Capitolo 3: Linguaggi liberi dal contesto

1 Albero di derivazione e derivazione "più a sinistra"

Ricordiamo che un linguaggio L è *acontestuale* (o *libero dal contesto*) se è generato da una grammatica di tipo 2, in cui cioè ogni regola di produzione è della forma $\alpha \rightarrow \beta$ dove α è un metasimbolo. Questa particolarità permette di dare una semplice descrizione visiva di una "derivazione " mediante il cosiddetto *albero di derivazione*.

Data una grammatica $G = <\Sigma, Q, P, S>$ di tipo 2 che genera il linguaggio L(G), un *albero di derivazione* della parola $w \in L(G)$ in G è un albero ordinato i cui nodi sono etichettati con simboli in $S \cup Q$, così che:

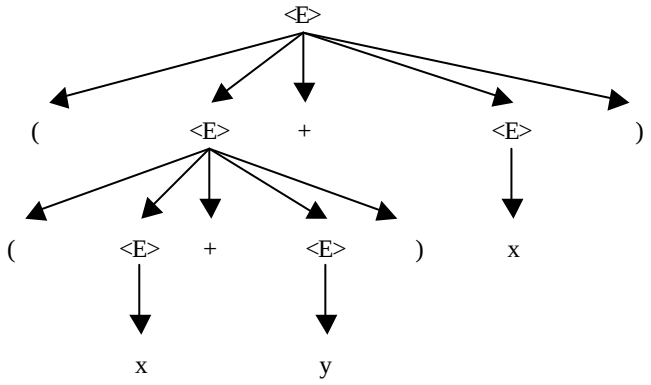
- ogni foglia è etichettata con un simbolo terminale, mentre ogni nodo interno è etichettato con un metasimbolo,
- la radice è etichettata con l'assioma S,
- se un nodo interno è etichettato col metasimbolo B e i suoi figli sono etichettati, in ordine, coi metasimboli $B_1 \dots B_m$, allora $B \rightarrow B_1 \dots B_m$ è una regola di produzione di G,
- leggendo le foglie in ordine prefisso, la sequenza di etichette forma la parola w.

Esempio 1.1

Data la grammatica G =< {(,),+,x.,y}},{E},{E→(E+E)/x/y},E> , si consideri la seguente derivazione della parola ((x+y)+x):

$$E \Rightarrow_G (E+E) \Rightarrow_G ((E+E)+E) \Rightarrow_G ((x+E)+E) \Rightarrow_G ((x+y)+E) \Rightarrow_G ((x+y)+x)$$

L'albero di derivazione corrispondente è:



Si osservi che diverse derivazioni possono avere associato lo stesso albero. Per esempio, riferendoci alla grammatica dell'esempio precedente, la seguente derivazione, diversa da quella presentata sopra, genera lo stesso albero:

$$E \Rightarrow_G (E+E) \Rightarrow_G (E+x) \Rightarrow_G ((E+E)+x) \Rightarrow_G ((E+y)+x) \Rightarrow_G ((x+y)+x)$$

Data una grammatica G, diremo che due derivazioni sono *equivalenti* se hanno associato lo stesso albero. Può essere interessante ottenere un elemento rappresentativo nella classe delle derivazioni equivalenti, compatibili quindi con lo stesso albero di derivazione. Una soluzione è quella di considerare la cosiddetta "*derivazione più a sinistra*", corrispondente alla visita dell'albero in profondità. Data una parola w contenente almeno un metasimbolo, è infatti univocamente individuato il

metasimbolo “più a sinistra” nella parola. Ad esempio, nella parola abAbbCcCB il metasimbolo più a sinistra è A. Questo porta alla seguente:

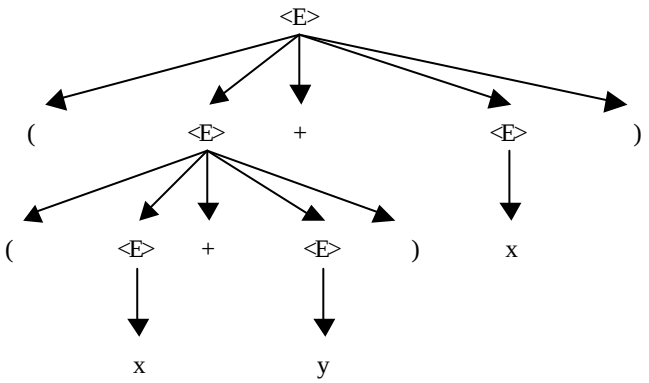
Definizione 1.1 data una grammatica $G = < \Sigma, Q, P, S >$ di tipo 2, sia $w \in \Sigma^*$ una parola derivabile dall’assioma S. La derivazione $S \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \dots \Rightarrow_G w_m \Rightarrow_G w$ è detta “derivazione più a sinistra” se, per ogni j ($1 \le j \le m$), w_{j+1} è ottenuta applicando una regola di produzione al metasimbolo più a sinistra in w_j .

