Il primo passo è inserire in S un insieme di produzioni che generino tutte e sole le stringhe che rappresentano possibili DI iniziali della MdT Z. Ciò è realizzabile con pochissime produzioni.

Il secondo passo è inserire in S una produzione del tipo 11.3 per ogni quintupla di Z, e due produzioni del tipo 11.4 per ogni stato di Z.

Il terzo e ultimo passo consiste nell'aggiungere, per ogni coppia (q_i, S_j) per cui non esiste alcuna quintupla in Z, una produzione che cancelli il simbolo di stato q_i. Da questo punto in avanti nessuna produzione di S sarà più applicabile. Queste stringhe "terminali" saranno quindi gli unici teoremi di S interamente composti da simboli tratti dall'alfabeto di Z, e coincideranno, per quanto visto nel paragrafo 11.3, con i possibili nastri terminali di Z.

12 ELEMENTI DI TEORIA DEI LINGUAGGI FORMALI

1. *Definizione di linguaggio formale*
2. *Relazione tra linguaggi e insieme di numeri*
3. *Operazioni fondamentali sui linguaggi*
4. *Rappresentazione dei linguaggi*
5. *Le grammatiche generative*
6. *Definizione formale di grammatica a struttura di frase*
7. *Classificazione delle grammatiche alla Chomsky*
8. *Alberi di derivazione e ambiguità*
9. *Sistemi di equazioni tra linguaggi*
10. *Grammatiche come equazioni tra linguaggi*

Nei capitoli precedenti si è affrontato il problema della definizione e caratterizzazione del concetto di algoritmo. Si sono quindi studiate le proprietà di calcolabilità e decidibilità che risultano come le più significative in quanto non dipendenti da una particolare formalizzazione. Si è visto come i diversi formalismi vengono specificati ciascuno con un suo proprio linguaggio per descrivere la struttura delle istruzioni (sintassi) e con un criterio (ad esempio un agente di calcolo) per associare a ciascun programma un preciso significato (semantica).

In effetti qualunque linguaggio di programmazione può essere pensato come un formalismo per esprimere algoritmi e nel quale si potrà quindi distinguere tra un aspetto sintattico ed uno semantico.

Il problema della descrizione della semantica dei programmi è un aspetto fondamentale dello studio dei linguaggi programmativi in quanto la fase di compilazione, ad esempio, che non è altro che un caso di traduzione automatica da un linguaggio ad alto livello ad un linguaggio macchina, non può prescindere dalla conoscenza del significato del testo del programma da compilare.

Il primo passo è inserire in S un insieme di produzioni che generino tutte e sole le stringhe che rappresentano possibili DI iniziali della MdT Z. Ciò è realizzabile con pochissime produzioni.

Il secondo passo è inserire in S una produzione del tipo 11.3 per ogni quintupla di Z, e due produzioni del tipo 11.4 per ogni stato di Z.

Il terzo e ultimo passo consiste nell'aggiungere, per ogni coppia (q_i, S_j) per cui non esiste alcuna quintupla in Z, una produzione che cancelli il simbolo di stato q_i. Da questo punto in avanti nessuna produzione di S sarà più applicabile. Queste stringhe "terminali" saranno quindi gli unici teoremi di S interamente composti da simboli tratti dall'alfabeto di Z, e coincideranno, per quanto visto nel paragrafo 11.3, con i possibili nastri terminali di Z.

12 ELEMENTI DI TEORIA DEI LINGUAGGI FORMALI

1. *Definizione di linguaggio formale*
2. *Relazione tra linguaggi e insieme di numeri*
3. *Operazioni fondamentali sui linguaggi*
4. *Rappresentazione dei linguaggi*
5. *Le grammatiche generative*
6. *Definizione formale di grammatica a struttura di frase*
7. *Classificazione delle grammatiche alla Chomsky*
8. *Alberi di derivazione e ambiguità*
9. *Sistemi di equazioni tra linguaggi*
10. *Grammatiche come equazioni tra linguaggi*

Nei capitoli precedenti si è affrontato il problema della definizione e caratterizzazione del concetto di algoritmo. Si sono quindi studiate le proprietà di calcolabilità e decidibilità che risultano come le più significative in quanto non dipendenti da una particolare formalizzazione. Si è visto come i diversi formalismi vengono specificati ciascuno con un suo proprio linguaggio per descrivere la struttura delle istruzioni (sintassi) e con un criterio (ad esempio un agente di calcolo) per associare a ciascun programma un preciso significato (semantica).

In effetti qualunque linguaggio di programmazione può essere pensato come un formalismo per esprimere algoritmi e nel quale si potrà quindi distinguere tra un aspetto sintattico ed uno semantico.

Il problema della descrizione della semantica dei programmi è un aspetto fondamentale dello studio dei linguaggi programmativi in quanto la fase di compilazione, ad esempio, che non è altro che un caso di traduzione automatica da un linguaggio ad alto livello ad un linguaggio macchina, non può prescindere dalla conoscenza del significato del testo del programma da compilare.

Per fare ciò è necessario dapprima effettuare una analisi del testo determinandone le componenti e la struttura in quanto presupposto per la comprensione del suo significato: bisogna cioè eseguirne una analisi sintattica.

Le tecniche sviluppate a tale scopo, oltre ai metodi di definizione della sintassi dei linguaggi e allo studio di loro proprietà sono alcuni degli argomenti della teoria dei linguaggi formali di cui ci occuperemo nei prossimi capitoli.

In particolare studieremo alcune classi di linguaggi in cui si tende a far rientrare la maggior parte dei linguaggi di programmazione in quanto per esse sono state sviluppate tecniche di riconoscimento e analisi particolarmente efficienti.

12.1. Definizione di linguaggio formale

Un linguaggio, da un punto di vista formale (sintattico), è costituito da un insieme di sequenze di simboli senza alcun riferimento a ciò che essi possano significare. Per darne una caratterizzazione più precisa premettiamo alcune definizioni.

Definizione 12.1.1. Un *alfabeto* è un insieme finito non vuoto i cui elementi sono detti *simboli* o *caratteri.*

Definizione 12.1.2. Una *stringa* o *parola* sull'alfabeto A è una sequenza finita o nulla di simboli di A ottenuta per giustapposizione. La lunghezza di una stringa è il numero di simboli che la compongono: la sequenza nulla ha lunghezza zero.

Definizione 12.1.3. La *concatenazione* di due stringhe è l'operazione che dà luogo alla stringa formata giustapponendo le stringhe.

Nel seguito useremo la lettera λ per indicare la stringa di lunghezza nulla. La concatenazione, che è un'operazione binaria sull'insieme di tutte le possibili stringhe viene indicata con $\cdot$, ma per brevità spesso verrà sottointesa. Per esempio se $x = ab$ e $y = ba$, allora $xy = abba$. La lunghezza della stringa x si indica con $|x|$. Siano x, y e z stringhe sull'alfabeto A e $w = xyz$: diremo che x è un *prefisso* e z un *suffisso* di w e che x, y, z sono *sottostringhe* di w. Notare come λ sia prefisso, suffisso e sottostringa di ogni parola.

Sia W l'insieme di tutte le stringhe finite su A.

L'operazione di concatenazione è un'operazione sull'insieme W che gode delle seguenti proprietà:

a) $\forall x, y \in W, \ (z = xy) \in W$: cioè W è chiuso rispetto alla concatenazione

b) $\forall v, x, y \ (v(xy) = (vx)y)$: proprietà associativa

c) $\forall x \ (x\lambda = \lambda x)$: λ è l'elemento identità

d) $\exists xy \ (xy \neq yx)$: la concatenazione non è commutativa

e) $\forall x, y (|xy| = |x| + |y|)$.

Le proprietà a) – c) permettono di affermare che l'insieme W è un semigruppo libero, con identità, rispetto all'operazione di concatenazione. Esso è noto come il semigruppo libero generato dall'alfabeto A.

Definizione 12.1.4. Un linguaggio L su un alfabeto A è un sottoinsieme di W.

12.2. Relazione tra linguaggi e insieme di numeri

Ovviamente se A è un alfabeto finito, W è numerabile(*). Ciò significa che esiste una corrispondenza biunivoca tra N e W. Inoltre una tale corrispondenza g può essere attuata in modo effettivo come segue.

Sia n la cardinalità di A e fissiamo una corrispondenza fra A e l'insieme $\{1, 2, \ldots, n\}$. Alla stringa $x = a_k \ldots a_1 a_0$ viene associato il numero:

$$g(x) = g(a_k \ldots a_1 a_0) = \sum_{i=0}^{k} c_i n^i$$

dove c_i è il numero associato al simbolo a_i.

Se $x = \lambda$ allora $g(x) = 0$.

Ad esempio sia $A = \{a, b, c\}$. Pensiamo associati gli elementi di

(*) W è numerabile anche se A è finito numerabile.

A rispettivamente 1, 2 e 3. Allora, usando la notazione decimale per i numeri, $g(acb) = 1 \times 3^2 + 3 \times 3 + 2 = 20$.

Notare inoltre come la corrispondenza introdotta associ anche ad ogni numero naturale esattamente un'unica stringa su A. Ciò si può ottenere, ad esempio, continuando a dividere il numero per n facendo in modo di ottenere sempre resti compresi fra 1 e n, fino ad ottenere una espressione del tipo $\Sigma\, c_i n^i$, $1 \leqslant c_i \leqslant n$. Ad esempio al numero 101 corrisponde la stringa su A:

$$101 = 3\times 33 + 2 = 3\times(3\times 10+3)+2 = 3\times(3\times 3+1)+3)+2 = 3\times 3^3 + 1\times 3^2 + 3\times$$
$$\times 3 + 2 = cacb\ .$$

In base a ciò si potrà parlare indifferentemente di insiemi di numeri come di linguaggi e proprietà degli uni saranno anche proprietà degli altri. Ad esempio l'insieme dei linguaggi su A, cioè di tutti i sottoinsiemi di W non è numerabile come non lo è l'insieme dei sottoinsiemi di N. Più in particolare, se un insieme di numeri I gode di una certa proprietà P, il linguaggio $L = g^{-1}(I)$ corrispondente ad I secondo g, godrà della proprietà $P \cdot g$: cioè se $I = \{x \in N \mid P(x)\}$, allora $L = g^{-1}(I) = \{y \in W \mid P(g(y))\}$, come illustrato dal diagramma.

$$\begin{tikzcd} N \arrow[r, "P"] & I \arrow[d, "g^{-1}"] \\ W \arrow[u, "g"] \arrow[r, "P\cdot g"'] & L \end{tikzcd}$$

Se pertanto P è una proprietà decidibile, cioè è un predicato calcolabile, lo stesso vale per $P \cdot g$, che è la corrispondente proprietà sulle stringhe, in quanto composizione di funzioni calcolabili. Naturalmente certe proprietà, significative per i numeri, possono non esserle sulle stringhe: ad esempio se $A = \{a, b\}$, la corrispondenza g è tale che alla proprietà P di essere numero pari corrisponderà la proprietà $P \cdot g$ di avere b come ultimo carattere. Ma alla proprietà di essere numero primo presumibilmente non corrisponde nessuna proprietà significativa sulle stringhe.

12.3. Operazioni fondamentali sui linguaggi

Consideriamo un alfabeto A e l'insieme W delle stringhe finite su A.

Essendo i linguaggi sottoinsiemi di W, possiamo applicare ad essi tutta una serie di proprietà e operazioni definite nel campo della teoria degli insiemi.

Definizione 12.3.1. Sia L un linguaggio. Il linguaggio *complementare* $\overline{L}$ e l'insieme

$$\overline{L} = \{w \mid w \in W,\ w \notin L\}$$

Definizione 12.3.2. Dati due linguaggi L_1 e L_2, l'*unione* $(L_1 + L_2)$ e l'*intersezione* $(L_1 \cap L_2)$ dei due linguaggi sono gli insiemi:

$$L_1 + L_2 = \{w \mid w \in W,\ w \in L_1 \text{ oppure } w \in L_2\}$$
$$L_1 \cap L_2 = \{w \mid w \in W,\ w \in L_1 \text{ e } w \in L_2\}.$$

Sia ϕ il linguaggio vuoto, cioè non contenente nessuna parola (da non confondersi con il linguaggio $\{\lambda\}$ costituito dalla sola parola vuota). ϕ rappresenta l'insieme complementare di W. La classe di tutti i linguaggi su A forma un'algebra booleana che coincide, ovviamente, con l'algebra booleana dei sottoinsiemi di W.

Oltre alle operazioni booleane hanno interesse due operazioni specifiche per i linguaggi: la concatenazione e chiusura.

Definizione 12.3.3. Dati due linguaggi L_1 e L_2, la *concatenazione* $(L_1 \cdot L_2)$ è l'insieme:

$$L_1 \cdot L_2 = \{w \mid w \in W,\ w = xy,\ x \in L_1 \text{ e } y \in L_2\}$$

Definizione 12.3.4. La *chiusura* di L, che si indica con L^*, è l'insieme:

$$L^* = \sum_{n \geqslant 0} L^n$$

dove $L^0 = \{\lambda\}$ e $L^n = L \cdot L^{n-1}$ per $n \geqslant 1$.

La *chiusura positiva* di L, che si indica con L^+, è:

$$L^+ = \sum_{n \geqslant 1} L^n$$

Notare che $L^+ = LL^* = L^* \cdot L$ e che $L^* = L^+ + \{\lambda\}$.

L'operatore $*$ si chiama *stella di Kleene*. L^* si chiama chiusura di L in quanto ogni linguaggio ottenuto da L mediante unione e concatenazione è contenuto in L^*. Ad esempio:

$$L' = ((L + L^2)^3 + L)(L^2 + \{\lambda\}) = (L^3 + L^4 + L^5 + L^6 + L)(L^2 + \{\lambda\})$$

$$= L + L^3 + L^4 + L^5 + L^6 + L^7 + L^8 \subseteq L^* .$$

Notare, infine come A^k sia l'insieme delle stringhe w tali che $|w| = k$, e $W = A^*$. Per questa ragione l'insieme di tutte le stringhe finite su A viene comunemente indicato con A^*.

Esempio. Sia $L = \{a, b\}$, allora

$$\begin{aligned} L^* &= L^0 + L^1 + L^2 + \ldots\ldots \\ &= \{\lambda\} + \{a, b\} + \{aa, ab, ba, bb\} + \ldots\ldots \\ &= \{\lambda, a, b, aa, ab, ba, bb, \ldots\ldots\} \end{aligned}$$

cioè L^* è l'insieme di tutte le parole su $\{a, b\}$.

Altre proprietà tipiche dei linguaggi sono:

$L + \phi = L$	ϕ è l'elemento nullo
$L . \phi = \phi . L = \phi$	
$L . \{\lambda\} = \{\lambda\} . L = L$	$\{\lambda\}$ è l'elemento unità
$\exists L_1, L_2 (L_1 . L_2 \neq L_2 . L_1)$	la concatenazione non è commutativa
$L_1 . (L_2 . L_3) = (L_1 . L_2) . L_3$	la concatenazione è associativa
$L_1 . (L_2 + L_3) = L_1 . L_2 + L_1 . L_3$	la concatenazione è distributiva rispetto all'unione.

12.4. Rappresentazione dei linguaggi

Definiti i linguaggi su un alfabeto A come sottoinsiemi di A^* il primo problema che si pone è quello della loro rappresentazione. Se un linguaggio è formato da un numero finito di stringhe, il problema si risolve elencandole. Tuttavia i linguaggi di un certo interesse sono costituiti da un numero illimitato di stringhe e noi, ovviamente, siamo interessati ad una rappresentazione finita.

Questo problema è stato affrontato da tre punti di vista diversi:

1. *generativo-sintetico*, in cui il linguaggio è visto come l'insieme delle stringhe generate da una struttura finita detta *grammatica*;
2. *riconoscitivo-analitico*, in cui il linguaggio è visto come l'insieme delle stringhe *riconosciute*, o *accettate*, da strutture finite chiamate *automi*;
3. *algebrico*, in cui il linguaggio è visto come soluzione di un sistema di relazioni algebriche oppure come rappresentato da un'espressione algebrica.

Nei prossimi paragrafi discuteremo più diffusamente i primi due approcci mentre limiteremo l'esame del terzo a quelle classi di linguaggi che richiedono strumenti algebrici elementari.

12.5. Le grammatiche generative

Prima di dare la definizione formale di queste grammatiche dette anche di Chomsky perchè furono da lui introdotte in vista del problema della traduzione automatica, vediamo di illustrarne l'idea fondamentale con un esempio tratto dalla linguistica naturale.

Nello studio della sintassi di una lingua il termine grammatica è inteso come la descrizione dei modi di comporre frasi senza alcun riferimento alla informazione che le frasi stesse possono contenere. La specificazione della grammatica avviene mediante la definizione di un certo numero di "categorie sintattiche" (nelle nostre scuole elementari chiamate "parti del discorso") e delle relazioni fra loro esistenti.

Tramite le categorie sintattiche si può riconoscere una "struttura" in ogni frase del linguaggio. Ad esempio una *frase* della lingua italiana può avere la seguente struttura: *soggetto* seguito da *predicato*. Il *soggetto* a sua volta potrà essere una *frase nominale* cioè un *nome proprio* oppure un *articolo* seguito da un *nome comune*. Il *predicato* è un *verbo* oppure un *verbo* seguito da una *frase nominale*.

Rappresentiamo questa definizione nella forma seguente:

1. < frase > → < soggetto> < predicato>
2. <soggetto> → <frase nominale>
3. <frase nominale> → <nome proprio>
4. <frase nominale> → <articolo> <nome comune>
5. <predicato> → <verbo>
6. <predicato> → <verbo> <frase nominale>

Per completare la definizione della categoria sintattica <frase> bisogna specificare alcune categorie attraverso le parole della lingua che la costituiscono:

7. <nome proprio> → Antonio (*)
8. <nome proprio> → Giuseppe
9. <verbo> → guarda
10. <verbo> → abbaia
11. <articolo> → il
12. <articolo> → la
13. <nome comune> → cane
14. <nome comune> → tavola

A questo punto interpretiamo le definizioni delle categorie sintattiche, nella forma $\alpha \rightarrow \beta$, come delle operazioni di riscrittura che consentano la sostituzione della sequenza α con la sequenza β. Il linguaggio generato dalla grammatica data è l'insieme delle parole derivabili dal la categoria sintattica <frase> mediante un numero finito di riscritture.

Ad esempio la frase "Antonio guarda Giuseppe" è ottenuta mediante i passi:

regola usata	<frase>
1	<soggetto> <predicato>
2	<frase nominale> <predicato>
3	<nome proprio> <predicato>
7	Antonio <predicato>
6	Antonio <verbo> <frase nominale>
9	Antonio guarda <frase nominale>

(*) Antonio, Giuseppe, guarda etc. sono gli elementi costitutivi del linguaggio che nella teoria sono detti simboli dell'alfabeto mentre nella terminologia comune sono detti parole del vocabolario.

3	Antonio guarda <nome proprio>
7	Antonio guarda Giuseppe

La grammatica data consente anche la derivazione della frase "la tavola abbaia la cane" che è sintatticamente corretta anche se insensata.

Aumentando opportunamente il numero delle categorie sintattiche è possibile evitare queste inesattezze grammaticali e ottenere anche frasi sensate con strutture molto complesse. Storicamente queste grammatiche sono state introdotte per la descrizione dei linguaggi naturali ma sono risultate intrinsecamente inadeguate in quanto una teoria ragionevole dei linguaggi naturali non può prescindere dalle relazioni fra il significato delle singole parole che compongono una frase. Comunque quello che più ci interessa in questo momento é che con un sistema di regole di riscrittura del tipo visto siamo in grado di generare un linguaggio.

Osserviamo infine che una categoria sintattica in realtà non è altro che un insieme di parole del vocabolario (simboli dell'alfabeto) e cioè un linguaggio. Ad esempio la categoria sintattica <nome comune> corrisponde all'insieme dei nomi comuni, nel nostro caso {cane, tavola}. Perciò una grammatica come quella data può essere vista come una serie di relazioni tra categorie sintattiche, ovvero come una serie di relazioni tra linguaggi. Così ad esempio la regola 4

<frase nominale> → <articolo> <nome comune>

afferma che il linguaggio corrispondente a <frase nominale> è costituito dalla concatenazione di quelli corrispondenti ad <articolo> e <nome comune>, che sono {il, la} e {cane, tavola}; e quindi è {il cane, la cane, il tavola, la tavola}.

12.6. Definizione formale di grammatica a struttura di frase

Nell'esempio del paragrafo precedente sono state introdotte due classi di simboli: quelli racchiusi fra parentesi angolate che rappresentano categorie sintattiche e quelli che rappresentano le parole che costituiscono la frase quando il processo di generazione termina. Poichè i simboli della seconda classe sono gli unici presenti alla fine della gene-

razione mentre quelli della prima appaiono solo nei passi intermedi, essi sono chiamati rispettivamente simboli *terminali* e *non terminali* (o *metavariabili* o *variabili*).

Il simbolo non terminale che è usato per iniziare il processo di generazione, nel nostro esempio frase, è chiamato *simbolo iniziale.*

Il processo di generazione consiste nell'applicazione, ad ogni passo, di una regola di riscrittura che chiameremo *produzione*(*). Questo processo trasforma una stringa in un'altra fino a che o nessuna produzione può essere applicata o la stringa è costituita solo da terminali. E' importante notare che ad ogni passo la sottostringa da sostituire con una produzione può essere scelta in modo arbitrario e che solo essa è alterata dalla trasformazione.

