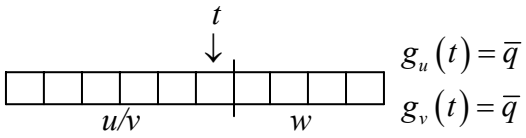


$$n=0 \Rightarrow \begin{cases} f_u(q)=\overline{q} \Rightarrow f_{uw}(q)=\overline{q} & \text{si esce a sinistra oppure si resta sempre dentro } u; \\ f_v(q)=\overline{q} \Rightarrow f_{vw}(q)=\overline{q} & \text{si esce a sinistra oppure si resta sempre dentro } v; \end{cases}$$

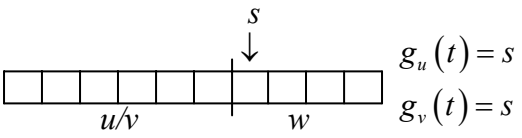
$$n=1 \Rightarrow \begin{cases} f_u(q)=p \Rightarrow f_{uw}(q)=\overline{q} \\ f_v(q)=p \Rightarrow f_{vw}(q)=\overline{q} \end{cases} \text{si attraversa una volta la linea e si arriva allo stato } p\text{; a questo punto } w \text{ è comune}^{51};$$



$$n=2 \Rightarrow \begin{cases} f_u(q)=t \Rightarrow f_{uw}(q)=\overline{q} \\ f_v(q)=t \Rightarrow f_{vw}(q)=\overline{q} \end{cases} \text{si è usciti una volta e rientrati una volta, arrivando nello stato } t;$$



$$n=3 \Rightarrow \begin{cases} f_u(q)=s \Rightarrow f_{uw}(q)=s \\ f_v(q)=s \Rightarrow f_{vw}(q)=s \end{cases} \text{si è usciti una prima volta, rientrati e riusciti una seconda volta, arrivando nello stato } s;$$



In questo caso sono nello stato s e sono della stessa stringa, quindi si comportano allo stesso modo:

$$g_{uw}(t)=g_{vw}(t)$$

Per come si sono definite le funzioni, ci si è garantito che ogni volta che si rientra nello stesso stato e si riesce si comporta allo stesso modo.

N.B.: questo teorema permette di costruire da un 2-DFA un 1-DFA. Il prezzo da pagare è in quantità di memoria, in quanto da n stati si passa (al massimo) a:

$$n \to \leq (n+1)^{2n}$$

Inversione

Presa la stringa:

$$w=a_1a_2...a_n \in \Sigma^* \quad \text{con} \begin{cases} a_i \in \Sigma \\ 1 \leq i \leq n \end{cases}$$

si definisce la stringa "scritta al contrario":

[51] Ed in particolare, se si rimane in uno si rimane anche nell'altro, mentre se si esce in uno si esce anche nell'altro.

$$\tilde{w} = a_n ... a_2 a_1 \in \Sigma^* \quad \text{con} \begin{cases} a_i \in \Sigma \\ 1 \le i \le n \end{cases}$$

Passando al linguaggio:

$$L \in \Sigma^*$$
$$\tilde{L} = \{\tilde{v} : v \in L\}$$

quindi:

$$L \in L(\text{DFA}) \Rightarrow \tilde{L} \in L(\text{DFA})$$

Passando all'automa DFA:

$$a_D = (\Sigma,\ Q,\ q_0,\ F,\ \delta)$$
$$L(a) = L$$

per costruire l'automa non deterministico $\tilde{a}$ occorre invertire le direzioni degli archi e fare in modo che gli stati di accettazione del primo diventino iniziali per il nuovo (e viceversa):

$$\tilde{a}_N = (\Sigma,\ Q,\ I,\ \{q_0\},\ \tilde{\delta}) \quad \text{con} \begin{cases} I = F \\ \tilde{\delta} \subseteq Q \times \Sigma \times Q \end{cases}$$
$$L(\tilde{a}) = \tilde{L}$$
$$(p,\ a,\ q) \in \tilde{\delta} \Leftrightarrow (q,\ a,\ p) \in \delta$$

Costruzione di un automa minimale

Si parte da un automa:

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

L'idea è legata al concetto di stati indistinguibili, in quanto si fanno degli "esperimenti" con l'automa, ossia lo si considera come una scatola nera alla quale si da un input e, tramite l'output corrispondente[52], si risale allo stato interno. Ad es.:

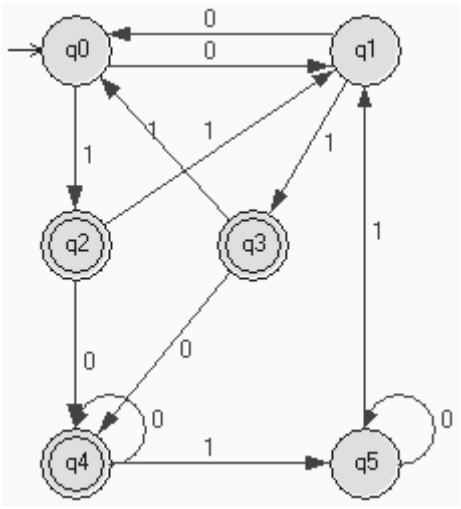


Figura 60 Automa d'esempio.

[52] Con 1=stato di accettazione e 0=stato non di accettazione.

- distinguere tra q_1 e q_2[53]:

$$1101 \Rightarrow (q_1,\ 1)$$
$$1101 \Rightarrow (q_2,\ 0)$$

- distinguere tra q_3 e q_4:

$$11 \Rightarrow (q_3,\ 1)$$
$$11 \Rightarrow (q_4,\ 0)$$

- non si distingue q_2 da q_3:

$$101001 \Rightarrow (q_2,\ 0)$$
$$101001 \Rightarrow (q_3,\ 0)$$
oppure
$$1101 \Rightarrow (q_2,\ 0)$$
$$1101 \Rightarrow (q_3,\ 0)$$

Si definiscono *stati indistinguibili* quando ad uno stesso input corrisponde lo stesso output. L'indistinguibilità è una relazione di equivalenza su Q, visto che se due stati sono indistinguibili allora appartengono alla stessa classe di equivalenza.

N.B.: se due stati sono indistinguibili, o sono entrambi d'accettazione o non lo sono.

Più formalmente, presi gli stati:

$$p\ \mathrm{I}\ q \Leftrightarrow \left(\delta^*(p,\ w) \in F \Leftrightarrow \delta^*(q,\ w) \in F\right) \quad \text{con} \begin{cases} p,\ q \in Q \\ \forall w \in \Sigma^* \\ \text{I equivalenza su } Q \end{cases}$$

03/12/2001

Riduzione

La *riduzione* non è altro che passare alle classi di equivalenza degli stati Q. In input si ha l'automa a, mentre in output si ottiene l'automa ridotto a_R:

$$a_R = (\Sigma,\ Q_R,\ [q_0],\ F_R,\ \delta_R)$$

con:

$Q_R =$ insieme delle classi di equivalenza di I
$F_R = \{[q] : q \in F\}$
$\delta_R([q],\ a) = [\delta(q,\ a)]$

Sono importanti le *proprietà* per costruire l'automa:

1)I è compatibile con F;
2) $p\ \mathrm{I}\ q \Rightarrow \delta(p,\ a)\ \mathrm{I}\ \delta(q,\ a) \quad \forall a \in \Sigma$

[53] Si dice che 1101 è un esperimento che permette di distinguere q_1 da q_2.

$$\delta^*\left(\delta\left(p,\ a\right),\ w\right)=\delta^*\left(p,\ aw\right)\in F \Leftrightarrow \delta^*\left(\delta\left(q,\ a\right),\ w\right)=\delta^*\left(q,\ aw\right)\in F$$

Prendendo l'automa precedente si ottengono la seguente partizioni:

$$1=\left\{q_0,\ q_1\right\}$$
$$2=\left\{q_2,\ q_3\right\}$$
$$3=\left\{q_4\right\}$$
$$4=\left\{q_5\right\}$$

e questi sono:

1)lo stato iniziale è la classe che conteneva lo stato iniziale;
2)lo stato d'accettazione è la classe che conteneva lo stato d'accettazione;

Le transizioni sono, prendendo un qualsiasi rappresentante delle partizioni 1, 2, 3 e 4:

$$\left.\begin{array}{c}\delta\left(q_0,\ 0\right)=q_1\\ \text{oppure}\\ \delta\left(q_1,\ 0\right)=q_0\end{array}\right\}\Rightarrow\delta\left(1,\ 0\right)=1$$
$$\delta\left(q_1,\ 1\right)=q_3\Rightarrow\delta\left(1,\ 1\right)=2$$

e, alla fine si ha che il grafo è:

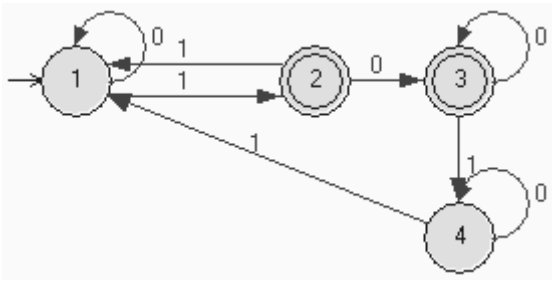


Figura 61 Automa ridotto dell'automa iniziale.