Ovviamente, ogni albero di derivazione in G ha associata un derivazione “più a sinistra” e viceversa: le derivazioni “più a sinistra” risultano allora in corrispondenza biunivoca con gli alberi di derivazione.

2 Grammatiche di tipo 2 ambigue e non ambigue

In vari contesti, data una grammatica G di tipo 2 che genera il linguaggio L(G) ed una parola $w \in L(G)$, risulta ragionevole interpretare un albero di derivazione di w in G come “*significato*” della parola w.

Consideriamo, a scopo esemplificativo, la grammatica $G =< \{(,),+,x.,y\},\{E\},\{E \rightarrow (E+E)/x/y\},E>$: possiamo interpretare le parole del linguaggio generato L(G) come particolari *espressioni aritmetiche* in un linguaggio di programmazione. Assegnando ai simboli x,y un valore numerico e interpretando + come operazione binaria, ad esempio la usuale somma, deve essere possibile attribuire un preciso valore ad ogni parola del linguaggio: in questo caso, l’albero di derivazione della parola stabilisce l’ordine di applicazione delle operazioni così da individuare univocamente il valore dell’espressione. Ad esempio, la parola ((x+y)+x) ha associato l’albero di derivazione:

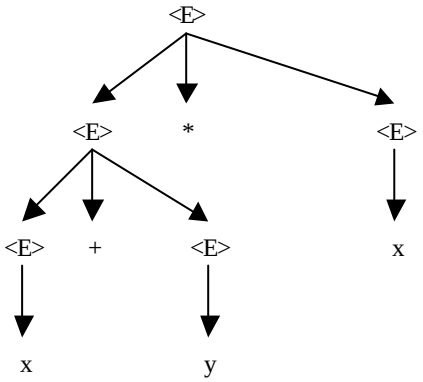


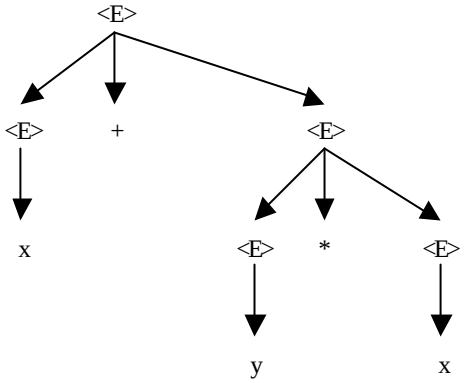
Se x=3 e y=5, l’albero di derivazione permette di calcolare il valore ((3+5)+3)=11 dell’espressione.

Una richiesta importante è che ogni parola in L(G) abbia un unico significato.
Attribuendo un albero di derivazione di w in G come “*significato*” della parola w, può succedere che la stessa parola abbia diversi alberi di derivazione, e quindi significati diversi, risultando *ambigua.* Questo porta alla seguente definizione:

Definizione 2.1 Una grammatica $G = <\Sigma, Q, P, S>$ di tipo 2 è detta *ambigua* se esiste una parola $w \in L(G)$ che ammette due diversi alberi di derivazione; una grammatica $G = <\Sigma, Q, P, S>$ di tipo 2 è detta *non ambigua* se ogni parola $w \in L(G)$ ammette un unico albero di derivazione.

Esempio 2.1
Data la grammatica G =< {+,*,x.,y}},{E},{E→ E+E/E*E/x/y},E> , si consideri la parola x+y*x, Essa ammette i seguenti due distinti alberi di derivazione:





Tale grammatica risulta dunque ambigua.

Esempio 2.2
Data la grammatica G =< {(,),+,*,x.,y}},{E},{E→(E+E)/(E*E)/x/y},E> , è possibile mostrare che ogni parola del linguaggio generato ammette esattamente un albero di derivazione (o, equivalentemente, una derivazione più a sinistra). Dimostriamolo per induzione sulla lunghezza della parola derivata.
Le parole di lunghezza 1 in L(G) sono x e y, che hanno un'unica derivazione (e quindi un unico albero di derivazione).
Sia $w \in L(G)$ una parola di lunghezza n>1. Essa è della forma $(w_1+ w_2)$ o alternativamente $(w_1* w_2)$, dove $|(w_1|,| w_2)|<n$. Supponiamo senza perdere di generalità che $w=(w_1+ w_2)$. Allora w è generata da una derivazione "più a sinistra" del tipo $E \Rightarrow_G (E+E) \Rightarrow_G^* (w_1+E) \Rightarrow_G^* (w_1+ w_2)$, Per ipotesi di induzione, esiste un'unica derivazione "più a sinistra" del tipo $E \Rightarrow_G^* w_1$ e del tipo $E \Rightarrow_G^* w_2$, quindi esiste un'unica derivazione "più a sinistra" di w.

Se L è generato da una grammatica ambigua G di tipo 3, è sempre possibile costruire una grammatica non ambigua G' di tipo 3 che genera L. Basta infatti costruire l'automa deterministico che riconosce L: la relativa grammatica non è ambigua.