Definizione 12.5.1. Una *grammatica* a *struttura di frase* (o *generale*) è una quadrupla $G = (V, T, S, P)$ dove:
V è un insieme finito non vuoto, il *vocabolario*, $T \subset V$ è l'insieme dei simboli terminali. $(V - T)$ è l'insieme dei simboli non terminali, $S \in (V - T)$ è il simbolo iniziale e P è un insieme finito di produzioni $\alpha \to \beta$ dove $\alpha \in V^+$ e $\beta \in V^*$(**).

Introduciamo ora due relazioni fra le stringhe su V. Esse saranno denotate rispettivamente con $\underset{G}{\to}$ e $\underset{G}{\overset{*}{\to}}$ ($\to$ e $\overset{*}{\to}$ se non ci sono ambiguità) e ci permetteranno di definire l'insieme delle stringhe generate dalla grammatica G.

Definizione 12.5.2. Una stringa γ *deriva direttamente* dalla stringa μ ($\mu \underset{G}{\to} \gamma$) o viceversa μ genera direttamente γ, se e solo se:

$$\mu = \sigma\alpha\tau, \quad \gamma = \sigma\beta\tau \quad \text{e} \quad \alpha \to \beta \in P$$

con σ e τ eventualmente vuote.

Definizione 12.5.3. Una stringa γ *deriva* da μ ($\mu \underset{G}{\overset{*}{\to}} \gamma$), o viceversa μ genera γ, se e solo se esiste una sequenza di stringhe $\omega_0, \omega_1, \ldots, \omega_n$, per $n \geqslant 0$, tale che $\mu = \omega_0$, $\gamma = \omega_n$ e per ogni $0 \leqslant i < n$, $\omega_i \underset{G}{\to} \omega_{i+1}$ con $\omega_i \in V^+$. La sequenza (ω_i) è una derivazione di γ in G. Con $\mu \underset{G}{\overset{+}{\to}} \gamma$ indicheremo una derivazione con $n > 0$.

(*) Il termine produzione è dovuto al matematico Emil Post che usò regole simili per caratterizzare l'aspetto sintattico nelle teorie logiche, cioè per descrivere il processo di dimostrazione di un teorema a partire dagli assiomi usando un numero finito di regole di deduzione.

(**) Useremo la seguente notazione: $A, B, C, \ldots, \in (V - T); \ldots, X, Y, Z \in V$; $a, b, c, \ldots, \in T$; $\ldots u, v, w, z \in T^*$; $\alpha, \beta, \gamma, \ldots, \in V^*$, $\alpha \to \beta_1 | \beta_2 | \ldots | \beta_n$ se esistono n produzioni $\alpha \to \beta_1$, $\alpha \to \beta_2, \ldots, \alpha \to \beta_n$.

Definizione 12.5.4. Il linguaggio L generato dalla grammatica G (denotato $L(G)$) è l'insieme delle stringhe sull'alfabeto T (stringhe terminali) che possono essere derivate da S in G:

$$L(G) = \{w \mid w \in T^*, \ S \underset{G}{\overset{*}{\to}} w\}$$

Definizione 12.5.5. Due grammatiche G_1 e G_2 sono equivalenti se $L(G_1) = L(G_2)$.

Esempio 12.1. Consideriamo la grammatica

$$G = (\{A, S, a, b\}, \{a, b\}, S, P) \qquad \text{con } P \text{ dato da:}$$

$$S \to aAb$$
$$aA \to aaAb$$
$$A \to \lambda$$

$S \overset{*}{\to} S$ con una derivazione di zero passi.
$S \overset{+}{\to} aabb$ e $S \overset{*}{\to} aabb$ con la derivazione:
$S \to aAb \to aaAbb \to aabb$. Si dimostra che $L(G) = \{a^n b^n \mid n \geqslant 1\}$.

Esempio 12.2. Consideriamo la grammatica $G = (\{S, A, B, a, b\}, \{a, b\}, S, P)$ con P dato da:

0. $S \to \cancel{aA}\,AB$
1. $A \to aA$
2. $A \to a$
3. $B \to Bb$
4. $B \to b$

La produzione (1) indica che l'insieme di stringhe associate a A è chiuso rispetto alla concatenazione a sinistra con "a". Poichè l'altra produzione che da informazioni sulla struttura dell'insieme A è la (2) è facile convincersi che A consiste di tutte le stringhe finite di "a": a, aa, aaa,..... Analogamente B consiste delle stringhe finite di b e il linguaggio generato dalla grammatica consiste di tutte le stringhe forma

te da una stringa di "a" seguita da una stringa di "b".

$$L(G) = \{a^n b^m \mid n, m \geqslant 1\}$$

Esempio 12.3. Consideriamo la grammatica

$$G = (\{S, A, a, b\}, \{a, b\}, S, P) \text{ con } P \text{ dato da:}$$

0. $S \to aSb$
1. $S \to ab$

Poichè l'alfabeto T è identico a quello dell'esempio precedente entrambi i linguaggi sono sottoinsiemi dell'insieme di stringhe contenenti "a" e "b". La differenza fra i due sta nel fatto che in questo caso le stringhe presentano lo stesso numero di a e b.

$$L(G) = \{a^n b^n \mid n \geqslant 1\}$$

In generale non è così semplice dare una descrizione del linguaggio generato da una grammatica. Vediamo un esempio.

Esempio 12.4. Consideriamo la grammatica

$G = (\{S, A, B, C, D, a, b\}, \{a, b\}, S, P)$ dove P è dato da:

$S \to CD$	$Ab \to bA$
$C \to aCA \mid bCB$	$Ba \to aB$
$AD \to aD$	$Bb \to bB$
$BD \to bD$	$C \to \lambda$
$Aa \to aA$	$D \to \lambda$

Un esempio di derivazione è:

$$S \to CD \to aCaD \to abCBaD \to abCaBD \to abCabD \to ababD \to abab$$

Si dimostra che $L(G) = \{ww \mid w \in T^*\}$.

Esempio 12.5. Consideriamo la grammatica

$G = (\{E, T, F, a, +, *, (,)\}, \{a, +, *, (,)\}, E, P)$ dove P è dato da:

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid a$

Un esempio di derivazione è:

$$E \to E+T \to E+T*F \to T+T*F \to T+T*(E) \to F+T*(E) \to$$
$$\to a+T*(E) \to a+T*(T) \to a+F*(T) \to$$
$$\to a+a*(F) \to a+a*(a)$$

L(G) è l'insieme delle espressioni aritmetiche esprimibili con i simboli a, +, *, (e).

Teorema 12.5.1.

Sia G una grammatica a struttura di frase. L(G) è un insieme di stringhe ricorsivamente enumerabile.

Dimostrazione. Descriviamo una procedura che genera L(G) costruendo la seguente catena induttiva di insiemi:

$$V_0 = \{S\}$$

$m > 0$, $V_m = V_{m-1} \cup \{\alpha \mid \alpha \in V^*$ e per qualche $\beta \in V_{m-1}$, $\beta \underset{G}{\to} \alpha\}$.

Ogni insieme V_m contiene tutte le stringhe derivabili da S in al più m passi e può essere costruito effettivamente applicando ciascuna produzione in tutti i modi possibili ad ogni parola di V_{m-1}. La procedura sì descritta che genera successivamente gli insiemi V_m, è effettiva ed enumera l'insieme L(G). ■

Corollario. I linguaggi generati da grammatiche a struttura di frase sono semidecibili.

Il prossimo teorema è una definitiva conferma di come due concetti apparentemente lontani fra loro come le grammatiche e le MdT siano intimamente legati.

Teorema 12.5.2.

Ogni insieme ricorsivamente enumerabile è generabile da una grammatica a struttura di frase.

Dimostrazione. Consideriamo un insieme ricorsivamente enumerabile $I \subset T^*$ e sia Z la MdT con stato finale q_F che calcola la sua funzione enumeratrice, cioè la funzione che associa ad ogni stringa di T^* una stringa di I: $I = \mathrm{cod}(f), f: T^* \to T^*$.

Costruiamo una grammatica G che generi tutte le possibili configurazioni iniziali della macchina Z delimitate da due Δ. Per fare ciò bastano le produzioni:

$$S \to \Delta\, q_0 P$$
$$P \to aP \mid \Delta \qquad \forall\, a \in T$$

A queste aggiungiamo delle produzioni che simulino i singoli passi di Z. Per simulare una quintupla del tipo:

$$q_i\ s_j\ q_{ij}\ s_{ij}\ D$$

possiamo usare le seguenti produzioni:

$$q_i\ s_j\ a \to s_{ij}\ q_{ij}\ a \qquad \text{per ogni } a \in T \text{ e}$$
$$q_i\ s_j\ \Delta \to s_{ij}\ q_{ij}\ s_0\ \Delta \qquad \text{dove } s_0 \text{ è il carattere bianco di } Z.$$

Quest'ultima produzione permette di inserire bianchi sulla destra della stringa in corrispondenza di mosse di Z sulla parte bianca del nastro. Per una quintupla del tipo:

$$q_i\ s_i\ q_{ij}\ s_{ij}\ S$$

usiamo le produzioni:

$$a\, q_i\ s_j \to q_{ij}\ a\ s_{ij} \qquad \forall\, a \in T$$
$$\Delta\, q_i\ s_j \to \Delta\, q_{ij}\ s_0\ s_{ij}$$

Aggiungiamo infine le produzioni che cancellano i due delimitatori e lo stato della stringa generata al momento in cui Z entra nello stato finale:

$$a\, q_F \to q_F\ a \qquad \forall\, a \in T$$
$$\Delta\, q_F \to \#$$
$$\#\, a \to a\, \# \qquad \forall\, a \in T$$
$$\#\, \Delta \to \lambda$$

La dimostrazione che $L(G) = I = \mathrm{cod}(f)$ si completa per induzione sul numero dei passi della computazione. ∎

Da questi teoremi e dalla esistenza di insiemi non ricorsivamente enumerabili scaturisce che esistono dei linguaggi che non sono generabili da nessun procedimento algoritmico. Siccome una grammatica è un procedimento algoritmico per generare linguaggi si conclude che esistono linguaggi non generabili da nessuna grammatica.

Nel prossimo paragrafo vedremo una gerarchia di grammatiche, e quindi dei linguaggi da esse generati, nota come la classificazione alla Chomsky: essa è ottenuta imponendo delle limitazioni alla forma delle produzioni. Vedremo, inoltre, come queste limitazioni comporteranno una variazione della potenza del formalismo.

12.7. Classificazione delle grammatiche alla Chomsky

Nella definizione di grammatica a struttura di frase non si impone nessun vincolo sulle produzioni. Le grammatiche di questo tipo sono anche chiamate grammatiche di *tipo 0.* Imponendo delle restrizioni sulla natura delle produzioni si ottengono altri tipi di grammatiche.
Una grammatica è detta *monotona* se per ogni produzione $\alpha \to \beta$, si ha che $|\alpha| \leqslant |\beta|$.

Ciò significa che ogniqualvolta si applica una produzione, la stringa risultante non è mai più corta di quella che l'ha generata.

Una grammatica è detta *dipendente dal contesto* se ogni produzione è del tipo $\gamma A \delta \to \gamma \beta \delta$, con $\gamma, \delta \in V^*$, $A \in (V - T)$: cioè A può essere sostituita da β solo se è presente nel contesto $\gamma A \delta$. $\beta \in V^+$

Una grammatica è detta *libera da contesto* o semplicemente *libera*, se ogni produzione è del tipo $A \to \beta$ con $\beta \in V^+$: cioè A può essere sostituita da β indipendentemente dal contesto in cui si trova.

Una grammatica è detta *lineare* se ogni produzione è del tipo $A \to \beta$, con $\beta \in V^+$ e β contenente al più un simbolo non terminale.

Una grammatica è detta *lineare destra* o *regolare* se ogni produzione è del tipo $A \to aB$ oppure $A \to a$. Se invece ogni produzione è del tipo $A \to Ba$ oppure $A \to a$, la grammatica è detta *lineare sinistra.*

In fig. 12.1 è mostrato il diagramma di inclusione delle varie classi di grammatiche. Notare come tutte le grammatiche ristrette sono definite in modo che il linguaggio generato non contiene la parola vuota. Si può però estendere la definizione ammettendo la produzione $S \to \lambda$, con la restrizione che il simbolo iniziale non compaia a destra di nessun' altra produzione. Si dimostra che questa modifica non altera le proprietà di queste grammatiche.

Mostriamo ora quali siano le relazioni esistenti fra i linguaggi generati da queste grammatiche.

Teorema 12.6.1. Per ogni grammatica monotona è possibile trovare una grammatica dipendente dal contesto equivalente.

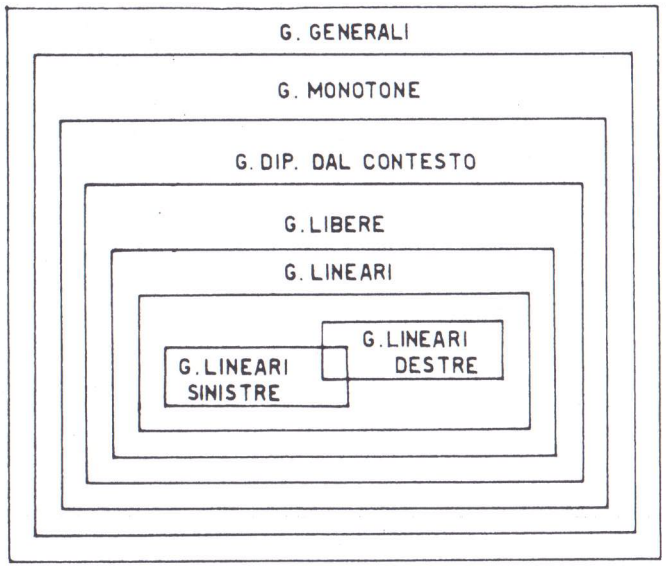


Fig. 12.1

Le grammatiche di questo tipo e i linguaggi da esse generati vengono anche rispettivamente chiamati *grammatiche* di *tipo 1* e *linguaggi di tipo 1*. Vedremo che i linguaggi di tipo 1 sono decidibili mentre il contrario non è vero, cioè esistono insiemi ricorsivi che non sono generabili da grammatiche tipo 1. Poichè abbiamo visto che i linguaggi tipo 0 sono semidecibili, possiamo concludere che l'insieme dei linguaggi di tipo 1 è strettamente contenuto nell'insieme dei linguaggi tipo 0.

Esempio 12.6. Consideriamo la grammatica monotona

$$G = (\{S, B, C, a, b, c\}, \{a, b, c\}, S, P\} \quad \text{e } P$$

costituito dalle seguenti produzioni:

$$S \to aSBc \mid abc$$
$$cB \to Bc$$
$$bB \to bb$$

Si dimostra che $L(G) = \{a^n b^n c^n \mid n \geqslant 1\}$.

Le grammatiche libere e i linguaggi da esse generati sono detti di *tipo 2*. Dimostreremo nel seguito che il linguaggio dell'esempio precedente non è di tipo 2 e quindi tale classe di linguaggi è strettamente contenuta in quella di tipo 1.

Esempio 12.7. Consideriamo la grammatica libera $G = (\{S, a, b\}, \{a, b\}, S, P\}$ e P costituito dalle seguenti produzioni

$$S \to aSb \mid ab$$

Ovviamente $L(G) = \{a^n b^n \mid n \geqslant 1\}$.

Teorema 12.6.2.

Per ogni grammatica lineare destra è possibile trovarne una lineare sinistra equivalente e viceversa. cercare prova

Queste grammatiche e i linguaggi da esse generati sono detti di *tipo 3*. Dimostreremo che il linguaggio $L = \{a^n b^n \mid n \geqslant 1\}$ generato dalla grammatica libera dell'Esempio 12.7 non è di tipo 3 e quindi tale classe di linguaggi è strettamente contenuta in quella di tipo 2.

Osserviamo infine che i linguaggi finiti sono una sottoclasse dei linguaggi tipo 3.

Esempio 12.8. Il linguaggio $L = \{a^n b^m \mid n, m \geq 1\}$ è regolare perchè generato dalla grammatica:

$G(\{S, A, B, a, b\}, \{a, b\}, S, P)$ e P costituito da:

$$S \rightarrow aA$$
$$A \rightarrow aA \mid aB$$
$$B \rightarrow bB \mid b$$

Le grammatiche lineari, infine, non hanno particolari proprietà. Esse hanno una potenza compresa fra quella delle grammatiche libere e delle grammatiche regolari.

Possiamo quindi concludere che detto L_i l'insieme dei linguaggi di tipo 1, si avrà:

$$L_0 \supset L_1 \supset L_2 \supset L_3 \supset L_{finiti}$$

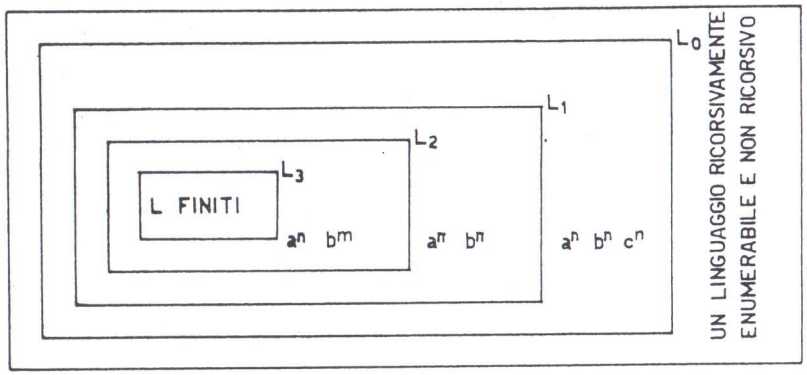


Fig. 12.2

Esempio 12.9. Una notazione molto diffusa per specificare la sintassi dei linguaggi di programmazione è la Backus Normal Form o Backus Naur Form (BNF). Essa fu introdotta per descrivere l'ALGOL 60 ed è una notazione per grammatiche libere. I simboli non terminali sono scritti racchiusi fra parentesi angolate e il segno $\rightarrow$ è rappresentato da : : = (che si legge "è sostituito da"). Modi alternativi di sostituire un non terminale sono separati da "|" (che si legge "oppure").

Usando questa notazione la grammatica dell'Esempio 12.7 diviene:

$$<S> :: = a <S> b \mid ab$$

e si legge: "il simbolo S è sostituito da a seguito da S seguito da b oppure da ab.

12.8. Alberi di derivazione e ambiguità

Assegnata una grammatica G per vedere se una parola $w \in L(G)$ bisogna trovare una derivazione del simbolo iniziale di G che generi w. In generale ad ogni passo della derivazione si possono applicare più produzioni e l'ordine in cui sono applicate per derivare w non interessa. L'aspetto importante di una derivazione è la sua struttura cioè le relazioni esistenti fra i simboli di w e le categorie sintattiche a cui essi appartengono. In particolare per le grammatiche libere, e quindi per quelle regolari che sono una sottoclasse, si può dare una rappresentazione ad albero che bene evidenzia la struttura del processo di generazione.

Data una grammatica libera $G = (V, T, S, P)$, l'*albero di derivazione* associato ad una stringa del linguaggio $L(G)$ risponderà ai seguenti requisiti:

a) ogni suo nodo ha un'etichetta che è un simbolo di V;
b) l'etichetta della radice è S;
c) ogni nodo non terminale ha un'etichetta che è in $(V - T)$;
d) ogni nodo terminale ha un'etichetta che è in T;
e) se un nodo con etichetta X ha come discendenti diretti, nell'ordine i nodi con etichetta $X_1, \ldots, X_k$ allora $X \rightarrow X_1 \ldots X_k$ è una produzione di P.

Dato un *albero di derivazione*, la parola generata è quella che si ottiene visitando l'albero in ordine anticipato tralasciando i simboli non terminali.

Esempio 12.10. Sia $G = (\{S, A, a, b\}, \{a, b\}, S, P)$ dove P contiene le seguenti produzioni:

$$S \rightarrow aAS \mid a$$
$$A \rightarrow SbA \mid SS \mid ba$$

Una derivazione della stringa aabbaa è:

$$S \to aAS \to aSbAS \to aSbAa \to aSbbaa \to aabbaa$$

Se ad ogni passo si sostituisce il simbolo non terminale più a sinistra si ha la *derivazione canonica sinistra.*

$$S \to aAS \to aSbAS \to aabAS \to aabbaS \to aabbaa$$

In modo analogo si definisce la *derivazione canonica destra*

$$S \to aAS \to aAa \to aSbAa \to aSbbaa \to aabbaa.$$

L'albero corrispondente è riportato in Fig. 12.3.

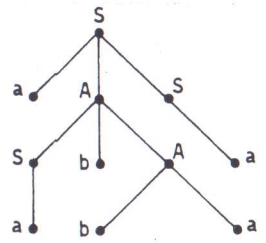


Fig. 12.3

La stringa associata alle foglie di un sottoalbero è una parola che appartiene alla categoria sintattica che etichetta la sua radice.

All'albero di derivazione di una stringa w corrispondono più derivazioni che differiscono per l'ordine in cui vengono applicate le produzioni. Poichè in una derivazione canonica sinistra l'ordine è unicamente determinato, esiste una unica derivazione canonica sinistra che corrisponde ad un dato albero di derivazione.

