grammatica	*forma delle regole*	*famiglia del linguaggio*	*tipo del riconoscitore*
Tipo 0	$\beta \rightarrow \alpha$ dove $\alpha, \beta \in (\Sigma \cup V)^+$ e $\beta \neq \varepsilon$	Ricorsivamente enumerabile	Macchina di Turing
Tipo 1 o contestuale o dipendente dal contesto (context sensitive)	$\beta \rightarrow \alpha$ dove $\alpha, \beta \in (\Sigma \cup V)^+$ e $\lvert\beta\rvert \leqslant \lvert\alpha\rvert$	Contestuale o dipendente dal contesto	Macchina di Turing con nastro di lunghezza limitata linearmente dalla lunghezza della stringa da riconoscere
Tipo 2 o libera dal contesto o non-contestuale (context-free) o BNF	$A \rightarrow \alpha$ dove A è un non-terminale e $\alpha \in (\Sigma \cup V)^*$	Linguaggi liberi (dal contesto) LIB o non contestuali o algebrici	Automa con memoria a pila
Tipo 3 o unilineare (lineare a destra o lineare a sinistra)	Lineare a destra: $A \rightarrow uB$, Lineare a sinistra: $A \rightarrow Bu$, dove A è un non-terminale, $u \in \Sigma^*$ e $B \in (V \cup \varepsilon)$	Regolari REG o razionali o a stati finiti	Automa finito

Le famiglie di linguaggi sono in relazione di contenimento stretto dall'alto verso il basso della tabella.

Le differenze tra le varie classi stanno, oltre che nella forma delle regole, nelle proprietà dei loro automi riconoscitori, che per i tipi 2 e 3 saranno studiati nei prossimi capitoli.

Dal punto di vista della memoria necessaria all'algoritmo che riconosce le frasi, si nota che, mentre per le classi 0, 1 e 2 essa è illimitata, per i linguaggi del tipo 3 basta una memoria finita.

Senza addentrarsi nello studio delle proprietà delle diverse classi, si citano alcuni altri punti.

Tutte e quattro le famiglie di linguaggi sono chiuse rispetto all'unione, al concatenamento, alla stella, all'inversione speculare, all'intersezione con linguaggi regolari. Ma le famiglie differiscono rispetto ad altre proprietà di chiusura: ad es. la chiusura risp. al complemento si ha per i tipi 1 e 3, ma non per i tipi 0 e 2.

Dal punto di vista della decidibilità, un salto algoritmico separa i linguaggi del tipo 0 da quelli del tipo 1, pur così simili nella struttura delle regole. Per i primi non esiste un algoritmo generale per decidere se una stringa appartiene al linguaggio (il problema è indecidibile o più precisamente semi-decidibile).

Invece per i linguaggi contestuali tale algoritmo esiste, anche se la sua complessità è non polinomiale.
Infine soltanto per il tipo 3 è decidibile se due grammatiche definiscono lo stesso linguaggio.

Si completa la presentazione con alcuni esempi di grammatiche e linguaggi del tipo 1 o contestuale.

Esempio 2.92. Grammatica contestuale del linguaggio a tre esponenti.
Un linguaggio non libero noto

$$L = \{a^n b^n c^n \mid n \geqslant l\}$$

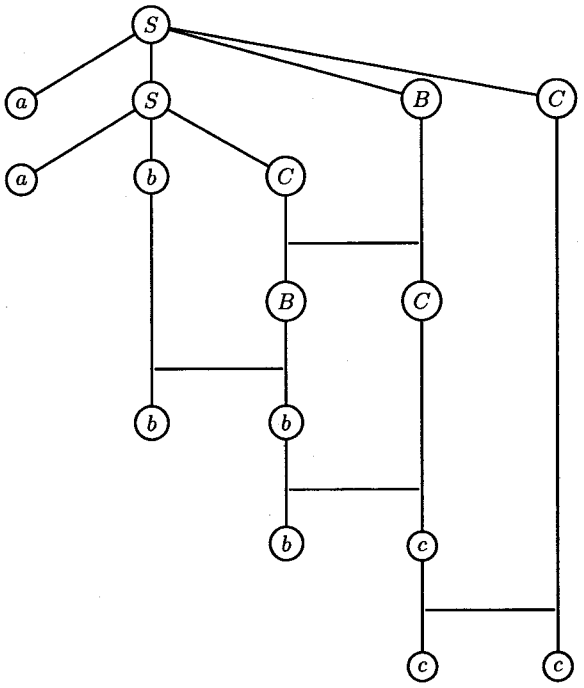
è generato dalla seguente grammatica di tipo 1:

1. $S \to aSBC$	3. $CB \to BC$	5. $bC \to bc$
2. $S \to abC$	4. $bB \to bb$	6. $cC \to cc$

Per le grammatiche dei tipi 1 e 0 la derivazione non è più rappresentabile come un albero, in quanto alla parte sinistra di una regola non corrisponde più un solo simbolo, bensì un insieme di simboli. Tuttavia si può disegnare la derivazione d'una frase sotto forma di un grafo, dove l'applicazione d'una regola come $BA \to AB$ appare come sostituzione simultanea di tutti i simboli della parte sinistra con quelli della parte destra.
Per generare la frase *aabbcc* sembra ragionevole cercare di costruire una derivazione sinistra:

$$S \Rightarrow aSBC \Rightarrow aabCBC \Rightarrow aabcBC \Rightarrow \text{ blocco!}$$

Purtroppo la derivazione si blocca, prima di riuscire a eliminare tutti i simboli nonterminali. Per generare la stringa voluta si deve usare una derivazione non sinistra, disegnata nel grafo seguente.



Intuitivamente la derivazione genera il numero voluto di lettere a mediante la regola 1 (del tipo 2 autoincassata) e poi la regola 2. I nonterminali B e C sono generati con il numero giusto di ripetizioni, ma mescolati tra loro. Per ottenere la stringa voluta, tutte le C devono essere spostate a suffisso, mediante la regola 3. $CB \to BC$. La prima applicazione della 3 riordina la forma di frase in modo che una B viene a essere contigua a una b, abilitando così la derivazione 4. $bB \Rightarrow bb$. Si procede allo stesso modo alternando i passi 3 e 4, finché nella forma di frase non restano che delle C come nonterminali. Esse sono trasformate nei terminali c, mediante la regola $bC \to bc$, seguita da ripetute applicazioni della regola $cC \to cc$.

Si è constatato nel primo tentativo che, a differenza delle grammatiche libere, per quelle contestuali non vale la proprietà che il linguaggio generato mediante derivazioni sinistre coincide con quello generato mediante derivazioni qualsiasi.
Queste grammatiche consentono di trattare le repliche (o le liste con concordanze), una struttura sintattica di interesse pratico già introdotta per esemplificare i limiti della famiglia LIB.

Esempio 2.93. Grammatica contestuale delle repliche.
Il linguaggio $L = \{ycy \mid y \in \{a,b\}^+\}$ contiene le frasi del tipo *aabcaab* in cui

vi è una marca centrale c e gli elementi di eguale posizione nel prefisso y e nel suffisso y devono essere eguali.
Per semplificare la scrittura delle regole, si suppone che le frasi siano terminate a destra dalla marca di fine stringa, $\dashv$.
La grammatica del tipo 1 è :

$$\begin{array}{lllll} S \to X \dashv & XA \to XA' & A'A \to AA' & A' \dashv\!\!\to a & B'a \to ba \\ X \to aXA & XB \to XB' & A'B \to BA' & B' \dashv\!\!\to b & B'b \to bb \\ X \to bXB & & B'A \to AB' & A'a \to aa & Xa \to ca \\ & & B'B \to BB' & A'b \to ab & Xb \to cb \end{array}$$

La grammatica segue questa strategia: dapprima genera un palindromo, quale $aabXBAA$, dove X marca il centro della frase e la seconda metà è scritta con simboli maiuscoli. Poi la seconda metà $B'AA$ viene invertita, trasformandola, in più passi, in $A'A'B'$. Infine i simboli maiuscoli apostrofati sono trascritti come aab, e il centro X è convertito in c.
Si mostra la derivazione della frase $aabcaab$, sottolineando per facilitare la lettura, la parte sinistra di ogni regola applicata:

$$\underline{S} \Rightarrow \underline{X} \dashv \Rightarrow a\underline{X}A \dashv \Rightarrow aa\underline{X}AA \dashv \Rightarrow aab\underline{XB}AA \dashv \Rightarrow$$

$$aabX\underline{B'A}A \dashv \Rightarrow aab\underline{XA}B'A \dashv \Rightarrow aabXA'\underline{B'A} \dashv \Rightarrow$$

$$aabX\underline{A'A}B' \dashv \Rightarrow aab\underline{XA}A'B' \dashv \Rightarrow aabXA'A'\underline{B' \dashv} \Rightarrow$$

$$aabXA'\underline{A'b} \Rightarrow aabX\underline{A'a}ab \Rightarrow aab\underline{Xa}ab \Rightarrow aabcaab$$

Si può osservare che la strategia di generazione delle frasi, applicata in senso opposto, permetterebbe di riconoscere se una stringa data appartiene al linguaggio. L'algoritmo dovrebbe memorizzare su di un nastro le stringhe via via ottenute, a partire da quella data; tale formulazione algoritmica corrisponde al calcolo svolto da una macchina di Turing, che non esce mai con la testina dal segmento di nastro occupato dalla stringa data.

Gli esempi dovrebbero convincere il lettore della difficoltà di applicare questo genere di grammatiche alla definizione di linguaggi di maggiori dimensioni. Infatti la grammatica si comporta come un algoritmo in cui le regole interagiscono in modo intricato e poco comprensibile.
Detto diversamente, le grammatiche del tipo 1 e 0 possono essere viste come una particolare notazione per programmare algoritmi qualsiasi, anche di natura matematica, facendo ricorso al solo meccanisno della trascrizione delle stringhe. Un caso tipico[25] è l'algoritmo, in forma di grammatica contestuale, che genera i numeri primi nella notazione unaria, ovvero il linguaggio $\{a^n \mid n \text{ numero primo}\}$. Tali sviluppi della teoria dei linguaggi formali nella direzione degli insiemi di natura matematica non hanno però utilità per il

[25] Presentato in [43].

progetto dei linguaggi dell'informatica.
Ciononostante le grammatiche contestuali hanno avuto alcuni rari fautori tra i progettisti dei linguaggi artificiali e dei compilatori. Si cita il linguaggio Algol 68, interamente definito mediante una particolare classe di grammatiche contestuali, dette a due livelli. [26]
In conclusione si deve ammettere che, allo stato dell'arte della teoria delle grammatiche formali, resta insoddisfatta l'esigenza di definire certi costrutti sintattici utili e frequenti come le repliche, che sfuggono alla capacità generativa delle grammatiche libere. Il progettista dei compilatori dovrà quindi definire tali costrutti con altri metodi detti semantici, che aggiungono dei vincoli di diversa natura alle regole grammaticali. Essi sono l'argomento del capitolo 5.

[26] Note anche come VW-grammatiche dal nome di Van Wijngarten[51]; vedasi anche Cleaveland e Uzgalis [12].