Esistono invece linguaggi acontestuali che possono essere generati solo da grammatiche di tipo 2 ambigue: tali linguaggi sono detti *inerentemente ambigui.*

Esempio 2.3
Si consideri il linguaggio

$$L = \{a^j b^s c^k \mid j=s \text{ oppure } k=s \}.$$

Esso è acontesuale, poiché è generato dalla grammatica con assioma S e produzioni del tipo:

$S \rightarrow XC \mid AY$, $Y \rightarrow aYb \mid \varepsilon$, $X \rightarrow bXc \mid \varepsilon$, $A \rightarrow aA \mid \varepsilon$, $C \rightarrow cC \mid \varepsilon$

Tale grammatica è ambigua poichè le parole del tipo $a^j b^j c^j$ ammettono due distinti alberi di derivazione. E' inoltre possibile mostrare che, per ogni grammatica G di tipo 2 che genera L, esistono infinite parole del tipo $a^j b^j c^j$ che ammettono due distinte derivazioni.

3 Forme normali di Chomsky e di Greibach

Sappiamo che se L è di tipo 2 con $\varepsilon \in L$, allora $L=L' \cup \{\varepsilon\}$, dove L' è generato da una grammatica con regole del tipo $A \rightarrow x$, con $x \in (\Sigma \cup Q)^+$. Pertanto in questo paragrafo prenderemo in considerazione solo linguaggi liberi dal contesto non contenenti la parola vuota.

Due importanti sottoclassi di grammatiche di tipo due sono le seguenti:

Definizione 3.1 Una grammatica $G = <\Sigma,Q,P,S>$ di tipo 2 è detta in forma normale di Chomsky se le sue regole di produzione sono del tipo $A \rightarrow BC$ oppure $A \rightarrow x$, dove A,B,C sono metasimboli e x è un simbolo terminale.

Definizione 3.2 Una grammatica $G = <\Sigma,Q,P,S>$ di tipo 2 è detta in forma normale di Greibach se le sue regole di produzione sono del tipo $A \rightarrow xW$, dove x è un simbolo terminale e W una parola (eventualmente vuota) di metasimboli.

E' possibile mostrare la seguente:

Proposizione 3.1 Per ogni linguaggio L libero dal contesto non contenente la parola vuota, esiste una grammatica G' in forma normale di Chomsky che genera L ed esiste una grammatica G'' in forma normale di Greibach che genera L.

Esempio 3.1
Si consideri la grammatica con assioma E e regole di produzione:

$$E \rightarrow (E+E) / x / y$$

Osserviamo che tale grammatica non è in forma normale di Chomsky, per via della regola $E \rightarrow (E+E)$.

Tale regola può tuttavia essere sostituita dalle regole:

$$E \rightarrow AEBEC \ , A \rightarrow (\ , \ B \rightarrow + \ , \ C \rightarrow)$$

ottenendo una grammatica equivalente. A sua volta, la regola $E \rightarrow AEBEC$ può essere sostituita dalle regole $E \rightarrow DBEC$, $D \rightarrow AE$, e, infine, la regola $E \rightarrow DBEC$ può essere sostituita dalle regole $E \rightarrow FG$, $F \rightarrow DB$, $G \rightarrow EC$, ottenendo una grammatica equivalente. In sostanza, si ottiene la grammatica equivalente con assioma E e con regole:

$$E \rightarrow FG, \quad F \rightarrow DB, \quad G \rightarrow EC, \quad D \rightarrow AE, \quad A \rightarrow (, \quad B \rightarrow +, \quad C \rightarrow), \quad E \rightarrow x, \ E \rightarrow y$$

Tale grammatica è in forma normale di Chomsky.

Esempio 3.2
Si consideri la grammatica con assioma E e regole di produzione:

$$E \rightarrow (E+E) / x / y$$

Osserviamo che tale grammatica non è in forma normale di Greibach, per via della regola $E \rightarrow (E+E)$.

Tale regola può tuttavia essere sostituita dalle regole:

$$E \rightarrow (EAEB \ , \ A \rightarrow + \ , \ B \rightarrow)$$

ottenendo una grammatica equivalente. Otteniamo allora una grammatica in forma normale di Greibach, equivalente alla precedente. Tale grammatica ha assioma E e regole di produzione:

$$E \rightarrow (EAEB, \quad A \rightarrow +, \quad B \rightarrow), \quad E \rightarrow x \, , \ E \rightarrow y$$