Definizione 12.8.1. Una grammatica libera G è *ambigua* se esiste una parola in L(G) che può essere generata con due derivazioni canoniche sinistre distinte. Un linguaggio libero L è non ambiguo se esiste una grammatica libera non ambigua che lo genera. Altrimenti L è *intrinsecamente ambiguo.*

Questi concetti si estendono agli altri tipi di grammatiche con una opportuna generalizzazione della definizione di albero di derivazione.

Esempio 12.11. Consideriamo la grammatica $G = (\{S, a, +, *\}, \{a, +, *\}, S, P)$ e P costituito da:

$$S \to S+S \mid S*S \mid a \mid b$$

Questa grammatica è ambigua perchè la stringa $a+b*a+b$ ha due derivazioni distinte:

$$S \to S+S \to a+S \to a+S*S \to a+b*S \to a+b*S+S \to a+b*a+S \to$$
$$\to a+b*a+b$$

$$S \to S*S \to S+S*S \to a+S*S \to a+b*S \to a+b*S+S \to a+b*a+S \to$$
$$\to a+b*a+b$$

Il linguaggio L(G) comunque non è intrinsecamente ambiguo perchè generato dalla seguente grammatica

$$S \to T+S \mid T$$
$$T \to T*T \mid a \mid b$$

Si dimostra che esistono linguaggi intrinsecamente ambigui.

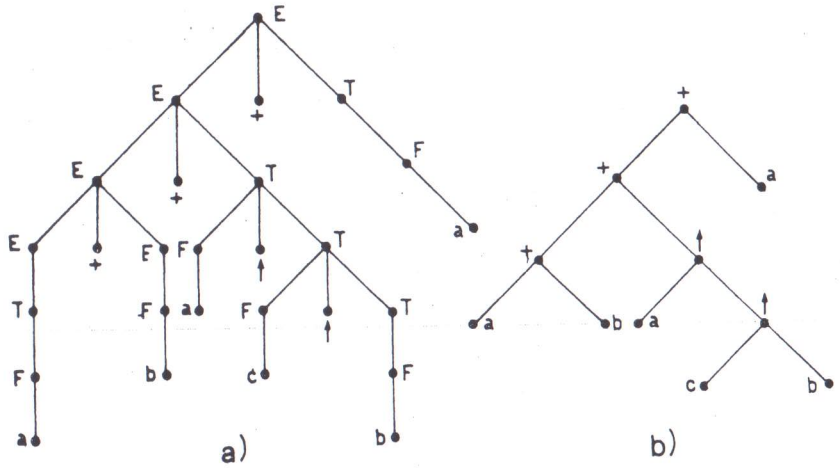
L'ambiguità non è una proprietà "interessante" nella teoria dei linguaggi formali. La sua importanza è legata essenzialmente all'uso di questa teoria nella realizzazione dei traduttori dei linguaggi di programmazione.

Esempio 12.12. Consideriamo la seguente grammatica libera che genera semplici espressioni aritmetiche con operatori + e ↑ e operandi a, b, c.

$$E \to E+T \mid T$$
$$T \to F \uparrow T \mid F$$
$$F \to a \mid b \mid c$$

In Fig. a è mostrato l'albero di derivazione corrispondente alla stringa $a+b+a\uparrow c\uparrow b+a$ e in Fig. b una versione semplificata, chiamato *albero sintattico*, che è la ben nota rappresentazione ad albero di una

espressione.



a) b)

Supponiamo assegnati ad a, b e c dei valori numerici e interpretiamo + e ↑ come la somma e l'elevamento a potenza. L'albero sintattico può essere usato per valutare l'espressione che rappresenta usando la seguente regola:

Se il nodo è una foglia con etichetta X allora il suo valore è il valore di X.

Altrimenti il suo valore è ottenuto applicando l'operatore ai valori associati ai figli sinistro e destro.

Con questa regola siamo in grado di associare un valore ad una qualsiasi stringa del linguaggio.

In altre parole si riesce a dare un "significato" ad un'espressione in base alla sua struttura espressa dall'albero di derivazione. Naturalmente si vorrebbe che questo valore coincidesse con quello che verrebbe assegnato alla stringa con le convenzioni standard dell'algebra e cioè:

a) le operazioni sono applicate in un ordine che riflette la precedenza fra gli operatori;

b) per ogni livello di precedenza deve esistere un'opportuna direzione di associatività per valutare in modo non ambiguo espressioni che coinvolgono sequenze di operatori di uguale precedenza. Ad esempio, l'espressione $a \uparrow b \uparrow c$ ha due valori diversi a seconda che venga interpretata come $((a \uparrow b) \uparrow c)$ oppure $(a \uparrow (b \uparrow c))$. Noi assumeremo per ↑ e + rispettivamente l'associatività a destra e a sinistra.

Mostriamo come la grammatica data imponga una struttura tale sulle stringhe del linguaggio che la valutazione abbia queste proprietà.

Nel nostro esempio alla radice dell'albero sintattico è associato il valore: $((a + b) + (a \uparrow (c \uparrow b))) + a$.

Entrambe le condizioni a, b sono soddisfatte, infatti:

– L'espressione $(a \uparrow c \uparrow b)$ viene interpretata come $(a \uparrow (c \uparrow b))$. Questo fatto è dovuto alla *ricorsione a destra* presente nella produzione $T \to F \uparrow T$: T che può generare altri elevamenti a potenza è alla destra di ↑ e quindi viene valutato prima. F invece è alla sinistra di ↑ e non può generare altri operatori.

– L'espressione $(a + b + a \uparrow c \uparrow b)$ viene interpretata come $((a + b) + a \uparrow c \uparrow b)$ per effetto della *ricorsione a sinistra* presente nella produzione $E \to E + T$. Se ci sono più somme adiacenti, per effetto della ricorsione a sinistra, essi vengono valutati da sinistra a destra.

– L'espressione $(a \uparrow c \uparrow b + a)$ viene interpretata come $((a \uparrow c \uparrow b) + a)$ rispecchiando così la precedenza di ↑ su +. Questa precedenza è espressa dalla produzione $E \to E + T$: prima di applicare + bisogna valutare T che può generare ↑. In generale se esiste una produzione del tipo $X \to X \rho Y$ e Y può generare l'operatore τ allora τ ha precedenza sull'operatore ρ.

12.9. Sistemi di equazioni tra linguaggi

Nel seguito vedremo come i linguaggi possano essere descritti anche come soluzioni di sistemi di equazioni. A questo scopo presentiamo qui alcune definizioni e osservazioni.

Un sistema di equazioni fra linguaggi su un alfabeto A è detto in forma normale quando è del tipo:

$$(1)\qquad \begin{array}{l} X_1 = F_1(X_1, X_2, \ldots, X_n) \\ \vdots \\ X_n = F_n(X_1, X_2, \ldots, X_n) \end{array}$$

dove le F_i sono funzioni di $U^n \to U$ con $U = 2^{A^*}$ (l'insieme dei linguaggi su A) e X_i variabili su U.

Chiameremo soluzione del sistema (1) una n-upla di linguaggi $(L_1, \ldots, L_n)$ che soddisfi le equazioni del sistema, cioè:

$$\begin{array}{l} L_1 = F_1(L_1, \ldots, L_n) \\ \vdots \\ L_n = F_n(L_1, \ldots, L_n) \end{array}$$

In generale possono esistere una, più di una o nessuna soluzione di un sistema. Ad esempio per l'equazione:

$$X = XX + \{aa\}$$

i linguaggi

$$L_k = \{a^{2n} \mid n \geqslant 0\} + \{a^{2n+k} \mid n \geqslant 0\}$$

sono tutti soluzioni per ogni valore di k.

Una soluzione $(\overline{L}_1, \overline{L}_2, \ldots, \overline{L}_n)$ si dice minima se per ogni altra soluzione $(L_1, L_2, \ldots, L_n)$ si ha che

$$\overline{L}_1 \subseteq L_1,\ \overline{L}_2 \subseteq L_2, \ldots, \overline{L}_n \subseteq L_n$$

Naturalmente la soluzione minima, se esiste, è unica. Nell'esempio precedente la soluzione minima è

$$\overline{L} = \{a^{2n} \mid n \geqslant 1\}$$

Per semplicità consideriamo ora il caso di una sola equazione in forma normale: tutto quanto diremo può essere esteso al caso generale considerando un sistema come un'unica equazione in forma vettoriale.

Una soluzione dell'equazione

$$X = F(X)$$

è dunque un punto fisso di $F : U \to U$ in quanto deve essere $L = F(L)$.

Nel caso in cui F sia una funzione continua, cioè per ogni sequenza di linguaggi

$$L_1 \subseteq L_2 \subseteq L_3 \subseteq \ldots$$

si abbia che

$$F\left(\sum_{i=1}^{\infty} L_i\right) = \sum_{i=1}^{\infty} F(L_i)$$

allora vale il teorema del punto fisso:

Teorema 12.8.1.

Se F è una funzione continua, allora il suo minimo punto fisso $\overline{L}$ esiste ed è dato da

$$\overline{L} = \sum_{i=1}^{\infty} F^i(\phi)$$

dove

$$F^i(\phi) = \underbrace{F(F(\ldots(F(\phi)\ldots))}_{i \text{ volte}}$$

Sono ad esempio funzioni continue quelle ottenute per composizione delle operazioni tra linguaggi definite precedentemente è cioè somma, concatenazione e iterazione.

Esempio 12.13. Le seguenti funzioni sono continue:

$$\begin{aligned} F_1(X) &= \{a\} + X \\ F_2(X) &= XX + \{aa\} \\ F_3(X) &= \{a\}X + X^*(\{b\} + X\{a\}X) \end{aligned}$$

Il minimo punto fisso di F_1, cioè la soluzione minima di $X = F_1(X)$ è dato da:

$$\begin{aligned} \overline{L}_1 &= \sum_{i=1}^{\infty} F_1^i(\phi) \\ &= F_1(\phi) + F_1(F_1(\phi)) + F_1(F_1(F_1(\phi))) + \ldots \\ &= \{a\} + (\{a\} + \{a\}) + (\{a\} + (\{a\} + \{a\})) + \ldots \\ &= \{a\} \end{aligned}$$

Notiamo che anche $\{a, b\}$, ad esempio, è punto fisso di F_1 in quanto:

$$F_1(\{a, b\}) = \{a\} + \{a, b\} = \{a, b\}$$

volta di una grammatica, che, come si vede, è di tipo 2:

$$G = (\{E\} \cup T, T, E, P) \quad \text{dove}$$
$$T = \{\lambda, \phi, +, *, \cdot,), (\} \cup A$$

e P è costituito dalle produzioni:

$$E \to (E + E) \mid (E \cdot E) \mid (E)^*$$
$$E \to \lambda \mid \phi \mid a_1 \mid a_2 \mid \ldots\ldots \mid a_n$$

Il linguaggio generato da questa grammatica costituisce l'insieme delle espressioni regolari sull'alfabeto A.

Esempio 13.1. Sono espressioni regolari sull'alfabeto $A = \{a, b\}$:

1) $(a \cdot b)$
2) $((a \cdot a) + b)$
3) $(\phi + ((a)^* \cdot b))$

Per semplificare la scrittura adottiamo la convenzione di dare precedenza nell'ordine agli operatori $*$, $\cdot$, $+$ in modo di evitare qualche parentesi, di sottintendere il $\cdot$ e di indicare $\underbrace{e \ldots\ldots e}_{n \text{ volte}}$ con e^n. Scriveremo quindi:

1') ab
2') $a^2 + b$
3') $\phi + a^*b$

La semantica di una espressione regolare e è il linguaggio L(e) ottenuto in base alle seguenti regole.

$$L[\phi] = \phi$$
$$L[\lambda] = \{\lambda\}$$
$$L[a_i] = \{a_i\}$$
$$L[e_1 + e_2] = L[e_1] + L[e_2]$$
$$L[e_1 \cdot e_2] = L[e_1] \cdot L[e_2]$$
$$L[e^*] = L[e]^*$$

Esempio 13.2. Per le espressioni dell'esempio precedente:

1) $L[ab] = L[a] . L[b] = \{a\} . \{b\} = \{ab\}$
2) $L[a^2 + b] = L[a^2] + L[b] = \{aa\} + \{b\} = \{aa, b\}$
3) $L[\phi + a^* b] = \phi + \{a\}^* \{b\} = \{b, ab, a^2 b, a^3 b, \ldots\ldots\}$

Sulle espressioni regolari si possono dimostrare alcune proprietà di tipo algebrico che qui riportiamo:

(a) $e_1(e_2 + e_3) = e_1 e_2 + e_1 e_3$	(g) $(e_1 + e_2) e_3 = e_1 e_3 + e_2 e_3$
(b) $e_1 + (e_2 + e_3) = (e_1 + e_2) + e_3$	(h) $\phi^* = \lambda$
(c) $e_1(e_2 e_3) = (e_1 e_2) e_3$	(i) $e^* + e = e^*$
(d) $e\lambda = \lambda e = e$	(j) $(e^*)^* = e^*$
(e) $\phi e = e\phi = \phi$	(k) $(e_1 + e_2)^* = (e_1^* e_2^*)^*$
(f) $e + e = e$	(l) $e + \phi = e$

Si osservi però che queste identità non riguardano le espressioni ma i linguaggi che esse rappresentano, analogamente a quanto avviene nel formalismo delle MdT dove si possono avere MdT diverse (sintatticamente) che calcolano la stessa funzione (e quindi "semanticamente" uguali).

13.2. Relazione tra espressioni regolari e grammatiche regolari

Data la particolare forma delle produzioni di una grammatica lineare destra, il sistema di equazioni ad esso associato risulta di tipo lineare, cioè ha la seguente forma:

$$(1) \qquad \begin{array}{l} X_1 = A_{10} + A_{11} X_1 + A_{12} X_2 + \ldots\ldots + A_{1n} X_n \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ X_n = A_{n0} + A_{n1} X_1 + A_{n2} X_2 + \ldots\ldots + A_{nn} X_n \end{array}$$

Ora ci occuperemo quindi di sistemi di questo genere cercando dapprima un metodo per risolvere l'equazione lineare:

$$(2) \qquad X = AX + B$$

dove A e B sono linguaggi noti.

La risposta ci viene data dal seguente

Teorema 13.2.1.

La soluzione minima dell'equazione (2) è data da

$$X = A^*B$$

Dimostrazione. La soluzione minima dell'equazione è il minimo punto fisso della funzione

$$F(X) = AX + B$$

e quindi è data, in base al teorema 12.8.1, da:

$$\overline{L} = \sum_{i=1}^{\infty} F^i(\phi)$$

Vediamo che:

$$F^1(\phi) = B$$
$$F^2(\phi) = F(F(\phi)) = B + AB = (\lambda + A)B$$
$$F^3(\phi) = B + A(B + AB) = B + AB + A^2B = (\lambda + A + A^2)B$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$F^i(\phi) = B + A(B + AB + \ldots\ldots + A^{i-1}B) = (\lambda + A + \ldots\ldots + A^i)B$$
$$\sum_{i=1}^{\infty} F^i(\phi) = (\lambda + A + A^2 + \ldots\ldots)\, B = A^*B$$

■

Teorema 13.2.2.

Se $\lambda \notin A$ la soluzione dell'equazione (2) è unica.

Dimostrazione. Se $\lambda \notin A$, per ogni altra soluzione L si ha che $L \subseteq A^*B = \overline{L}$ e quindi, poichè quest'ultima è quella minima, si avrà che $L = A^*B$. Infatti, se $w \in L$ e $|w| = k$; siccome abbiamo che

$$L = AL + B \qquad \text{si avrà che}$$
$$L = A(AL + B) + B = A^2L + AB + B$$

e ripetendo k volte queste sostituzioni otteniamo

$$L = A^{k+1}L + A^kB + A^{k-1}B + \ldots\ldots + AB + B$$

Se $\lambda \notin A$, allora tutte le stringhe di $A^{k+1}X$ hanno almeno lunghezza $k + 1$.

Perciò, se $w \in L$ deve essere che $w \in A^iB$ per qualche i

$k \geqslant i \geqslant 0$, e quindi $w \in A^*$ dato che $A^iB \subseteq A^*B$. ■

Se la condizione $\lambda \notin A$ non è soddisfatta, A^*B è comunque la soluzione minima (teorema 13.2.1) ma anche $L = A^*(B + C)$ è una soluzione, qualunque sia C, e quindi si hanno infinite soluzioni.

Vediamo ora come si risolve il sistema (1).

Ricaviamo X_1 dalla prima equazione sfruttando il teorema 13.2.1:

$$X_1 = A_{11}^*(A_{10} + A_{12}X_2 + \ldots\ldots + A_{1n}X_n)$$

e sostituiamo nelle successive. Quindi semplifichiamo:

$$X_2 = A_{20} + A_{21}A_{11}^*A_{10} + (A_{22} + A_{21}A_{11}^*A_{12})X_2 + \ldots\ldots + (A_{2n} + A_{21}A_{11}^*A_{1n})X_n$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$X_{11} = A_{n0} + A_{n1}A_{11}^*A_{10} + (A_{n2} + A_{n1}A_{11}^*A_{12})X_2 + \ldots\ldots + (A_{nn} + A_{n1}A_{11}^*A_{1n})X_n$$

In tal modo otteniamo un sistema di $n - 1$ equazioni in $n - 1$ variabili.

Possiamo perciò iterare il procedimento fino ad ottenere una espressione senza incognite per il linguaggio X_n. Le espressioni per $X_1, \ldots, X_{n-1}$ si ottengono poi per sostituzione(*).

I linguaggi soluzione minima di un sistema lineare sono quindi ottenuti dai linguaggi coefficienti A_{ij} per mezzo di operazioni di $+$, $.$ e $*$. Possiamo perciò affermare che

Teorema 13.2.3.

I linguaggi generati da grammatiche regolari sono rappresentabili mediante espressioni regolari.

Dimostrazione. Da quanto detto, se i linguaggi coefficienti di un sistema lineare possono essere rappresentati mediante espressioni regolari, ciò sarà vero anche per i linguaggi soluzione minima. Nel caso di un sistema associato ad una grammatica regolare questi sono o simboli terminali o ϕ, quindi tale condizione è soddisfatta. Pertanto il linguaggio

(*) Si può dimostrare (vedi Appendice) che la soluzione così ottenuta è proprio quella minima.

generato dalla grammatica, che è il linguaggio corrispondente al simbolo iniziale della grammatica nella soluzione minima del sistema a questa associato, è rappresentabile mediante una espressione regolare. ■

Osserviamo infine che se $\lambda \notin A_{ij}$, allora la soluzione del sistema lineare (1) è unica in quanto ad ogni passo del procedimento ci ritroviamo con sistemi lineari in cui i coefficienti delle variabili non contengono λ e quindi vale la condizione di unicità del teorema. Questa condizione è chiaramente soddisfatta nel caso il sistema sia quello associato ad una grammatica lineare.

Esempio 13.3. Dalla grammatica $G = (\{X, Y, a, b\}, \{a, b\}, X, P)$ dove P sono le produzioni:

$$X \to aX \mid bY \mid a$$
$$Y \to bX \mid bY \mid b$$

si ottiene il sistema:

$$X = aX + bY + a$$
$$Y = bX + bY + b$$

Ricaviamo Y dalla seconda equazione e sostituiamo nella prima:

$$Y = b^*(bX + b)$$
$$X = (a + bb^*b)X + a + bb^*b$$

Quindi abbiamo che

$$X = (a + bb^*b)^*(a + bb^*b)$$

è l'espressione che rappresenta il linguaggio generato da G.

13.3. Procedura di riconoscimento

Proponiamoci di risolvere il seguente problema: trovare una procedura che, per ogni stringa, o parola, su un assegnato alfabeto, stabilisca se essa appartiene o no ad un certo linguaggio. Come si è già accennato, è possibile risolvere questo problema ogni qualvolta il linguaggio assegnato sia generato da una grammatica dipendente dal contesto o da un'altra meno potente. Ora, comunque, vogliamo vedere più in generale che caratteristiche deve avere una tale procedura iniziando con una descrizione molto semplice.

Con un discorso molto approssimativo possiamo dire che una tale procedura deve in un qualche modo esaminare uno dopo l'altro i caratteri che formano la parola e ad ogni carattere letto modificare un "qualcosa", che chiameremo "stato", in modo che giunti alla fine della parola lo stato risultante indichi se la parola appartiene o non al linguaggio.

Per precisare un po' meglio queste cose diamo una descrizione modellistica di questa procedura di riconoscimento che chiameremo "automa". Il modello consiste di un dispositivo di controllo, con un numero finito di stati, una testina di lettura e un nastro infinito diviso in celle Fig. 13.1. La stringa di ingresso è scritta sul nastro con un carattere per ogni cella, con il carattere iniziale a sinistra, e tutte le altre celle a destra della stringa contengono il carattere bianco. All'inizio del riconoscimento, la testina è posizionata sul carattere iniziale e il controllo è nello stato iniziale.

Supponiamo che ad ogni carattere che si legge corrisponde un passo del funzionamento della procedura. La cosa più semplice che viene in mente è di prescrivere che il passo successivo ad uno considerato consista nella lettura del carattere successivo (il nastro viene esaminato da sinistra a destra) e nel passaggio ad un altro stato, determinato da una *funzione di transizione* esplicitamente assegnata e, naturalmente, definita per ogni coppia (stato, carattere). Esaminata tutta la stringa in ingresso l'automa si arresta e a seconda del suo stato decideremo se la parola appartiene o no al linguaggio.