N.B.: se si riapplica la riduzione ad a_R, si ottiene nuovamente a_R, ossia nell'automa ridotto, l'indistinguibilità I coincide con l'identità:

$$p\ \mathrm{I}\ p\Rightarrow p=q$$

Teorema 14

$$L\left(a_R\right)=L\left(a\right)$$

Dim.:

Per definizione si ha che:

$$L\left(a_R\right)=\left\{w\in\Sigma^*:\delta_R^*\left(q_{0_R},\ w\right)\in F_R\right\}=\left\{w\in\Sigma^*:\delta_R^*\left(\left[q_0\right],\ w\right)\in F_R\right\}=$$
$$=\left\{w\in\Sigma^*:\left[\delta^*\left(q_0,\ w\right)\right]\in F_R\right\}=\left\{w\in\Sigma^*:\delta^*\left(q_0,\ w\right)\in F\right\}=L\left(a\right)$$

Teorema 15

a_R è l'automa minimale

Dim.:

Preso un automa a, si passa all'equivalenza e si associa ρ_a:

$$u\rho_a v \Leftrightarrow \delta^*(q_0,\ u) = \delta^*(q_0,\ v) \qquad \rho_L \le \rho_a$$
$$u\rho_L v \Leftrightarrow (uw \in L \Leftrightarrow vw \in L) \qquad \forall w \in \Sigma^*$$

in particolare, se si prende l'automa ridotto:

$$u\rho_{a_R} v \Leftrightarrow \delta_R^*(q_{0_R},\ u) = \delta_R^*(q_{0_R},\ v)$$

Si deve ora dimostrare che:

$$\rho_L = \rho_{a_R}$$

e visto che $\rho_L \le \rho_a$, si deve dimostrare soltanto che $\rho_L \ge \rho_{a_R}$, cioè:

$$u\rho_L v \Rightarrow u\rho_{a_R} v \qquad \forall u,\ v \in \Sigma^*$$

Preso:

$$\left.\begin{array}{l} u\rho_L v \Rightarrow (uw \in L \Leftrightarrow vw \in L) \Rightarrow \left(\delta_R^*(q_{0_R},\ uw) \in F_R \Leftrightarrow \delta_R^*(q_{0_R},\ vw) \in F_R\right) \Rightarrow \\ \Rightarrow \left(\delta_R^*\left(\underbrace{\delta_R^*(q_{0_R},\ u)}_{p},\ w\right) \in F_R \Leftrightarrow \delta_R^*\left(\underbrace{\delta_R^*(q_{0_R},\ v)}_{s},\ w\right) \in F_R\right) \Rightarrow \end{array}\right\} \quad \forall w \in \Sigma^*$$

questa non è altro che la relazione d'indistinguibilità tra p ed s:

$$p\ \mathrm{I}\ s \Leftrightarrow \left(\delta^*(p,\ w) \in F \Leftrightarrow \delta^*(s,\ w) \in F\right) \quad \text{con} \begin{cases} p,\ s \in Q \\ \forall w \in \Sigma^* \\ \text{I equivalenza su } Q \end{cases}$$

ma, nell'automa ridotto, l'indistinguibilità è l'identità; quindi:

$$\delta_R^*(q_{0_R},\ u) = \delta_R^*(q_{0_R},\ v) \Rightarrow u\rho_{a_R} v$$

In definitiva, da qualunque automa si parte (in un insieme di automi che rappresentano lo stesso linguaggio) si arriva sempre allo stesso automa minimale a_R.

Due stati sono indistinguibili secondo I_k[54]

$$p\ \mathrm{I}_k\ q \Leftrightarrow \left(\delta^*(p,\ w) \in F \Leftrightarrow \delta^*(q,\ w) \in F\right) \quad \text{con} \begin{cases} p,\ q \in Q \\ \forall w \in \Sigma^* \\ k \ge 0 \\ |w| \le k \end{cases}$$

Le *proprietà* sono:

1)se due stati sono indistinguibili per I_h, lo saranno anche per I_k:

$$\mathrm{I}_k \le \mathrm{I}_h \qquad \forall k \le h$$

2)considerando l'indice:

[54] Questa proprietà è algoritmicamente probabile, mentre l'altra non lo è.

$$i_k = i(\mathrm{I}_k) \Rightarrow i_k \le i_h \qquad \forall k \le h$$

3) $p\ \mathrm{I}_0\ q \Leftrightarrow (p \in F \Leftrightarrow q \in F) \qquad \text{con } i_0 = 2$

andando a vedere la successione non decrescente delle equivalenze si ha che:

$$\mathrm{I}_0 \le \mathrm{I}_1 \le \mathrm{I}_2 \le \ldots$$
$$i_0 \le i_1 \le i_2 \le \ldots \le n = Card(Q)$$

cioè è limitata superiormente.

Teorema 16

$$\exists k : \mathrm{I}_k = \mathrm{I}_{k+1} \Rightarrow \mathrm{I}_k = \mathrm{I}_{k+n} \Rightarrow \mathrm{I}_k = \mathrm{I} \qquad \forall n \ge 0$$

Dim.:

$$p\ \mathrm{I}_{k+1}\ q \Leftrightarrow \begin{cases} w = 0 \Rightarrow & p\ \mathrm{I}_0\ q \\ w \ge 1 \Rightarrow & \left(\delta^*(p,\ w) \in F \Leftrightarrow \delta^*(q,\ w) \in F\right) \end{cases} \quad \text{con} \begin{cases} \forall w \in \Sigma^* \\ 1 \le |w| \le k+1 \end{cases}$$

ponendo:

$$w = av \quad \begin{cases} \forall a \in \Sigma \\ \forall v \in \Sigma^* \\ |v| \le k \end{cases}$$

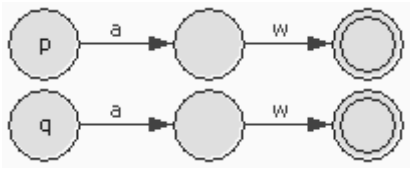
si ha che p e q sono k+1 equivalenti:

$$p\ \mathrm{I}_{k+1}\ q \Leftrightarrow \begin{cases} w = 0 \Rightarrow & p\ \mathrm{I}_0\ q \\ w \ge 1 \Rightarrow & \left(\delta^*(p,\ av) \in F \Leftrightarrow \delta^*(q,\ av) \in F\right) \end{cases} \quad \text{con} \begin{cases} \forall a \in \Sigma \\ \forall v \in \Sigma^* \\ |v| \le k \end{cases}$$

$$\Leftrightarrow \begin{cases} w = 0 \Rightarrow & p\ \mathrm{I}_0\ q \\ w \ge 1 \Rightarrow & \left(\delta^*(\delta(p,\ a),\ v) \in F \Leftrightarrow \delta^*(\delta(q,\ a),\ v) \in F\right) \end{cases} \quad \text{con} \begin{cases} \forall a \in \Sigma \\ \forall v \in \Sigma^* \\ |v| \le k \end{cases}$$

$$\Leftrightarrow \begin{cases} w = 0 \Rightarrow & p\ \mathrm{I}_0\ q \\ w \ge 1 \Rightarrow & \left(\delta(p,\ a)\ \mathrm{I}_k\ \delta(q,\ a)\right) \end{cases} \quad \text{con} \begin{cases} \forall a \in \Sigma \\ \forall v \in \Sigma^* \\ |v| \le k \end{cases}$$

Graficamente si ha che:



$|w| \le k+1$

Figura 62 Due stati indistinguibili per una parola *w*.

Supponiamo:

$$\mathrm{I}_k = \mathrm{I}_{k+1} \Rightarrow p\ \mathrm{I}_{k+2}\ q \Leftrightarrow \begin{cases} w = 0 \Rightarrow & p\ \mathrm{I}_0\ q \\ w \geq 1 \Rightarrow & \left(\delta(p,\ a)\ \mathrm{I}_{k+1}\ \delta(q,\ a)\right) \end{cases} \quad \text{con} \begin{cases} \forall a \in \Sigma \\ \forall v \in \Sigma^* \\ |v| \leq k \end{cases}$$

cioè:

$$\mathrm{I}_k = \mathrm{I}_{k+1} = \mathrm{I}_{k+2}$$

Il caso peggiore si avrà quando:

$$\mathrm{I}_0 \leq \mathrm{I}_1 \leq \mathrm{I}_2 \leq \ldots$$
$$i_0 \leq i_1 \leq i_2 \leq \ldots \leq n = Card(Q)$$

e, partendo da i_0=2, al più può crescere n−2:

$$\mathrm{I} = \mathrm{I}_{n-2}$$

quindi, se si ha un automa con n stati, basta verificare l'indistinguibilità fino a parole di lunghezza n−2. Da un punto di vista pratico, questo non è corretto; si fa allora riferimento agli stati distinguibili.

Stati distinguibili (per una stringa di lunghezza k)

$$p\ \mathrm{D}\ q \Leftrightarrow \exists v \in \Sigma^* : \begin{cases} \left(\delta(p,\ v) \in F \wedge \delta(q,\ v) \notin F\right) \\ \text{oppure} \\ \left(\delta(p,\ v) \notin F \wedge \delta(q,\ v) \in F\right) \end{cases}$$

cioè, se si trovano due stati distinguibili, si va indietro con lo stesso simbolo:

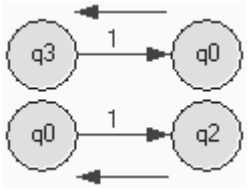


Figura 63 Due stati distinguibili.