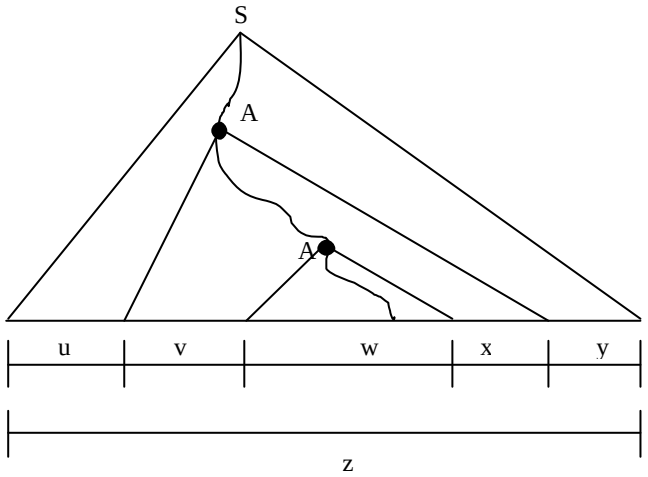
4 “Pumping lemma” e linguaggi non liberi dal contesto

Osserviamo che ogni albero di derivazione di una grammatica in forma normale di Chomsky è un albero binario. Poiché un albero binario con n foglie ha altezza almeno $\log_2 n$, si ottiene una condizione (pumping lemma) necessaria affinché un linguaggio sia acontestuale, che può essere utilizzata per provare che certi linguaggi non sono liberi dal contesto:

Proposizione 4.1 (pumping lemma) Per ogni linguaggio libero dal contesto L esiste una costante H tale che ogni parola $z \in L$ con lunghezza $|z|>H$ può essere decomposta nella forma z=uvwxy tale che:

1. $|vx| \geq 1$ (almeno una parola tra x e v è diversa da ε)
2. $|vwx| \leq H$
3. per ogni $k \geq 0$ vale che $uv^k wx^k y \in L$

Dimostrazione: Sappiamo che esiste una grammatica libera dal contesto G in forma normale di Chomsky che genera $L-\{\varepsilon\}$; sia h il numero di metasimboli di G e sia S l’assioma di G. Posto $H=2^{h+1}$, sia z una parola in L con $|z|>H$. Poiché $z \in L$, esiste un albero di derivazione di z con H foglie e, quindi, con altezza almeno $\log_2 H$; consideriamo ora il più lungo cammino C dalla radice a una foglia, avente quindi lunghezza almeno $h+1 = \log_2 H$. Consideriamo ora un sottocammino di C composto dagli ultimi h+1 nodi consecutivi prima della foglia; poiché G ha h metasimboli, nel sottocammino ci sarà un metasimbolo (diciamo A) ripetuto in due nodi differenti. Chiamiamo w la parola ottenuta dalle foglie del sottoalbero che ha come radice la seconda ripetizione di A; analogamente chiamiamo vwx la parola ottenuta dalle foglie del sottoalbero che ha come radice la prima ripetizione di A. Risulta quindi che z=uvwxy per opportune parole u ed y.



Risulta inoltre:

1. $|vx| \geq 1$. Infatti almeno una parola tra x e v è diversa da ε, altrimenti i due nodi (distinti) etichettati con A sarebbero coincidenti.
2. $|vwx| \leq H$. Infatti l’albero che ha come radice la prima ripetizione di A ha altezza al più h+1, e quindi possiede al più $H=2^{h+1}$ foglie.
3. per ogni $k \geq 0$ vale che $uv^k wx^k y \in L$. Infatti dall’analisi dell’albero di derivazione si deduce che: $S \Rightarrow_G^* uAy$, $A \Rightarrow_G^* vAx$, $A \Rightarrow_G^* w$. Allora, per ogni $k \geq 0$ vale che $S \Rightarrow_G^* uAy \Rightarrow_G^* u\,vAx\,y \Rightarrow_G^* u\,v^2\,Ax^2\,y \Rightarrow_G^* \ldots \Rightarrow_G^* u\,v^k\,Ax^k\,y \Rightarrow_G^* u\,v^k\,w\,x^k\,y$.

ÿ

Il risultato precedente viene usato per dimostrare che certi linguaggi non sono liberi dal contesto; a tale scopo, basta infatti provare che un dato linguaggio non verifica il "pumping lemma".

Esempio 4.2
Consideriamo il linguaggio $L = \{a^n b^n c^n \mid n \geq 1\}$ e dimostriamo che questo linguaggio non è acontestuale, come affermato in Proposizione 4.1 di Cap.1. A tale scopo, supponiamo per assurdo che lo sia e che quindi verifichi il "pumpimg lemma"; esiste quindi una costante H tale che la parola $z=a^H b^H c^H \in L$ con lunghezza $|z|= 3H>H$ può essere decomposta nella forma z=uvwxy tale che $|vwx| \leq H$, $|vx| \geq 1$ e $uv^2wx^2y \in L$. La parola vwx non può contenere sia a che c infatti, se così fosse, dovrebbe contenere anche tutti i b, che sono H (per l'ipotesi $|z|=3H$) e ciò sarebbe in contrasto con l'ipotesti $|vwx| \leq H$. Supponiamo quindi che c non sia contenuto in vwx. Il numero di simboli c contenuti in uv^2wx^2y è allora uguale al numero di simboli c contenuti in z = uvwxy, cioè H. Sappiamo che $uv^2wx^2y \in L$ e che la lunghezza di una parola in L è il triplo del numero di c contenuti, quindi $|uv^2wx^2y| = 3H$. Questo è assurdo, poiché $3H=|uv^2wx^2y|=| uvwxy|+|vx|=3H+|vx|>3H$, essendo $|vx| \geq 1$.