Questa descrizione fa subito pensare alle Macchine di Turing, ma ci sono delle differenze che, come vedremo, rendono questo tipo di automa meno potente. La differenza fondamentale sta nel fatto che un automa può solo *leggere* dal nastro di ingresso e non può scriverci utilizzando cioè come una memoria esterna illimitata.

Esempio 13.4. Consideriamo un automa con quattro stati che accetta in ingresso parole sull'alfabeto $\{a, b\}$. Lo stato iniziale e finale è q_0 e la funzione di transizione è data dalla tabella in Fig. 13.2. Lo stato iniziale e quello finale sono contrassegnati rispettivamente con una freccia e con un cerchio.

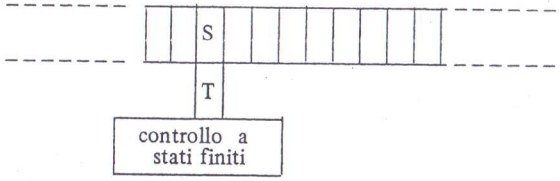


Fig. 13.1

E' facile verificare cone il linguaggio riconosciuto dall'automa sia quello costituito o dalla parola vuota o dalle parole con un numero pari di a e/o di b.

Q \ A	a	b
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Fig. 13.2

Passiamo ora a descrivere in precisi termini algebrici un automa a stati finiti e il suo funzionamento.

13.4. Automi a stati finiti deterministici

Si chiama *automa a stati finiti deterministico* (per brevità lo indicheremo con ASFD) un sistema M definito come segue

$$M = (Q, A, t, q_0, F)$$

dove:

Q è un insieme finito di stati

A è un insieme finito di caratteri che costituiscono l'alfabeto

$t: Q \times A \to Q$ è una funzione che associa ad ogni coppia (stato, carattere) uno stato.

q_0 è lo *stato iniziale* e $q_0 \in Q$

F è l'insieme degli *stati finali*, $F \subseteq Q$.

Per completare la definizione di automa come riconoscitore di parole sull'alfabeto A, consideriamo un'altra funzione $t': Q \times A^* \to Q$, definita, sull'insieme degli stati Q e sull'insieme delle parole A^*, e data ricorsivamente da:

$$t'(q, \lambda) = q$$
$$t'(q, xa) = t(t'(q, x), a) \qquad \text{con} \qquad x \in A^* \text{ e } a \in A$$

ad esempio
$$\begin{aligned} t'(q_0, abc) &= t(t'(q_0, ab), c) \\ &= t(t(t'(q_0, a), b), c) \\ &= t(t(t(t'(q_0, \lambda), a), b), c) \\ &= t(t(t(q_0, a), b), c) = q \end{aligned}$$

cioè il significato di $t'(q_0, y) = q$, con $y \in A^*$, è che l'automa inizia il suo funzionamento nello stato q_0 e con ingresso y, e si troverà nello stato q con la testina posizionata alla destra di y. Mentre con t si indica una singola transizione dell'automa, con t' si indicano tutte le transizioni effettuate dall'automa per effetto della parola in ingresso. Nel seguito, quando questo non darà luogo ad ambiguità, useremo il simbolo t per indicare entrambe le funzioni.

A questo punto possiamo dare la definizione di linguaggio riconosciuto (o accettato) da un ASFD. Sia $L \subseteq A^*$ un linguaggio sull'alfabeto A, si dice che L è *riconoscibile mediante un automa finito* quando esiste un automa $M = (Q, A, t, q_0, F)$ per cui $L = L(M) = \{x \mid x \in A^*,\ t(q_0, x) \in F\}$.

Gli automi usati come riconoscitori di linguaggi sono chiamati anche *accettori*. Notare che, poichè un automa a stati finiti dopo un numero finito di passi uguale alla lunghezza della stringa di ingresso termina *sempre* il suo funzionamento, noi saremo sempre in grado di decidere se una parola appartiene o no all'insieme L.

Quindi L è un insieme ricorsivo e l'automa realizza la sua funzione caratteristica. Nel seguito mostreremo come l'insieme dei linguaggi riconoscibili mediante automi finiti coincide con l'insieme dei linguaggi regolari su A, cioè del tipo 3.

13.5. Grafi di transizione

Di un automa si può dare una rappresentazione grafica mediante un *grafo di transizione*, simile a quella vista per le Macchine di Turing. Il contenuto della tabella di transizione si rappresenta con un grafo orientato, i nodi del quale rappresentano gli stati e gli archi orientati rappresentano le transizioni da uno stato ad un altro causate dal carattere letto. Ogni arco è marcato con il carattere che ha generato la transizione. Ad esempio l'automa considerato nell'Esempio 13.4 viene rappresentata dal grafo di transizione di Fig. 13.3. Lo stato iniziale è mostra

to con una freccia e quello finale (che in questo caso coincidono), da due cerchi. Notare che da ogni nodo escono tanti archi quanti sono i caratteri dell'alfabeto. Ogni cammino dallo stato iniziale ad uno stato finale rappresenta una parola riconosciuta dall'automa, e l'insieme di tutti i possibili cammini, di questo tipo, forniscono il linguaggio riconosciuto dall'automa.

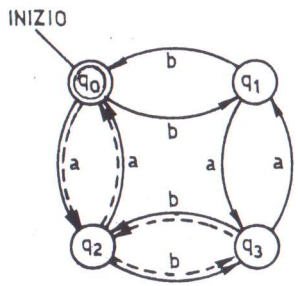


Fig. 13.3

Ad esempio, il cammino disegnato a tratti in Fig. 13.3, corrisponde alla parola abba.

13.6. Relazioni di equivalenza e automi finiti

Richiamiamo alcuni concetti dall'algebra.

Dati due insiemi A e B che possono anche coincidere, una relazione binaria R fra essi è un sottoinsieme dell'insieme prodotto A $\times$ B. Se $a \in B$ e $b \in B$ e la coppia $(a, b) \in R$, allora indicheremo questo fatto con la notazione aRb.

Una relazione binaria R su un insieme S è chiamata una *relazione di equivalenza* se sono verificate le tre seguenti condizioni:

1) *è riflessiva*, cioè $\forall s \in S$, sRs

2) *è simmetrica*, cioè $\forall s_1 s_2 \in S$, se $s_1 R s_2$ allora $s_2 R s_1$.

3) *è transitiva*, cioè $\forall s_1 s_2 s_3 \in S$, se $s_1 R s_2$ e $s_2 R s_3$ allora $s_1 R s_3$.

Sia R una relazione di equivalenza su S. Allora R divide l'insieme S in sottoinsiemi di elementi equivalenti. Questi sottoinsiemi sono chiamati *classi di equivalenza* e il loro numero è detto *indice* della relazione di equivalenza.

Una *partizione* di un insieme S è una famiglia di sottoinsiemi S_i di S tali che $\forall i, j\ S_i \cap S_j = \emptyset$ e $\bigcup_{0}^{n}{}_i S_i = S$. Cioè data una partizione di S nessun elemento appartiene a più di un sottoinsieme S_i e ogni elemento appartiene ad un S_i.

E' facile verificare che una relazione di equivalenza su S determina una partizione di S e viceversa.

Ritorniamo agli automi finiti e definiamo una relazione di equivalenza R sull'insieme A^* delle parole sull'alfabeto A: $\forall x, y \in A^*$, xRy se e solo se $t(q_0, x) = t(q_0, y)$. In altri termini due parole sull'alfabeto A sono equivalenti se portano l'automa nello stesso stato.

Come risultato immediato si ha che l'insieme A^* viene suddiviso in classe di equivalenza $S_i = \{x \mid t(q_0, x) = q_i\}$, di *indice* uguale al più al numero degli stati dell'automa, che costituiscono una partizione di A^*. Questa relazione di equivalenza su A^* gode di una importante proprietà, chiamata *invarianza a destra*, così definita:

$$\forall z \in A^* \text{ se } xRy \text{ allora } xzRyz$$

Infatti se xRy allora $t(q_0, x) = t(q_0, y)$ e quindi $t(q_0, xz) = t(t(q_0, x), z) = t(t(q_0, y), z) = t(q_0, yz)$.

Questa proprietà nel caso degli automi finiti ha una interessante interpretazione. Consideriamo l'automa rappresentante in Fig. 13.3.

Le parole x = ab e y = ba sono equivalenti perchè entrambe portano l'automa nello stato q_3. Qualunque altra parola z pensiamo concatenata ad x ed y l'automa, durante la lettura di z, "transiterà", a partire da q_3, per certi stati in modo completamente indipendente dal fatto che aveva raggiunto q_3 per effetto di x o di y. Cioè una volta raggiunto uno stato, tutta la "storia" del passato è "codificata" da quello stato e la parte di stringa di ingresso già letta non influenza più il "futuro".

Vediamo ora un teorema che da una prima caratterizzazione dei linguaggi riconoscibili da ASFD.

Teorema 13.6.1.

Un linguaggio $L \subseteq A^*$ è riconosciuto da un ASFD se e solo se L è l'unione di alcune delle classi di equivalenza di una relazione invariante destra di indice finito.

Dimostrazione. "*parte solo se*". Supponiamo che L sia riconosciuto da un ASFD: $M = \{Q, A, t, q_0, F\}$. Sia R la relazione di equivalen-

za $xRy \Longleftrightarrow t(q_0, x) = t(q_0, y)$.

Sappiamo che R è invariante a destra e determina una partizione di A^* di indice finito. Consideriamo le classi di equivalenza $S_i = \{x \mid x \in A^*, t(q_0, x) = q_i \in F\}$. La loro unione definisce l'insieme di tutte le stringhe che mandano l'automa in uno stato finale e quindi esattamente il linguaggio L.

"*parte se*". Assumiamo che L sia l'unione di alcune delle classi di equivalenza di una relazione invariante destra e mostriamo come sia possibile definire un ASFD $M = \{Q, A, t, q_0, F\}$ tale che $L = L(M)$.

L'alfabeto A di M è lo stesso su cui è definito L.

Per comodità, pensiamo associato ad ogni classe di equivalenza un simbolo q_i; l'insieme di questi simboli lo assumiamo come l'insieme Q di M. Lo stato iniziale q_0 è il simbolo associato alla classe di equivalenza $[\lambda]$ che contiene la stringa vuota. Con $[x]$ si indica la classe di equivalenza contenente la parola x.

L'insieme degli stati finali F è costituito dai simboli associati alle classi di equivalenza che contengono parole di L: $F = \{[x] \mid x \in L\}$.

La funzione di transizione è definita dalla relazione:

$$t([x], a) = [xa]$$

Il significato di questa relazione è il seguente. Per definire t occorre associare uno stato ad ogni coppia (stato, carattere). In questo caso gli stati sono rappresentati da classi di equivalenza e la t si definisce associando ad ogni coppia (classe di equivalenza, carattere) un'altra classe di equivalenza che è quella che contiene la parola xa. Le transizioni così definite non dipendono dalla parola x scelta ma dalla classe di equivalenza a cui essa appartiene. Infatti per ogni altra parola y tale che xRy, cioè $[x] \equiv [y]$, allora xaRya cioè $[xa] \equiv [ya]$ per l'invarianza destra di R.

A questo punto M è completamente specificato e $L = L(M)$ per come è stato costruito. ■

Vediamo un esempio per chiarire la seconda parte del teorema.

Esempio 13.4. Sia L un linguaggio costituito dall'insieme di parole su

$A = \{a, b\}$ che sono sequenze alternate di a e b e che iniziano per a o per b.

$$L = \{a, b, ab, ba, aba, bab, abab, \ldots\}$$

Sia R una relazione di equivalenza su A^* che determini le seguenti classi di equivalenza:

q_0 $[\lambda]$
q_1 [a, aba, ababa,.....]
q_2 [ab, abab, ababab,.....]
q_3 [b, bab, babab,.....]
q_4 [ba, bab, bababa,.....]
q_5 [bb, aa, bbba,.....]

Assumiamo che questa sia una partizione di A^* e che R sia invariante destra. E' facile verificare entrambe le cose per parole scelte a caso. Inoltre si vede anche che L coincide con l'unione della seconda, terza, quarta e quinta classe di equivalenza. Allora per il teorema dimostrato possiamo costruire l'automa che riconosce L e noi ne daremo il grafo di transizione.

Si disegnano tanti nodi quante sono le classi di equivalenza e si marcano con $[\lambda]$, [a] [ab],..... oppure per semplicità con $q_0, q_1, \ldots\ldots$ (l'automa associato ad un grafo di transizione non dipende dai simboli usati per contrassegnare i nodi).

I nodi delle classi di equivalenza "finali" si disegnano con due cerchi.

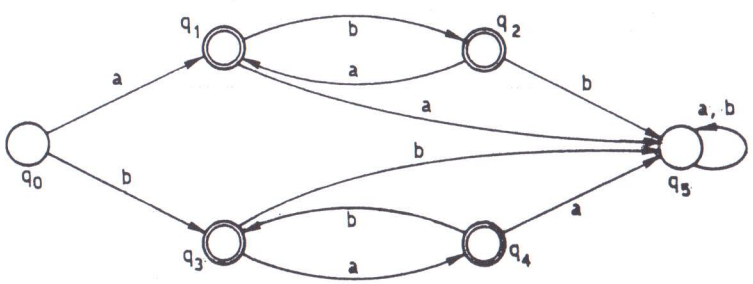


Fig. 13.4

Vediamo ora come si completa il grafo con gli archi marcati. Consideriamo ad esempio il nodo q_3 e determiniamo la transizione per gli ingressi a e b;

$$t(q_3, a) = t([b], a) = [ba] = q_4$$
$$t(q_3, b) = t([b], b) = [bb] = q_5$$

Continuando in questo modo per tutti gli altri nodi, il grafo si completa come in Fig. 13.4.

Vediamo ora un'altra relazione di equivalenza su A^* che ci consentirà di giungere ad una conclusione molto interessante.

Definizione 13.6.1. Sia $L \subseteq A^*$ ed E una relazione di equivalenza così definita: $(\forall x, \forall y, \forall z) \in A^*$, xEy se e solo se ogni qualvolta $xz \in L$ allora anche $yz \in L$ e viceversa.

Lemma. E è invariante destra.

Infatti se xEy allora $(\forall w, \forall z) \in A^*$, $xwz \in L$ se e solo se anche $ywz \in L$ e quindi xwEyw.

Teorema 13.6.2.

Se L è l'unione di alcune classi di equivalenza di una relazione R invariante destra di indice finito, allora E è di indice finito, e viceversa.

Dimostrazione. Mostriamo come ogni classe di equivalenza di R è interamente contenuta in una di E, cioè che xRy implica xEy. Assumiamo che xRy. Poichè R è invariante destra, $\forall z \in A^* \rightarrow xzRyz$ e se $xz \in L$ anche $yz \in L$, cioè xEy. Quindi E risulta di indice inferiore o uguale a quello di R.

Il viceversa è immediato perchè E è invariante destra ed $L = \Sigma [x]$ con $x \in L$.

Teorema 13.6.3.

La relazione E determina l'*automa minimo* che riconosce L.

Dimostrazione. Per il teorema 13.6.1, E determina un automa che riconosce L. Se esistesse un altro automa con meno stati, allora L sarebbe dato dall'unione di alcune classi di equivalenza di una relazione R che partizionerebbe A^* in un numero inferiore di classi di E. Questo contraddice quanto risulta nella dimostrazione del teorema 13.6.2. ■

Teorema 13.6.4.

L'automa minimo che riconosce un linguaggio L è unico (a meno dei nomi assegnati agli stati).

Dimostrazione. L'automa minimo è individuato dalla relazione E definita precedentemente. Se esistesse un altro automa, con lo stesso numero n di stati, che riconosce L, questo individuerebbe una relazione R di indice n. Le classi di R sono contenute ciascuna in una classe di E, ma siccome sono nello stesso numero, non possono che coincidere e di conseguenza anche i due automi coincidono. ■

Esempio 13.5. Applichiamo il teorema 13.6.2. per determinare l'automa minimo che riconosce il linguaggio L, insieme delle parole su $A = \{a, b\}$ che sono sequenze alternate di a e b che iniziano per a.

$$L = \{a, ab, aba, abab, \ldots\}$$

Per trovare la partizione di A^* generata dalla relazione E vediamo prima se L sia tale da poter appartenere ad un'unica classe di equivalenza. Questo non è possibile e infatti per $z = ab$, $x = a$ e $y = ab$ si ha che xEy e *non* è vero che $xz \in L$. L va perciò partizionato in più classi di equivalenza. Consideriamo una qualsiasi parola $z \in L$ e cerchiamo di capire che condizioni devono soddisfare due stringhe di L perchè

$$xz \in L \Longleftrightarrow yz \in L$$

E' facile convincersi che è sufficiente che le due parole terminino con lo stesso carattere. Quindi basta partizionare L in due classi: $\{a, aba, ababa, \ldots\}$ $\{ab, abab, \ldots\}$. La partizione di A^* che soddisfa il teorema 13.6.2. si completa poi con le classi $[\lambda]$ e [tutte le altre

stringhe che non $\in L$]. Il passaggio all'automa minimo è immediato.

Se un ASFD è assegnato con un grafo di transizione o con la relazione R', allora esiste un algoritmo per ricavare l'automa minimo equivalente: è, sostanzialmente, la procedura per minimizzare il numero degli stati di una macchina sequenziale completamente specificata(*).

Esempio 13.6. Utilizziamo il teorema 1.4.2 per dimostrare che il linguaggio $L = \{a^n b^n$ con $n > 1\}$ non è riconoscibile da un automa a stati finiti. Procediamo per assurdo; cioè supponiamo che $a^n b^n$ sia dato dall'unione di un numero finito di classi di equivalenza di una relazione E di indice finito. Essendo le possibili classi di equivalenza in cui è stato partizionato A^* in numero finito allora esisteranno due interi $n \neq m$ tali che $a^n E a^m$. Ma allora per la proprietà invariante destra di E ne segue che $a^n b^n E a^m b^n$ e questo è falso perchè $a^n b^n \in L$ e $a^m b^n \notin L$.

13.7. Automi a stati finiti e Grammatiche Regolari

Tra gli insiemi $L \subseteq A^*$ riconosciuti da un ASFD e i linguaggi generati dalle grammatiche regolari esiste una corrispondenza che è espressa dal seguente teorema:

Teorema 13.7.1.

Dato un automa a stati finiti M, esiste una grammatica regolare tale che $L(G) = L(M)$.

Dimostrazione. Assegnato un ASFD, $M = (Q, A, t, q_0, F)$ si costruisce una grammatica regolare con la seguente procedura:

1) l'insieme dei simboli terminali V_T coincide con A;

2) l'insieme dei simboli non terminali V_N coincide o è in corrispondenza biunivoca con Q;

3) il simbolo iniziale S di G è il simbolo non terminale che corrisponde a q_0.

(*) Vedi ad esempio A. Grasselli, Teoria delle Reti Logiche, ETS-Pisa.

4) l'insieme delle produzioni P è dato da:
ad ogni transizione dallo stato q_i allo stato q_j per effetto del carattere a_h si associa una produzione $N_i \to a_h N_j$, con N_i e N_j simboli non terminali corrispondenti a q_i e q_j.
Se $q_j \in F$ si aggiunge la produzione $N_i \to a_h$ (notare come il tipo di produzione sia quello delle grammatiche regolari).

Mostriamo ora come per ogni parola $x = a_1 a_2 \ldots a_n$, con $n > 1$, accettata da M esiste una derivazione dal simbolo iniziale di G che la produce. Se $x = L(M)$ allora esiste una sequenza di stati $q_0, q_1, \ldots, q_n$ tale che $t(q_0, a_1) = q_1$, $t(q_1, a_2) = q_2, \ldots, t(q_n, a_n) \in F$.
Per come è stata costruita G allora esisterà la sequenza delle produzioni $N_0 \to a N_1$, $N_1 \to a_2 N_2, \ldots, N_n \to a_n$ che costituiscono la derivazione di x in G.

In modo analogo si fa vedere che per ogni derivazione in G $N_0 \overset{*}{\to} x$ esiste una sequenza di transizioni dell'automa tali che $t(q_0, x) \in F$.

Se l'automa riconosce anche la parola vuota, allora la grammatica che si ottiene va modificata per tenerne conto. ■

Esempio 13.7. Ricaviamo la grammatica regolare equivalente all'automa descritto dal grafo di transizione di Fig. 13.5:

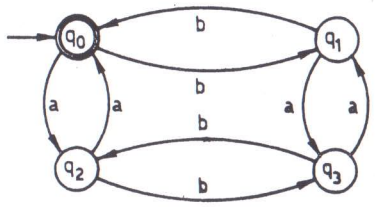


Fig. 13.5

$N_0 \to aN_2$	$N_2 \to bN_3$
$N_0 \to bN_1$	$N_2 \to aN_0$
$N_1 \to bN_0$	$N_2 \to a$
$N_1 \to b$	$N_3 \to aN_1$
$N_1 \to aN_3$	$N_3 \to bN_2$

Poichè l'automa riconosce anche la parola vuota occorre aggiungere le produzioni $\bar{N}_0 \to \lambda$, $\bar{N}_0 \to aN_2$, $\bar{N}_0 \to bN_1$ e assumere $\bar{N}_0$ come simbolo iniziale.