06/12/2001

L'idea dell'algoritmo è quella di prendere le parole distinguibili dalla parola vuota:

$p\ \mathrm{D}_k\ q$ stati p e q distinguibili dalla stringa di lunghezza k;

$p\ \mathrm{D}_0\ q$ stati p e q distinguibili dalla stringa di lunghezza 0 (=ε);

nel caso in cui si abbia una coppia D_k distinguibile (p, q) ed inoltre (r, s):

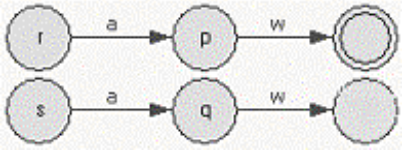


Figura 64 Degli stati distinguibili.

$$p \; \mathrm{D} \; q \Rightarrow r \; \mathrm{D} \; s$$
$$p \; \mathrm{D}_k \; q \Rightarrow r \; \mathrm{D}_{k+1} \; s$$

cioè esiste un parametro *k* per verificare la distinguibilità tra i due stati.

Esempio 18

Riferendosi all'automa visto nel paragrafo riguardante la "*Costruzione di un automa minimale*" si prende una coppia (q_x, q_y) in cui uno è di accettazione e l'altro no, e si va a ritroso vedendo se ci sono archi che entrano con un certo carattere "*c*". Si prosegue quindi passando da D_0 (0 distinguibili) a D_1 (1 distinguibili) e così via. Ad es.:

- guardando (q_0, q_2) ed andando indietro con "1" si ottiene che (q_0, q_3) sono distinguibili:

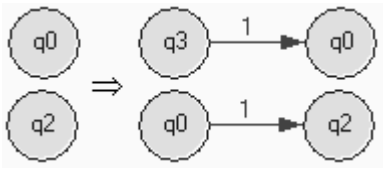


Figura 65 Gli stati (q_0, q_3) sono distinguibili.

- guardando (q_1, q_2) ed andando indietro con "1" si ottiene che (q_0, q_5) sono distinguibili:

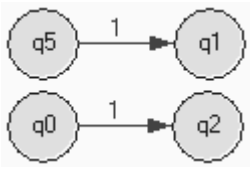


Figura 66 Gli stati (q_0, q_5) sono distinguibili.

- guardando (q_1, q_3) ed andando indietro con "1" si ottiene che (q_1, q_5) sono distinguibili:

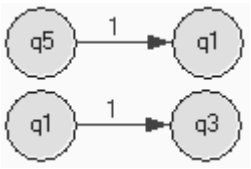


Figura 67 Gli stati (q_1, q_5) sono distinguibili.

- guardando (q_0, q_5) ed andando indietro con "1" si ottiene che (q_3, q_4) sono distinguibili:

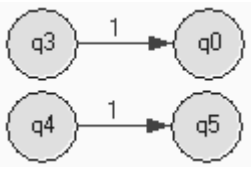


Figura 68 Gli stati (q_3, q_4) sono distinguibili.

In questo modo si può completare la tabella:

q1					
q2	D_0	D_0			
q3	D_0	D_0			
q4	D_0	D_0	D_2	D_2	
q5	D_1	D_1	D_0	D_0	D_0
	q0	***q1***	***q2***	***q3***	***q4***

Tabella 4 Tabella degli stati distinguibili dell'automa.

dove le caselle bianche rappresentano le coppie di stati indistinguibili.

Esempio 19

Data una grammatica lineare destra (Tipo 3) con le seguenti regole:

1)$A \rightarrow bA$;
2)$A \rightarrow aB$;
3)$B \rightarrow aB$;
4)$B \rightarrow bC$;
5)$C \rightarrow aC$;
6)$C \rightarrow bC$;
7)$C \rightarrow a$;
8)$C \rightarrow b$;

un esempio (degli infiniti possibili) di derivazione è essere:

$$A \overset{1}{\Rightarrow} bA \overset{1}{\Rightarrow} bbA \overset{2}{\Rightarrow} bbaB \overset{3}{\Rightarrow} bbaaB \overset{3}{\Rightarrow} bbaaaB \overset{4}{\Rightarrow} bbaaabC \overset{6}{\Rightarrow} bbaaabbC \overset{7}{\Rightarrow} bbaaabba$$

e si arriva alla stringa derivata composta soltanto da simboli terminali.

Da grammatica ad automa[55]

Il meccanismo di derivazione è molto simile al riconoscimento di una stringa; l'automa parte da uno stato iniziale (A) e legge da sinistra a destra. I prefissi delle varie stringhe intermedie equivale alla parte di parola già letta, ed ogni regola della grammatica corrisponde ad un arco dell'automa corrispondente. Presi la grammatica e l'automa dell'Esempio 19:

$$G = \{\Sigma,\ V,\ P,\ \Omega\} \Rightarrow a = (\Sigma,\ Q,\ q_0,\ F,\ \delta)$$

con:

$$Q = V \cup \{T\} \qquad \text{con } T \notin V$$
$$q_0 = \Omega$$
$$F = \{T\}$$
$$\delta(q,\ a) = \begin{cases} q = t & \text{non definito} \\ q = A \in V \text{ et} \begin{cases} \exists A \rightarrow aB \Rightarrow B \\ \exists A \rightarrow a \Rightarrow T \end{cases} \end{cases}$$

Ad es.:

$$\delta(A,\ a) = B$$
$$\delta(A,\ b) = A$$

e l'automa corrispondente alla grammatica è:

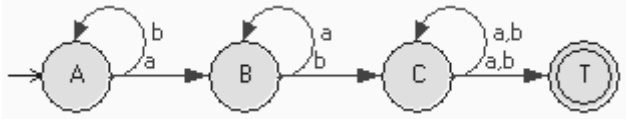


Figura 69 Automa dell'Esempio 19.

Da automa a grammatica

[55] La grammatica deve essere lineare destra (Tipo 3).

Al contrario si ha:

$$a = (\Sigma,\ Q,\ q_0,\ F,\ \delta) \Rightarrow G = \{\Sigma,\ V,\ P,\ \Omega\}$$

con:

$$V = Q$$

$$P = \begin{cases} \delta(q,\ a) = P \notin F & q \rightarrow aP \\ \delta(q,\ a) = P \in F & q \rightarrow a \end{cases}$$

$$\Omega = q_0$$

N.B.: ogni regola corrisponde ad un arco e viceversa.

N.B.: non è detto che si ricostruisca la stessa grammatica.

Le regole sono quindi:

1)$A \rightarrow bA$;
2)$A \rightarrow aB$;
3)$B \rightarrow aB$;
4)$B \rightarrow bC$;
5)$C \rightarrow aC$;
6)$C \rightarrow bC$;
7)$C \rightarrow aT$;
8)$C \rightarrow bT$;
9)$C \rightarrow a$;
10)$C \rightarrow b$;

ma dato che nessuna regola ha come input T, la "7" e la "8" si possono eliminare e si hanno le stesse regole della grammatica originaria.

Albero di derivazione (di una stringa w rispetto ad una grammatica G)

E' un albero[56] etichettato[57] ed ordinato[58] in cui:

1)la radice ha come etichetta l'assioma;
2)i nodi hanno come etichetta simboli non terminali;
3)le foglie hanno come etichetta simboli terminali;
4)se nell'albero c'è un nodo A con n figli (x_1, x_2, … x_n) allora nella grammatica G c'è la regola:

$$A \rightarrow x_1,\ x_2,\ \ldots,\ x_n$$

La stringa w è quindi la lettura delle foglie da sinistra a destra.

07/12/2001

Esempio 20 (linguaggio delle palindrome)

Il *linguaggio delle palindrome* è definito come:

[56] E' un grafo non orientato, connesso ed aciclico.
[57] Ogni nodo ha un'etichetta.
[58] Le etichette hanno un certo ordine.

$$\left.\begin{array}{l} w = a_1 a_2 ... a_n \\ \tilde{w} = a_n ... a_2 a_1 \end{array}\right\} w \text{ è palindroma se } w = \tilde{w}$$

quindi:

$$L(P) = \left\{ w \in \Sigma^* : w = \tilde{w} \right\} \qquad \text{con } \Sigma = \{a,\ b\}$$

Si può dare una *definizione ricorsiva* del linguaggio:

- base = *a*, *b*, *aa*, *bb* sono palindrome;
- induzione = *v* è palindroma $\Rightarrow$ *ava* e *bvb* sono palindrome;

quindi tutte le palindrome si ottengono da quelle base applicando la regola di induzione.

Utilizzando una grammatica CF per le palindrome, si ha che:

$$\Sigma = \{a,\ b\}$$
$$V = \{\Omega\}$$
$$P : A \to \alpha$$

e la definizione ricorsiva si può scrivere come:

1)$\Omega \to a$;
2)$\Omega \to b$;
3)$\Omega \to aa$;
4)$\Omega \to bb$;
5)$\Omega \to a\Omega a$;
6)$\Omega \to b\Omega b$;

Un es. può essere:

$$abaab \underbrace{bb}_{\text{base}} baaba$$

In questa grammatica il meccanismo di derivazione, che rappresenta la ricorsività, va dall'esterno all'interno ed è del tipo:

$$\Omega \Rightarrow a\Omega a \Rightarrow ab\Omega ba \Rightarrow aba\Omega aba \Rightarrow abaa\Omega aaba \Rightarrow abaab\Omega baaba \Rightarrow abaabbbbaaba$$

Espressioni aritmetiche BNF

Sono espressioni aritmetiche con i simboli +, * e nelle variabili *x*, *y*, *z*. Secondo la notazione BNF si ha che, ricorsivamente:

$$\langle \text{espr. aritmetica} \rangle ::= \langle \text{espr. aritmetica} \rangle + \langle \text{espr. aritmetica} \rangle$$
$$\langle \text{espr. aritmetica} \rangle ::= \langle \text{espr. aritmetica} \rangle * \langle \text{espr. aritmetica} \rangle$$
$$\langle \text{espr. aritmetica} \rangle ::= x$$
$$\langle \text{espr. aritmetica} \rangle ::= y$$
$$\langle \text{espr. aritmetica} \rangle ::= z$$