5 Riconoscitori per linguaggi liberi dal contesto

Sappiamo che esistono linguaggi acontestuali, ad esempio $L = \{a^n b^n \mid n \geq 1\}$, non riconoscibili con automi a stati finiti. Il problema è che un automa a stati finiti possiede una memoria limitata dal numero degli stati, e quindi non è in grado di riconoscere il linguaggio $L = \{a^n b^n \mid n \geq 1\}$ per il semplice fatto che non può "contare" il numero di simboli a in una parola per poterne poi operare il confronto col numero di simboli b. Per riconoscere linguaggi liberi dal contesto è allora necessario considerare dispositivi con una memoria aggiuntiva potenzialmente infinita: consideriamo qui il caso della memoria a pila (stack).

Una memoria a pila permette di memorizzare una qualsiasi parola su un alfabeto K. Sulla parola z memorizzata possiamo operare come segue:

1. E' possibile controllare se la parola z è vuota, col predicato ISEMPTY(z) che vale 1 se $z=\varepsilon$ e vale 0 se $z \neq \varepsilon$.
2. E' possibile leggere il primo simbolo in z con l'operazione TOP(z). Se $z=\varepsilon$ allora TOP(z) è indefinita, altrimenti se z è della forma ax con $a \in K$, e vale TOP(ax)=a
3. E' possibile cancellare il primo simbolo in z con l'operazione POP(z). Se $z=\varepsilon$ allora POP(z) è indefinita, altrimenti se z è della forma ax con $a \in K$, e vale POP(ax)=x
4. E' possibile aggiungere un simbolo $a \in K$ in testa a z con l'operazione PUSH(a,z) = az.

La possibilità di costruire riconoscitori per linguaggi liberi dal contesto utilizzando una memoria a pila viene ben evidenziata dalle seguenti considerazioni. Se L è un linguaggio libero dal contesto, allora può essere generato da una grammatica G in forma normale di Greibach, cioè con regole di produzione del tipo $A \rightarrow aW$, dove a è un simbolo terminale e W una parola (eventualmente vuota) di metasimboli.

Sappiamo che possiamo limitarci, senza perdita di generalità, a derivazioni in cui le regole di produzione vengono applicate al metasimbolo "più a sinistra". Applicando, a partire dall'assioma S, le regole al metasimbolo più a sinistra, poiché la grammatica è in forma normale di Greibach si derivano in generale parole della forma: $S \Rightarrow_G^* x_1\ x_2\ \ldots..\ x_m\ X_1\ X_2\ \ldots..\ X_n$, dove $x_1\ x_2\ \ldots..\ x_m$ è una parola di simboli terminali mentre $X_1\ X_2\ \ldots..\ X_n$ è una parola di metasimboli.

L'applicazione alla parola $x_1\ x_2\ \ldots\ x_m\ X_1\ X_2\ \ldots\ X_n$ di una regola del tipo $X_1 \rightarrow aW$ (dove W è una parola eventualmente vuota di metasimboli) porta alla parola $x_1\ x_2\ \ldots\ x_m\ a\ W\ X_2\ \ldots\ X_s$; si osservi che la nuova parola di metasimboli $W\ X_2\ \ldots\ X_s$ è ottenuta dalla parola precedente $X_1\ X_2\ \ldots\ X_s$ operando come segue:

1. Si cancella il primo metasimbolo X_1 (mediante una operazione POP)
2. Si aggiunge la parola W in testa alla pila (mediante eventuali operazioni PUSH)

Si osservi inoltre che la possibilità di applicare "più a sinistra" una eventuale regola del tipo $A \rightarrow aV$ alla parola $x_1\ x_2\ \ldots\ x_m\ X_1\ X_2\ \ldots\ X_n$ richiede solo di applicare a $X_1\ X_2\ \ldots\ X_n$ l'operazione TOP: dopo aver ottenuto $X_1 = TOP(X_1\ X_2\ \ldots\ X_n)$ basta controllare se $X_1=A$. Si osservi infine che una derivazione del tipo $S \Rightarrow_G^* x_1\ x_2\ \ldots\ x_m$, che attesta che $x_1\ x_2\ \ldots\ x_m \in L$, corrisponde al caso in cui $X_1\ X_2\ \ldots\ X_n$ è la parola vuota.