Per generare il passaggio inverso, grammatica regolare-automa finito, si potrebbe pensare di "invertire" la procedura vista. Ad esempio se la grammatica contiene una produzione del tipo $N_1 \to aN_2$ allora si introduce una transizione $t(q_1, a) = q_2$. Ma supponiamo di avere in G anche la produzione $N_1 \to aN_1$. Se si introducesse la transizione $t(q_1, a) = q_1$ avremmo però un caso di transizione non univocamente determinata. Una situazione di questo tipo veniva esclusa nel caso delle macchine di Turing per evitare il funzionamento non deterministico.

Ora invece vogliamo proprio indagare sulle proprietà presentate da un automa a stati finiti quando questa restrizione viene abbandonata. Vedremo che con il non determinismo non si aumenta la capacità riconoscitiva dell'automa ma questo concetto ci permetterà di dimostrare più facilmente il passaggio grammatica regolare-ASFD.

13.8. Automi a stati finiti non deterministici

Un *automa a stati finiti non deterministico* (per brevità ASFND) è un sistema M definito come segue:

$$M = (Q, A, t, q_0, F)$$

dove

Q, A, q_0, F sono definiti come per gli automi finiti

$t: Q \times A \to 2^Q$ con 2^Q insieme potenza di Q o insieme di tutti i sottoinsiemi di Q.

In questo modo ad ogni transizione non è associato un unico stato ma un insieme finito di stati, quindi se $t(q_0, a) = \{q_1, q_2, q_3\}$ questo significa che la macchina nello stato q_0 se legge a può transitare in uno di tre possibili stati.

La funzione di transizione si estende al dominio $Q \times A^*$ definendo:

$$t'(q, \lambda) = \{q\} \quad e \quad t'(q, xa) = \bigcup_{p \in t'(q,x)} t(p, a)$$

e al dominio $2^Q \times A^*$ con

$$t''(\{p_1, p_2 \ldots\ldots p_k\}, x) = \bigcup_{i=1}^{k} t'(p_i, x)$$

Nel seguito indicheremo una qualsiasi di queste funzioni di transizioni con t se dal contesto sarà chiaro a quale delle tre si fa riferimento.

La definizione di parola riconosciuta da questo tipo di automa è: M riconosce una parola x se esiste almeno una sequenza di stati che inizia con q_0 e termini con uno stato finale e tale che le singole transizioni siano consistenti con la parola letta e la descrizione dell'automa.

Il linguaggio riconosciuto da M è l'insieme:

$$L(M) = \{x \mid p \in Q, \text{ tale che } p \in t(q_0, x) \wedge p \in F\}$$

E' opportuno far notare che un ASFND sebbene abbia più alternative ad ogni passo, non usa nessun meccanismo probabilistico per decidere la transizione. Invece di fare *una* scelta di tipo probabilistico che potrebbe portare al *non* riconoscimento di una parola di L, l'automa analizza tutte le possibili transizioni. Se almeno una di esse lo porta in uno stato finale, la parola è riconosciuta. Infine si ricordi che mentre allo ASFD corrisponde un modello fisico che è la rete sequenziale a quello non deterministico non corrisponde nessuna specifica realizzazione e il suo interesse è solo teorico.

Esempio 13.8. Un modo per visualizzare cosa accade quando un automa non deterministico analizza una parola è quello di ricorrere ad una rappresentazione ad albero che chiameremo *albero delle alternative.* La radice di questo albero è marcata con q_0. Ogni nodo ha tanti figli quanti sono gli stati raggiungibili, dallo stato ad esso associato, per effetto di una semplice transizione. Ad esempio, per l'automa dato dalla tabella di Fig. 13.6.a, se la parola da riconoscere è aabb, l'albero che ne risulta dopo aver letto a è riportato in Fig. 13.6.b. A questo punto si potrebbe pensare di procedere in due modi diversi per completare l'analisi della parola.

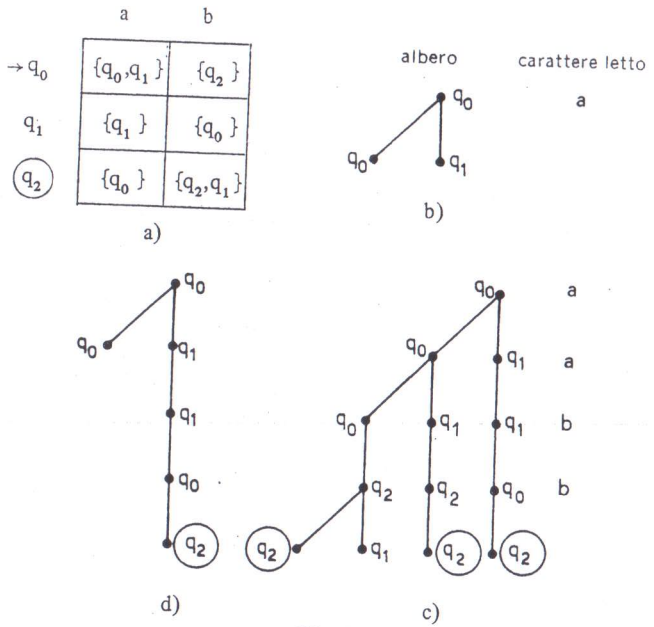


Fig. 13.6

1) *in parallelo*: per ogni successivo carattere letto si aggiungono i figli di tutti i nodi generati. Dopo aver letto l'ultimo carattere la parola è riconosciuta se fra i nodi terminali dell'albero ce n'è almeno uno marcato con uno stato finale. Fig. 13.6.c.

2) *in profondità*: per ogni successivo carattere letto si sviluppa l'albero considerando solo un nodo fra quelli generati, ad esempio, quello più a destra (Fig. 13.6.d). Dopo aver letto l'ultimo carattere la parola è riconosciuta se fra i nodi terminali di questa parte dell'albero ce n'è almeno uno marcato con uno stato finale. Altrimenti si "risale" nell'albero al primo nodo padre con non tutti i figli esaminati, se ne sceglie uno e si prosegue come sopra.

Esempio 13.9. Un altro modo per visualizzare il funzionamento di un ASFND è quello di immaginare che esistano più copie dello stesso automa. Si inizia ad esaminare la parola di ingresso con un automa solo. Ogni qualvolta sono possibili più di una transizione, si creano tanti au-

tomi quante sono le alternative presenti ed ognuno di essi viene inizializzato in uno stato diverso. Invece, se non è possibile nessuna transizione, l'automa viene soppresso. Da questo istante in poi tutti gli automi attivi proseguono leggendo i rimanenti caratteri della parola. Perciò ad ogni passo esiste un *insieme* di automi tutti in stati diversi (non serve tenere più copie di automi nello stesso stato, in quanto il loro comportamento da quel momento in poi sarà uguale). La parola sarà riconosciuta se alla fine almeno una delle macchine è in uno stato finale.

Esempio 13.10. Anche il funzionamento di un automa non deterministico può essere descritto con un grafo delle transizioni. In questo caso però da un nodo possono uscire più archi marcati con lo stesso simbolo o può addirittura mancare l'arco per altri simboli (Fig. 13.7). Entrambi questi fatti scaturiscono dalla definizione della funzione di transizione.

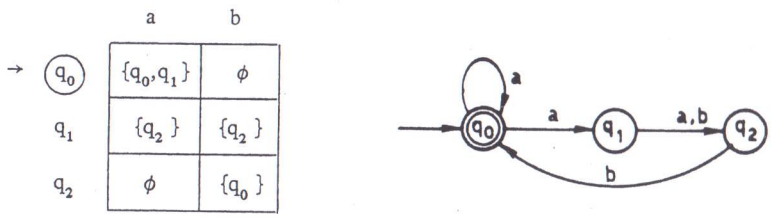


Fig. 13.7

Le transizioni del tipo $t(q_0, b) = \phi$ vanno pensate come una condizione di arresto. Naturalmente per decidere se la parola è riconosciuta o no, vanno considerate anche altre alternative. Ad esempio con la parola abb l'automa si arresta con la sequenza degli stati $q_0 q_0$, mentre la riconosce con q_0, q_1, q_2, q_0.

Viene naturale chiedersi se gli automi a stati finiti non deterministici siano più potenti di quelli deterministici, cioè se esistono dei linguaggi riconoscibili da un tipo di automa e non dall'altro. La risposta è data dal seguente teorema.

Teorema 13.8.1.

Per ogni ASFND, N, ne esiste uno equivalente deterministico, D,

cioè tale che $L(N) = L(D)$.

Dimostrazione. Dato N, definiamo D come segue:

1) Gli stati Q_D rappresentano tutti i possibili sottoinsiemi di Q_N. Indicheremo con $[q_0, q_1, \ldots, q_k]$ lo stato corrispondente all'insieme $\{q_0, q_1, \ldots, q_k\}$.

2) Gli stati finali F_D sono tutti i sottoinsiemi di Q_N che contengono almeno un elemento appartiene ad uno stato finale di N.

3) Lo stato iniziale è quello che corrisponde all'insieme $\{q_0\}$ con q_0 stato iniziale di N.

4) La funzione di transizione è data da:

$$t_D([q_0, \ldots, q_i], a) = [t_N(q_0, a) \cup t_N(q_1, a) \cup \ldots \cup t_N(q_i, a)]$$

con a il generico simbolo dell'alfabeto.

5) L'alfabeto è identico a quello di N.

Rimane da dimostrare che D riconosce tutte le parole riconosciute da N e viceversa.

Per come è stata definita t_D, ad ogni passo l'automa D tiene conto di tutte le possibili transizioni di N. Inoltre D raggiunge uno stato finale ogni qualvolta esiste almeno una sequenza di transizioni in N che porta ad uno stato finale.

Quindi D riconosce tutte le parole riconosciute da N.

Viceversa assumiamo che $x = a, \ldots, a_n \in L(D)$ allora esiste una sequenza di stati che inizia con $[q_0]$ e termina con $[q_m \ldots]$ che contiene un $q \in F_N$. Ogni altro stato $[q_i \ldots q_k]$ nella sequenza, per come è stata definita t_D, contiene un q_h tale che esiste una sequenza di transizioni di N che inizia con q_0 e termina con q_h nello stesso numero di passi occorrenti a D per raggiungere $[q_i \ldots q_k]$. Per induzione sulla sequenza degli stati se D riconosce x esiste una sequenza di passi di N che porta al riconoscimento della stessa parola. ■

Esempio 13.11. La procedura descritta può generare una tabella che contiene stati non raggiungibili dallo stato iniziale per nessuna sequenza di ingresso, come, ad esempio, gli stati ϕ e $[q_1]$ in Fig. 13.8.

Questi stati non raggiungibili possono essere eliminati esplorando

	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_2\}$
q_1	$\{q_2\}$	$\{q_0\}$
q_2	$\{q_0\}$	$\{q_2, q_1\}$

ASFND

a)

	a	b
ϕ	ϕ	ϕ
→ $[q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_1]$	$[q_1]$	$[q_0]$
$[q_2]$	$[q_0]$	$[q_1, q_2]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$
$q_0, q_1, q_2]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$

ASFD

b)

Fig. 13.8

la tabella a partire dallo stato iniziale $[q_0]$ e marcando le righe corrispondenti agli stati da esso raggiungibili. Si passa poi al successivo stato marcato e si prosegue. Alla fine di questo processo gli stati non marcati si possono eliminare. Questa osservazione suggerisce una procedura per ottenere direttamente la tabella semplificata:

1) Si inizia la tabella che stiamo costruendo con lo stato $[q_0]$;

2) Si completa la riga con gli stati raggiungibili con una transizione elementare. Ogni insieme di stati successori, se non compare già, si aggiunge alla colonna degli stati;

3) Si completa la riga successiva come al passo 2 e si continua fino a che vengono aggiunte nuove righe.

13.9. Equivalenza fra automi non deterministici e espressioni regolari

In questo paragrafo completiamo le dimostrazioni dell'equivalenza fra tutti e quattro i modi possibili di rappresentare un linguaggio tipo 3 mostrando come si costruisce l'ASFND che riconosce lo stesso linguag-

gio definito da una espressione regolare.

Diamo prima la seguente definizione:

Definizione 13.9.1. Un ASFND è *semplice* se e solo se ha un unico stato iniziale q_I, un unico stato finale q_F, con $q_I \neq q_F$, e tale che non esistano nè transizioni che portano in q_I nè transizioni che partano da q_F (Fig. 13.9).

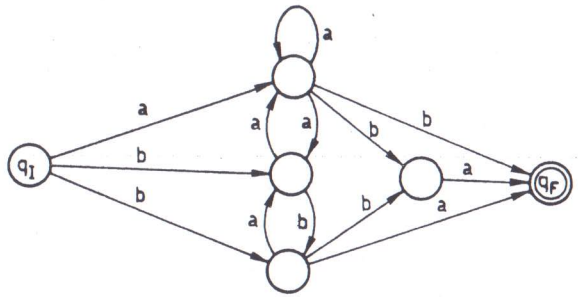


Fig. 13.9

Teorema 13.9.1.

Sia $M = (Q, A, t, q_0, F)$ un ASFND. Allora esiste un ASFND semplice M' tale che $L(M) = L(M')$.

Dimostrazione. Definiamo l'automa $M' = (Q', A, t', q_I, q_F)$ come segue:

1) $Q' = Q \cup \{q_I, q_F\}$ con $q_I, q_F \notin Q$, $q_I \neq q_F$
2) Per ogni transizione da q_0:

a) Se $t(q_0, a_i) \not\subseteq F$ allora

$$t'(q_I, a_i) = t(q_0, a_i)$$

Pensando al grafo di transizione questa condizione significa che se dal nodo iniziale non si transita direttamente in uno stato finale, allora il nodo q_I viene collegato a tutti i nodi successori diretti di q_0:

b) Se la condizione precedente non è verificata,

$$t'(q_I, a_i) = t(q_0, a_i) \cup \{q_F\}$$

c) Se $q_0 \in F$ si aggiunge la transizione:

$$t'(q_I, \lambda) = \{q_F\}$$

d) Per ogni $q_i \in \overline{Q}$ diverso da q_I e q_F

$$t'(q_i, a_i) = t(q_i, a_i) \quad \text{se} \quad t(q_i, a_i) \not\subseteq F$$

e) Tutte le transizioni che portano in uno stato finale vengono "sdoppiate":

$$t'(q_i, a_i) = t(q_i, a_i) \cup \{q_F\} \quad \text{se} \quad t(q_i, a_i) \subseteq F$$

f) Dal nuovo stato finale non ci sono archi uscenti: $t'(q_F, a_i) = \phi$.

Evidentemente Q' è un ASFND semplice per come è stato costruito. Per dimostrare l'equivalenza dei due automi bisogna far vedere che se $x \in L(M)$ allora $x \in L(M')$ e viceversa. Se $x = a_1 a_2 \ldots a_n \in L(M)$ allora esiste in M una sequenza di stati $q_0, q_1 \ldots q_n \in F$ e $q_i \in t(q_{i-1}, a_{i-1})$, $1 \leqslant i \leqslant n$.

A questa sequenza ne corrisponde una di M' data da $q_I, q_1, \ldots, q_{n-1}, q_F$ per come è stato costruito M'. Quindi $x \in L(M')$ e in modo analogo si dimostra il viceversa. ■

Esempio 13.12.

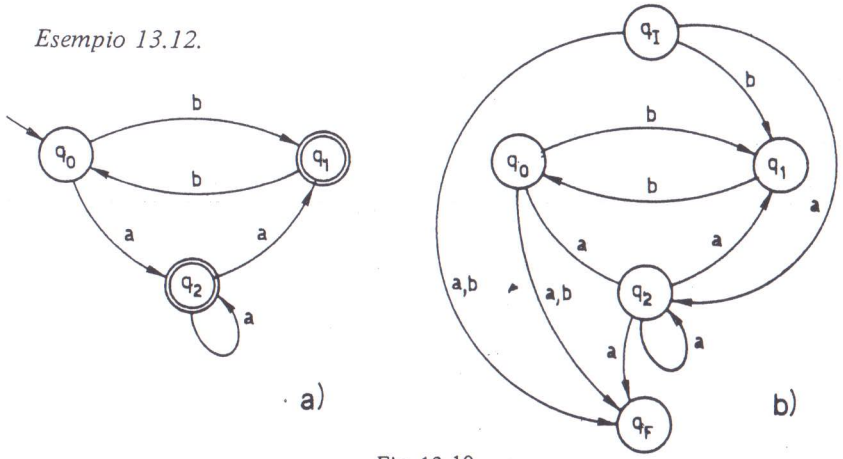


Fig. 13.10

10

Osservazione. Questo metodo, per la regola c, può condurre a grafi di trandizione con archi marcati da λ. Queste transizioni sono comode da usarsi per rendere il grafo semplice ma si possono eliminare facilmente. Se da un nodo q_i parte un arco marcato con λ ad un nodo q_j, allora questa transizione si può eliminare aggiungendo a q_i tutte le transizioni da q_j. Inoltre se q_j è uno stato finale anche q_i lo diventa (Fig. 13.11).

Queste regole si giustificano in base alle eguaglianze

$$t(q_i, \lambda\, a_i) = t(t(q_i, \lambda), a_i) = t(q_j, a_i)$$

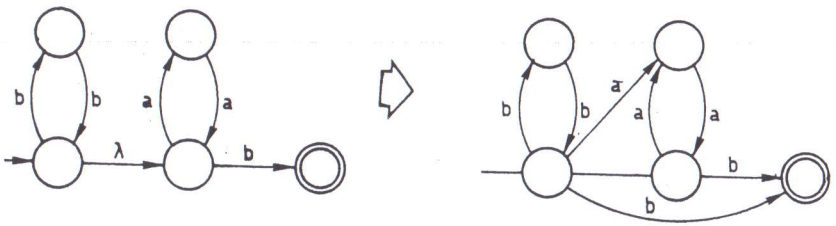


Fig. 13.11

Il prossimo teorema ci fornisce un metodo per costruire un ASFND da un'espressione regolare.

Teorema 13.9.2.

Se α è un'espressione regolare allora esiste un ASFND, N, tale che L(N) è l'insieme di parole rappresentate da α.

Dimostrazione. Dimostriamo il teorema mostrando come si costruisce N.

Se α è o un simbolo a_i dell'alfabeto o λ oppure Φ i corrispondenti grafi sono riportati in Fig. 13.12.

Fig. 13.12

Supponiamo di avere gli ASFND semplici N_1 e N_2 corrispondenti alle espressioni regolari β e γ.

E' immediato verificare che gli automi, per $\beta+\gamma$, $\beta\cdot\gamma$ e β^* sono quelli riportati in Fig. 13.13. Notare come l'automa corrispondente a β^* non è semplice ma questo non è richiesto dal teorema. ■

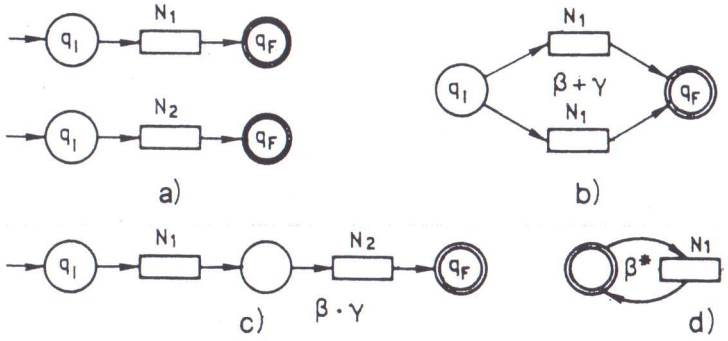


Fig. 13.13

Esempio 13.13. Trovare l'ASFND che riconosce il linguaggio definito da:

$$\alpha = ((ab + c^*)b + a)\,.$$

In Fig. 13.14.a sono riportati i grafi, tutti semplici, di a, b, c.

Le espressioni ab e c^* determinano i grafi di Fig. 13.14.b. Poichè il grafo di c^* non è semplice per ottenere il grafo di $ab + c^*$ si fa prima la trasformazione riportata in Fig. 13.14.c. I passaggi successivi sono mostrati nelle altre figure.

Abbiamo finora visto quattro modi diversi per rappresentare i linguaggi tipo L_3: con grammatiche regolari, con espressioni regolari, con ASFND deterministici e ASFD con automi deterministici.

Inoltre abbiamo dato degli algoritmi per passare da un tipo di rappresentazione ad un'altra come è sintetizzato dal diagramma di Fig. 13.15. Fra tutte queste rappresentazioni quella con automi deterministici offre un ulteriore vantaggio che è quello di avere una rappresentazione "minima" che dipende univocamente dal linguaggio che si vuole rappresentare. Questo risultato, come vedremo nel prossimo paragrafo, sarà molto utile per decidere se due qualsiasi rappresentazioni sono equivalenti, cioè se

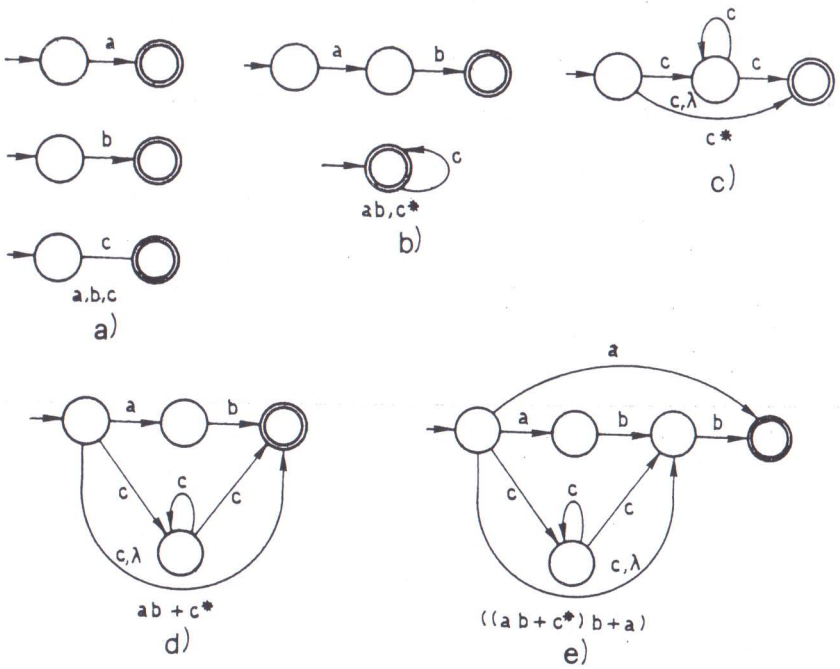


Fig. 13.14

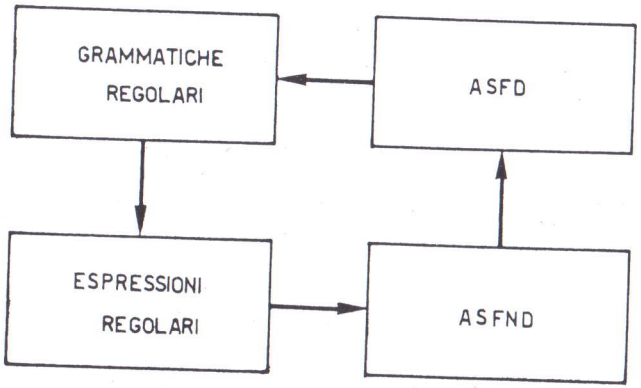


Fig. 13.15

rappresentano lo stesso linguaggio: basta trovare i due automi minimi e vedere se sono identici a meno dei nomi usati per rappresentare gli stati.

13.10. Proprietà decidibili dei linguaggi regolari

Per ogni linguaggio si possono formulare un certo numero di domande di interesse generale che non sempre hanno una risposta.

In questo paragrafo esamineremo quelle più significative nel caso dei linguaggi regolari.

Una caratterizzazione algebrica della classe di questi linguaggi è data dal seguente teorema:

Teorema 13.10.1.

La classe dei linguaggi regolari è la più piccola famiglia dei sottoinsiemi di A^* che contiene gli insiemi finiti ed è chiusa rispetto alle operazioni di unione, prodotto (o concatenazione) e stella di Kleene (o iterazione).

Questo risultato deriva immediatamente dal fatto che tali linguaggi sono quelli rappresentabili con espressioni regolari.

Consideriamo ora anche l'operazione di complementazione:

$$\overline{L} = \{y \mid y \in A^* \wedge y \notin L\}$$

Se L è un linguaggio regolare, il suo complementare è anche regolare? La risposta è data dal seguente teorema:

Teorema 13.10.2.

Se L è un linguaggio regolare, allora anche $\overline{L}$ è regolare.

Dimostrazione. Per ipotesi L è accettato da un automa a stati finiti $M = (Q, A, t, q_0, F)$. L'insieme delle parole $y \in \overline{L}$ è tale che $\forall y \in A^*$, $t(q_0, y) \notin F$ e quindi tale che $t(q_0, y) \in Q - F$. Ne segue che $\overline{L}$ è accettato dall'automa $M' = (Q, A, t, q_0, Q - F)$ e quindi è un linguaggio regolare. ■

Un'altra operazione definita sui linguaggi è l'intersezione:

$$L_1 \cap L_2 = \{z \mid z \in L_1 \wedge z \in L_2\}$$

Teorema 13.10.3.

Se L_1 e L_2 sono linguaggi regolari, anche l'intersezione $L_1 \cap L_2$ è un linguaggio regolare.

La dimostrazione scaturisce immediatamente dal fatto che i linguaggi regolari per il teorema 1 e 2, sono chiusi rispetto alle operazioni di unione e complementazione e dalla ben nota relazione $L_1 \cap L_2 = \overline{L_1} + \overline{L_2}$.

Infine poichè anche l'insieme vuoto Φ e A^* sono insiemi regolari ne segue l'importante corollario:

Corollario. La classe dei linguaggi regolari costituisce un'algebra booleana di insiemi.

Supponiamo che ci venga assegnato un automa a stati finiti M senza ulteriori specificazioni. E' possibile decidere se il linguaggio riconosciuto dall'automa è non vuoto?

E se L(M) è non vuoto, è possibile decidere se è finito o infinito?

Le risposte sono date dai due teoremi che seguono.

Teorema 13.10.4.

Sia M un automa a stati finiti. Allora L(M) è non vuoto se e solo se M accetta una parola di lunghezza inferiore al numero n degli stati di M.

Dimostrazione. La parte "se" del teorema è ovvia, vediamo quella del "solo se". Assumiamo per assurdo che $x \in L(M)$, con $|x| \geqslant n$ e che $|x| \leqslant |y|$, $\forall y \in L(M)$. Cioè x è la parola di lunghezza minima di L.

Poichè $|x| \geqslant n$, esisteranno tre parole y, z, w con $0 < |z| \leqslant n$, tali che $x = yzw$, e $t(q_0, y) = t(q_0, yz)$. L'ultima condizione si verifica perchè M ha n stati e quindi per stringhe x più lunghe di n, l'automa passerà per uno stesso stato due volte e z è la stringa tra due successivi passaggi per quello stato. Ora sia $\overline{x} = yw$. Ovviamente $|\overline{x}| < |x|$ e $t(q_0, x) = t(q_0, yzw) = t(t(q_0, yz), w) = t(t(q_0, y) = t(t(q_0, y), w) = t(q_0, \overline{x})$.

Quindi anche $\overline{x} \in L(M)$ contraddicendo l'ipotesi che x è la parola più corta in L. ■

Come immediata conseguenza di questo teorema si ha che esiste una procedura effettiva per decidere se, per un assegnato automa, il linguaggio da esso riconosciuto è vuoto oppure no: basta provare con tutte le parole lunghe al più n.

Teorema 13.10.5.

Sia M un automa a stati finiti con n stati e sia $x \in L(M)$, $|x| \geqslant n$. Allora esistono tre parole y, z, w tali che $x = yzw$, con $0 < |z| \leqslant n$, e $\forall m \geqslant 0$, $yz^m w \in L(M)$.

Dimostrazione. Abbiamo già visto nella dimostrazione del precedente teorema che $t(q_0, y) = t(q_0, yz)$.

Prendiamo come ipotesi di induzione che $t(q_0, y) = t(q_0, yz^m)$ e mostriamo che è vero anche $t(q_0, y) = t(q_0, yz^{m+1})$

$$\begin{aligned} t(q_0, y) &= t(q_0, yz) \\ &= t(t(q_0, y), z) \\ &= t(t(q_0, yz^m), z) = t(q_0, yz^{m+1}) \end{aligned}$$

usando questo risultato:

$$\begin{aligned} t(q_0, x) &= t(q_0, yzw) \\ &= t(t(q_0, yz), w) \\ &= t(t(q_0, yz^m), w) = t(q_0, yz^m w) . \end{aligned}$$ ■

Esempio 13.14. Usiamo questo teorema per dimostrare che L = $L = \{a^n b^n \mid n \geqslant 1\}$ non è regolare. Supponiamo che un automa a stati finiti con N stati lo riconosca. Allora $a^N b^N$ può essere scritto come

yzw, con $0 < |z| \leqslant N$, e tale che $\forall\, m \geqslant 0$ $yz^m w \in L$. Se z contiene solo a o solo b, allora $yw = yz^0 w \notin L$. Se z contiene a e b allora $yzzw \notin L$. Quindi L non può essere regolare.

Abbiamo così scoperto che se M accetta una parola $|x| \geqslant n$, allora $L(M)$ è infinito. Questo risultato però non basta per impostare una procedura effettiva per decidere se $L(M)$ è infinito. Quello che ci occorre però è dato dal teorema che segue (Rabin-Scott (1959)):

Teorema 13.10.6.

Sia M un automa a stati finiti con n stati. Allora $L(M)$ è infinito se e solo se esiste una parola $x \in L(M)$, tale che $n \leqslant |x| < 2n$.

Dimostrazione. La parte del "se" discende dal teorema precedente. Mostriamo come se L è infinito allora esiste una parola $x \in L$ con $n \leqslant |x| < 2n$.

Se L è infinito, contiene parole di lunghezza arbitraria.

Sia w la parola più corta maggiore o uguale di n. Se $|w| < 2n$ l'asserto è dimostrato, altrimenti scomponiamo w come nel teorema precedente: $|xyz| \geqslant 2n$, $0 < |y| \leqslant n$. Siccome in base allo stesso teorema anche $xz \in L$ si ha che $|xz| \geqslant n$ contraddicendo l'ipotesi di aver scelto w come la parola più corta. ∎

Infine consideriamo il problema dell'equivalenza fra due qualunque rappresentazioni dei linguaggi tipo 3. Ad esempio in termini di grammatiche che li generano il problema può essere così formulato: dato due grammatiche regolari, è decidibile se esse generano lo stesso linguaggio? La risposta è affermativa: basta costruire i due automi minimi corrispondenti e controllare se essi sono identici a meno dei nomi usati per gli stati.

13.11. Complessità dell'algoritmo di analisi

Per tutti i riconoscitori di linguaggi è molto interessante sapere qualcosa sulla complessità dell'*algoritmo di riconoscimento* (o di *analisi*

o *parsing* che è il termine inglese corrispondente).

Assumeremo come criterio di valutazione della complessità il numero dei passi impiegato dall'algoritmo e cercheremo, se esiste, una relazione fra questa grandezza e la lunghezza della stringa in ingresso nel caso generale.

Per i linguaggi regolari il problema ha una semplice soluzione. Infatti un qualunque algoritmo di riconoscimento sarà una procedura che simula il modello dell'automa deterministico a stati finiti e quindi riconoscerà una qualsiasi parola in un numero di passi proporzionale al numero delle transizioni prodotte dalla parola in ingresso e quindi alla lunghezza della parola. Vedremo come passando a linguaggi di tipo superiore questa semplice relazione di proporzionalità verrà, in generale, a mancare.

Problemi

1) Procedendo come nell'esempio 1.3., trovare l'automa minimo che riconosce il linguaggio su $A = \{a, b\}$ che contiene tutte le sequenze alternate di a e b che iniziano per a o per b.

2) Descrivere un metodo per minimizzare un ASFD assegnato con un grafo di transizione o con la relazione R ed applicarlo all'automa dell'esempio 1.2.

3) Sia L il linguaggio regolare rappresentato dalla seguente espressione regolare: $((ab)^* + b(bb)^*)$.
Individuare la relazione di equivalenza E tra le stringhe su $\{a, b\}^*$ e costruire l'ASFD minimo che riconosce L.

4) Dimostrare che $L = \{a^n b^n ;\ n, m \geqslant 1\}$ è riconoscibile da un ASFD e costruire l'automa minimo.

5) Dimostrare che i seguenti linguaggi non sono riconoscibili da automi a stati finiti:

$$L_1 = \{a^n\, ba^n : n \geqslant 1\}$$
$$L_2 = \{a^n : n \text{ primo}\}$$

6) Dare un algoritmo per passare direttamente da una espressione regolare ad una grammatica lineare destra ed applicarlo a:

$$(ab + bc)\,(b + c)^*$$

(Trovare prima le grammatiche che generano i singoli simboli dell'alfabeto, λ e Φ. Dare poi le regole per combinare le grammatiche associate a due espressioni regolari α_1, α_2 per trovare la G associata a $\alpha_1 + \alpha_2$, $\alpha_1 . \alpha_2$, α_1^*).

7) a) Dare un algoritmo per passare direttamente da una grammatica regolare all'ASFND equivalente ed applicarlo all'esempio che segue:

$V = \{S, A, B, a, b\}$	$S \to bA$	$A \to a$
	$S \to bB$	$A \to b$
	$A \to aA$	$A \to bS$
	$A \to aB$	$B \to aA$
		$B \to a$

b) Trovare l'ASFD equivalente.

c) Trovare un'altra grammatica per L dalla descrizione dell'ASFD.

8) Sia L il linguaggio costituito dalle stringhe legali di parentesi, e cioè ad esempio da (() ()) o da (()) (), ma non da ()) (().

Caratterizzare le classi di equivalenza della relazione M tra stringhe su $V = \{(,)\}$ definita come segue:

$$\alpha M \beta \longleftrightarrow \forall \gamma (\alpha\gamma \in L \Longleftrightarrow \beta\gamma \in L) .$$

Dimostrare quindi che L non è un linguaggio regolare.

9) La *derivata* di una espressione regolare α rispetto ad un simbolo dell'alfabeto a si indica con $D_a\alpha$ ed è definita come segue:

$$D_a\alpha = \{x \mid ax \in \alpha\}$$

cioè è un'altra espressione regolare $\overline{\alpha}$ tale che per ogni parola $x \in A^*$ da essa rappresentata, $a . x$ è rappresentata da α.

Esempio $\alpha = (ab + bc)\,(b + c)^* = ab . (b + c)^* + bc . (b + c)^*$

$$D_a\alpha = b . (b + c)^* \qquad D_b\alpha = c . (b + c)^* \qquad D_c\alpha = \Phi$$

La derivata ha le seguenti proprietà:

$D_a a = \lambda$

$D_a t = \Phi$ con $t = \lambda, \Phi, \forall a_i \in A \wedge a_i \neq a$

$D_a(\alpha + \beta) = D_a\alpha + D_a\beta$

$D_a(\alpha . \beta) = (D_a\alpha) \cdot \beta + \delta(\alpha) \cdot (D_a\beta)$

$\delta(\alpha) = \lambda$ se α rappresenta anche la parola vuota altrimenti vale Φ.

$D_a\alpha^* = (D_a\alpha)\,\alpha^*$

$D_\lambda\alpha = \alpha$

$D_{a_1 \ldots\ldots a_k}\,\alpha = D_{a_k}(D_{a_1 \ldots\ldots a_{k-1}}\,\alpha)$

Si dimostra che assegnata una espr. regolare α su $A = \{a_1 \ldots\ldots a_k\}$, la sequenza delle derivate:

$$D_{a_1}\alpha,\ D_{a_2}\alpha, \ldots\ldots D_{a_k}\alpha,\ D_{a_1 a_2}\alpha, \ldots\ldots, D_{a_1 a_k}\alpha,\ D_{a_2 a_1}\alpha, \ldots\ldots D_{a_2 a_k}\alpha, \ldots\ldots$$
$$\ldots\ldots, D_{a_k a_k}\alpha,\ D_{a_1 a_2 a_3}\alpha, \ldots\ldots, D_{\underbrace{a_k a_k \ldots a_k}_{k \text{ volte}}}\alpha .$$

presenta un numero finito di espr. regolari $\neq \Phi$.

Utilizzare questo risultato per trovare la grammatica regolare equivalente alle espressioni regolari a^*b, $(ab + bc)\,(b + c)^*$.

[Suggerimento: una produzione di una grammatica regolare può essere interpretata come segue: se i simboli non terminali si pensano associati ad espressioni regolari allora $N_i \to a_j N_k$ significa che le parole rappresentate da N_i sono ottenibili concatenando a_j a quelle rappresentate da N_k.

Quindi ad N_k è associata l'espressione regolare ottenibile da quella associata ad N_i per derivazione rispetto ad a_j. In modo analogo si procede per le produzioni che hanno a sinistra N_k.

Trovare il criterio per generare le produzioni terminali].

14 LINGUAGGI LIBERI DAL CONTESTO

1. *Grammatiche libere e sistemi di equazioni*
2. *Proprietà decidibili dei linguaggi liberi*
3. *Problemi indecidibili per i linguaggi liberi*
4. *Semplificazione delle GL. Forme Normali*
5. *Proprietà di chiusura dei linguaggi liberi*
6. *Automi a pila*
7. *Automi a pila non deterministici e linguaggi liberi*
8. *Complessità dell'algoritmo di analisi*

Tra le classi di grammatiche nella classificazione alla Chomsky, le grammatiche libere dal contesto (GL) sono le più importanti per le applicazioni ai linguaggi programmativi. E' ben noto che i programmi scritti nei linguaggi come il FORTRAN, ALGOL, PL1, ecc., per essere eseguiti da un calcolatore devono prima essere tradotti nel inguaggio macchina. Questo processo di traduzione viene effettuato da un programma chiamato *compilatore*, mediante una fase di riconoscimento delle istruzioni e una fase di generazione del codice. Quest'ultima utilizza la descrizione strutturale delle istruzioni ottenuta durante la fase di riconoscimento che viene così ad assumere un ruolo di particolare importanza per tutto il processo di traduzione.

I primi linguaggi di programmazione venivano specificati con regole ad hoc e questo si rifletteva nella definizione del riconoscitore che risultava specifico per il linguaggio. Quando invece si iniziarono a descrivere le istruzioni con una grammatica formale, si poterono sviluppare, in modo sistematico, procedure di riconoscimento basate sull'uso della definizione sintattica e si intravide la possibilità di organizzare dei riconoscitori che avessero in ingresso sia la descrizione della grammatica del linguaggio che le frasi da riconoscere e che producessero la strut-

tura sintattico delle frasi corrette.

Tali riconoscitori sono, in un certo senso, indipendenti dal linguaggio: l'algoritmo di analisi non è specifico per un linguaggio ma per il tipo di grammatica che lo descrive.

Nella pratica, comunque, la fase di riconoscimento dei compilatori non è mai così generale, ma è invece orientata verso quell'unico linguaggio per cui è previsto il compilatore. E' ancora lontano il giorno in cui si avrà in commercio un compilatore adatto a tradurre una intera classe di linguaggi.

Il primo esempio di uso di una grammatica libera per la definizione di un linguaggio di programmazione si è avuto con l'ALGOL 60. Purtroppo tale tipo di grammatica non è sufficiente per specificare tutta la struttura sintattica di un qualunque linguaggio che, come l'ALGOL, contenga una parte dichiarativa che impone delle restrizioni sui tipi di operandi che si possono usare in certi contesti. Tuttavia nella pratica ci si può ricondurre al caso dei linguaggi liberi con l'ausilio di tecniche opportune.

Prima di vedere le caratteristiche della procedura di riconoscimento studiamo alcune proprietà di questi linguaggi.

14.1. Grammatiche libere e sistemi di equazioni

Il sistema di equazioni associato ad una GL risulta essere in forma normale. Per risolvere sistemi di questo tipo non esiste un metodo semplice come quello visto per le GR che consenta di ottenere la soluzione in una forma esplicita. Descriveremo perciò solamente dei metodi iterativi che forniscono approssimazioni successive della soluzione minima: essi vengono chiamati metodo "dall'alto al basso" e metodo "dal basso all'alto".

Il metodo "dall'alto al basso" è una semplice riedizione del meccanismo di generazione del linguaggio definito da una GL G. Esso consiste nel ricavare successivamente nuovi sistemi ottenuti a partire da quello originario sostituendo ciascuna variabile con la corrispondente espressione.

Formalmente ciò corrisponde a formare, a partire dal sistema

$$\left|\begin{array}{l} X_1 = F_1(X_1, \ldots\ldots, X_n) \\ \ldots\ldots\ldots\ldots\ldots \\ X_n = F_n(X_1, \ldots\ldots, X_n) \end{array}\right.$$

la successione così definita:

$$\left|\begin{array}{l} X_1^0 = X_1 \\ \ldots\ldots\ldots \\ X_n^0 = X_n \end{array}\right.$$

e per induzione

$$\left|\begin{array}{l} X_1^{i+1} = F_1(X_1^i, \ldots\ldots, X_n^i) \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ X_n^{i+1} = F_n(X_1^i, \ldots\ldots, X_n^i) \end{array}\right.$$

Ad esempio partendo dal sistema

$$\left|\begin{array}{l} X_1 = a + X_1 X_2 \\ X_2 = b + aX_2 \end{array}\right.$$

Si ha:

$$\left|\begin{array}{l} X_1^1 = a + X_1 X_2 \\ X_1^1 = b + aX_2 \end{array}\right.$$

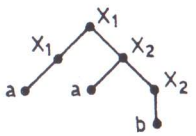
$$\left|\begin{array}{l} X_1^2 = a + (a+X_1 X_2)(b+aX_2) = a+ab+X_1 X_2 b+a^2 X_2 + X_1 X_2 aX_2 \\ X_2^2 = b+a(b+aX_2) = b+ab+a^2 X_2 \end{array}\right.$$

$$\left|\begin{array}{l} X_1^3 = a+ab+(a+X_1 X_2)(b+aX_2)b+a^2(b+aX_2)+(a+X_1 X_2)(b+aX_2)a(b+aX_2) \\ \quad = a+ab+ab^2+a^2 b+abab+\ldots\ldots \\ X_2^3 = b+ab+a^2(b+aX_2) = b+ab+a^2 b+a^3 X_2 \end{array}\right.$$

E' facile convincersi che i termini costanti che appaiono nell'equazione per X_i^k sono esattamente quelle stringhe derivabili da X_i in G con un albero di derivazione non più profondo di $k^{(*)}$. Nel nostro esempio il linguaggio X_1 contiene una stringa, a, derivabile con albero profondo uno; una stringa, ab, derivabile con albero profondo due;

(*) Per profondità di un albero si intende la lunghezza massima dei cammini dalla radice alle foglie.

tre stringhe, ab^2, a^2b e abab derivabili con albero profondo tre, ecc. Ad esempio, l'albero di derivazione di a^2b, così come si ricava dalla storia delle sostituzioni, è mostrato in figura



E' quindi chiaro che in questo modo ogni stringa di X_i apparirà, da un certo punto in poi, nella parte costante di tutte le successive equazioni per X_i.

Nel caso invece in cui si vogliano generare solo le stringhe di X_i di lunghezza minore o uguale a k, basta cancellare, dopo ogni iterazione; tutti i termini (costanti o meno) di lunghezza maggiore di k dai quali non si possono derivare stringhe lunghe meno di k. In tal caso, appena il metodo converge, cioè l'ultima equazione generata è uguale precedente, i termini costanti forniscono il linguaggio desiderato(*). Ad esempio, nel nostro caso, per X_1 e per k = 4 abbiamo:

$$X_1^1 = a+X_1X_2$$
$$X_1^2 = a+(a+X_1X_2)(b+aX_2) = a+ab+X_1X_2b+a^2X_2+X_1X_2aX_2$$
$$X_1^3 = a+ab+ab^2+X_1X_2b^2+a^2X_2b+a^2b+a^3X_2+abab$$
$$X_1^4 = a+ab+ab^2+ab^3+a^2b^2+a^2b+a^3b+abab$$

Il metodo iterativo "dal baso all'alto" è quello che si ricava dalla espressione stessa del minimo punto fisso del teorema:

$$\bar{L} = \sum_{i=0}^{\infty} F^i(\phi)$$

per cui ϕ, $F(\phi)$, $F(F(\phi))$ ecc. costituiscono approssimazioni della soluzione minima.

Nel caso di un sistema

$$\left| \begin{array}{l} X_1 = F_1(X_1,\ldots\ldots,X_n) \\ \ldots\ldots\ldots\ldots\ldots \\ X_n = F_n(X_1,\ldots\ldots,X_n) \end{array} \right.$$

(*) Nel caso la grammatica contenga produzioni del tipo $X \rightarrow Y$, non sempre il metodo converge, tuttavia, come si vedrà più avanti, si può facilmente ricavare una grammatica equivalente che non contenga produzioni di tale tipo.

La prima approssimazione sarà $(\phi,\ldots\ldots,\phi)$, la seconda $(F_1(\phi,\ldots\ldots,\phi), F_2(\phi,\ldots\ldots,\phi),\ldots\ldots,F_n(\phi,\ldots\ldots,\phi))$, la terza e le successive si ricavano sostituendo nelle equazioni le approssimazioni del livello precedente. Nell'esempio di prima:

$$\left| \begin{array}{l} X_1^0 = \phi \\ X_2^0 = \phi \end{array} \right.$$

$$\left| \begin{array}{l} X_1^1 = a+X_1^0X_2^0 = a \\ X_2^1 = b+aX_2^0 = b \end{array} \right.$$

$$\left| \begin{array}{l} X_1^2 = a+X_1^1+X_2^1 = a+ab \\ X_2^2 = b+aX_2^1 = b+ab \end{array} \right.$$

$$\ldots\ldots\ldots\ldots\ldots$$

$$\left| \begin{array}{l} X_1^{k+1} = a+X_1^kX_2^k \\ X_2^{k+1} = b+aX_2^k \end{array} \right.$$

Anche qui è facile convincersi che X_i^k è il sottoinsieme di X_i composto esattamente dalle stringhe con albero di derivazione di profondità non superiore a k. Soltanto che, mentre con il primo metodo iterativo gli alberi di derivazione venivano fatti crescere "verso il basso", cioè espandendo le foglie, qui gli alberi di derivazione vengono fatti crescere "verso l'alto", cioè congiungendo più alberi alla radice.

Pure con il metodo "dal basso verso l'alto" si possono generare tutte le stringhe di lunghezza non maggiore di k, eliminando ogni volta le stringhe troppo lunghe. Ad esempio, per k = 4, abbiamo ancora:

$$\left| \begin{array}{l} X_1^0 = \phi \\ X_2^0 = \phi \end{array} \right.$$

$$\left| \begin{array}{l} X_1^1 = a \\ X_2^1 = b \end{array} \right.$$

$$\left| \begin{array}{l} X_1^2 = a+ab \\ X_2^2 = b+ab \end{array} \right.$$

$$\begin{cases} X_1^3 = a+(a+ab)\,(b+ab) = a+ab+ab^2+a^2b+abab \\ X_2^3 = b+a(b+ab) = b+ab+a^2b \end{cases}$$

$$\begin{cases} X_1^4 = a+ab+ab^2+ab^3+a^2b^2+a^2b+abab+a^3b \\ X_2^4 = b+ab+a^2b+a^3b \end{cases}$$

$$\begin{cases} X_1^5 = X_1^4 \\ X_2^5 = X_2^4 \end{cases}$$

14.2. Proprietà decidibili dei linguaggi liberi

Dimostriamo alcuni teoremi che caratterizzano le principali proprietà decidibili dei linguaggi liberi.

Teorema 14.2.1.

E' decidibile se il linguaggio generato da una GL è vuoto.

Dimostrazione. Sia $G = (V, T, S, P)$ una GL. Definiamo la seguente catena induttiva di insiemi:

$$T_0 = T$$
$$T_i = T_{i-1} \cup \{A \mid A \in V,\ x \in T_{i-1}^*,\ A \to x\}$$

Cioè T_1 è costituito dai terminali e da quei non terminali che generano direttamente parole su T; T_2 contiene T_1 più quei non terminali che generano parole su T in due passi, e così via. Se n è il numero dei simboli non terminali di G, poichè ad ogni passo aggiungiamo all'insieme precedente dei simboli non terminali, ne segue che: a) la catena T_i è superiormente limitata da T_n; b) $T_n = T_{n+k}$ ($k = 1, 2, \ldots$); e c) in T_n ci saranno tutti i simboli non terminali che generano parole non nulle su T^*.

Per la definizione di linguaggio generato da G:

$$x \in L \iff (S \overset{*}{\to} x,\ x \in T^*) \iff (\exists\, m:\ S \in T_m) \iff S \in T_n$$

e quindi L è vuoto se e solo se $S \notin T_n$.

Siccome la costruzione degli insiemi T_i si può realmente effettuare guardando le produzioni di G, il procedimento di decisione descritto è algoritmo. Di qui l'asserto per la tesi di Curch. ■

Esempio 14.1. La grammatica $G = (\{E, T, F\} \cup V, E, P)$ dove $V = \{a, +, *, (,)\}$ e P costituito dalle produzioni:

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid a$$

genera espressioni algebriche con $+$, $*$ e parentesi, sull'unica variabile a. Applichiamo a G il metodo del teorema:

$$T_0 = \{a, +, *, (,)\}$$
$$T_1 = \{a, +, *, (,), F\}$$
$$T_2 = \{a, +, *, (,), F, T\}$$
$$T_3 = \{a, +, *, (,), F, T, E\}$$
$$T_4 = T_3$$

Siccome $E \in T_4$ allora $L(G) \neq \phi$.

Teorema 14.2.2.

E' decidibile se una parola appartiene ad un linguaggio generato da una GL.

Dimostrazione. Poichè una GL è anche una grammatica monotona, il teorema discende come corollario dall'analogo teorema per i linguaggi di tipo 1 che vedremo nel seguito. ■

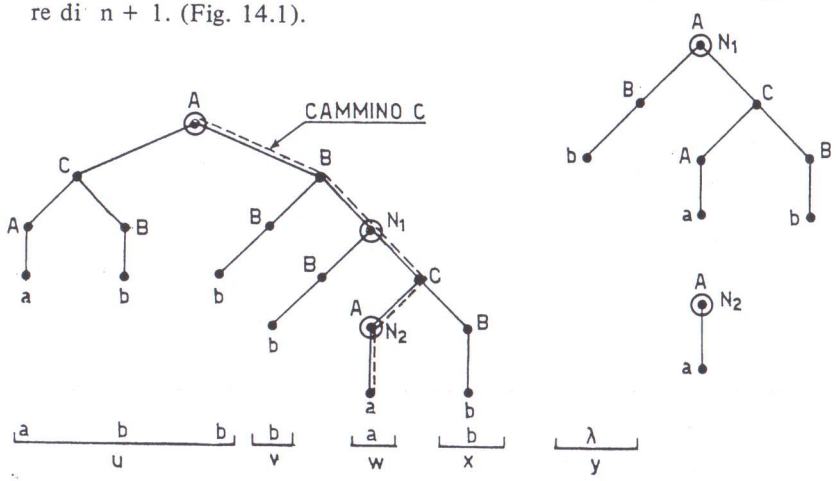
Il teorema che segue è molto importante per decidere se un linguaggio è generabile da una GL perchè impone una struttura sulle parole che appartengono ad un linguaggio libero. Esso è simile a quello visto per i linguaggi regolari che afferma che per ogni linguaggio regolare L esiste un intero n tale che una qualsiasi parola $x \in L$ con $|x| > n$ può essere decomposta in $x = yzw$ tale che $z \neq \lambda$ e ogni altra parola $yz^iw \in L$, per $i \geqslant 0$.

Teorema 14.2.3. ("uvwxy")

Sia L un linguaggio generato da una GL.

Esiste una costante p, dipendente solo da L, tale che se esiste una parola $z \in L$, con $|z| > p$, allora z può essere scritta come $z = uvwxy$, dove $|vwx| \leqslant p$ e $vx \neq \lambda$, in modo tale che $uv^iwx^iy \in L$, $i = 0, 1, 2, \ldots$.

Dimostrazione. Sia $L = L(G)$ con $G = (V, T, S, P)$ e sia $n = \#(V\text{-}T)$. Consideriamo tutti gli alberi di derivazione di G con un qualunque simbolo non terminale come radice e con profondità minore o uguale di $n + 1$. L'insieme di tali alberi è finito. Indichiamo con p la lunghezza massima delle parole ottenibili da tali alberi. Allora, se esiste una parola $z \in L(G)$ con $|z| > p$, questa può essere generata solo con alberi di profondità maggiore di $n + 1$. Consideriamo quindi l'albero di derivazione per z col minimo numero di nodi e su questo consideriamo un cammino C dalla radice alle foglie di lunghezza maggiore di $n + 1$. (Fig. 14.1).



$G = (\{A, B, C, a, b\}, \{a, b\}, A, \{A \to CB, A \to a, B \to BA, B \to b, C \to AB\})$

Fig. 14.1 - Albero di derivazione di abbbab e sottoalberi di radice N_1 ed N_2

In C ci devono essere due nodi, N_1 ed N_2, tali che:

1) N_1 ed N_2 hanno la stessa etichetta, diciamo A;

2) N_2 è discendente di N_1;

3) La porzione del cammino C da N_1 alla foglia è di lunghezza al più $n + 1$.

Ciò si verifica poichè se si considerano gli ultimi $n + 2$ nodi di C, l'ultimo ha per etichetta un terminale, gli altri $n + 1$ invece simboli non terminali e quindi almeno un simbolo deve essere ripetuto. Ora il sottoalbero con radice N_2 rappresenta la derivazione $A \overset{*}{\to} w$, quello con radice N_1 la derivazione $A \overset{*}{\to} vAx \overset{*}{\to} vwx$, mentre tutto l'albero rappresenta $S \overset{*}{\to} uAy \overset{*}{\to} uvAxy \overset{*}{\to} uvwxy$. Perciò se si sostituisce il sottoalbero con radice N_1 a quello con radice N_2, si ottiene ancora un albero di derivazione, che rappresenta $S \overset{*}{\to} uAy \overset{*}{\to} uvAxy \overset{*}{\to} \overset{*}{\to} uvvAxxy \overset{*}{\to} uv^2wx^2y$ (vedi Fig. 14.2).

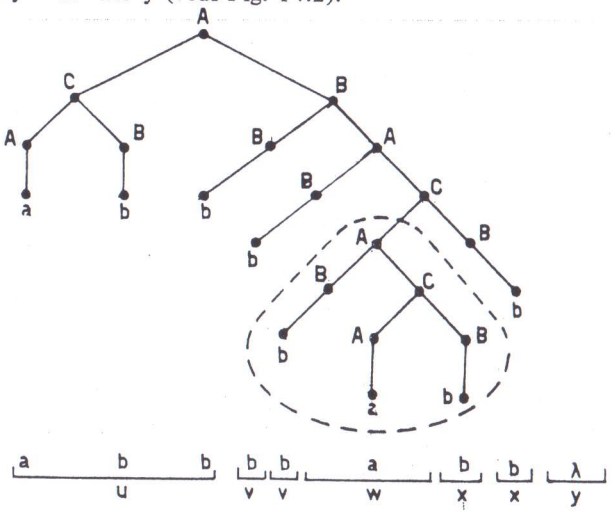


Fig. 14.2 - Albero di derivazione di uv^2wx^2y

Ripetendo il processo si può ottenere $S \overset{*}{\to} uAy \overset{*}{\to} uvAxy \overset{*}{\to} \ldots$ $\ldots \overset{*}{\to} uv^iAx^iy$ per ogni $i \geqslant 0$, e di conseguenza che $uv^iwx^iy \in L(G)$.

Siccome il sottoalbero con radice N_1 ha profondità minore o uguale ad $n + 1$, risulta che $|vwx| \leqslant p$. Se fosse $vx = \lambda$, sostituendo il sottoalbero con radice N_2 al posto di quello con radice N_1, si ottiene ancora un albero di derivazione per z con un numero di nodi inferiore contraddicendo l'ipotesi di aver scelto quello col minore numero di nodi. ∎

Come applicazione di questo teorema dimostriamo che:

Teorema 14.2.4.

Il linguaggio $L = \{a^n b^n c^n \mid n \geqslant 1\}$ non è libero.

Dimostrazione. Supponiamo per assurdo che L sia libero e sia p la costante del teorema precedente. Consideriamo $z = a^p b^p c^p$. Questa senz'altro ha lunghezza maggiore di p, perciò dovrebbe essere possibile decomporla nella sequenza: $z = uvwxy$. Siccome però $|vwx| \leqslant p$, vwx non può contenere sia "a" che "c", quindi o u contiene a^p o y contiene c^p. Perciò uwy, che appartiene a L in base al teorema (per $i = 0$), contiene p "a" oppure p "c". Tuttavia non contiene p volte ciascuno dei tre simboli, in quanto $|uwy| < 3p$, e quindi non stà in L. ■

Usando lo stesso risultato si risolve il problema di decidere se un linguaggio libero è finito o meno.

Teorema 14.2.5.

E' decidibile se il linguaggio generato da una GL è finito.

Dimostrazione. Sia p la costante del teorema 2.1.3 per il linguaggio L. Se L contiene una parola z con $|z| > p$, allora è infinito: infatti se $z = uvwxy$, tutte le parole uv^iwx^iy appartengono ad L e sono diverse tra loro in quanto $vx \neq \lambda$. Facciamo vedere ora che se L è infinito, contiene parole di lunghezza ℓ con $p < \ell \leqslant 2p$.

Difatti contiene parole di lunghezza arbitraria e quindi anche maggiori di p. Tra queste prendiamo la più breve, z. Se $|z| \leqslant 2p$ l'asserto è dimostrato, altrimenti scomponiamo z come nel teorema 2.1.3. Avremo che $|uvwxy| > 2p$ e $|vwx| \leqslant p$. Siccome in base allo stesso teorema $uwy \in L$, si ha che $|z| = |uvwxy| > |uwy| > p$, contraddicendo il fatto di aver scelto z come la più breve.

In definitiva L è infinito se e solo se contiene parole di lunghezza ℓ con $p < \ell \leqslant 2p$. Siccome è decidibile se una parola appartiene ad un dato linguaggio libero, un algoritmo per stabilire se L è finito o meno consiste nel verificare se, tra tutte le parole di lunghezza compresa tra p e 2p, ve ne sia una appartenente ad L. ■

Riportiamo qui un altro risultato sulla decidibilità di alcuni problemi, che servirà per ottenere delle GL in forma semplificata.

Teorema 14.2.6.

Sia $G = (V, T, S, P)$ una GL. Sono decidibili, per ogni $A \in (V - T)$, e $\gamma \in V^*$ le seguenti relazioni:

a) $A \overset{*}{\rightarrow} \gamma$

b) $\exists\, \alpha, \beta \in V^*$, tali che $A \overset{*}{\rightarrow} \alpha\, x\, \beta$.

Il primo risultato è una semplice generalizzazione del teorema 14.2.2. Il secondo si dimostra costruendo la catena induttiva di insiemi che al passo i contengono tutte le parole σ, con $|\sigma| \leqslant |\gamma|$, tali che $A \overset{*}{\rightarrow} \rho\, \sigma\, \delta$ in meno di i passi di derivazione, per qualche ρ, $\delta \in V^*$.

Come casi particolari risultano decidibili:

– se $A \overset{*}{\rightarrow} B$, con $B \in (V - T)$

– se $S \overset{*}{\rightarrow} \alpha A \beta$.

Chiusura algebrica di linguaggi liberi

14.3. Problemi indecidibili per i linguaggi liberi

Anche se diversi fatti sono decidibili per i linguaggi liberi, molti altri restano indecidibili. Ne diamo qui alcuni esempi.

Teorema 14.3.1.

Dati due linguaggi liberi L_1 ed L_2, i problemi di determinare se:

a) $L_1 \cap L_2 = \phi$;

b) $L_1 \cap L_2$ è infinito;

c) $L_1 \cap L_2$ è un linguaggio libero;

d) $L_1 \cap L_2$ è un linguaggio regolare;

e) $L_1 \subseteq L_2$;
f) $L_1 = L_2$ (problema dell'equivalenza);

sono indecidibili.

Teorema 14.3.2.

Dato un linguaggio libero L, i problemi di determinare se:

a) $\overline{L} = \phi$;
b) $\overline{L}$ è infinito;
c) $\overline{L}$ è un linguaggio libero;
d) $\overline{L}$ è un linguaggio regolare;
e) L è regolare;
f) $L = T^*$;

sono indecidibili.

Ricordiamo infine che non è decidibile se una GL è ambigua.

14.4. Semplificazione delle GL. Forme Normali

Data una qualsiasi GL è univocamente determinato il linguaggio da essa generato. Non vale però il viceversa: dato un linguaggio esistono più GL che lo generano. Fra tutte queste grammatiche equivalenti ne possono esistere alcune ridondanti nel senso che presentano simboli non terminali (o semplicemente variabili), e relative produzioni, che non verranno mai usate nella derivazione di una parola del linguaggio o che vengono usate in produzioni del tipo $A \to A$. Viene quindi naturale chiedersi se non esistano degli algoritmi che, data una GL, la riducano ad una forma più semplice secondo certi criteri.

Teorema 14.4.1.

Data una GL $G = (V, T, S, P)$, si può costruire effettivamente una GL ridotta $G' = (V', T, S, P')$ tale che:

$$L(G) = L(G'),\ V' \subseteq V,\quad P' \subseteq P \quad e$$

1) $\forall A \in (V' - T),\ A \neq S,\quad \exists w \in T^*$ tale che $A \overset{*}{\underset{G}{\to}} w$

2) $\forall A \in (V' - T),\quad \exists \alpha,\ \beta \in V^*$ tali che $S \overset{*}{\underset{G'}{\to}} \alpha A \beta$

3) $\forall A, B \in (V' - T)$ non esistono produzioni $A \to B$

La condizione 1) ci assicura che sono eliminabili tutte le variabili dalle quali non è possibile derivare parole del linguaggio; la 2) che si possono eliminare quelle variabili che non verranno usate in nessuna derivazione dal simbolo iniziale, e la 3) che sono eliminabili le produzioni che coinvolgono solo simboli non terminali.

Siccome le proprietà richieste, per il teorema 14.2.6, sono decidibili, le nuove grammatiche si ottengono verificando quali simboli o produzioni non le posseggono e quindi eliminandoli o modificandole opportunamente.

Oltre ad eliminare variabili o produzioni inutili, si può richiedere che le produzioni siano tutte di forma particolare, senza tuttavia ridurre l'espressività delle grammatiche. Le GL in forma normale sono particolarmente utili nei problemi di analisi sintattica.

Forma Normale di Chomsky: Ogni linguaggio libero può essere generato da una grammatica le cui produzioni sono tutte del tipo $A \to BC$ oppure $A \to a$, dove A, B e C sono variabili ed a terminale.

Forma Normale di Greibach: Ogni linguaggio libero può essere generato da una grammatica le cui produzioni sono tutte del tipo $A \to a\alpha$, dove $A \in (V - t)$, $a \in T$, $\alpha \in (V - T)^*$.

Si dimostra anche che data una GL è possibile costruirne effettivamente una equivalente in Forma Normale di Chomsky e di Greibach.

Esempio 14.2. Consideriamo la grammatica dell'esempio 14.1.

Le grammatiche equivalenti in Forma Normale di Chomsky e in Forma Normale di Greibach sono:

E → EA | TB | LC | a	E → aMT | a | (ERMT | (ER | aMTPE | aPE | (ERMTPE | (ERPE
A → PT	T → aMT | a | (ERMT | (ER
P → +	M → *
T → TB | LC | a	R →)
B → MF	P → +
M → *	

$F \to LC \mid a$
$L \to ($
$C \to ER$
$R \to)$

Esempio 14.3. Il linguaggio costituito dalle parole con lo stesso numero di a e b è generato dalla grammatica in forma normale di Greibach:

$S \to bA \mid aB$
$A \to a \mid aS \mid bAA$
$B \to b \mid bS \mid aBB$

e dalla grammatica in forma normale di Chomsky:

$S \to FA \mid CB$	$H \to a$
$A \to HS \mid LD \mid a$	$L \to b$
$B \to MS \mid NE \mid b$	$C \to a$
$D \to AA$	$M \to b$
$E \to BB$	$N \to a$
$F \to b$	

14.5. Proprietà di chiusura dei linguaggi liberi

Le proprietà di chiusura, oltre ad essere interessanti da un punto di vista teorico, possono servire a dimostrare che certi linguaggi non sono liberi.

Teorema 14.5.1.

La classe dei linguaggi liberi è chiusa rispetto all'operazione di unione.

Dimostrazione. Siano $G_1 = (V_1, T, S_1, P_1)$ e $G_2 = (V_2, T, S_2, P_2)$ due GL.

Supponiamo, senza ledere la generalità, che le due grammatiche non abbiano simboli non terminali in comune, cioè $(V_1 - T) \cap (V_2 - T) = \phi$. (Se così non fosse basterebbe ridenominare i simboli, ottenendo grammatiche equivalenti). Definiamo $G = (V_1 \cup V_2 \cup \{S\}, T, S, P_1 \cup$

$\cup P_2 \cup \{S \to S_1, S \to S_2\})$ dove S è un nuovo simbolo con $S \notin V_1 \cup V_2$. Si dimostra che $L(G) = L(G_1) \cup L(G_2)$. Infatti, se ad esempio $x \in L(G_1)$ significa che $S_1 \overset{*}{\underset{G_1}{\to}} x$, quindi $S \underset{G}{\to} S_1 \overset{*}{\underset{G_1}{\to}} x$, ed anche $S \overset{*}{\underset{G}{\to}} x$, dato che G contiene tutte le produzioni di G_1.

Viceversa se $x \in L(G)$, allora $S \overset{*}{\underset{G}{\to}} x$, ma siccome ad S si possono applicare solo le produzioni $S \to S_1$ o $S \to S_2$, si avrà, diciamo, $S \underset{G}{\to} S_1 \overset{*}{\underset{G}{\to}} x$. Ma per l'ipotesi fatta che $(V_1 - T) \cap (V_2 - T) = \phi$, nella derivazione di x da S_1 si potranno applicare solo produzioni di G_1, pertanto si avrà anche $S_1 \overset{*}{\underset{G_1}{\to}} x$. ■

Teorema 14.5.2.

La classe dei linguaggi liberi è chiusa rispetto all'operazione di concatenazione.

Dimostrazione. Siano G_1 e G_2 come nel teorema precedente. Sia $G = (V_1 \cup V_2 \cup \{S\}, T, S, P_1 \cup P_2 \cup \{S \to S_1 S_2\})$ dove $S \notin V_1 \cup V_2$. Con dimostrazione analoga a quella del teorema precedente si vede che $L(G) = L(G_1) \cdot L(G_2)$. ■

Teorema 14.5.3.

La classe dei linguaggi liberi è chiusa rispetto all'operazione stella di Kleene.

Dimostrazione. Sia G_1 come sopra. Definiamo $G = (V_1 \cup \{S\}, T, S, P \cup \{S \to S_1 S, S \to S_1\})$ con $S \notin V_1$. Come nei teoremi precedenti si vede che $L(G) = L(G_1)^* - \{\lambda\}$ (*). ■

Teorema 14.5.4.

La classe dei linguaggi liberi non è chiusa rispetto all'operazione di intersezione.

Dimostrazione. I linguaggi $L_1 = \{a^n b^n c^i \mid n \geqslant 1 \text{ e } i \geqslant 0\}$
$L_2 = \{a^j b^n c^n \mid n \geqslant 1 \text{ e } j \geqslant 0\}$

(*) Si tenga presente l'osservazione fatta nel paragrafo 12.7 per quanto riguarda la parola vuota nei linguaggi generati da grammatiche ristrette.

sono liberi in quanto generati rispettivamente dalle grammatiche:

$G_1 = (\{S, T, a, b, c\}, \{a, b, c\}, S, \{S \to Sc, S \to T, T \to aTb, T \to ab\})$

$G_2 = (\{S, T, a, b, c\}, \{a, b, c\}, S, \{S \to aS, S \to T, T \to bTc, T \to bc\})$

Tuttavia $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geqslant 1\}$ non è libero (teorema 14.2.4. ■

Questo porta anche a concludere che:

Teorema 14.5.5.

La classe dei linguaggi liberi non è chiusa rispetto all'operazione di complementazione.

Dimostrazione. Se così fosse, siccome è chiusa rispetto all'unione, lo sarebbe anche rispetto all'intersezione poichè $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. ■

Menzioniamo infine un teorema di chiusura tra linguaggi liberi e regolari.

Teorema 14.5.6.

La classe dei linguaggi liberi è chiusa rispetto all'intersezione con un linguaggio regolare.

14.6. Automi a pila

Nel capitolo precedente abbiamo visto come, in sostanza, la limitazione nella capacità di riconoscimento degli ASFD dipendesse dal fatto che questi posseggono una memoria, costituita dagli stati, finita.

Per questo motivo essi non sono in grado di riconoscere linguaggi che, per la loro struttura, richiedono di ricordare una quantità di "informazioni" non limitate. Come nel caso del linguaggio $a^n b^n$ che richiede di ricordare il numero di "a" letti per poter controllare che li segua uno stesso numero di "b".

Per aumentare la capacità di riconoscimento dell'automa a stati finiti, occorre avere la possibilità di ricordare una quantità illimitata di informazione e far dipendere l'azione da compiere non solo dal carattere letto e dallo stato ma anche da un altro parametro che dipenda in qualche modo dalla informazione memorizzata. Precisiamo meglio questi concetti descrivendo un modello che chiameremo *automa a pila* (AP) o "push-down automaton". Esso è costituito da un controllo a stati finiti, da un nastro di ingresso e da una memoria ausiliaria a pila di lunghezza infinita alla quale si può quindi accedere *solo* dalla testa (Figura 14.3).

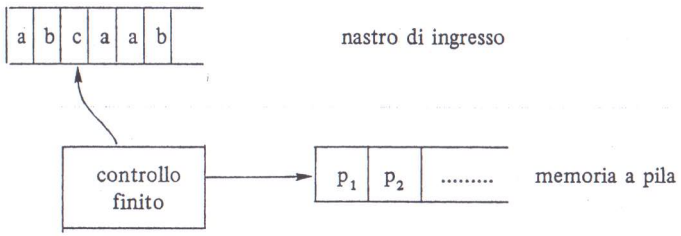


Fig. 14.3

In ogni situazione l'AP può compiere due tipi di mosse:

1) Leggere il contenuto di una cella del nastro ed il simbolo in cima alla pila e, in base a ciò e allo stato in cui si trova, passare in un nuovo stato e sostituire il simbolo letto dalla pila con una stringa (eventualmente λ), mentre la testina sul nastro di ingresso avanza di una posizione a destra. In cima alla pila apparirà ora il primo simbolo della stringa che vi è appena stata scritta.

2) Come nella mossa del primo tipo, ma senza leggere alcun simbolo dal nastro (viene letto λ) e senza avanzamento della testina.

La mossa di tipo 2 permette all'AP di manipolare il contenuto della pila senza leggere ulteriori simboli dal nastro; tuttavia, qualora lo automa intendesse leggere qualche simbolo in posizione interna nella pila, esso perderebbe completamente traccia di quanto lo precedeva.

Per definire il linguaggio riconosciuto da un AP vi sono due possibilità distinte. Una parola è accettata da un AP se, dopo averla letta:

1) l'automa si trova in uno stato finale; oppure

(1) $(q_0,abba,Z) \vdash (q_0,bba,AZ) \vdash (q_0,ba,BAZ) \vdash$
$\vdash (q_0,a,BBAZ) \vdash (q_0,\lambda,ABBAZ)$

(2) $(q_0,abba,Z) \vdash (q_0,bba,AZ) \vdash (q_0,ba,BAZ) \vdash$
$\vdash (q_1,a,AZ) \vdash (q_1,\lambda,Z) \vdash (q_1,\lambda,\lambda)$

Quindi nonostante la prima serie di mosse non porti a vuotare la pila, non è vero che abba non viene accettata. Invece, siccome la seconda serie conduce alla pila vuota, la parola abba è riconosciuta.

Un automa a pila è deterministico quando la funzione di transizione t specifica non più di una mossa per ogni configurazione ed in particolare le mosse di tipo 2 sono possibili solo quando quelle di tipo 1 non sono permesse. La classe dei linguaggi riconosciuti da APD è più ristretta di quella dei linguaggi riconosciuti da APND. Quest'ultima infatti coincide, come vedremo nel prossimo paragrafo, con la classe dei linguaggi liberi, mentre la prima coincide con la classe dei linguaggi generati da un particolare tipo di grammatiche libere: le grammatiche LR(k). Si dimostra che il linguaggio dell'Esempio 2.5 non è riconoscibile da un APD.

14.7. Automi a pila non deterministici e linguaggi liberi.

Vogliamo far vedere che la classe dei linguaggi riconosciuti da APND coincide con la classe dei linguaggi liberi.

Teorema 14.7.1. Se L è un linguaggio libero, esiste un APND M tale che L = N(M), ed M legge ad ogni passo un simbolo dal nastro.

Dimostrazione. Sia G = (V, T, S, P) una grammatica libera in forma normale di Greibach che genera L. Sia

$$M = (\{q\}, T, V-T, t, q, S, \phi)$$

dove t(q, a, A) contiene (q, α) se e solo se $A \to a\alpha$ è in P: di conseguenza tale automa legge un simbolo ad ogni passo.

Mostriamo come L(G) = N(M). Si noti che se esiste una derivazione in G del tipo:

$$x A \beta \xrightarrow[G]{} x a\alpha \beta \quad \text{con} \quad \alpha, \beta \in (V-T)^*$$

allora esisterà una mossa di M tale che:

$$(a, a, A\beta) \vdash (q, \lambda, \alpha\beta)$$

e viceversa.

Per induzione sul numero di passi della derivazione ne segue che

$$x A \xrightarrow[G]{*} x y \alpha \beta, \quad \text{dove} \quad x, y \in T^*, A \in V-T \text{ e } \alpha, \beta \in (V-T)^*,$$

se e solo se esiste una seguenza di mosse di M tali che:

$$(q, y, A\beta) \overset{*}{\underset{M}{\vdash}} (q, \lambda, \alpha\beta)$$

Perciò $S \xrightarrow[G]{*} y$ se e solo se $(q, y, S) \overset{*}{\underset{M}{\vdash}} (q, \lambda, \lambda)$. ■

In base a tale risultato possiamo concludere che per quel che concerne la capacità di riconoscimento ci si può limitare agli APND che non eseguono mosse di tipo 2. Ad automi di questo tipo ci si può sempre ricondurre in maniera effettiva e di fatto tale problema corrisponde a quello di portare una GL in forma normale di Greibach.

Per ottenere il viceversa è conveniente sfruttare il seguente lemma, che ci garantisce che gli APND con un solo stato non sono meno potenti degli APND generali.

Lemma 14.7.2. Se L = N(M) per qualche APDN M, allora L = $N(M_1)$ per qualche APND M_1 con un solo stato.

La dimostrazione di questo fatto si ottiene costruendo un automa M_1 con un solo stato i cui simboli dell'alfabeto della pila sono costituiti da coppie stato-simbolo di M, e la cui funzione di transizione estenda quella di M.

Teorema 14.7.3. Se L = N(M) per qualche APND M ad un solo stato, allora L è un linguaggio libero.

Dimostrazione. Sia M = $(\{q\}, A, R, t, q, Z_0, \phi)$. Definiamo la GL G = $(A \cup R, R, Z_0, P)$ dove P è l'insieme di produzioni:

$$Z \to a Z_1 Z_2 \cdots Z_n \text{ se } t(q, a, Z) \text{ contiene } (q, Z_1 \cdots Z_n).$$

Risulta immediatamente che

$$(q, a, Z\alpha) \vdash_M (q, \lambda, Z_1 \ldots Z_n \alpha) \text{ se e solo se } Z\alpha \xrightarrow[G]{} a Z_1 \ldots Z_n \alpha$$

e quindi per induzione che per ogni $x \in A^*$

$$(q, x, Z_0) \vdash_M^* (q, \lambda, \lambda) \text{ se e solo se } z_0 \xrightarrow[G]{*} x . \qquad \blacksquare$$

Esempio 14.6. In Fig. 14.4 b) è riportata la tabella di transizione dell'AP corrispondente alla grammatica in forma normale di Greibach di Fig. 14.4a).

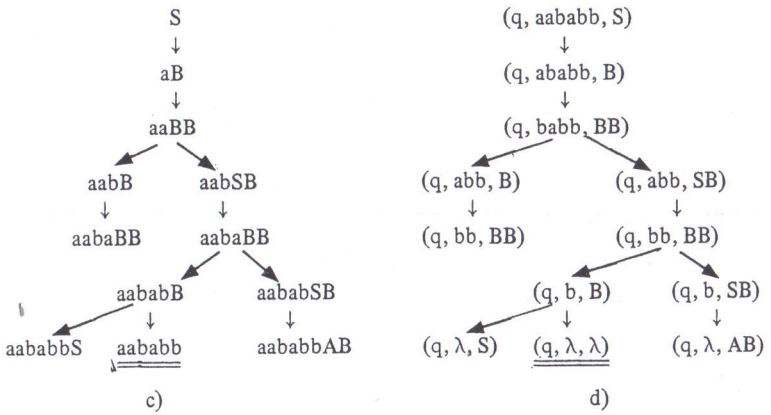
$S \to aB$	$A \to a$
$S \to bA$	$B \to bS$
$A \to aS$	$B \to aBB$
$A \to bAA$	$B \to b$

a)

	λ	a	b
S		q,B	q,A
A		q,λ q,S	q,AA
B		q,BB	q,λ q,S

q

b)



c) d)

Fig. 14.4

In fig. 14.4 c, d sono rappresentati con due alberi la derivazione in G e il riconoscimento con l'AP della parola aababb.

Si osservi la similitudine tra derivazioni della G nella forma normale di Greibach e mosse dell'automa. Le produzioni generano ad ogni passo un carattere terminale della parola finale che si vuole ottenere mentre l'automa ad ogni mossa legge un carattere e costruisce nella pila la stessa stringa di simboli non terminali presente nella corrispondente derivazione. Inoltre ogni stringa nella derivazione coincide con la stringa ottenuta concatenando i caratteri letti con i caratteri presenti nella pila. Si noti infine come l'automa analizzi la parola con il metodo dall'alto al basso costruendo l'albero di derivazione.

14.8. Complessità dell'algoritmo di analisi.

Come si è visto nel paragrafo precedente, un APND è un modello della procedura di riconoscimento per i linguaggi liberi.

Tuttavia per essere realizzato in pratica, tale modello deve essere simulato deterministicamente e ciò comporta il dove seguire tutte le possibili scelte dell'APND. Quindi in generale ad ogni passo si moltiplicano le alternative e di conseguenza la complessità di un tale algoritmo cresce esponenzialmente con la lunghezza della stringa da riconoscere. Mostriamo ora invece un algoritmo di analisi più efficiente la cui complessità cresce col cubo della lunghezza della parola, e che utilizza le grammatiche in forma normale di Chomsky. Vediamone prima il funzionamento sulla grammatica dell'Esempio 14.2.

Data la parola $x = a_1 a_2 \ldots a_n$ che si vuole analizzare, si riempie una tabella triangolare di lato n, ponendo nella casella T_{ij} tutte e sole le variabili che generano la sottostringa di x di lunghezza i a partire dal j-esimo simbolo.

Vediamo come si costruisce la tabella per l'analisi di $(a + a) * a$

Le caselle della prima riga contengono quelle variabili che generano direttamente il corrispondente simbolo: così $T_{15} = \{R\}$ poiché $R \xrightarrow[G]{}$).

$E \to EA \mid TB \mid LC \mid a$

$A \to PT$

$P \to +$

$T \to TB \mid LC \mid a$

$B \to MF$

$M \to *$

$F \to LC \mid a$

$L \to ($

$C \to ER$

$R \to ($

	(	a	+	a	)	*	a
1	L	E T F	P	E T F	R	M	E T F
2	–	–	A	C	–	B	7
3	–	E	–	–	–	6	
4	–	C	–	–	5		
5	E T F	–	–	4			
6	–	–	3				
7	E T	2					
	1						

Fig. 14.5

Nella j-esima casella della seconda riga vanno messe quelle variabili che generano sottostringhe di lunghezza due e che iniziano con il j-esimo simbolo della parola in esame. Essendo la grammatica nella forma normale di Chomsky queste variabili saranno quelle che generano coppie di non terminali appartenenti il primo alla cella immediatamente sopra, il secondo a quella a destra di quest'ultima: così $T_{24} = \{C\}$ poiché $C \xrightarrow[G]{} ER$ ed $E \in T_{14}$ ed $R \in T_{15}$. Le celle della terza riga devono contenere variabili che generano sottostringhe di x di lunghezza 3, ad esempio T_{32} quelle che generano a + a. Questa stringa può essere suddivisa in due parti non vuote in due soli modi: a, +a oppure a+, a. Siccome T_{12} e T_{23} contengono le variabili che generano rispettivamente a e a+, T_{32} dovrà contenere tutte le variabili A che appaiono in produzioni $A \to B_1 B_2$ con $B_1 \in T_{12}$ e $B_2 \in T_{23}$. Dovrà però contenere anche quelle variabili A per cui $B_1 \in T_{22}$ e $B_2 \in T_{14}$ e che corrispondono all'altra possibile suddivisione di a+a.

Passando alla riga 4, T_{42}, ad esempio, sarà formato dalle variabili A tali che $A \longrightarrow B_1 B_2$ e inoltre $B_1 \in T_{12}$ e $B_2 \in T_{33}$, oppure $B_1 \in T_{22}$ $B_2 \in T_{24}$ oppure $B_1 \in T_{32}$ e $B_2 \in T_{15}$. Cioè per ognuna delle (4-1) suddivisioni della stringa a + a) = $x_1 \cdot x_2$, B_1 e B_2 sono le variabili che generano x_1 e x_2.

E' facile a questo punto ricavare la regola generale e descrivere quindi formalmente l'algoritmo.

Sia G = (V, T, S, P) una grammatica libera in forma normale di Chomsky e sia $x = a_1 a_2 \cdots a_n$ la parola che si desidera riconoscere. Si costruiscono gli insiemi T_{ij}, per ogni i e j con $i + j \leqslant n + 1$, nel seguente modo:

1) $T_{ij} = \{A \in V \mid A \xrightarrow[G]{} a_j\}$ per ogni $1 \leqslant j \leqslant n$

2) $T_{ij} = \{A \in V \mid A \xrightarrow[G]{} B_1 B_2,\ B_1 \in T_{kj}$ e $B_2 \in T_{i-k,j+k}$ per qualche $1 \leqslant k < i\}$ per ogni $i > 1$.

Si dimostra a questo punto che:

Teorema 14.8.1. Per gli insiemi T_{ij} costruiti come sopra, vale che

$$T_{ij} = \{A \in V \mid A \overset{*}{\underset{G}{\longrightarrow}} a_j a_{j+1} \cdots a_{j+i-1}\}$$

e di conseguenza $x \in L(G) \Leftrightarrow S \in T_{n1}$.

Dimostrazione. Si effettua per induzione su i. Per i = 1 la relazione è vera per definizione. Supponendo che sia vera per $i \leqslant k$, dimostriamola per i = k + 1. Essendo la grammatica in forma normale di Chomsky, $A \overset{*}{\underset{G}{\longrightarrow}} a_j a_{j+1} \cdots a_{j+k}$ se e solo se $A \xrightarrow[G]{} B_1 B_2$ e $B_1 \overset{*}{\underset{G}{\longrightarrow}} a_j \cdots \cdots a_{j+r-1}$ e $B_2 \overset{*}{\longrightarrow} a_{j+r} \cdots a_{j+k}$ per qualche $1 \leqslant r \leqslant k$. Ma per ipotesi di induzione $B_1 \in T_{rj}$ e $B_2 \in T_{k-r+1,j+r}$ essendo $r \leqslant k$ e $k - r + 1 \leqslant k$; quindi $A \in T_{k+1,j}$. ■

Notare infine come ripercorrendo all'indietro la tabella si possono ricavare tutti gli alberi di derivazione della stringa data.

Vediamo di calcolare la complessità dell'algoritmo descritto .

Consideriamo come operazione elementare il confronto di due caselle della matrice per ricavare le variabili che generano direttamente coppie di simboli della prima e della seconda cella.

La prima riga della matrice di riconoscimento ha n elementi, e ciascuno richiede un passo per essere calcolato.

La seconda riga ha n-1 elementi e per ciascuno occorre una operazione.