Esempio 21

Le regole di produzione di questa grammatica *E* sono:

1)E → E+E;
2)E → E*E;

3)E → *x*;
4)E → *y*;
5)E → *z*;

inoltre si ha:

$$V = \{E\}$$
$$\Sigma = \{x,\ y,\ z,\ +,\ *\}$$

Ad es. si ha:

$$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow x + E * E \Rightarrow x + y * E \Rightarrow x + y * z$$

Linguistica

Passando alla linguistica si ha:

1)F → P_nP_v	Frase → Parte nominale + Parte verbale
2)P_n → A_rNA_g	Parte nominale → Articolo + Nome + Aggettivo
3)P_v → VP_n	Parte verbale → Verbo + Parte nominale

e prendendo la frase:

$$\overbrace{\overbrace{\text{il}}^{A_r}\ \overbrace{\text{gatto}}^{N}\ \overbrace{\text{bianco}}^{A_g}}^{P_n}\ \overbrace{\overbrace{\text{insegue}}^{V}\ \overbrace{\text{il topo nero}}^{P_n}}^{P_v}$$

si ha:

- N → gatto, topo;
- A_r → il;
- A_g → bianco, nero;
- V → insegue;

Più che come meccanismo di analisi, si può vedere come generazione di una frase:

$$F \Rightarrow P_nP_v \Rightarrow A_rNA_gP_v \Rightarrow A_rNA_gVP_v$$

visto che, con la stessa struttura, si può creare la frase:

il gatto nero insegue il topo bianco

N.B.: in questo modo si possono generare frasi corrette sintatticamente ma scorrette semanticamente.

N.B.: la grammatica è quindi un meccanismo per generare frasi, l'automa, invece, è un modello per riconoscere le frasi.

Esempio 22

L'albero di derivazione per la stringa riferita alla grammatica precedente è:

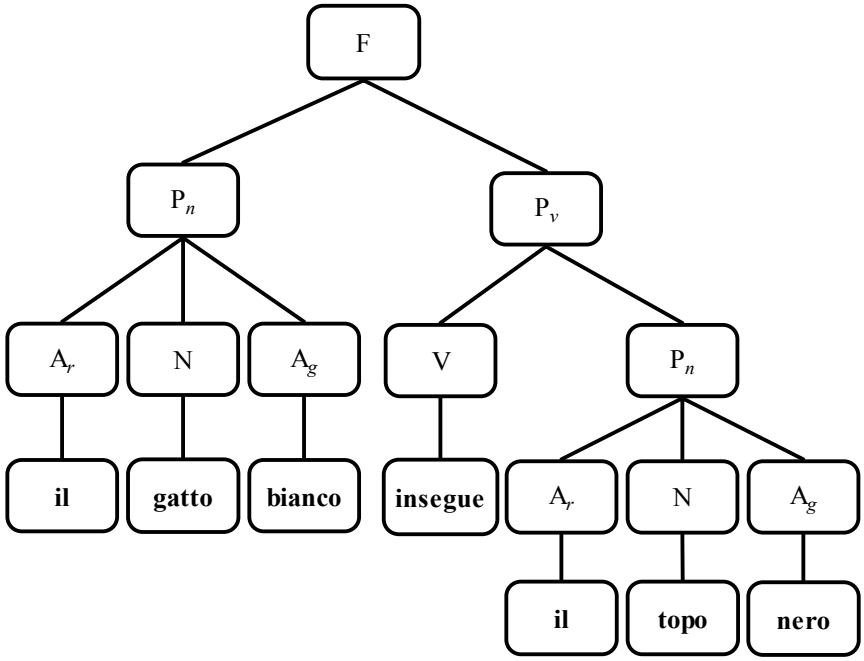


Figura 70 Albero di derivazione per la grammatica precedente.

Esempio 23

Un modo di catturare il meccanismo di generazione, indipendente dall'ordine delle sostituzioni, può essere, per es. prendendo la grammatica *E* dell'Esempio 21:

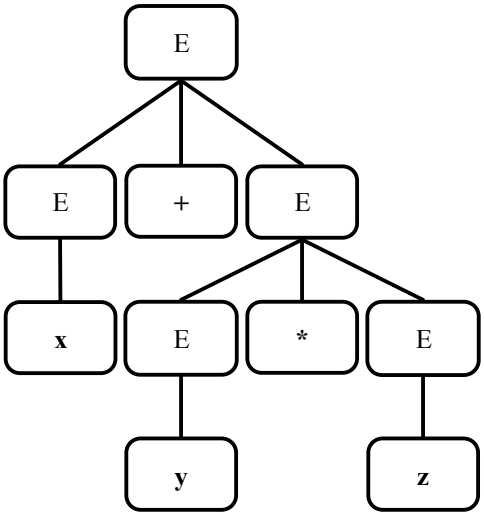


Figura 71 Albero di derivazione per la grammatica *E*.

N.B.: l'albero di derivazione è relativo ad una particolare stringa prodotta da una grammatica e non alla grammatica stessa.

Esempio 24 (linguaggio delle palindrome)

Riferendosi al linguaggio delle palindrome (Esempio 20), per la parola *abbabba* l'albero di derivazione è:

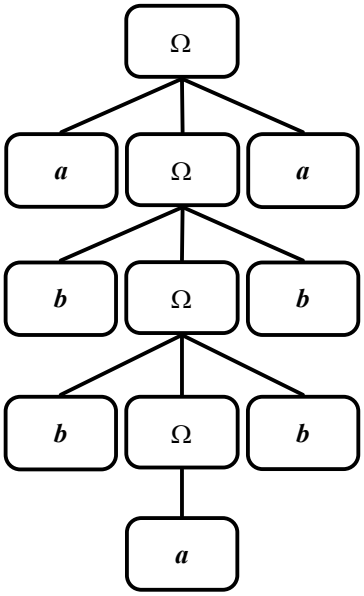


Figura 72 Albero di derivazione per la stringa palindroma *abbabba*.

nel quale si vede la struttura ad incapsulamento ed il simmetrismo.

Ambiguità

Una grammatica si dice *ambigua* se:

$$\exists w \in L(G) : w \text{ ammette due diversi alberi di derivazione}$$

Questo problema si può risolvere passando dalla sintassi alla semantica:

1)si introducono le parentesi;
2)si da una priorità ad ogni operazione;

Esempio 25

La grammatica *E* è ambigua, visto che si possono trovare due alberi di derivazione per la stessa stringa[59]:

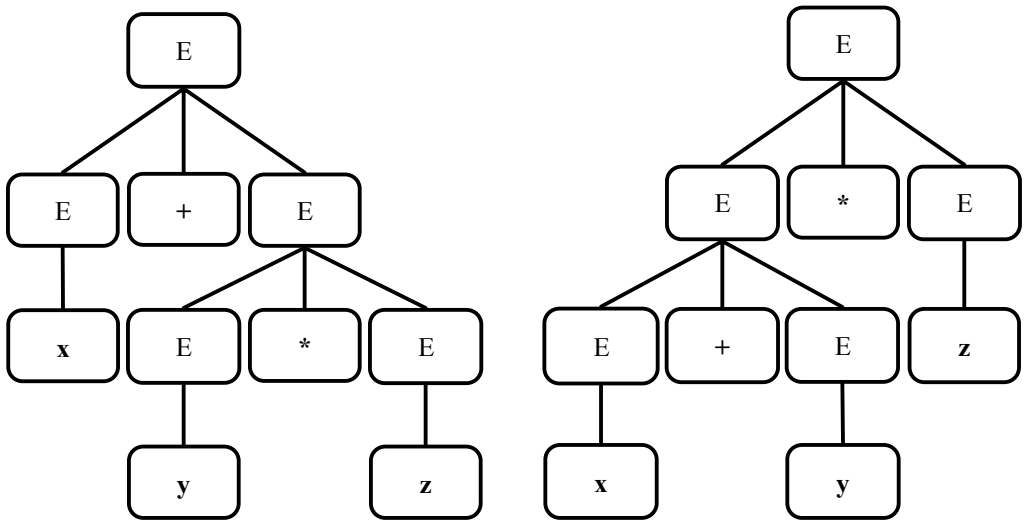


Figura 73 Esempio di grammatica ambigua.

[59] Anche se il secondo è semanticamente sbagliato.

Teorema 17 (Indecidibilità del problema dell'ambiguità)

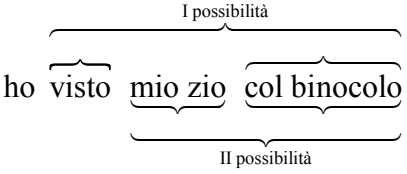
$\nexists$ alcun algoritmo per decidere se una certa grammatica G è ambigua

Linguaggio inerentemente ambiguo

Se tutte le grammatiche che lo generano sono ambigue; l'ambiguità è quindi legata al linguaggio.

Esempio 26

Questa grammatica è ambigua:



10/12/2001

Forma normale[60] di Chomsky (CNF)

Una grammatica non contestuale G è CNF se le tutte regole di produzione P sono della forma:

$$A \to BC$$
$$A \to a$$

Teorema 18

$$\text{data una grammatica } G \Rightarrow \begin{cases} \exists \text{ una grammatica equivalente in CNF} \\ \exists \text{ un algoritmo per generarla} \end{cases}$$

Teorema 19

ogni grammatica CF è equivalente ad una grammatica CNF

Dim.:

La dimostrazione procede per costruzione. Per ogni regola della grammatica CF della forma:

$$A \to X_1, \ldots, X_{i_1}, \ldots, X_{i_k}, \ldots, X_n \qquad \text{con } X_i \in \Sigma \cup V$$

in cui i simboli X_i possono essere terminali o meno. Si prendono quelli terminali (in totale k) e si sostituiscono con k nuovi simboli non terminali:

$$B_1, \ldots, B_k$$

ad es. $X_{i_1}, \ldots, X_{i_k}$ terminali; si sostituiscono:

$$A \to X_1, \ldots, B_1, \ldots, B_k, \ldots, X_n$$

[60] E' simile al concetto di riduzione per gli automi.

e si introducono le nuove regole:

$$\begin{aligned} B_1 &\to X_{i_1} \\ &\vdots \quad \vdots \quad \vdots \\ B_k &\to X_{i_k} \end{aligned}$$

si ottiene quindi una grammatica con le regole del tipo:

$$\begin{aligned} A &\to X_1, \ \ldots\ , X_n \qquad \text{con } X_i \in V \\ A &\to a \qquad\qquad\quad \text{con } a \in \Sigma \end{aligned}$$

Per ridurla ad una CNF, bisogna sostituire la prima regola introducendo $n-2$ nuovi simboli non terminali:

$$D_1, D_2, \ \ldots\ , D_{n-2}$$

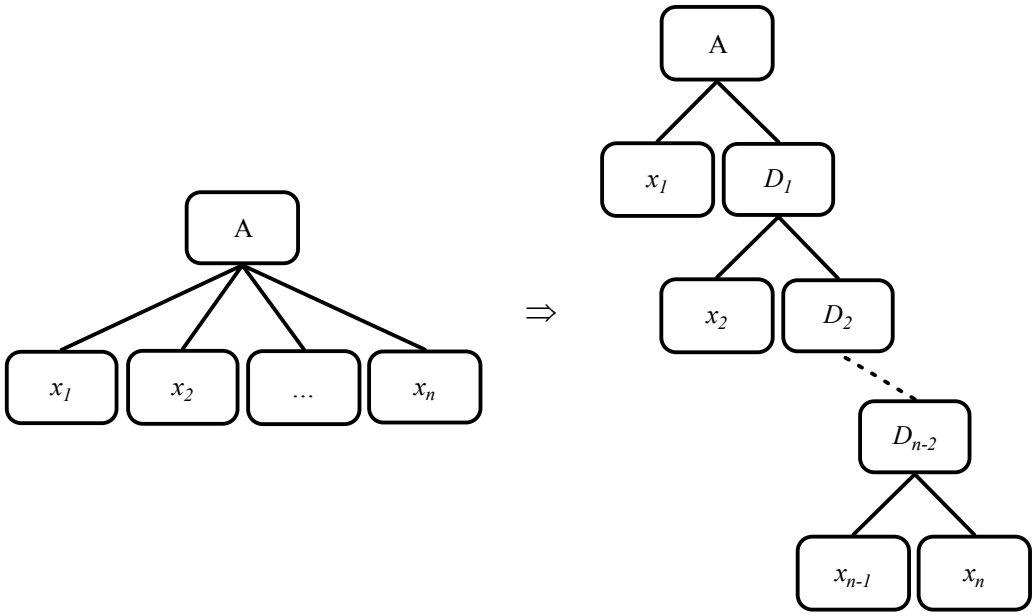
e l'insieme di regole diventa:

$$\begin{aligned} A &\to X_1 D_1 \\ D_1 &\to X_2 D_2 \\ &\vdots \quad \vdots \quad\ \ \vdots \\ D_{n-3} &\to X_{n-2} D_{n-2} \\ D_{n-2} &\to X_{n-1} X_n \end{aligned}$$

N.B.: in una grammatica CNF, se le sole regole sono del tipo:

$$\begin{aligned} A &\to BC \\ A &\to a \end{aligned}$$

i possibili alberi di derivazione sono binari. Si ha quindi che, applicando in serie le regole, si trasforma l'albero *k*-ario in uno binario:



Esempio 27 (linguaggio delle palindrome)

Riferendosi al linguaggio delle palindrome (Esempio 20), la grammatica diventa:

G:
1)$\Omega \to a\Omega a$;

G':
1)$\Omega \to B_a \Omega\, B_a$;

2)$\Omega \rightarrow b\Omega b$;
3)$\Omega \rightarrow aa$;
4)$\Omega \rightarrow bb$;
5)$\Omega \rightarrow a$;
6)$\Omega \rightarrow b$;

2)$B_a \rightarrow a$;
3)$\Omega \rightarrow B_b\Omega\, B_b$;
4)$B_b \rightarrow b$;
5)$\Omega \rightarrow B_a\, B_a$;
6)$\Omega \rightarrow B_b\, B_b$;
7)$\Omega \rightarrow a$;
8)$\Omega \rightarrow b$;

Dalla prima regola:

$$\Omega \rightarrow B_a D_1$$
$$D_1 \rightarrow \Omega B_a$$

La seconda si ricopia in quanto è già in CNF:

$$B_a \rightarrow a$$

La terza diventa:

$$\Omega \rightarrow B_b D_2$$
$$D_2 \rightarrow \Omega B_b$$

Alla fine si ha che le regole sono in forma normale di Chomsky:

$$\Omega \rightarrow B_a D_1$$
$$D_1 \rightarrow \Omega B_a$$
$$B_a \rightarrow a$$
$$\Omega \rightarrow B_b D_2$$
$$D_2 \rightarrow \Omega B_b$$
$$B_b \rightarrow b$$
$$\Omega \rightarrow B_a B_a$$
$$\Omega \rightarrow B_b B_b$$
$$\Omega \rightarrow a$$
$$\Omega \rightarrow b$$

cioè tutte portano o a due simboli non terminali o a uno terminale.

N.B.: il prezzo da pagare è nel numero di regole (visto che se ne devono aggiungere).

Forma normale di Geibach (GNF)

Una grammatica non contestuale G è GNF se tutte le regole di produzione P sono della forma:

$$A \rightarrow \alpha B \qquad \text{con } B \in V^*$$

come ad es.:

$$A \rightarrow aB_1,\ B_2,\ \ldots,\ B_k \qquad \text{con } B_i \in V$$

N.B.: B può anche non esistere; quindi la regola:

$$A \rightarrow \alpha \Leftrightarrow A \rightarrow \alpha B \qquad \text{con } B = \varepsilon$$

Teorema 20

ogni grammatica CF è equivalente ad una grammatica GNF

Derivazione left-most/right-most (canonica sinistra/destra)

Data una stringa con simboli terminali e non, si può sostituire il simbolo più a sinistra/destra. Ad es.:

$$\Omega \Rightarrow \Omega + \Omega \Rightarrow x + \Omega \Rightarrow x + \Omega * \Omega \Rightarrow x + y * \Omega \Rightarrow x + y * z$$

Teorema 21

$\forall$ stringa derivata $\exists!$ derivazione left-most/right-most

N.B.: si ha una grammatica ambigua se questa ammette due derivazioni distinte left-most/right-most.

Esempio 28

Avendo le due regole:

1)$B_1 \rightarrow b$;
2)$B_2 \rightarrow aC_1C_2$;

la derivazione left-most è:

$$\Omega \Rightarrow aB_1B_2B_3...B_k \Rightarrow abB_2B_3...B_k \Rightarrow abaC_1C_2B_3...B_k \Rightarrow$$

Andando a guardare la forma delle stringhe intermedie così costruite si vede che saranno composte da simboli terminali seguiti da simboli non terminali:

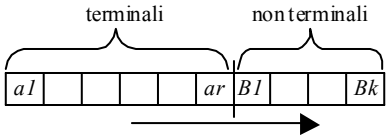


Figura 74 I simboli non terminali tendono a scomparire.

e, alla fine del computo, la parte della stringa con simboli non terminali scomparirà.

N.B.: questo meccanismo è quello di un automa a pila PDA.

Teorema 22

un linguaggio è generato da una grammatica CF $\Leftrightarrow$ è riconosciuto da un automa a pila PDA non deterministico[61]

Notazione polacca (prefissa)

Permette di eliminare l'ambiguità in una grammatica. Ad es. l'espressione:

$$(x * y) * (x * ((y + y) + z)) \Rightarrow * + xy * x + + yyz \qquad \text{con } \Sigma = \{+, *, x, y, z\}$$

Definizione ricorsiva di notazione polacca ed automa a pila

[61] In una grammatica si possono avere più scelte nella sostituzione.

Un'espressione in forma polacca si definisce come:

1)la concatenazione di due espressioni in forma polacca preceduta da +;
2)la concatenazione di due espressioni in forma polacca preceduta da *;
3)*x*;
4)*y* ;
5)*z*;

Sotto forma di regole di una grammatica si ha :

1)$\Omega \rightarrow +\Omega\Omega$;
2)$\Omega \rightarrow *\Omega\Omega$;
3)$\Omega \rightarrow x$;
4)$\Omega \rightarrow y$;
5)$\Omega \rightarrow z$;

Il corrispettivo albero di derivazione è :

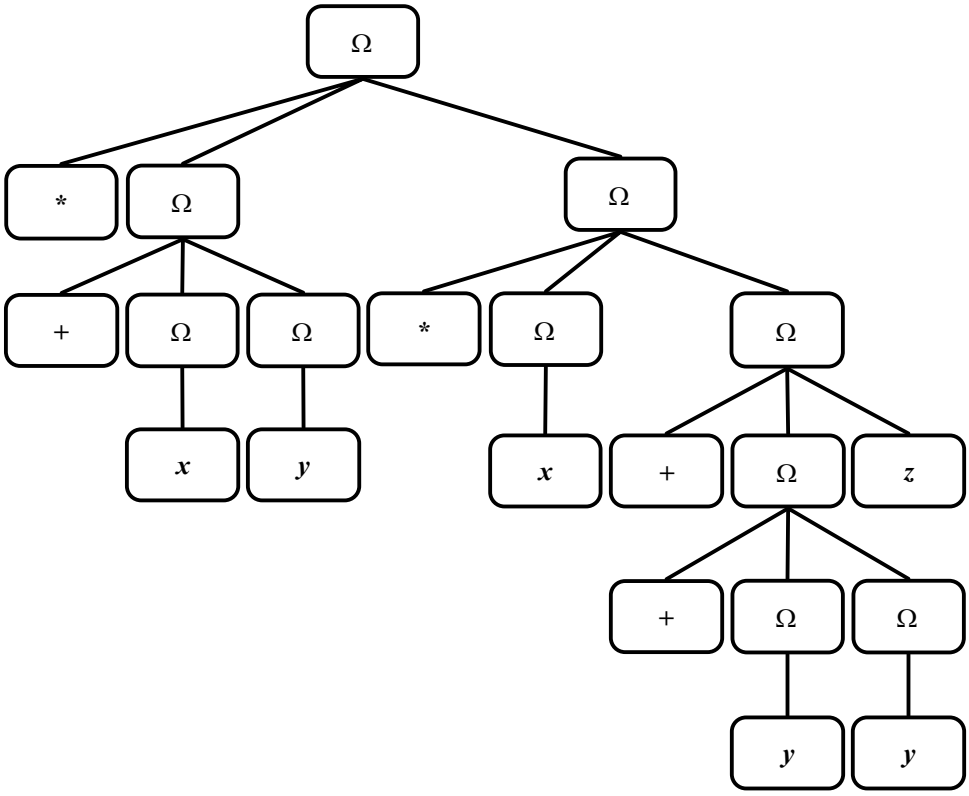


Figura 75 Albero di derivazione per un'espressione in forma polacca.

mentre la derivazione left-most è:

$$\Omega \Rightarrow *\Omega\Omega \Rightarrow *+\Omega\Omega\Omega \Rightarrow *+x\Omega\Omega \Rightarrow *+xy\Omega \Rightarrow *+xy*\Omega\Omega \Rightarrow$$
$$\Rightarrow *+xy*x\Omega \Rightarrow *+xy*x+\Omega\Omega \Rightarrow *+xy*x++\Omega\Omega\Omega \Rightarrow$$
$$\Rightarrow *+xy*x++y\Omega\Omega \Rightarrow *+xy*x++yy\Omega \Rightarrow *+xy*x++yyz$$

Si può costruire l'automa a pila che riconosce la stringa. Questo legge un simbolo e guarda la cima della pila:

1)se in cima c'è un simbolo terminale allora:

- se è diverso da quello che sta leggendo $\Rightarrow$ la stringa non è accettata e si ferma;
- se è uguale a quello che sta leggendo $\Rightarrow$ lo cancella dalla pila e sposta la testa a destra;

2)se in cima c'è un simbolo non terminale *A* e sta leggendo *a* allora guarda se nella

grammatica G c'è una regola della forma $A \to a\beta$:

- sì $\Rightarrow$ sostituisco nella pila A con $a\beta$;
- no $\Rightarrow$ la stringa non è accettata e mi fermo;

XXX(esempio PDA)

13/12/2001

Esempio 29

$$L(\text{FSA}) \subset L(\text{CF})$$

infatti:

$$L = \{a^n b^n : n \geq 1\} \Rightarrow L \notin L(\text{FSA})$$

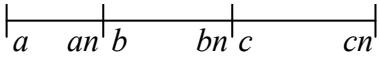
$$\left.\begin{array}{l} \Omega \to a\Omega b \\ \Omega \to ab \end{array}\right\} \Rightarrow L \in L(\text{CF})$$

Lemma di iterazione per linguaggi CF

Presa una qualsiasi parola di un linguaggio CF, si ha una coppia di fattori (*coppia iterante*) che può essere sempre individuata e tale che, se la si ripete n volte (con $n \geq 0$), la stringa ottenuta appartiene ancora al linguaggio CF. I due fattori si possono scegliere non troppo lontani. XXX

Esempio 30

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



Questo linguaggio $\in L$(CF) perché, presi due blocchi, il terzo gruppo di lettere è diverso XXX.

Teorema 23 (fattorizzazione)

$$L \in L(CF) \Rightarrow \exists k \in \mathbb{N} : w \text{ si fattorizza in:}$$

$$w = xuyvz$$

con:

$$\forall w \in L$$

$$x,\ y,\ z \in \Sigma^*$$

$$|w| > k$$

$$|uv| > 0$$

$$|y| < k$$

cioè x, y, z possono essere la parola vuota, mentre u, v non possono essere entrambi la parola vuota. Si ha:

$$xu^n yv^n z \in L \qquad \forall n \geq 0$$

Dim.:

Presa una grammatica G in CNF:

$$\begin{aligned} &G = \{\Sigma,\ V,\ P,\ \Omega\} \\ &L(G) = L \end{aligned} \qquad \text{con} \begin{cases} w \in L \\ |w| > k \\ k = 2^{|V|} \end{cases}$$

e si ha che:

$$\left.\begin{aligned} &|w| > k = 2^{|V|} \\ &2^P > 2^{|V|} \end{aligned}\right\} \Rightarrow P > |V| \qquad \text{con} \begin{cases} w = \text{numero di foglie dell'albero} \\ P = \text{profondità dell'albero} \end{cases}$$

Presa una parola w di lunghezza $>k$, il suo albero di derivazione ed un cammino, in questo si avranno, ad es., due nodi con la stessa etichetta (visto che la profondità dall'albero è $P>|V|$). Si prendono i sottoalberi con radici uguali ai nodi aventi le due etichette ripetute:

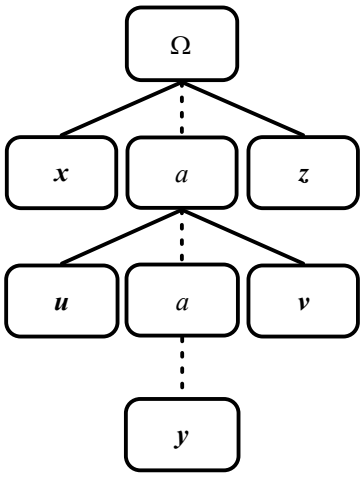


Figura 76 Albero di derivazione di w con le due etichette ripetute a.

In questo modo si prendono coppie vicine, dato che si prendono le coppie di nodi più profonde con etichetta uguale; questo si traduce in stringhe più vicine, cioè:

$$|y| < k$$

Questo fatto soddisfa la definizione data. Partendo dall'assioma Ω, si ha la derivazione con le seguenti regole:

1) $\Omega \overset{*}{\Rightarrow} xAz$

2) $A \overset{*}{\Rightarrow} uAv$

3) $A \overset{*}{\Rightarrow} y$

Esempio 31

Utilizzando le regole precedenti si ha:

a) $\Omega \underset{1}{\overset{*}{\Rightarrow}} xAz \underset{2}{\overset{*}{\Rightarrow}} xuAvz \underset{3}{\overset{*}{\Rightarrow}} xuyvz = w \in L \qquad \text{con } n = 1$

b) $\Omega \underset{1}{\overset{*}{\Rightarrow}} xAz \underset{3}{\overset{*}{\Rightarrow}} xyz$

c) $\Omega \underset{1}{\overset{*}{\Rightarrow}} xAz \underset{2}{\overset{*}{\Rightarrow}} xuAvz \underset{2}{\overset{*}{\Rightarrow}} xuuAvvz \underset{3}{\overset{*}{\Rightarrow}} xuuyvvz$

e, i corrispettivi alberi di derivazione sono:

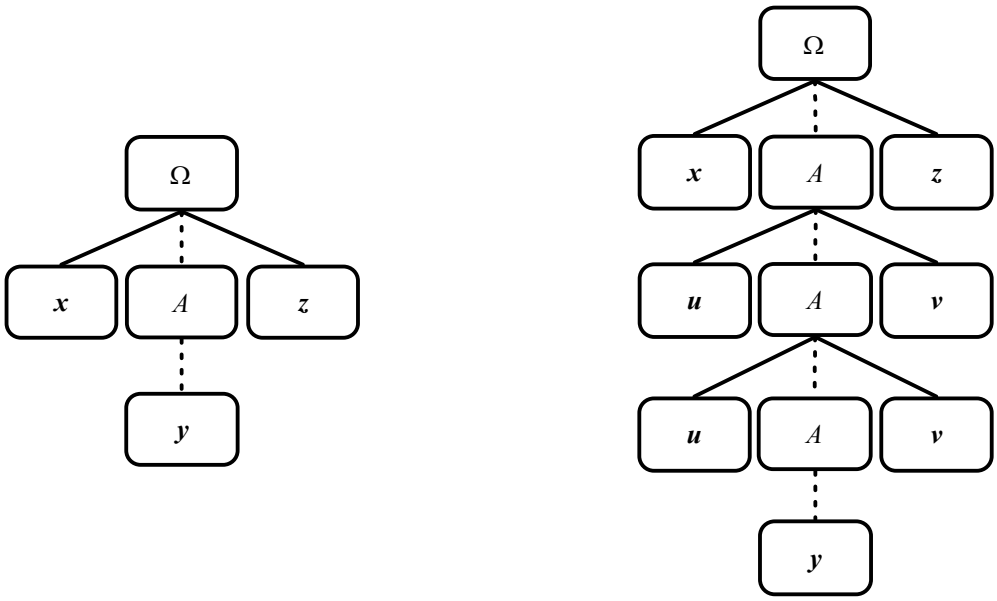


Figura 77 Alberi di derivazione per le derivazioni b e c.

Esempio 32 (linguaggio delle palindrome di lunghezza pari)

Definiamo le palindrome (di lunghezza) pari:

$$PP = \left\{ w\tilde{w} : w \in \Sigma^* \right\} \Rightarrow PP \in L(\mathrm{CF})$$

che si ottiene a partire dalla più piccole palindrome pari ("*aa*" e "*bb*") con le seguenti regole:

1)$\Omega \rightarrow a\Omega a$;
2)$\Omega \rightarrow b\Omega b$;
3)$\Omega \rightarrow aa$;
4)$\Omega \rightarrow bb$;

Esempio 33 (linguaggio Copy)

Definiamo il linguaggio Copy come segue:

$$Copy = \left\{ ww : w \in \Sigma^* \right\} \Rightarrow Copy \notin L(\mathrm{CF})$$

Presa una stringa molto lunga:

$$V = \underbrace{a^n b^m}_{w}\,\underbrace{a^n b^m}_{w}$$

$$a \quad an \quad b \quad bm \quad a \quad an \quad b \quad bm$$

e una coppia di fattori iteranti, questi si dovrebbero mettere nei due blocchi di *a*; ma questi sono molto lontani tra loro, visto che in mezzo c'è *b* con *m>k*. Dovrei quindi prenderli tra *a* e *b*, ma cioè è impossibile, quindi non esiste una coppia iterante.

Proprietà di chiusura di L(CF)

1)<u>*Unione*</u> = $L_1,\ L_2 \in L(\mathrm{CF}) \Rightarrow L_1 \cup L_2 \in L(\mathrm{CF})$

cioè:

$$L_1 = L(G_1) \in L(\mathrm{CF}) \quad \text{con } G_1 = \{\Sigma,\ V_1,\ P_1,\ \Omega_1\}$$
$$L_2 = L(G_2) \in L(\mathrm{CF}) \quad \text{con } G_2 = \{\Sigma,\ V_2,\ P_2,\ \Omega_2\}$$

si prende allora la grammatica:

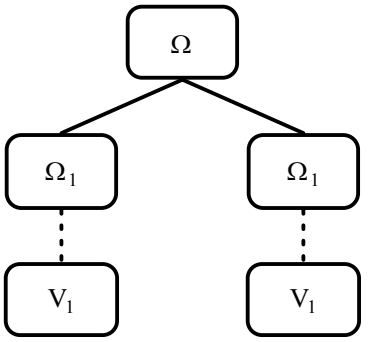
$$G = \{\Sigma,\ V,\ P,\ \Omega\} \quad \text{con} \begin{cases} V_1 \cap V_2 = \varnothing \\ V = (V_1 \cup V_2 \cup \{\Omega\}) \\ P = (P_1 \cup P_2 \cup \{\Omega \to \Omega_1,\ \Omega \to \Omega_2\}) \\ \Omega \notin V_1 \\ \Omega \notin V_2 \end{cases}$$

2)<u>*Concatenazione*</u> = $L_1,\ L_2 \in L(\mathrm{CF}) \Rightarrow L_1 L_2 \in L(\mathrm{CF})$

cioè la grammatica è definita come:

$$G = \{\Sigma,\ V,\ P,\ \Omega\} \quad \text{con} \begin{cases} V_1 \cap V_2 = \varnothing \\ V = (V_1 \cup V_2 \cup \{\Omega\}) \\ P = (P_1 \cup P_2 \cup \{\Omega \to \Omega_1\Omega_2\}) \\ \Omega \notin V_1 \\ \Omega \notin V_2 \end{cases}$$

e l'albero di derivazione è del tipo:



$$\text{con} \begin{cases} V_1 \in L_1 \\ V_2 \in L_2 \end{cases}$$

N.B.: non valgono l'intersezione ed il complemento:

3)<u>*Intersezione*</u> = $L_1,\ L_2 \in L(\mathrm{CF}) \Rightarrow L_1 \cap L_2 \notin L(\mathrm{CF})$

Esempio 34

Dati due linguaggi

$$L_1 = \{a^n b^n c^m : n,\ m \geq 1\} = \{a^n b^n : n \geq 1\} \cdot \{c^m : m \geq 1\}$$
$$L_2 = \{a^n b^m c^m : n,\ m \geq 1\} = \{a^n : n \geq 1\} \cdot \{b^m c^m : m \geq 1\}$$

l'intersezione è:

$$L_1 \cap L_2 = \{a^n b^n c^n : n \geq 1\} \notin L(\mathrm{CF})$$ [62]

[62] Vedi "*Esempio 29*".

4)<u>*Complemento*</u> = $L_1 \in L(\mathrm{CF}) \Rightarrow L_1^c \notin L(\mathrm{CF})$

14/12/2001

Grammatiche di Tipo 1 (Context sensitive CS)

Si vogliono ricondurre regole della forma:

$$P : \alpha \to \beta \Leftrightarrow |\alpha| \le |\beta|$$

in regole che cambiano in base al contesto:

$$\begin{aligned} &\gamma_1 A \gamma_2 \Rightarrow \gamma_1 \alpha_1 \gamma_2 \\ &\beta_1 A \beta_2 \Rightarrow \beta_1 \alpha_2 \beta_2 \end{aligned} \qquad \text{con } |\gamma_1 \alpha_1 \gamma_2| \ge 0$$

Quindi, da:

$$AB \to CDE$$

si ha:

$$\begin{aligned} AB &\to CB \\ CB &\to CDE \end{aligned}$$

cioè

A diventa *C* se alla sua destra c'è una *B*;
B diventa *CE* se alla sua sinistra c'è una *C*;

che, componendole, riportano alla regola iniziale.

Proprietà di chiusura di L(CS)

Immerman dimostrò, negli anni '90, che i linguaggi CS sono chiusi per il complemento.

Conseguenze del lemma di iterazione

1) $L(G) \ne \varnothing \Leftrightarrow \exists w \in L(G) \quad \text{con } |w| \le k$

2) $L(G) = \infty \Leftrightarrow \exists w \in L(G) \quad \text{con } \begin{cases} k \le |w| \le 2k \\ k = 2^{|v|} \end{cases}$

N.B.: queste due conseguenze permettono di dire che EP ed FP sono decidibili per le grammatiche di Tipo 2, cioè per gli automi PDA.

Prove scritte[63]

29 Gennaio 2002

1)Scrivere un'espressione regolare per ognuno dei seguenti linguaggi sull'alfabeto $\Sigma=\{a, b\}$:

a)linguaggio riconosciuto da tutte le stringhe che non contengono più di 3 volte la lettera *a*;
b)linguaggio riconosciuto da tutte le stringhe che contengono un numero di *a* divisibile per 3;

2)Quali tra le seguenti affermazioni sono vere? (motivare la risposta)

a)$baa \in a^*b^*a^*b^*$

b)$b^*a^* \cap a^*b^* = a^* \cup b^*$

c)$a^*b^* \cap b^*c^* = \varnothing$

d)$abcd \in \left(a^*\left(cd\right)^*b\right)^*$

3)Costruire un DFA minimale per il linguaggio L, sull'alfabeto $\Sigma=\{a, b\}$, costituito da tutte le parole che contengono sia *ab* che *ba* come fattore;

4)Costruire un DFA minimale per il linguaggio L, sull'alfabeto $\Sigma=\{a, b\}$, costituito da tutte le parole il cui penultimo carattere sia una *b*;

5)Sia T il linguaggio, sull'alfabeto $\Sigma=\{0, 1, 2\}$, costituito da tutte le stringhe che, in base 3, rappresentano numeri pari. Costruire, se esiste, un DFA che riconosce T;

6)Una parola *v* sull'alfabeto $\Sigma=\{a, b, c\}$ si dice "senza quadrati" se:

$$v = xyyz \Rightarrow y = \varepsilon \qquad \forall x, y, z \in \{a, b, c\}^*$$

Sia SQ il linguaggio delle parole senza quadrati su $\Sigma=\{a, b, c\}$. Esiste un DFA che riconosce SQ? (Motivare la risposta);

7)Denotiamo con:

- *a* = parentesi tonda aperta (;
- *b* = parentesi tonda chiusa);

Determinare una grammatica in forma normale di Chomsky che genera il linguaggio delle espressioni corrette di parentesi tonde;

8)Siano *a* e *b* come nell'esercizio precedente e denotiamo inoltre con:

- *c* = parentesi quadra aperta [;
- *d* = parentesi quadra chiusa];

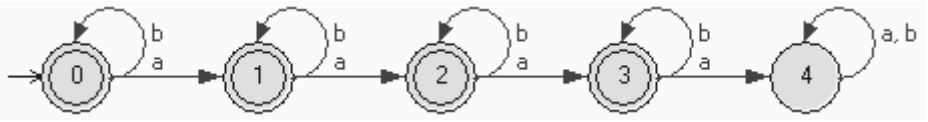
Determinare una grammatica context-free che genera il linguaggio delle espressioni corrette di parentesi tonde e quadre;

9)Esiste una grammatica context-free che genera il linguaggio SQ definito nell'esercizio 6?