Per le considerazioni fatte, si può costruire un riconoscitore per un linguaggio L generato da una grammatica G in forma normale di Greibach gestendo una pila. Informalmente, il riconoscitore è costituito da un nastro, che contiene la parola $w = x_1\ x_2\ \ldots\ x_m$ da riconoscere, e da una pila, che inizialmente contiene l'assioma S. Il riconoscitore scandisce in ordine, partendo dalla prima, le lettere della parola da riconoscere e ad ogni scansione modifica il contenuto della pila. Più precisamente, se il contenuto della pila è la parola $X_1\ X_2\ \ldots\ X_n$ e il riconoscitore legge il simbolo x_k, allora effettua la seguente mossa:

1. prende in considerazione, se ne esiste almeno una, una regola di G del tipo $X_1 \rightarrow x_k W$
2. cancella dalla pila il simbolo X_1 e pone in testa alla pila la parola W, cosicché il nuovo contenuto della pila sia $W\ X_2\ \ldots\ X_s$.

Si osservi che il riconoscitore così costruito è in generale non deterministico, poiché, riguardo al punto 1., possono esistere più regole con lo stesso metasimbolo a sinistra e lo stesso simbolo terminale a destra. La parola w è accettata se esiste almeno una sequenza di mosse per cui il riconoscitore, dopo aver scandito tutta la parola, ha la pila vuota.

Più formalmente:

Definizione 5.1 Un *riconoscitore a pila* è descritto da un sistema $\Phi = <\Sigma, K, \delta, S>$ dove;

1. Σ è un alfabeto di simboli terminali
2. K è un alfabeto disgiunto da Σ ed S è un elemento di K.
3. δ è una funzione da ΣxK ai sottoinsiemi finiti di K*

L'interpretazione di $\delta(a,A)=\{W_1, \ldots, W_m\}$ è la seguente: è letto il simbolo a e il simbolo in testa alla pila è A, allora nella pila viene cancellato A e inserita in testa una parola W_k scelta in $\{W_1, \ldots, W_m\}$.

Formalmente, una *configurazione* del riconoscitore a pila è una parola X in K* (da interpretare come contenuto della pila). Se $a \in \Sigma$ e $W_k \in \delta(a,A)$, allora porremo:

$$a: AY \Rightarrow W_k Y$$

Se inoltre abbiamo che:

$$a_j: \Lambda_j \Rightarrow \Lambda_{j+1} \qquad \text{per ogni j tra 1 ed n}$$

allora porremo:

$$a_1 a_2\ \ldots\ a_n : \Lambda_1 \Rightarrow^* \Lambda_{n+1}$$

Definizione 5.2: Il linguaggio $L(\Phi)$ riconosciuto (o accettato) dall'automa a pila Φ è il linguaggio:

$$L(\Phi) = \{w \mid w: S \Rightarrow^* \varepsilon\}$$

Vale la seguente:

Proposizione 5.1 L è acontestuale se e solo se è accettato da un riconoscitore a pila.
Dimostrazione: Se L è libero dal contesto, esiste una grammatica G = < S,Q,*P*,S> in forma normale di Greibach che lo genera. Utilizzando G, costruiamo il riconoscitore a pila Φ = <Σ, K, δ, S> dove:

1. K = Q
2. δ(a,A) = {W | A → aW è una regola di produzione di G }

Si prova facilmente per induzione che S $\Rightarrow_G$* w se e solo se, in Φ, vale w: S $\Rightarrow$* ε. Questo implica che L(Φ) =L

Se viceversa L è riconosciuto da un riconoscitore a pila Φ = <Σ, K, δ, S>, allora è generato dalla grammatica G = < S,Q,*P*,S> dove:

1. Q = K
2. A→ aW è una regola di produzione di G se e solo se W$\in$ δ(a,A).

ÿ

Esempio 5.1
Si consideri il linguaggio L formato dalle espressioni generate dalla grammatica con assioma E e regole di produzione:

E→ (E+E) / x/ y

Possiamo trovare una grammatica in forma normale di Greibach, equivalente alla precedente. Tale grammatica ha assioma E e regole di produzione:

E→ **(**EAEB , A→ + , B→ **)** , E→x , E→ y

Da questa grammatica otteniamo l'automa riconoscitore Φ = <{**(** , **)** , x , y , +}, {E,A,B}, δ, E>, dove:
δ(**(** , E) = EAEB , δ(+ , A) = δ(**)** , B) = δ(x , E) =δ(y , E) = ε. La funzione δ è indefinita in ogni altro caso.
Applicando ad esempio il riconoscitore alla parola (x+(y+x)), la parola è accettata perchè:
(: E $\Rightarrow$ EAEB
x : EAEB $\Rightarrow$ AEB
+ : AEB $\Rightarrow$ EB
(: EB $\Rightarrow$ EAEBB
y : EAEBB $\Rightarrow$ AEBB
+ : AEBB $\Rightarrow$ EBB
x : EBB $\Rightarrow$ BB
) : BB $\Rightarrow$ B
) : B $\Rightarrow$ ε

6 Linguaggi di markup

I linguaggi di markup sono utilizzati per la descrizione di documenti. Le parole di questi linguaggi sono i documenti corredati da marcatori, chiamati *tag*, che dovrebbero fornire informazioni su porzioni dei documenti stessi.

6.1 HTML

Un noto linguaggio di markup è l'HTML (Hypertext Markup Language), nato per presentare documenti sul World Wide Web. Esso permette sia di curare la veste grafica (formattazione) di testi, sia di creare collegamenti tra i diversi documenti.

Esempio 6.1
Consideriamo il seguente documento, consistente in un testo seguito da un elenco:

Lista della spesa:
1. zucchero
2. latte
3. pane

La sua descrizione in HTML risulta essere la seguente:

```
<html>
<body>Lista della spesa:
<ol>
    <li>zucchero</li>
    <li>latte</li>
    <li>pane</li>
</ol>
</body>
</html>
```

dove <html>, <body>, <ol>, <li>, </html>, </body>, , sono tag. Essi indicano rispettivamente:
- <html> l'inizio e <\html> la fine di un documento HTML;
- <body> l'inizio e <\body> la fine del testo visibile;
- l'inizio e <\ol> la fine di una lista ordinata;
- l'inizio e <\li> la fine di un elemento di una lista.

I tag standard dell'HTML saranno poi interpretati dai browser che visualizzeranno il documento avendo in ingresso la sua descrizione in HTML.

Non tutte le parole formate da testo e tag sono descrizioni HTML corrette di documenti. In particolare le descrizioni HTML corrette possono essere formalmente generate da una opportuna grammatica acontestuale.

Ad esempio, la porzione di grammatica che permentte di descrivere elenchi numerati è la seguente:

Car → a | A | ...
Testo → ε | Car Testo
Elnum → Lista
Lista → ε | Elemento Lista
Elemento → Testo

E' possibile costruire un *analizzatore sintattico* corrispondente alla grammatica di HTML, cioé un algoritmo che riconosce se una stringa è generata da tale grammatica e, in caso affermativo, ne costruisce l'albero di derivazione. Questo analizzatore è implementato nei vari browser.

6.2 XML e DTD

Un altro linguaggio di markup è l'XML (eXtensible Markup Language). Obiettivo di XML è quello di estendere HTML, dando la possibilità all'autore di un documento di "inventare" tag che veicolano particolari significati in termini di struttura del documento stesso.

Le parole di XML sono essenzialmente sequenze di simboli corrispondenti a parentesizzazioni ben formate. I simboli usati per le parentesi sono un insieme potenzialmente infinito; le parentesi sono classificate in parentesi aperte (tag di inizio) o chiuse (tag di fine). Esse possono essere date dalla seguente grammatica:

Car → a | A | ...
nome-tag → Car nome-tag | ε
parentesi-aperta → <nome-tag>
parentesi-chiusa → </nome-tag>

<x> è la parentesi aperta il cui corrispettivo è la parentesi chiusa </x>.

Esempio 6.2

- <mail> è la parentesi aperta il cui corrispettivo è la parentesi chiusa </mail>;
- <da> è la parentesi aperta il cui corrispettivo è la parentesi chiusa </da>.

Un documento XML si dice *ben formato* se le parentesi soddisfano le seguenti condizioni:

1. Esiste S tale che tutto il documento è contenuto tra <S> e </S>;
2. Ogni parentesi aperta <x> deve essere seguita dalla chiusa </x>;
3. Le parentesi devono essere annidate correttamente: se <x> è aperta prima di <y>, prima dovrò chiudere con la parentesi </y> e poi con la parentesi </x>.

L'insieme dei documenti ben formati non è un linguaggio acontestuale, poiché le potenziali parentesi sono infinite. Tuttavia, fissando a priori un insieme *finito* di parentesi, il linguaggio dei documenti ben formati limitati a quelle parentesi risulta essere acontestuale.

Esempio 6.3

Consideriamo solo due parentesi differenti: <a>, e , che per semplicità riscriveremo come: (,), {, }.

- "( ) { }" non è ben formato (non soddisfa la condizione 1);
- "({))" non è ben formato (non soddisfa la condizione 2);
- "({) }" non è ben formato (non soddisfa la condizione 3);
- "({ } ( ))" è ben formato (soddisfa le condizioni 1, 2 e 3).

I documenti ben formati con le parentesi dell'esempio 6.3 formano il linguaggio acontestuale generato dalla seguente grammatica:

S → (X)
X → (A | { B
A → (AA | { BA |)
B → { BB | (AB | }

In questo esempio, riducendo la grammatica in forma di Greibach e costruendo l'automa a pila corrispondente si ottiene un automa deterministico che permette di eseguire l'analisi sintattica di un testo in tempo lineare.