Soluzioni

1a)L'automa DFA è:

[63] I testi sono quelli dati dal Prof., mentre le soluzioni sono state elaborate da me e dai miei colleghi; ci possono quindi essere degli errori.



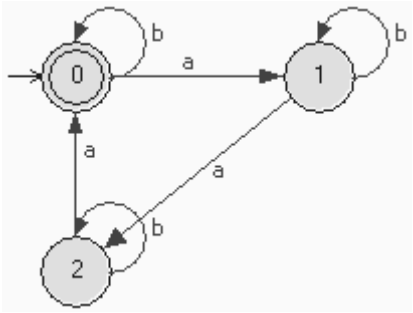
l'espressione regolare è:

$$b^*(a \cup \varepsilon)b^*(a \cup \varepsilon)b^*(a \cup \varepsilon)b^*$$

mentre la grammatica è composta dalle regole:

1)$\Omega \to a$;
2)$\Omega \to b$;
3)$\Omega \to aA$;
4)$\Omega \to b\Omega$;
5)$A \to a$;
6)$A \to b$;
7)$A \to aB$;
8)$A \to bA$;
9)$B \to a$;
10)$B \to b$;
11)$B \to bB$;
12)$\Omega \to \varepsilon$;

1b)L'automa DFA è:



l'espressione regolare è:

$$\left(b^*ab^*ab^*ab^*\right)^*$$

mentre la grammatica è composta dalle regole:

1)$\Omega \to aA$;
2)$\Omega \to b\Omega$;
3)$A \to aB$;
4)$A \to bA$;
5)$B \to a\Omega$;
6)$B \to bB$;
7)$\Omega \to \varepsilon$;

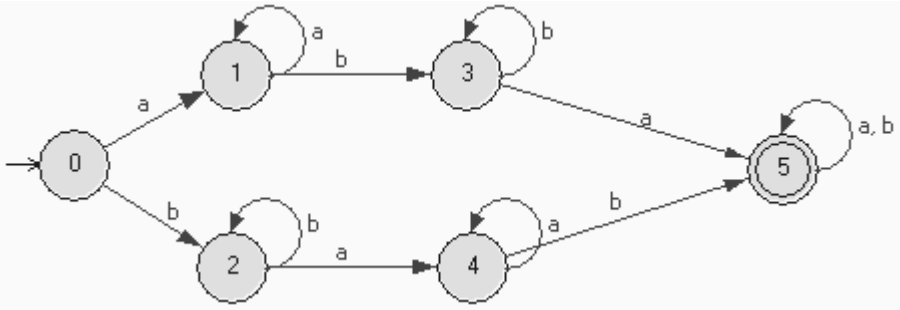
2a)Sì $\Rightarrow$ dato che $x^* = \{\varepsilon,\ a,\ aa,\ aaa,\ ...\} \Rightarrow a^0b^1a^2b^0 = \varepsilon baa\varepsilon$

2b)Sì $\Rightarrow$ $b^0a^* \cap a^0b^* = a^* \cup b^*$

2c)No $\Rightarrow$ $a^0b^* \cap b^*c^0 = b^*$

2d)No $\Rightarrow$ la stringa termina per *d*, mentre l'espressione regolare permette di generare stringhe che finiscono per *b*.

3)L'automa DFA è:



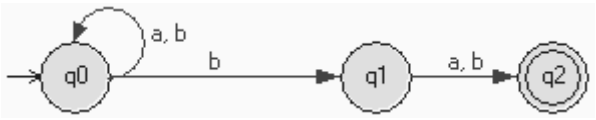
l'espressione regolare è:

$$\left(aa^*bb^*aa^*b^*\right)^* \cup \left(bb^*aa^*ba^*b^*\right)^*$$

mentre la grammatica è composta dalle regole:

1)$\Omega \rightarrow aA$;
2)$\Omega \rightarrow bB$;
3)$A \rightarrow aA$;
4)$A \rightarrow bC$;
5)$B \rightarrow aD$;
6)$B \rightarrow bB$;
7)C $\rightarrow aE$;
8)$C \rightarrow bC$;
9)$D \rightarrow bE$;
10)$D \rightarrow aD$;
11)$E \rightarrow aE$;
12)$E \rightarrow bE$;
13)$E \rightarrow \varepsilon$;

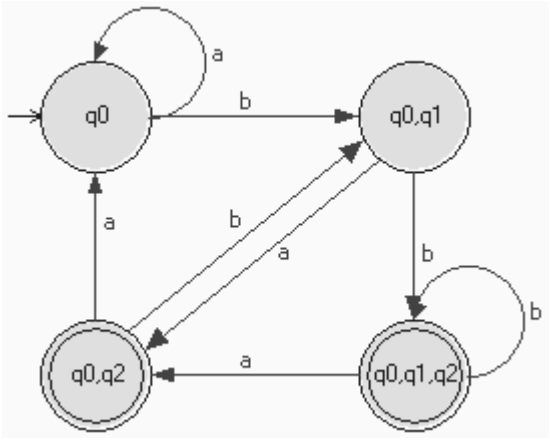
4)L'automa NFA è:



si passa quindi alla subset construction:

$$\delta_D\left(\{q_0\},\ a\right)=\{q_0\}$$
$$\delta_D\left(\{q_0\},\ b\right)=\{q_0,\ q_1\}$$
$$\delta_D\left(\{q_0,\ q_1\},\ a\right)=\{q_0,\ q_2\}$$
$$\delta_D\left(\{q_0,\ q_1\},\ b\right)=\{q_0,\ q_1,\ q_2\}$$
$$\delta_D\left(\{q_0,\ q_2\},\ a\right)=\{q_0\}$$
$$\delta_D\left(\{q_0,\ q_2\},\ b\right)=\{q_0,\ q_1\}$$
$$\delta_D\left(\{q_0,\ q_1,\ q_2\},\ a\right)=\{q_0,\ q_2\}$$
$$\delta_D\left(\{q_0,\ q_1,\ q_2\},\ b\right)=\{q_0,\ q_1,\ q_2\}$$

e l'automa DFA risultante è:



L'espressione regolare è:

$$DFA \Rightarrow \left(a^* bbb^*\right) \cup \left(a^* ba\right)$$

oppure

$$NFA \Rightarrow \left(a^* \cup b^*\right) b \left(a \cup b\right)$$

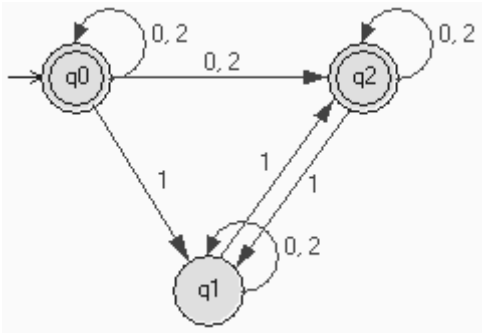
mentre la grammatica è composta dalle regole:

1)$\Omega \to a\Omega$;
2)$\Omega \to bA$;
3)$A \to a\Omega$;
4)$A \to bB$;
5)$B \to a$;
6)$B \to b$;

5)Basta notare che:

n. di "1" pari $\Rightarrow$ numero pari;
n. di "1" dispari $\Rightarrow$ numero dispari;

Quindi l'automa NFA è:

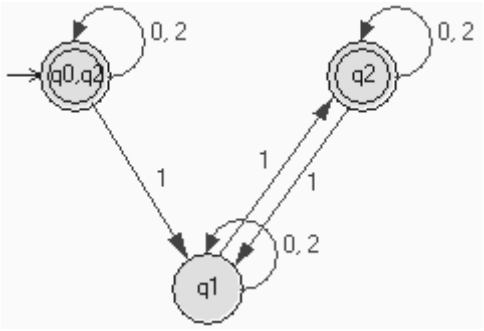


si passa quindi alla subset construction:

$$\delta_D\left(\{q_0\},\ 0\right) = \{q_0,\ q_2\}$$
$$\delta_D\left(\{q_0\},\ 1\right) = \{q_1\}$$
$$\delta_D\left(\{q_0\},\ 2\right) = \{q_0,\ q_2\}$$
$$\delta_D\left(\{q_1\},\ 0\right) = \{q_1\}$$
$$\delta_D\left(\{q_1\},\ 1\right) = \{q_2\}$$
$$\delta_D\left(\{q_1\},\ 2\right) = \{q_1\}$$

$$\delta_D\left(\{q_0,\ q_2\},\ 0\right)=\{q_0,\ q_2\}$$
$$\delta_D\left(\{q_0,\ q_2\},\ 1\right)=\{q_1\}$$
$$\delta_D\left(\{q_0,\ q_2\},\ 2\right)=\{q_0,\ q_2\}$$

e l'automa DFA risultante è:



6)XXX

7)La grammatica CNF è composta dalle regole[64]:

1)$\Omega \to A\Omega B$;
2)$\Omega \to AB$;
3)$\Omega \to \Omega\Omega$;
4)$A \to a$;
5)$B \to b$;

8)La grammatica CF è composta dalle regole:

1)$\Omega \to a\Omega$b;
2)$\Omega \to ab$;
3)$\Omega \to \Omega\Omega$;
4)$\Omega \to c\Omega$d;

9)XXX

[64] Vedi il "*Linguaggio delle parentesi*".

Appunti del corso di "*Informatica teorica*"

Università di Palermo

C.d.L. in Informatica

A.A. 2001/2002

Docente: Prof. Restivo

Autore: Dadà

Testo: J.E. Hopcraft – R. Motwani – J.D. Ullman "*Introduction to Automata theory, Languages, and Computation*", Addison-Wesley

e-mail: davide666@excite.com

URL: http://utenti.tripod.it/dada/