Come abbiamo visto, sia i documenti HTML che le parentesizzazioni ben formate (limitate a un numero finito di parentesi) di XML sono linguaggi acontestuali e quindi descrivibili da grammatiche di tipo 2. Questo non è in sé stupefacente, poiché praticamente tutti i linguaggi di programmazione possono essere essenzialmente descritti da grammatiche acontestuali. In XML, tuttavia, le grammatiche acontestuali giocano un ruolo di maggior impegno. E' infatti possibile specificare sottoclassi di documenti ben formati utilizzando grammatiche di tipo 2.

Un documento t sarà detto *corretto*, se è una parentesizzazione ben formata; ulteriormente sarà detto *valido* rispetto alla grammatica G se è generato dalla grammatica G stessa. Il documento assumerà quindi la forma <G, t>, dovendosi specificare contemporaneamente la grammatica e il testo.

La possibilità di una trattazione automatica alla validazione del documento è offerta da un algoritmo che, avendo in ingresso la grammatica G e il testo t, decide se t è generato da G e, in tal caso, fornisce l'albero di derivazione.

La grammatica G, descritta con opportuna notazione, viene detta DTD (Document Type Definition).
Le regole di produzione sono del tipo:

1. <!ELEMENT nome-tag (descrizione) >,
   oppure
2. <!ELEMENT nome-tag (#PCDATA) > .

Nel primo caso nome-tag è un metasimbolo, mentre (descrizione) è una stringa di metasimboli intercalati dalla virgola, la quale indica il prodotto di concatenazione. La regola di produzione denotata è la seguente:

1. nome-tag → <nome-tag> descrizione </nome-tag>

Esempio 6.4
La seguente regola <!ELEMENT mail (a, da, oggetto, testo) > denota la regola di produzione:
mail → <mail> a da oggetto testo </mail >.

Ogni metasimbolo in descrizione deve comparire sulla sinistra di una qualche regola di produzione del DTD. Ad esempio, nel DTD che contiene la regola per mail dell'esempio 6.4 dovranno essere presenti regole per i metasimboli a, da, oggetto e testo.
In generale, (descrizione) può essere qualcosa di più complicato che una semplice parola di metasimboli; ad esempio è possibile esprimere un'unione di parole mediante il simbolo |, e sono anche consentiti i simboli *, +, ?, con l'usuale significato visto per le espressioni regolari in Unix. Pertanto è possibile definire regole di produzione in cui le parti destre risultano particolari espressioni regolari.

Nel secondo caso nome-tag è, come prima, un metasimbolo, e #PCDATA indica un testo Testo che non contiene tag. La regola di produzione denotata è la seguente:

2. nome-tag → <nome-tag> Testo </nome-tag>

Esempio 6.5
La seguente regola <!ELEMENT oggetto (#PCDATA) > denota la regola di produzione:
oggetto → <oggetto> Testo </oggetto >, dove Testo è una sequenza di caratteri priva di tag.

Qui ci limiteremo a considerare solo una sottoclasse di DTD caratterizzati da regole di tipo 1. o 2.

Esempio 6.6
Il seguente file mail.xml è un esempio di documento XML:

```
<?xml version="1.0"?>
<mail>
<a>Claudia</a>
<da>Maurizio</da>
<oggetto>Invito a cena</oggetto>
<testo>Sei libera questa sera?</testo>
</mail>
```

L'istruzione <?xml version="1.0"?> indica la versione della specifica XML rispetto alla quale mail.xml è stato scritto.
Un DTD per mail.xml è dato dal seguente file mail.dtd:

```
<!ELEMENT mail    (a, da, oggetto, testo) >
<!ELEMENT a       (#PCDATA) >
<!ELEMENT da      (#PCDATA) >
<!ELEMENT oggetto (#PCDATA) >
<!ELEMENT testo   (#PCDATA) >
```

Per quanto detto precedentemente, l'analizzattore sintattico validante di XML richiede in ingresso la coppia <mail.dtd, mail.xml>. Ciò può essere fatto riportando all'interno del file mail.xml il contenuto di mail.dtd, oppure inserendo nel file mail.dtd l'istruzione:

<!DOCTYPE mail SYSTEM "mail.dtd">.

Inoltre è possibile associare fogli di stile (file CSS) ai documenti XML. I fogli di stile consentono di specificare la formattazione dei documneti. Con l'istruzione:

<?xml-stylesheet type="text/css" href="mail.css" ?>

associamo al documento mail.xml il seguente foglio di stile mail.css:

```
a     {
      display: block;
      text-align: center;
      }
da    {
      display: block;
      text-align: center;
      color: white;
      background-color: green;
      }
```

```
oggetto {
      display: block;
      font-weight: bold;
      }
testo {
      display: block;
      border-color: red;
      border-style: solid;
      }
```

in cui per ogni metasimbolo viene indicato tra parentesi graffe il formato di visualizzazione.

Ad esempio "da { display: block; text-align: center; color: white; background-color: green;}" consente di visualizzare il testo contenuto tra <da> e </da> in un blocco separato, dove il testo è centrato, il carattere bianco e lo sfondo verde. Di seguito viene riportato il browser Netscape e l'immagine riprodotta: