Grammatiche e Linguaggi Liberi dal Contesto

- Abbiamo visto che molti linguaggi non sono regolari. Consideriamo allora classi piu' grandi di linguaggi.

- *Linguaggi Liberi dal Contesto* (CFL = Context-Free Languages) sono stati usati nello studio dei linguaggi naturali dal 1950, e nello studio dei (generatori di) compilatori dal 1960.

- Le *grammatiche libere dal contesto* (CFG = Context-Free Grammars) sono la base della sintassi BNF (Backus-Naur-Form), usate per i linguaggi di programmazione.

- Oggi i CFL sono importanti anche per XML.

Studieremo: CFG, i linguaggi che generano, gli alberi sintattici, gli automi a pila, e le proprieta' di chiusura dei CFL.

Esempio informale di CFG

Consideriamo $L_{pal} = \{w \in \Sigma^* : w = w^R\}$ palindrome

Per esempio: $\texttt{otto} \in L_{pal}$, $\texttt{ara} \in L_{pal}$.

Sia $\Sigma = \{0, 1\}$ e supponiamo che L_{pal} sia regolare.

Sia n dato dal pumping lemma. Allora $0^n10^n \in L_{pal}$. Nel leggere 0^n il FA deve passare per un loop. Se omettiamo il loop, contraddizione.

Definiamo L_{pal} induttivamente:

Base: ϵ, 0, e 1 sono palindromi.

Induzione: Se w e' una palindrome, anche $0w0$ e $1w1$ lo sono.

Nessun altra stringa e' una palindrome.

Le CFG sono un modo formale per definizioni come quella per L_{pal}.

1. $S \rightarrow \epsilon$
2. $S \rightarrow 0$
3. $S \rightarrow 1$
4. $S \rightarrow 0S0$
5. $S \rightarrow 1S1$

0 e 1 sono *terminali*

S e' una *categoria sintattica* (o, piu' tecnicamente, *variabile*)

S e' in questa grammatica anche la categoria sintattica *iniziale*.

1–5 sono *produzioni* (o *regole*)

Definizione formale di CFG

Una *grammatica libera dal contesto* e' una quadrupla

$$G = (V, T, P, S)$$

dove

V e' un insieme finito di *variabili* (o *non-terminali*).

T e' un insieme finito di *terminali*.

P e' un insieme finito di *produzioni* della forma $A \to \alpha$, dove A e' una variabile e $\alpha \in (V \cup T)^*$

S e' una variabile distinta chiamata *variabile iniziale*.

Esempio: $G_{pal} = (\{S\}, \{0, 1\}, P, S)$, dove $P =$
$\{S \to \epsilon, S \to 0, S \to 1, S \to 0S0, S \to 1S1\}$.

A volte raggruppiamo le produzioni con la stessa testa: $P =$
$\{S \to \epsilon|0|1|0S0|1S1\}$.

Esempio: espressioni (semplici) in un tipico linguaggio di programmazione. Gli operatori sono $+$ e $*$, e gli operandi sono identificatori, cioe' stringhe in $L((\boldsymbol{a}+\boldsymbol{b})(\boldsymbol{a}+\boldsymbol{b}+\mathbf{0}+\mathbf{1})^*)$

Le espressioni sono definite dalla grammatica

$$G = (\{E, I\}, T, P, E)$$

dove $T = \{+, *, (,), a, b, 0, 1\}$ e P e' il seguente insieme di produzioni:

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Derivazioni usando le grammatiche

Sia $G = (V, T, P, S)$ una CFG, $A \in V$, $\{\alpha, \beta\} \subset (V \cup T)^*$, e $A \to \gamma \in P$.

Allora scriviamo

$$\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$$

o, se e' ovvia la G,

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

e diciamo che da $\alpha A \beta$ *si deriva* $\alpha \gamma \beta$.

Definiamo $\overset{*}{\Rightarrow}$ la chiusura riflessiva e transitiva di $\Rightarrow$, cioe':

Base: Sia $\alpha \in (V \cup T)^*$. Allora $\alpha \overset{*}{\Rightarrow} \alpha$.

Induzione: Se $\alpha \overset{*}{\Rightarrow} \beta$, e $\beta \Rightarrow \gamma$, allora $\alpha \overset{*}{\Rightarrow} \gamma$.

Esempio: Derivazione di $a*(a+b00)$ da E nella grammatica delle espressioni:

$$E \Rightarrow E*E \Rightarrow I*E \Rightarrow a*E \Rightarrow a*(E) \Rightarrow$$

$$a*(E+E) \Rightarrow a*(I+E) \Rightarrow a*(a+E) \Rightarrow a*(a+I) \Rightarrow$$

$$a*(a+I0) \Rightarrow a*(a+I00) \Rightarrow a*(a+b00)$$

Nota: ad ogni passo potremmo avere varie regole tra cui scegliere, ad esempio
$I*E \Rightarrow a*E \Rightarrow a*(E)$, oppure
$I*E \Rightarrow I*(E) \Rightarrow a*(E)$.

Nota: non tutte le scelte portano a derivazioni di una particolare stringa, per esempio

$$E \Rightarrow E+E$$

non ci fa derivare $a*(a+b00)$.

Derivazioni a sinistra e a destra

Derivazione a sinistra $\underset{lm}{\Rightarrow}$: rimpiazza sempre la variabile piu' a sinistra con il corpo di una delle sue regole.

Derivazione a destra $\underset{rm}{\Rightarrow}$: rimpiazza sempre la variabile piu' a destra con il corpo di una delle sue regole.

Der. a sinistra: quella del lucido precedente.

A destra:

$$E \underset{rm}{\Rightarrow} E * E \underset{rm}{\Rightarrow}$$

$$E * (E) \underset{rm}{\Rightarrow} E * (E + E) \underset{rm}{\Rightarrow} E * (E + I) \underset{rm}{\Rightarrow} E * (E + I0)$$

$$\underset{rm}{\Rightarrow} E * (E + I00) \underset{rm}{\Rightarrow} E * (E + b00) \underset{rm}{\Rightarrow} E * (I + b00)$$

$$\underset{rm}{\Rightarrow} E * (a + b00) \underset{rm}{\Rightarrow} I * (a + b00) \underset{rm}{\Rightarrow} a * (a + b00)$$

Possiamo concludere che $E \overset{*}{\underset{rm}{\Rightarrow}} a * (a + b00)$

Il linguaggio di una grammatica

Se $G(V,T,P,S)$ e' una CFG, allora il *linguaggio di* G e'

$$L(G) = \{w \in T^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

cioe' l'insieme delle stringhe su T^* derivabili dal simbolo iniziale.

Se G e' una CFG, chiameremo $L(G)$ un *linguaggio libero dal contesto*.

Esempio: $L(G_{pal})$ e' un linguaggio libero dal contesto.

Alberi sintattici

- Se $w \in L(G)$, per una CFG, allora w ha un *albero sintattico*, che ci dice la **struttura (sintattica)** di w.

- w potrebbe essere un programma, una query SQL, un documento XML, ...

- Gli alberi sintattici sono una **rappresentazione alternativa alle derivazioni**.

- Ci possono essere diversi alberi sintattici per la stessa stringa.

- Idealmente ci dovrebbe essere **solo un albero sintattico** (la "vera" struttura), cioè la CFG dovrebbe essere **non ambigua**.

- Sfortunatamente, non sempre possiamo rimuovere l'ambiguita'.

Costruzione di un albero sintattico

Sia $G = (V, T, P, S)$ una CFG. Un albero e' un *albero sintattico* per G se:

1. Ogni **nodo interno** e' etichettato con una **variabile** in V.
2. Ogni foglia e' etichettata con un simbolo in $V \cup T \cup \{\epsilon\}$. Ogni foglia etichettata con ϵ e' l'unico figlio del suo genitore.
3. Se un nodo interno e' etichettato A, e i suoi figli (da sinistra a destra) sono etichettati

   $$X_1, X_2, \ldots, X_k,$$

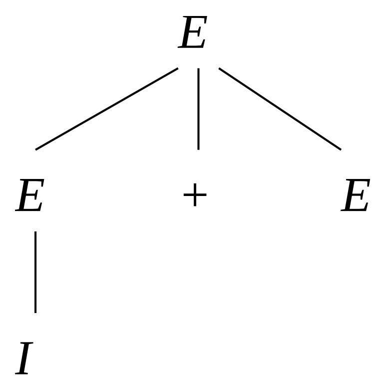
   allora $A \to X_1 X_2 \ldots X_k \in P$.

Esempio: nella grammnatica

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

$\vdots$

il seguente e' un albero sintattico:

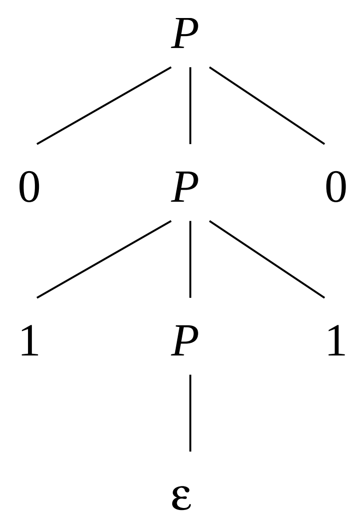


Questo albero sintattico mostra la derivazione $E \stackrel{*}{\Rightarrow} I + E$

Esempio: nella grammatica

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

il seguente e' un albero sintattico:



Mostra la derivazione $P \stackrel{*}{\Rightarrow} 0110$.

Il prodotto di un albero sintattico

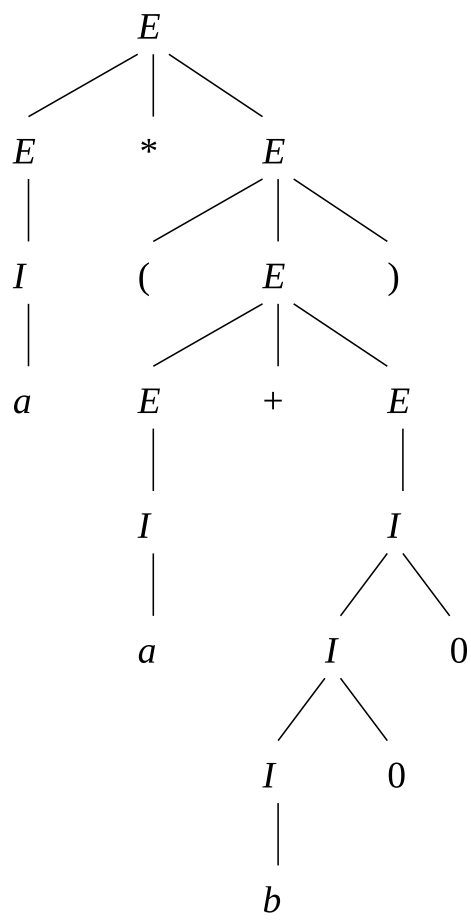
Il *prodotto* di un albero sintattico e' la **stringa di foglie da sinistra a destra**.

Importanti sono quegli alberi sintattici dove:

1. Il prodotto e' **una stringa terminale**.
2. La radice e' etichettata dal **simbolo iniziale**.

L'insieme dei **prodotti di questi alberi sintattici** e' il **linguaggio della grammatica**.

Esempio:



Il prodotto e' $a * (a + b00)$.

Sia $G = (V, T, P, S)$ una CFG, e $A \in V$. I seguenti sono equivalenti:

1. $A \stackrel{*}{\Rightarrow} w$
2. $A \underset{lm}{\stackrel{*}{\Rightarrow}} w$, e $A \underset{rm}{\stackrel{*}{\Rightarrow}} w$
3. C'e' un albero sintattico di G con radice A e prodotto w.

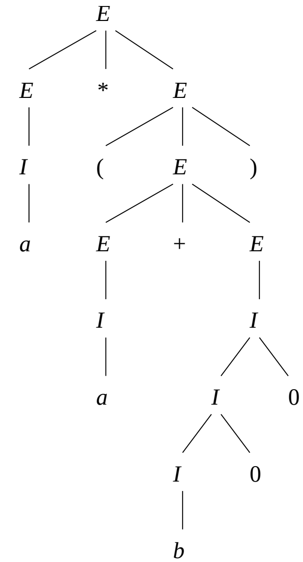
Per provare l'equivalenza, usiamo il seguente piano:

- Dagli alberi alle derivazioni a sinistra (destra): visito l'albero da sinistra a destra (da destra a sinistra)

- Una derivazione sinistra (o destra) è anche una derivazione

- Leggendo la derivazione costruisco l'albero derivazione = derivazione canonica

Esempio: Costruiamo la derivazione a sinistra per l'albero



Supponiamo di aver induttivamente costruito la deriv. a sinistra

$$E \underset{lm}{\Rightarrow} I \underset{lm}{\Rightarrow} a$$

corrispondente al sottoalbero piu' a sinistra, e la deriv. a sinistra

$$E \underset{lm}{\Rightarrow} (E) \underset{lm}{\Rightarrow} (E+E) \underset{lm}{\Rightarrow} (I+E) \underset{lm}{\Rightarrow} (a+E) \underset{lm}{\Rightarrow}$$

$$(a+I) \underset{lm}{\Rightarrow} (a+I0) \underset{lm}{\Rightarrow} (a+I00) \underset{lm}{\Rightarrow} (a+b00)$$

corrispondente al sottoalbero piu' a destra.

Per la derivazione corrispondente all'intero albero, iniziamo con $E \underset{lm}{\Rightarrow} E * E$ e espandiamo la prima E con la prima derivazione e la seconda E con la seconda derivazione:

$$
\begin{aligned}
& E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} \\
& I * E \underset{lm}{\Rightarrow} \\
& a * E \underset{lm}{\Rightarrow} \\
& a * (E) \underset{lm}{\Rightarrow} \\
& a * (E + E) \underset{lm}{\Rightarrow} \\
& a * (I + E) \underset{lm}{\Rightarrow} \\
& a * (a + E) \underset{lm}{\Rightarrow} \\
& a * (a + I) \underset{lm}{\Rightarrow} \\
& a * (a + I0) \underset{lm}{\Rightarrow} \\
& a * (a + I00) \underset{lm}{\Rightarrow} \\
& a * (a + b00)
\end{aligned}
$$

Ambiguita' in Grammatiche e Linguaggi

Nella grammatica

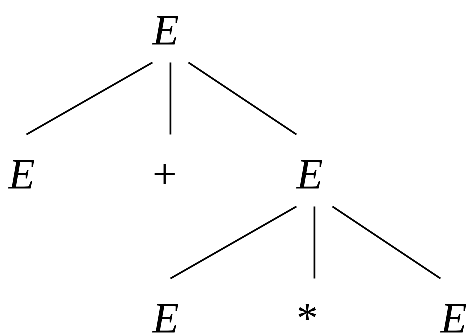
1. $E \to I$
2. $E \to E + E$
3. $E \to E * E$
4. $E \to (E)$

$\cdots$

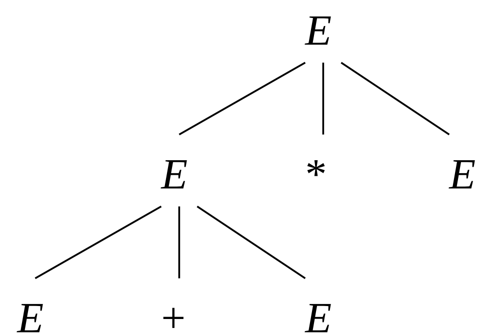
$E + E * E$ ha due derivazioni:

$$E \Rightarrow E + E \Rightarrow E + E * E \quad \text{e} \quad E \Rightarrow E * E \Rightarrow E + E * E$$

Questo ci da' due alberi sintattici:



(a)



(b)

L'esistenza di varie *derivazioni* di per se non e' pericolosa, e' l'esistenza di vari alberi sintattici che rovina la grammatica.

Esempio: Nella stessa grammatica

$$
\begin{aligned}
&5.\ I \rightarrow a\\
&6.\ I \rightarrow b\\
&7.\ I \rightarrow Ia\\
&8.\ I \rightarrow Ib\\
&9.\ I \rightarrow I0\\
&10.\ I \rightarrow I1
\end{aligned}
$$

la stringa $a + b$ ha varie derivazioni:

$$E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b$$

e

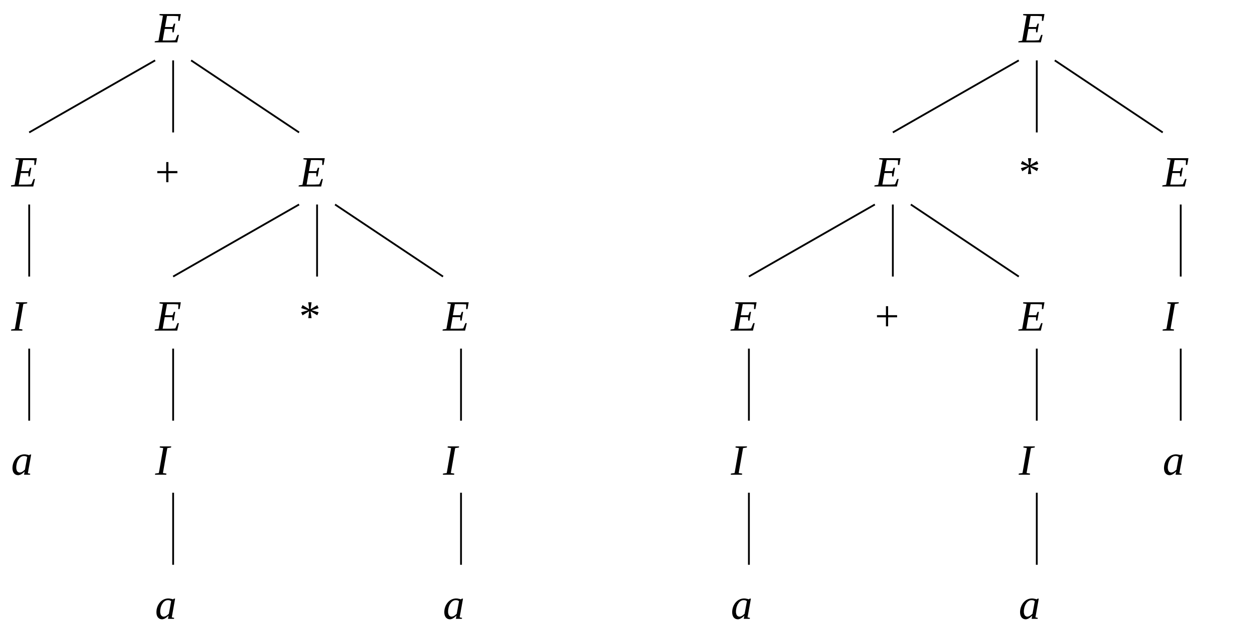
$$E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b$$

Pero' il loro albero sintattico e' lo stesso (anche per le altre possibili derivazioni di $a + b$): la struttura di $a + b$ e' quindi non ambigua.

Definizione: Sia $G = (V, T, P, S)$ una CFG. Diciamo che G e' *ambigua* se esiste una stringa in T^* che ha piu' di un albero sintattico.

Se ogni stringa in $L(G)$ ha un unico albero sintattico, G e' detta *non-ambigua*.

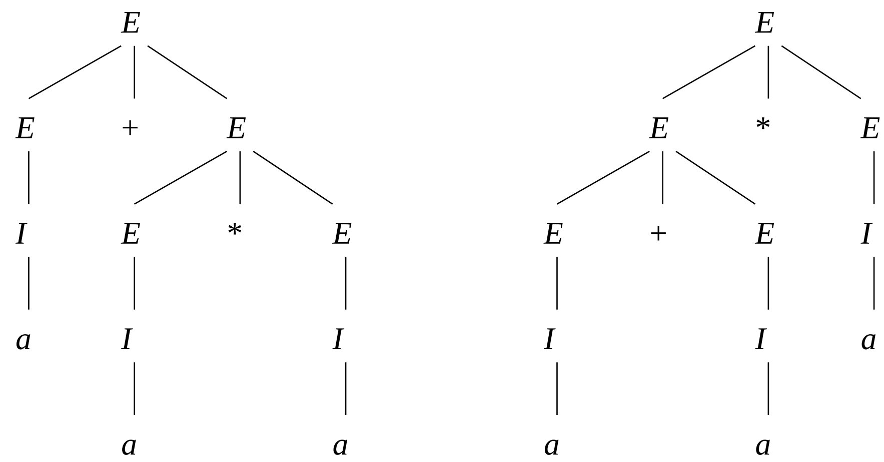
Esempio: La stringa terminale $a + a * a$ ha due alberi sintattici:



(a) (b)

Derivazioni a sinistra e ambiguita'

I due alberi sintattici per $a + a * a$



(a) (b)

danno luogo a due derivazioni:

$$E \underset{lm}{\Rightarrow} E + E \underset{lm}{\Rightarrow} I + E \underset{lm}{\Rightarrow} a + E \underset{lm}{\Rightarrow} a + E * E$$
$$\underset{lm}{\Rightarrow} a + I * E \underset{lm}{\Rightarrow} a + a * E \underset{lm}{\Rightarrow} a + a * I \underset{lm}{\Rightarrow} a + a * a$$

e

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} E + E * E \underset{lm}{\Rightarrow} I + E * E \underset{lm}{\Rightarrow} a + E * E$$
$$\underset{lm}{\Rightarrow} a + I * E \underset{lm}{\Rightarrow} a + a * E \underset{lm}{\Rightarrow} a + a * I \underset{lm}{\Rightarrow} a + a * a$$

In generale:

- Ad un albero sintattico corrispondono molte derivazioni, ma

- ad ogni (diverso) albero sintattico corrisponde un'unica (diversa) derivazione *a sinistra*.

- ad ogni (diverso) albero sintattico corrisponde un'unica (diversa) derivazione *a destra*.

Teorema 5.29: Data una CFG G, una stringa terminale w ha due distinti alberi sintattici se e solo se w ha due distinte derivazioni a sinistra dal simbolo iniziale.

Rimuovere l'ambiguita' dalle grammatiche

Buone notizie: a volte possiamo rimuovere l'ambiguita'

Cattive notizie: non c'e' nessun algoritmo per farlo in modo sistematico

Ancora cattive notizie: alcuni CFL hanno solo CFG ambigue

Studiamo la grammatica

$$E \rightarrow I \mid E+E \mid E*E \mid (E)$$
$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

Bisogna modificarla in modo da stabilire:

1. Chi ha precedenza tra * e +

2. Come si raggruppano sequenze di uno stesso operatore: $E+E+E$ e' inteso come $E+(E+E)$ o come $(E+E)+E$?

Soluzione: Introduciamo una gerarchica di variabili che stabilisca un ordine di precedenza tra gli operatori

1. *espressioni* E: composizioni di uno o più *termini* T tramite $+$

2. *termini* T: composizioni di uno o più *fattori* F tramite $*$

3. *fattori* F:

    (a) *identificatori* I

    (b) *espressioni* E racchiuse tra parentesi

I termini T non possono generare $+$ che siano fuori da parentesi: questa gerarchia stabilisce che $*$ **ha precedenza rispetto a** $+$.

Esempio: l'unico modo di generare $a+a*a$, visto in precedenza, e' considerando $a*a$ come un termine T (albero sintattico di sinistra)

Formalmente:

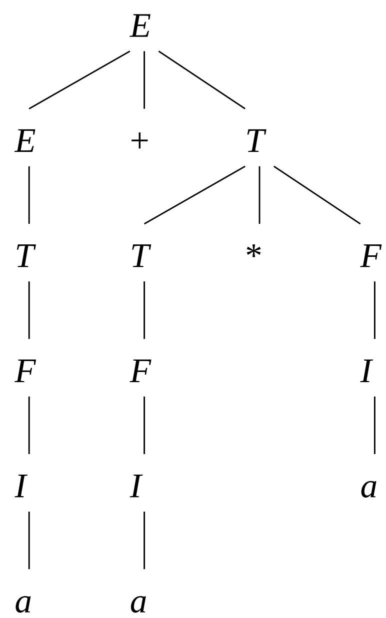
1. $E \rightarrow T \mid E + E$
2. $T \rightarrow F \mid T * T$
3. $F \rightarrow I \mid (E)$
4. $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Questa grammatica e' non ambigua?

No! Dobbiamo anche imporre un ordine per raggruppamento operatori allo stesso livello. Es con associativita' a sinistra:

1. $E \rightarrow T \mid E + T$
2. $T \rightarrow F \mid T * F$
3. $F \rightarrow I \mid (E)$
4. $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Grammatica non ambigua, es. unico albero sintattico di $a + a * a$ è



Ambiguita' inerente

Un CFL L e' *inerentemente ambiguo* se *tutte* le grammatiche per L sono ambigue.

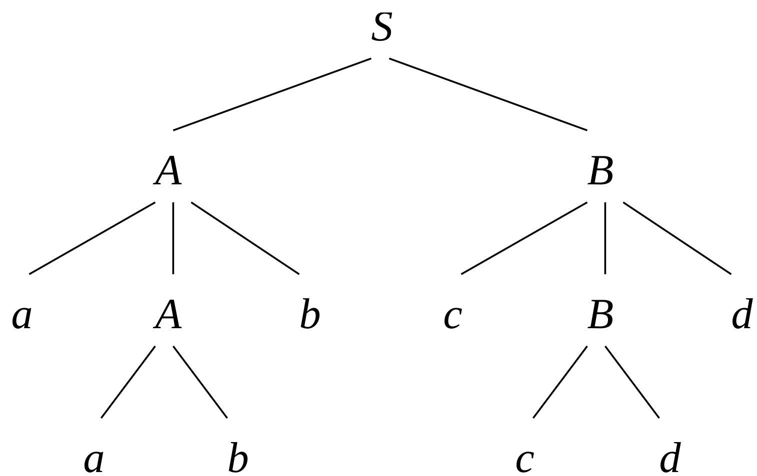
Esempio: Consideriamo $L =$

$$\{a^n b^n c^m d^m : n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\}.$$

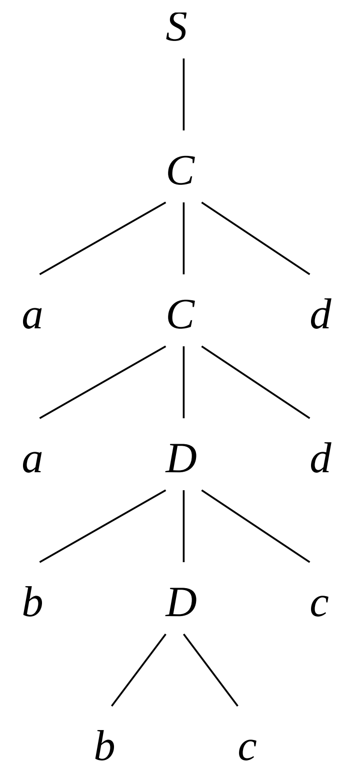
Una grammatica per L e'

$$\begin{aligned}
S &\rightarrow AB \mid C \\
A &\rightarrow aAb \mid ab \\
B &\rightarrow cBd \mid cd \\
C &\rightarrow aCd \mid aDd \\
D &\rightarrow bDc \mid bc
\end{aligned}$$

Guardiamo la struttura sintattica della stringa $aabbccdd$.



(a)



(b)

Vediamo che ci sono due derivazioni a sinistra:

$$S \underset{lm}{\Rightarrow} AB \underset{lm}{\Rightarrow} aAbB \underset{lm}{\Rightarrow} aabbB \underset{lm}{\Rightarrow} aabbcBd \underset{lm}{\Rightarrow} aabbccdd$$

e

$$S \underset{lm}{\Rightarrow} C \underset{lm}{\Rightarrow} aCd \underset{lm}{\Rightarrow} aaDdd \underset{lm}{\Rightarrow} aabDcdd \underset{lm}{\Rightarrow} aabbccdd$$

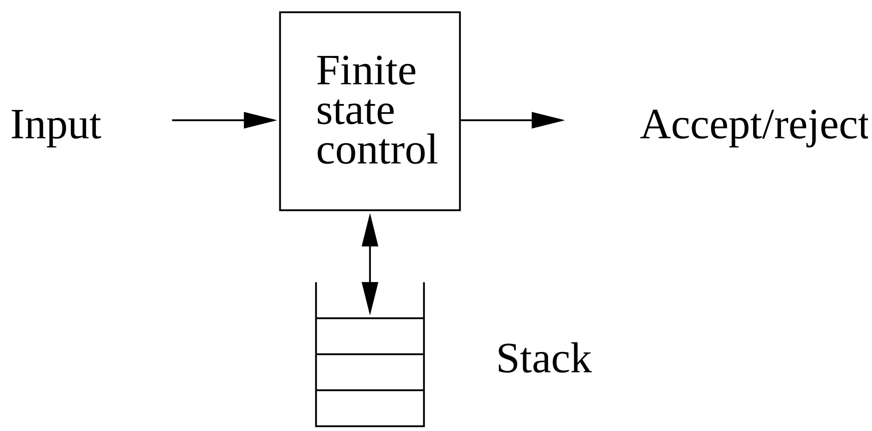
Puo' essere provato che *ogni* grammatica per L si comporta come questa. Il linguaggio L e' quindi inerentemente ambiguo.

Automi a pila

Un automa a pila (PDA) e' in pratica un ϵ-NFA con una pila.

In una transizione un PDA:

1. Consuma un simbolo di input o esegue una transizione ϵ.
2. Va in un nuovo stato (o rimane dove e').
3. Rimpiazza il top della pila con una stringa (consuma il carattere in cima, e mette al suo posto una stringa, eventualmente vuota o uguale al carattere consumato lasciando quindi la pila inalterata)



Esempio: Consideriamo

$$L_{wwr} = \{ww^R : w \in \{0,1\}^*\},$$

con "grammatica" $P \to 0P0$, $P \to 1P1$, $P \to \epsilon$. Un PDA per L_{wwr} ha **tre stati**, e funziona come segue:

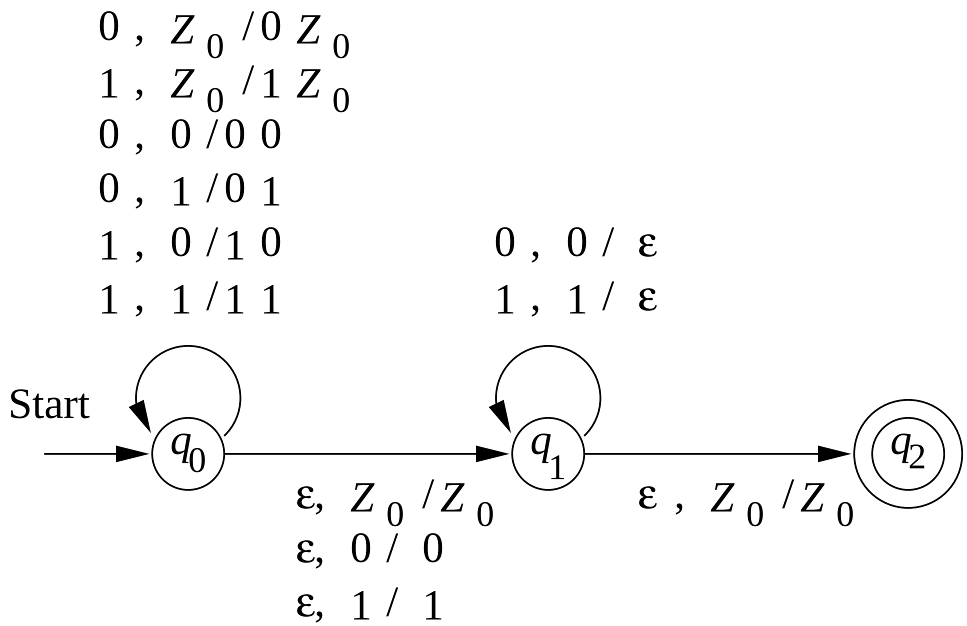
1. Legge w un simbolo alla volta, rimanendo nello stato q_0, e aggiungendo il simbolo di input alla pila.

2. Decide non deterministicamente che sta nel mezzo di ww^R e va nello stato q_1.

3. Legge w^R un simbolo alla volta e lo paragona col simbolo al top della pila: Se sono uguali, fa un pop della pila, e rimane nello stato q_1. Se non sono uguali, si blocca.

4. Se la pila non ha piu' simboli (0 o 1), va nello stato q_2 e accetta.

Il PDA per L_{wwr} come diagramma di transizione:



Definizione formale di PDA

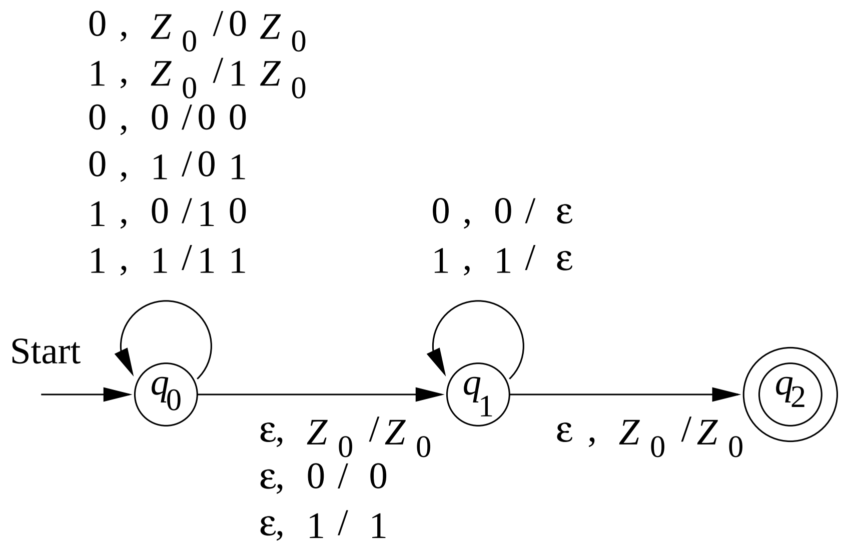
Un PDA e' una tupla di 7 elementi:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

dove

- Q e' un insieme finito di stati,
- Σ e' un *alfabeto finito di input,*
- Γ e' un *alfabeto finito di pila,*
- δ e' una *funzione di transizione* da $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ a sottinsiemi di $Q \times \Gamma^*$,
- q_0 e' lo *stato iniziale,*
- $Z_0 \in \Gamma$ e' il *simbolo iniziale* per la pila, e
- $F \subseteq Q$ e' l'insieme di *stati di accettazione.*

Esempio: Il PDA



e' la 7-tupla

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\}),$$

dove δ e' data dalla tabella seguente:

	$0, Z_0$	$1, Z_0$	0,0	0,1	1,0	1,1	ϵ, Z_0	$\epsilon, 0$	$\epsilon, 1$
$\rightarrow q_0$	$\{(q_0, 0Z_0)\}$	$\{(q_0, 1Z_0)\}$	$\{(q_0, 00)\}$	$\{(q_0, 01)\}$	$\{(q_0, 10)\}$	$\{(q_0, 11)\}$	$\{(q_1, Z_0)\}$	$\{(q_1, 0)\}$	$\{(q_1, 1)\}$
q_1	$\emptyset$	$\emptyset$	$\{(q_1, \epsilon)\}$	$\emptyset$	$\emptyset$	$\{(q_1, \epsilon)\}$	$\{(q_2, Z_0)\}$	$\emptyset$	$\emptyset$
$\star q_2$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Descrizioni istantanee

Un PDA passa da una configurazione ad un'altra configurazione:

- consumando un simbolo di input (o tramite transizione ϵ),
- consumando la cima dello stack sostituendolo con una stringa (eventualmente vuota).

Per ragionare sulle computazioni dei PDA, usiamo delle *descrizioni istantanee* (ID) del PDA. Una ID e' una tripla

$$(q, w, \gamma)$$

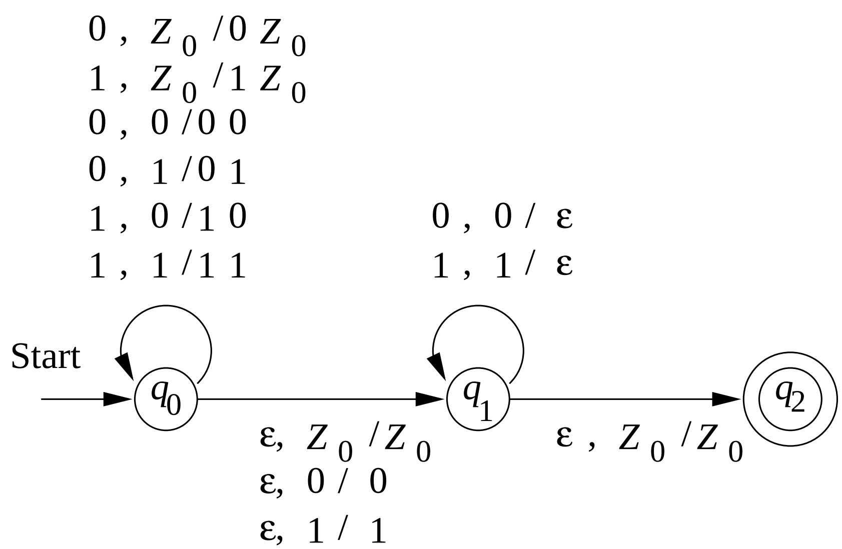
dove q e' lo stato, w l'input rimanente, e γ il contenuto della pila.

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Allora $\forall w \in \Sigma^*, \beta \in \Gamma^*$:

$$(p, \alpha) \in \delta(q, a, X) \Rightarrow (q, aw, X\beta) \vdash (p, w, \alpha\beta).$$

Definiamo $\overset{*}{\vdash}$ la chiusura riflessiva e transitiva di $\vdash$.

Esempio: Su input 1111 il PDA



ha le seguenti sequenze di computazioni:

$$
\begin{tikzcd}
(q_0, 1111, Z_0) \arrow[d] \arrow[dr] & & \\
(q_0, 111, 1Z_0) \arrow[d] \arrow[dr] & (q_1, 1111, Z_0) \arrow[r] & (q_2, 1111, Z_0) \\
(q_0, 11, 11Z_0) \arrow[d] \arrow[dr] & (q_1, 111, 1Z_0) \arrow[r] & (q_1, 11, Z_0) \arrow[d] \\
(q_0, 1, 111Z_0) \arrow[d] \arrow[dr] & (q_1, 11, 11Z_0) \arrow[dr] & (q_2, 11, Z_0) \\
(q_0, \varepsilon, 1111Z_0) \arrow[d] & (q_1, 1, 111Z_0) \arrow[d] & (q_1, 1, 1Z_0) \arrow[d] \\
(q_1, \varepsilon, 1111Z_0) & (q_1, \varepsilon, 11Z_0) & (q_1, \varepsilon, Z_0) \arrow[d] \\
 & & (q_2, \varepsilon, Z_0)
\end{tikzcd}
$$

Accettazione per stato finale

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Il *linguaggio accettato da P per stato finale* e'

$$L(P) = \{w : (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \alpha),\ q \in F\}.$$

Esempio: Il PDA di prima accetta esattamente L_{wwr}.

Accettazione per pila vuota

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Il *linguaggio accettato da P per pila vuota* e'

$$N(P) = \{w : (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon)\}.$$

Nota: q puo' essere uno stato qualunque.

Domanda: come modificare il PDA per ww^R per accettare lo stesso linguaggio per pila vuota?

Stringa tutta letta e pila vuota accetta, non ci interessa dello stato

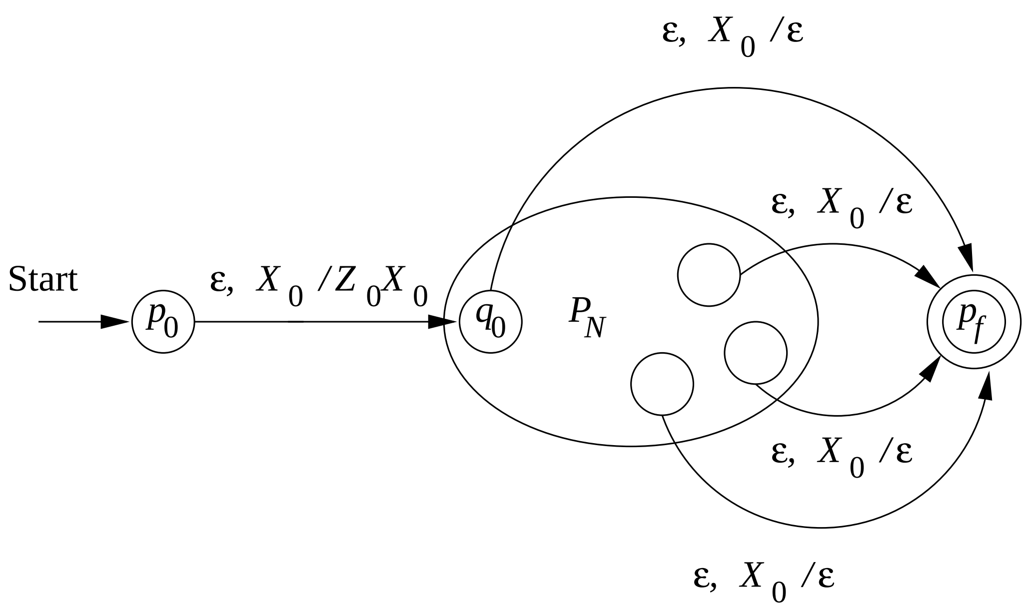
Da pila vuota a stato finale

Teorema 6.9: Se $L = N(P_N)$ per un PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, allora $\exists$ PDA P_F, tale che $L = L(P_F)$.

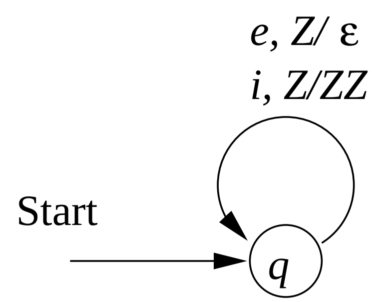
Prova: Sia

$$P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$$

dove $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$, e per ogni $q \in Q, a \in \Sigma \cup \{\epsilon\}, Y \in \Gamma$: $\delta_F(q, a, Y) = \delta_N(q, a, Y)$, e inoltre $(p_f, \epsilon) \in \delta_F(q, \epsilon, X_0)$.



Consideriamo il seguente automa a pila:



Formalmente,

$$P_N = (\{q\}, \{i, e\}, \{Z\}, \delta_N, q, Z),$$

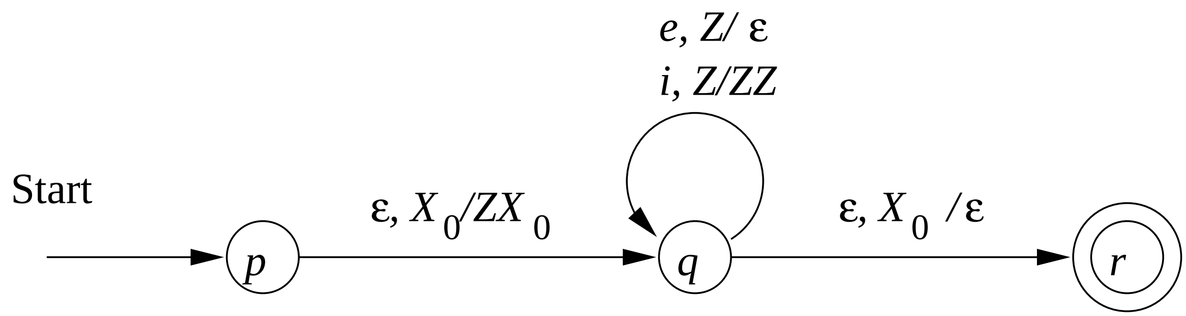
dove $\delta_N(q, i, Z) = \{(q, ZZ)\}$, e $\delta_N(q, e, Z) = \{(q, \epsilon)\}$.

Da P_N possiamo costruire

$$P_F = (\{p, q, r\}, \{i, e\}, \{Z, X_0\}, \delta_F, p, X_0, \{r\}),$$

dove
$\delta_F(p, \epsilon, X_0) = \{(q, ZX_0)\}$,
$\delta_F(q, i, Z) = \delta_N(q, i, Z) = \{(q, ZZ)\}$,
$\delta_F(q, e, Z) = \delta_N(q, e, Z) = \{(q, \epsilon)\}$, and
$\delta_F(q, \epsilon, X_0) = \{(r, \epsilon)\}$

Il diagramma per P_F e'



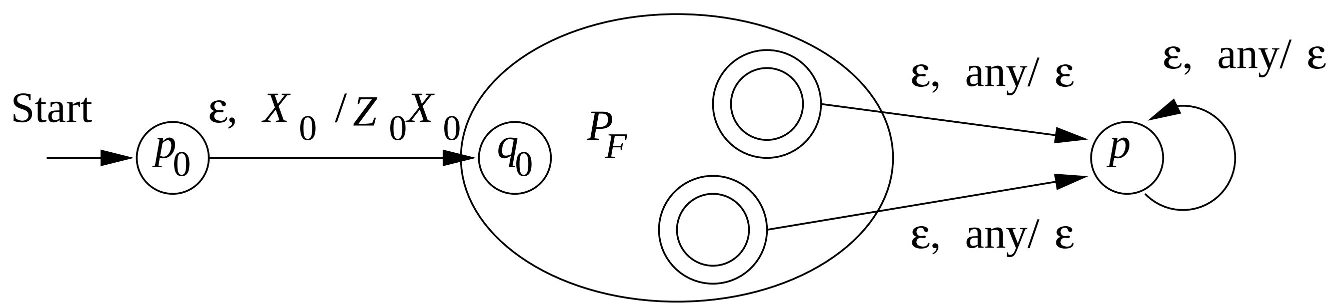
Da stato finale a pila vuota

Teorema 6.11: Sia $L = L(P_F)$, per un PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Allora $\exists$ PDA P_N, tale che $L = N(P_N)$.

Prova: Sia

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

dove $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$, $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$, per $Y \in \Gamma \cup \{X_0\}$, e per tutti i $q \in Q$, $a \in \Sigma \cup \{\epsilon\}, Y \in \Gamma$: $\delta_N(q, a, Y) = \delta_F(q, a, Y)$, e inoltre $\forall q \in F$, e $Y \in \Gamma \cup \{X_0\}$: $(p, \epsilon) \in \delta_N(q, \epsilon, Y)$.



Equivalenza di PDA e CFG

Un linguaggio e'

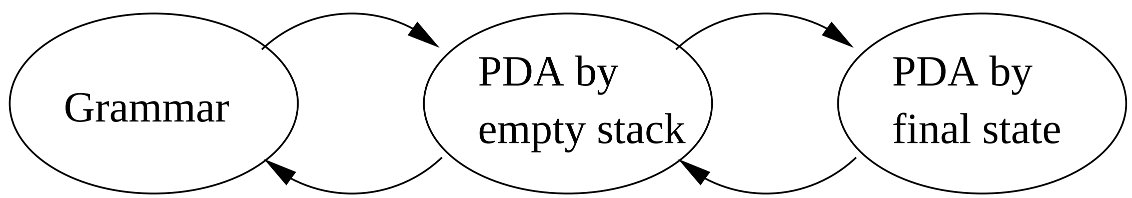
generato da una CFG

se e solo se e'

accettato da un PDA per pila vuota

se e solo se e'

accettato da un PDA per stato finale



Sappiamo gia' andare da pila vuota a stato finale.

Da CFG a PDA

Idea: data G, costruiamo un PDA che simula $\overset{*}{\underset{lm}{\Rightarrow}}$.

Scriviamo le stringhe ottenute lungo una *derivazione sinistra* come

$$xA\alpha$$

dove A e' la variabile *piu' a sinistra*. Ad esempio,

$$\underbrace{(a+}_{x}\underbrace{\underbrace{E}_{A}\underbrace{)}_{\alpha}}_{\text{tail}}$$

Sia $xA\alpha \underset{lm}{\Rightarrow} x\beta\alpha$ (a causa di una produzione $A \rightarrow \beta$ della CFG). Questo corrisponde al PDA che, dopo aver consumato input x, e essersi ritrovato con $A\alpha$ sulla pila, ora esegue una transizione ϵ che elimina A e mette al suo posto β sulla pila.

Piu' formalmente, sia w la stringa data in *input* al PDA e y tale che $w = xy$. Allora il PDA va non deterministicamente dalla configurazione $(q, y, A\alpha)$ alla configurazione $(q, y, \beta\alpha)$.

Alla configurazione $(q, y, \beta\alpha)$ il PDA si comporta come prima, a meno che ci siano *terminali* nel prefisso di β. In questo caso, il PDA li elimina, se *li legge nell'input* (se fanno match con l'input).

Se tutte le scommesse sono giuste (consentono di matchare l'input), il PDA finisce l'input con la *pila vuota*.

Quindi **la trasformazione è la seguente**.

Sia $G = (V, T, Q, S)$ una CFG. Definiamo P_G come

$$(\{q\}, T, V \cup T, \delta, q, S),$$

dove

$$\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \ \in \ Q\},$$

per $A \in V$, e

$$\delta(q, a, a) = \{(q, \epsilon)\},$$

per $a \in T$.

Esempio:

Consideriamo la grammatica

$$S \to \epsilon | SS | iS | iSe.$$

Il PDA corrispondente e'

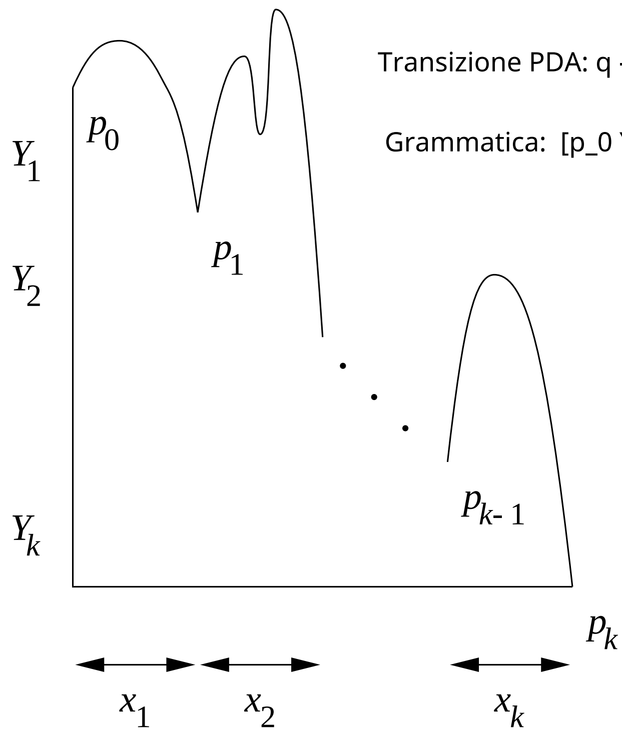
$$P = (\{q\}, \{i, e\}, \{S, i, e\}, \delta, q, S),$$

dove $\delta(q, \epsilon, S) = \{(q, \epsilon), (q, SS), (q, iS), (q, iSe)\}$, $\delta(q, i, i) = \{(q, \epsilon)\}$, e $\delta(q, e, e) = \{(q, \epsilon)\}$.

Da PDA a CFG

Idea: comportamento dei PDA per rimuovere simbolo Y dalla pila (usando una transizione che sostituisce Y con $Y_1 Y_2 \cdots Y_k$)

Quando leggo la sottostringa x_1
elimino y_1 dalla pila, passando
da stato p_0 a stato p_1



Transizione PDA: q --> {a , Y | y_1, y_2 ... y_k} --> p_0

Grammatica: [p_0 Y p_k] --> a [p_0 y_1 p_1] [p_1 y_2 p_2] ... [p_{k_1} y_k p_k]

Definiremo una grammatica con variabili della forma $[p_{i-1} Y_i p_i]$ che rappresentano il passaggio da p_{i-1} a p_i con l'effetto di eliminare Y_i.

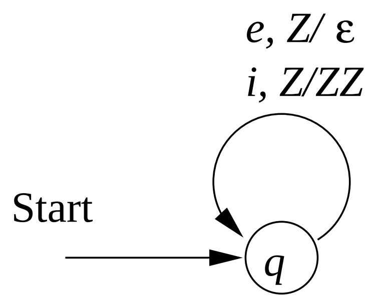
Quindi stringa terminale generata da variabile $[pXq]$ rappresenta:
input letto da PDA andando da p a q e rimuovendo X da pila

Formalmente, sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ un PDA. Definiamo $G = (V, \Sigma, R, S)$, con

$$
\begin{aligned}
V &= \{[pXq] : \{p, q\} \subseteq Q, X \in \Gamma\} \cup \{S\} \\
R &= \{S \to [q_0 Z_0 p] : p \in Q\} \cup \\
&\quad \{[\mathbf{q} X r_k] \to a[\mathbf{r} Y_1 r_1] \cdots [r_{k-1} Y_k r_k] : \\
&\qquad\qquad a \in \mathbf{\Sigma} \cup \{\epsilon\}, \\
&\qquad\qquad \{r_1, \ldots, r_k\} \subseteq Q, \\
&\qquad\qquad (\mathbf{r}, Y_1 Y_2 \cdots Y_k) \in \delta(\mathbf{q}, a, X)\}
\end{aligned}
$$

dove, in caso $k = 0$ si ha: $Y_1 Y_2 \cdots Y_k = \epsilon$ e $r_k = \mathbf{r}$

Esempio: Convertiamo



$$P_N = (\{q\}, \{i, e\}, \{Z\}, \delta_N, q, Z),$$

dove $\delta_N(q, i, Z) = \{(q, ZZ)\}$,
e $\delta_N(q, e, Z) = \{(q, \epsilon)\}$

in una grammatica

$$G = (V, \{i, e\}, R, S),$$

dove $V = \{[qZq], S\}$ e
$R = \{S \to [qZq], [qZq] \to i[qZq][qZq], [qZq] \to e\}$.

Se rimpiazziamo $[qZq]$ con A otteniamo le produzioni $S \to A$ e $A \to iAA|e$.

Esempio: Convertiamo $P = (\{p, q\}, \{0, 1\}, \{X, Z_0\}, \delta, q, Z_0)$, dove δ e' data da

1. $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$
2. $\delta(q, 1, X) = \{(q, XX)\}$
3. $\delta(q, 0, X) = \{(p, X)\}$
4. $\delta(q, \epsilon, X) = \{(q, \epsilon)\}$
5. $\delta(p, 1, X) = \{(p, \epsilon)\}$
6. $\delta(p, 0, Z_0) = \{(q, Z_0)\}$

in una CFG.

Schema di produzione tranzizione 1:
[q Z_0 p_2] --> 1 [q X p_1] [p_1 Z_0 p_2]

Produzioni:
[q Z_0 p] --> 1 [q X p] [p_1 Z_0 p]
[q Z_0 p] --> 1 [q X q] [p_1 Z_0 p]
[q Z_0 q] --> 1 [q X p] [p_1 Z_0 q]
[q Z_0 q] --> 1 [q X q] [p_1 Z_0 q]

k = numero simboli nella pila
k = |X, Z_0| = 2
numero produzione = n^k = 4

Otteniamo $G = (V, \{0, 1\}, R, S)$, dove

$$V = \{[qZ_0q], [pZ_0q], [qZ_0p], [pZ_0p], [qXq], [pXq], [qXp], [pXp], S\}$$

e le produzioni in R sono

$S \to [qZ_0q] | [qZ_0p]$

Dalla transizione (1) $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$ si ha:

$[qZ_0q] \to 1[qXq][qZ_0q]$
$[qZ_0q] \to 1[qXp][pZ_0q]$
$[qZ_0p] \to 1[qXq][qZ_0p]$
$[qZ_0p] \to 1[qXp][pZ_0p]$

Dalla transizione (2) $\delta(q, 1, X) = \{(q, XX)\}$ si ha:

$[qXq] \to 1[qXq][qXq]$
$[qXq] \to 1[qXp][pXq]$
$[qXp] \to 1[qXq][qXp]$
$[qXp] \to 1[qXp][pXp]$

Dalla transizione (3) $\delta(q, 0, X) = \{(p, X)\}$ si ha:

$[qXq] \rightarrow 0[pXq]$
$[qXp] \rightarrow 0[pXp]$

Dalla transizione (4) $\delta(q, \epsilon, X) = \{(q, \epsilon)\}$ si ha:

$[qXq] \rightarrow \epsilon$

Dalla transizione (5) $\delta(p, 1, X) = \{(p, \epsilon)\}$ si ha:

$[pXp] \rightarrow 1$

Dalla transizione (6) $\delta(p, 0, Z_0) = \{(q, Z_0)\}$ si ha:

$[pZ_0q] \rightarrow 0[qZ_0q]$
$[pZ_0p] \rightarrow 0[qZ_0p]$

PDA deterministici

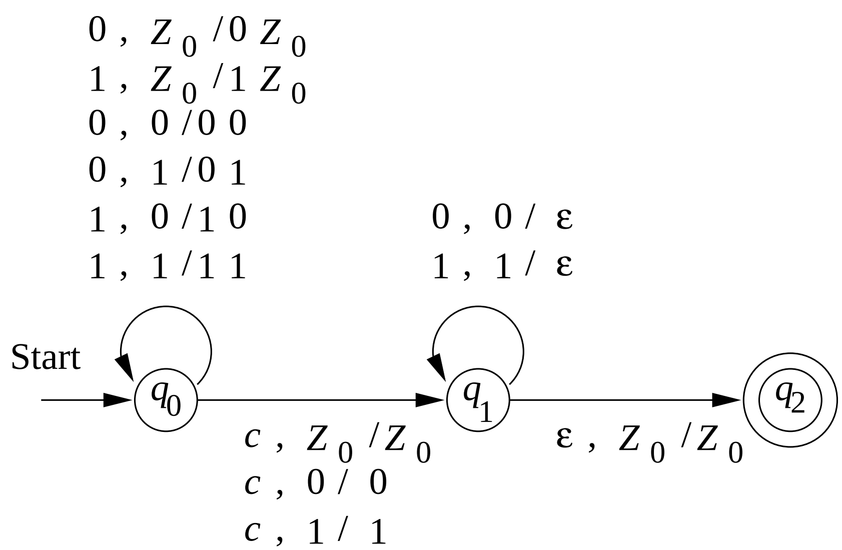
Un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ e' *deterministico* se e solo se:

1. ogni $\delta(q, a, X)$, con $a \in \Sigma \cup \{\epsilon\}$, contiene *al piu'* un elemento
2. se $\delta(q, a, X)$ non vuoto per un $a \in \Sigma$, allora $\delta(q, \epsilon, X)$ vuoto.

Esempio: Definiamo

$$L_{wcwr} = \{wcw^R : w \in \{0,1\}^*\}$$

Allora L_{wcwr} e' riconosciuto dal seguente DPDA



DPDA che accettano per stato finale

Mostreremo che Regolari $\subset L(\text{DPDA}) \subset$ CFL

Teorema 6.17: Se L e' regolare, allora $L = L(P)$ per qualche DPDA P.

Prova: Dato che L e' regolare, esiste un DFA A tale che $L = L(A)$. Sia

$$A = (Q, \Sigma, \delta_A, q_0, F)$$

definiamo il DPDA

$$P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F),$$

dove

$$\delta_P(q, a, Z_0) = \{(\delta_A(q, a), Z_0)\},$$

per tutti i $p, q \in Q$ e $a \in \Sigma$.

Un'induzione su $|w|$ ci da'

$$(q_0, w, Z_0) \overset{*}{\vdash} (p, \epsilon, Z_0) \Leftrightarrow \hat{\delta}_A(q_0, w) = p$$

- Abbiamo visto che Regolari $\subseteq L$(DPDA).
- $L_{wcwr} \in L$(DPDA)$\setminus$ Regolari   \ = sottrazione
- Ci sono linguaggi in CFL$\setminus L$(DPDA).

  Si, per esempio L_{wwr}.

DPDA che accettano per pila vuota

E i DPDA che accettano per pila vuota?

Possono riconoscere solo linguaggi con la **proprieta' del prefisso**.

Un linguaggio L ha la *proprieta' del prefisso* se **non** esistono due stringhe distinte in L, tali che una e' un prefisso dell'altra.

Esempio: L_{wcwr} ha la proprieta' del prefisso.

Esempio: $\{0\}^*$ non ha la proprieta' del prefisso.

Teorema 6.19: L e' $N(P)$ per qualche DPDA P se e solo se L ha la proprieta' del prefisso e L e' $L(P')$ per qualche DPDA P'.

DPDA e non ambiguità

L(DPDA) coincide con i CFL aventi grammatiche **non ambigue** (cioe' non inerentemente ambigui)? **No**. Per esempio:

> L_{wwr} ha una grammatica non ambigua $S \to 0S0|1S1|\epsilon$ ma non e' L(DPDA).

L'inverso invece vale! Abbiamo, preliminarmente:

Teorema 6.20: Se $L = N(P)$ per qualche DPDA P, allora L ha una CFG non ambigua.

Prova: Applicando la costruzione vista da PDA a CFG, se la costruzione e' applicata ad un DPDA, il risultato e' una CFG con derivazioni a sinistra uniche per ogni stringa.

Teorema 6.20 puo' essere rafforzato:

Teorema 6.21: Se $L = L(P)$ per qualche DPDA P, allora L ha una CFG non ambigua.

Prova: Sia $\$$ un simbolo fuori dell'alfabeto di L, e sia $L' = L\{\$\}$.
E' facile modificare P per riconoscere L' (PDA ancora deterministico); inoltre L' ha la proprieta' del prefisso.
Per il teorema 6.19 abbiamo $L' = N(P')$ per qualche DPDA P'.
Per il teorema 6.20 L' puo' essere generato da una CFG G' non ambigua
Modifichiamo G' in G, tale che $L(G) = L$, aggiungendo la produzione

$$\$ \rightarrow \epsilon$$

(e considerando \$ una variabile anziche' un terminale)

Dato che G' ha derivazioni a sinistra uniche, anche G le avra' uniche, dato che l'unica cosa nuova e' l'aggiunta di derivazioni

$$w\$ \underset{lm}{\Rightarrow} w$$

alla fine.

Proprieta' dei CFL

- *Semplificazione* di una CFG. Se un linguaggio e' un CFL, ha una grammatica in una possibile forma speciale.

- *Pumping Lemma per CFL.* Simile ai linguaggi regolari.

- *Proprieta' di chiusura.* Solo alcune delle proprieta' di chiusura dei linguaggi regolari valgono anche per i CFL.

- *Proprieta' di decisione.* Possiamo controllare l'appartenenza e l'essere vuoto, ma, per esempio, l'equivalenza di CFL e' non verificabile tramite un algoritmo (indecidibile).

Forma normale di Chomsky

Ogni CFL (senza ϵ) e' generato da una CFG dove tutte le produzioni sono della forma

$$A \to BC, \text{ o } A \to a$$

dove $A, B,$ e C sono variabili, e a e' un simbolo terminale. Questa e' detta forma normale di Chomsky (CNF), e per ottenerla dobbiamo innanzitutto "pulire" la grammatica:

- Eliminare i *simboli inutili*, quelli che non appaiono in nessuna derivazione $S \stackrel{*}{\Rightarrow} w$, per simbolo iniziale S e terminale w.
- Eliminare le produzioni ϵ, della forma $A \to \epsilon$.
- Eliminare le *produzioni unita'*, cioe' produzioni della forma $A \to B$, dove A e B sono variabili.

Eliminazione simboli inutili

- Un simbolo X e' *utile* per una grammatica $G = (V, T, P, S)$, se esiste una derivazione

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha X \beta \overset{*}{\underset{G}{\Rightarrow}} w$$

per una stringa di terminali w. Simboli che non sono utili sono detti *inutili*.

- Un simbolo X e' *generante* se $X \overset{*}{\underset{G}{\Rightarrow}} w$, per qualche $w \in T^*$

- Un simbolo X e' *raggiungibile* se $S \overset{*}{\underset{G}{\Rightarrow}} \alpha X \beta$, per qualche $\{\alpha, \beta\} \subseteq (V \cup T)^*$

Se in G (con $L(G) \neq \emptyset$) eliminiamo prima i simboli non generanti, e poi quelli non raggiungibili, rimarranno solamente simboli utili.

Esempio: Sia G la grammatica

$$S \rightarrow AB|a, \ A \rightarrow b$$

S e A sono generanti, B non lo e'. Se eliminiamo B dobbiamo eliminare $S \rightarrow AB$, riducendo la grammatica

$$S \rightarrow a, \ A \rightarrow b$$

Ora, solo la variabile S e' raggiungibile. Eliminando A rimane solo

$$S \rightarrow a$$

con linguaggio $\{a\}$.

Nota Se eliminiamo prima i simboli non raggiungibili, si ha che tutti i simboli sono raggiungibili. Da

$$S \rightarrow AB|a, \ A \rightarrow b$$

eliminiamo B in quanto non generante, e rimane la grammatica

$$S \rightarrow a, \ A \rightarrow b$$

che contiene ancora simboli inutili

Eliminazione produzioni ϵ

Si ha che se L e' un CFL, allora $L \setminus \{\epsilon\}$ ha una grammatica priva di produzioni ϵ (cio' mostra anche che $L \setminus \{\epsilon\}$ e' CFL).

La variabile A e' *annullabile* se $A \stackrel{*}{\Rightarrow} \epsilon$.

Sia A annullabile. Rimpiazzeremo una regola del tipo

$$B \to \alpha A \beta$$

con

$$B \to \alpha A \beta, \ B \to \alpha \beta$$

(rimpiazzando in tal modo anche le nuove regole via via ottenute) e cancelleremo tutte le regole con corpo ϵ.

Indichiamo con $n(G)$, l'insieme dei simboli annullabili di una grammatica $G = (V, T, P, S)$

Esempio: Sia G la grammatica

$$S \to AB,\ A \to aAA|\epsilon,\ B \to bBB|\epsilon$$

Abbiamo $n(G) = \{A, B, S\}$. La prima regola diventa

$$S \to AB|A|B$$

la seconda

$$A \to aAA|aA|aA|a$$

e la terza

$$B \to bBB|bB|bB|b$$

Eliminiamo le regole con corpo ϵ, ed otteniamo la grammatica G_1 :

$$S \to AB|A|B,\ A \to aAA|aA|a,\ B \to bBB|bB|b$$

Eliminazione produzioni unita'

$$A \rightarrow B$$

e' una produzione *unita'*, nel caso in cui A e B siano variabili.

Produzioni unita' possono essere eliminate.

Si consideri la grammatica

$$\begin{array}{lcl} E & \rightarrow & T \mid E + T \\ T & \rightarrow & F \mid T * F \\ F & \rightarrow & I \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

ha le produzioni unita' $E \rightarrow T$, $T \rightarrow F$, e $F \rightarrow I$

Si consideri la **produzione unità** $\mathbf{E \rightarrow T}$. Si trasforma tale produzione con il seguente procedimento a **espansione**.

Si espande $E \rightarrow T$ ottenendo le produzioni:

$$E \rightarrow F,\ E \rightarrow T * F$$

Poi, espandendo $E \rightarrow F$, si ottiene:

$$E \rightarrow I|(E)|T * F$$

Infine, espandendo $E \rightarrow I$, si ottiene:

$$E \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F$$

Si considerano poi le altre produzioni unità $\mathbf{T \rightarrow F}$ e $\mathbf{F \rightarrow I}$ della grammatica e, per ciascuna, si applica analogo procedimento.

La grammatica inziale

$$
\begin{aligned}
E &\rightarrow T \mid E + T \\
T &\rightarrow F \mid T * F \\
F &\rightarrow I \mid (E) \\
I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

viene quindi modificata trasformando:

$E \rightarrow T$ in

$$E \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F$$

$T \rightarrow F$ in

$$T \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E)$$

$F \rightarrow I$ in

$$F \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

Quindi, eliminando le produzioni unità, la grammatica diviene

$$
\begin{aligned}
E &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F \mid E + T \\
T &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F \\
F &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \\
I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

Consideriamo ancora il **procedimento a espansione usato per trasformare le produzioni unità**. Ad esempio per $E \to T$:

Si espande $E \to T$ ottenendo le produzioni:

$$E \to F,\ E \to T * F$$

Poi, espandendo $E \to F$, si ottiene:

$$E \to I|(E)|T * F$$

Infine, espandendo $E \to I$, si ottiene:

$$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F$$

Questo procedimento funziona per qualsiasi grammatica?

No! Se ci sono **cicli** il procedimento visto non consente di giungere alla rimozione delle produzioni unità che via via si generano!

Si consideri per esempio la grammatica: $A \to B,\ B \to C,\ C \to A$

Soluzione:
Se durante il procedimento sopra **si genera una produzione unità che si ha già espanso** la si può semplicemente **rimuovere** (espandendola si otterrebbero di nuovo produzioni gia' generate)

Esempio: si consideri la grammatica

$A \to B \mid a$
$B \to C \mid b$
$C \to A \mid c$

Comincio trasformando la **produzione unità** $\mathbf{A \to B}$.

Si espande $A \to B$ ottenendo le produzioni:

$$A \to C \,|\, b$$

Poi, espandendo $A \to C$, si ottiene:

$$A \to A \,|\, c \,|\, b$$

Infine, espandendo $A \to A$, si ottiene:

$$A \to B \,|\, a \,|\, c \,|\, b$$

Andando avanti ottengo ovviamente sempre produzioni che ho già, quindi mi posso **fermare ed eliminare** $\mathbf{A \to B}$.

La produzione unità $A \to B$ si trasforma quindi in:

$$A \to a \,|\, c \,|\, b$$

La grammatica inziale

$$
\begin{aligned}
&A \to B \mid a\\
&B \to C \mid b\\
&C \to A \mid c
\end{aligned}
$$

viene quindi modificata trasformando:

$A \to B$ in

$$A \to a \,|\, c \,|\, b$$

$B \to C$ in

$$B \to b \,|\, a \,|\, c$$

$C \to A$ in

$$C \to c \,|\, b \,|\, a$$

Quindi, eliminando le produzioni unità, la grammatica diviene

$$
\begin{aligned}
&A \to a \mid b \mid c\\
&B \to a \mid b \mid c\\
&C \to a \mid b \mid c
\end{aligned}
$$

Sommario

Per “pulire” una grammatica si deve

1. Eliminare le produzioni ϵ
2. Eliminare le produzioni unita’
3. Eliminare i simboli inutili

in questo ordine.

Esercizio. Trovare una grammatica in cui cambiando l’ordine non si ottiene una versione “pulita”

Forma Normale di Chomsky, CNF

Ogni CFL non vuoto, che non contiene ϵ, ha una grammatica G priva di simboli inutili, con produzioni nella forma

- $A \to BC$, dove $\{A, B, C\} \subseteq V$, o
- $A \to a$, dove $A \in V$, e $a \in T$.

Per ottenerla, si effettuano le seguenti trasformazioni su una qualsiasi grammatica per il CFL

1. "Pulire" la grammatica
2. Modificare le produzioni con 2 o piu' simboli in modo tale che siano tutte variabili
3. Ridurre il corpo delle regole di lunghezza superiore a 2 in cascate di produzioni con corpi da 2 variabili.

- Per il passo 2, per ogni terminale a che compare in un corpo di lunghezza ≥ 2, creare una nuova variabile, ad esempio A, e sostituire a con A in tutti i corpi, e aggiungere la nuova regola $A \rightarrow a$.

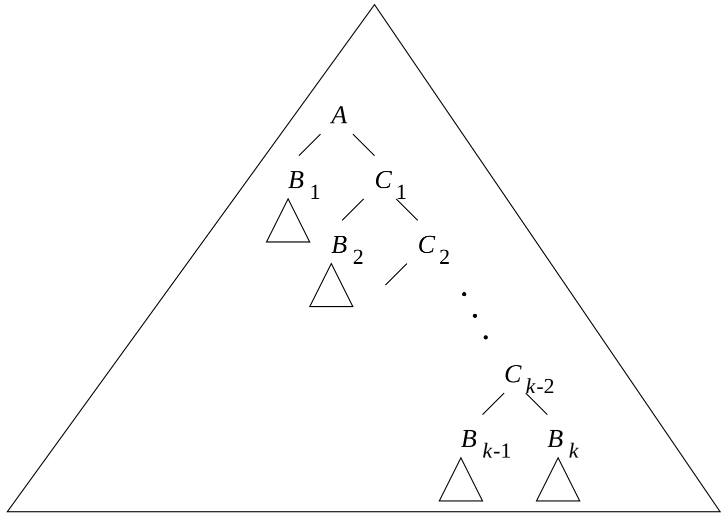
- Per il passo 3, per ogni regola nella forma

$$A \rightarrow B_1 B_2 \cdots B_k,$$

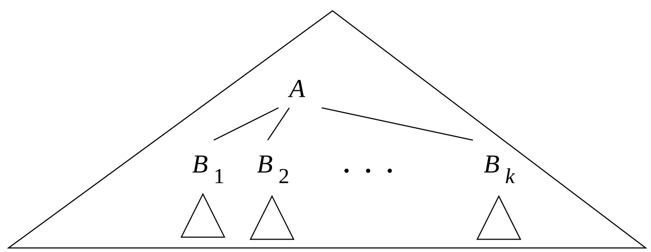
$k \geq 3$, introdurre le nuove variabili $C_1, C_2, \ldots C_{k-2}$, e sostituire la regola con

$$\begin{array}{rcl} A & \rightarrow & B_1 C_1 \\ C_1 & \rightarrow & B_2 C_2 \\ & \cdots & \\ C_{k-3} & \rightarrow & B_{k-2} C_{k-2} \\ C_{k-2} & \rightarrow & B_{k-1} B_k \end{array}$$

Illustrazione dell'effetto del passo 3.



(a)



(b)

Esempio

Iniziamo dalla grammatica

$$
\begin{aligned}
&E \to E+T \mid T*F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1\\
&T \to T*F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1\\
&F \to (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1\\
&I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

Per il passo 2 usiamo le regole
$A \to a, B \to b, Z \to 0, O \to 1$
$P \to +, M \to *, L \to (, R \to)$
e otteniamo la grammatica

$$
\begin{aligned}
&E \to EPT \mid TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO\\
&T \to TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO\\
&F \to LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO\\
&I \to a \mid b \mid IA \mid IB \mid IZ \mid IO\\
&A \to a, B \to b, Z \to 0, O \to 1\\
&P \to +, M \to *, L \to (, R \to)
\end{aligned}
$$

Per il passo 3, rimpiazziamo

$E \to EPT$ con $E \to EC_1, C_1 \to PT$

$E \to TMF, T \to TMF$ con
$E \to TC_2, T \to TC_2, C_2 \to MF$

$E \to LER, T \to LER, F \to LER$ con
$E \to LC_3, T \to LC_3, F \to LC_3, C_3 \to ER$

La grammatica in CNF finale e'

$E \to EC_1 \mid TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$T \to TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$F \to LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$I \to a \mid b \mid IA \mid IB \mid IZ \mid IO$
$C_1 \to PT, C_2 \to MF, C_3 \to ER$
$A \to a, B \to b, Z \to 0, O \to 1$
$P \to +, M \to *, L \to (, R \to)$

Pumping Lemma per CFL

Pumping Lemma per linguaggi regolari: per una stringa del linguaggio abbastanza lunga da causare un ciclo nel relativo DFA si può “ripetere” il ciclo e scoprire una infinita’ di stringhe che appartengono al linguaggio

Pumping Lemma per CFL (un po’ piu’ complicato): per una stringa del linguaggio sufficientemente lunga e’ sempre possibile trovare due pezzi distinti da ripetere “in tandem”:

> ripetendoli lo stesso numero di volte “i”, otteniamo, per ogni “i”, una nuova stringa appartenente al linguaggio

Enunciato del Pumping Lemma per CFL

Pumping Lemma:

Sia L un CFL. Allora $\exists n \geq 1$ che soddisfa:

ogni $z \in L : |z| \geq n$ è scomponibile in 5 stringhe $z = uvwxy$ tali che:

1. $|vwx| \leq n$
2. $|vx| > 0$
3. per ogni $i \geq 0$, $uv^iwx^iy \in L$

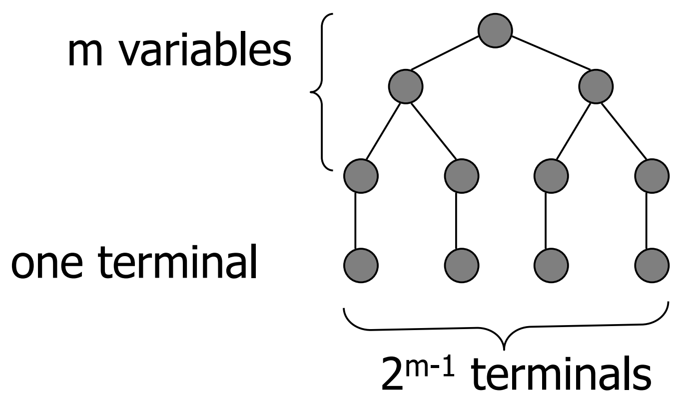
Dimostrazione Pumping Lemma per CFL

Prova:

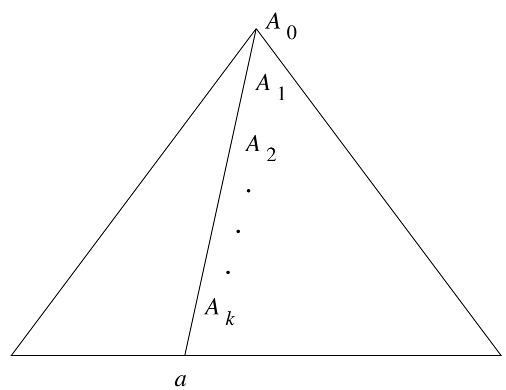
- Si consideri una grammatica per $L \setminus \{\epsilon\}$ in CNF
- Assumiamo che la grammatica abbia m variabili. Sia $n = 2^m$
- Sia $z \in L$ una qualsiasi stringa tale che $|z| \geq n = 2^m$. Si ha che ogni albero sintattico di z contiene un cammino di lunghezza $\geq m + 1$

Lemma 1: Se tutti i cammini dell'albero sintattico hanno lunghezza $\leq m$, allora la stringa generata ha lunghezza $\leq 2^{m-1}$

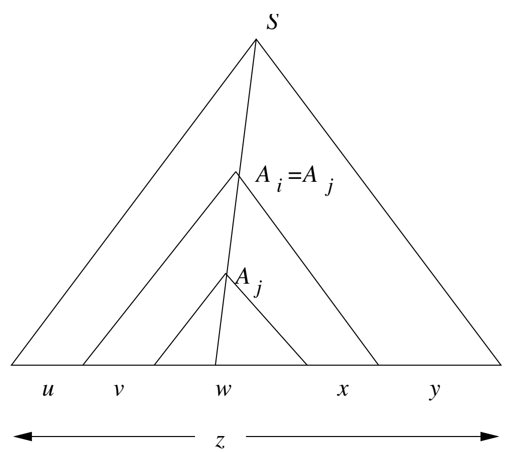
Prova:



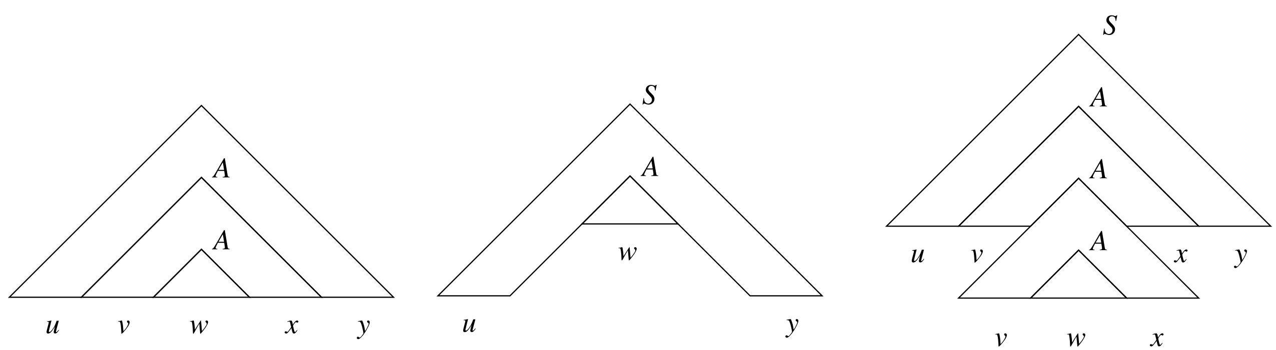
• Consideriamo un cammino $A_0 A_1 \ldots A_k a$ di lunghezza massima: ha lunghezza $\geq m+1$.



Esistono $i \neq j$ tali che $A_i = A_j$ (assumiamo che i, j siano fra le **ultime** $m+1$ variabili del cammino)



- **Osservazione 1:** l'albero radicato in A_i ha altezza $\leq m+1$, quindi la stringa corrispondente ha lunghezza $\leq 2^m = n$ (cioe' $|vwx| \leq n$)
- **Osservazione 2:** le stringhe v e x non possono essere entrambe vuote in quanto A_i (essendo la grammatica in CNF) genera due variabili entrambe non annullabili (quindi $|vx| > 0$)
- **Osservazione 3:** l'albero sintattico ottenuto ripetendo un numero arbitrario di volte (possibilmente anche 0 volte) la parte di albero radicato in A_i meno l'albero radicato in A_j, continua ad essere un albero sintattico corretto (quindi per ogni $i \geq 0$, $uv^iwx^iy \in L$)



Applicazioni del Pumping Lemma per CFL

Come per i linguaggi regolari, il Pumping Lemma per CFL puo' essere usato per dimostrare che un dato linguaggio non e' libero

Esempio 1: Si consideri $L = \{0^m10^m10^m : m \geq 1\}$. Dimostrare che L non e' un CFL.

Prova: Assumiamo, per assurdo, che L sia CFL. Sia n la costante del Pumping Lemma. Si consideri la stringa $z = 0^n10^n10^n$. Si ha che $z \in L$ e $|z| \geq n$. Allora, per Pumping Lemma, $z = uvwxy$ con $|vwx| \leq n$, $|vx| > 0$ e $uv^iwx^iy \in L$ per ogni $i \geq 0$. Consideriamo ora i due seguenti casi:

- vx contiene almeno un 1.
  In questo caso $uwy \notin L$ visto che ha al piu' un solo 1

- vx contiene solo 0.
  Ci sono solo due casi. O vx include 0 tutti appartenenti ad uno stesso gruppo di 0 oppure v appartiene ad un gruppo ed x ad un altro.

  In entrambi i casi $uwy \notin L$ in quanto almeno un gruppo di 0 mantiene lunghezza n ed un'altro si riduce a lunghezza $< n$

Esempio 2: Si consideri $L = \{0^{k^2} : k \geq 1\}$. Dimostrare che L non e' un CFL.

Prova: Assumiamo, per assurdo, che L sia CFL. Sia n la costante del Pumping Lemma. Si consideri la stringa $z = 0^{n^2}$. Si ha che $z \in L$ e $|z| \geq n$. Allora, per Pumping Lemma, $z = uvwxy$ con $|vwx| \leq n$, $|vx| > 0$ e $uv^iwx^iy \in L$ per ogni $i \geq 0$. Consideriamo ora il seguente caso:

- $uv^2wx^2y \in L$ per Pumping Lemma;

- $n^2 < |uv^2wx^2y| \leq n^2{+}n < n^2{+}2n{+}1 = (n{+}1)^2$. Non essendoci quadrati perfetti strettamente inclusi fra n^2 e $(n+1)^2$ allora $uv^2wx^2y \notin L$, contraddicendo il punto precedente.

Proprieta' di chiusura dei CFL

Teorema 7.24: I CFL sono chiusi rispetto ai seguenti operatori (i) : unione, (ii) : concatenazione e (iii) : chiusura di Kleene e chiusura positiva $+$

Prova: Per esercizio.

Teorema: Se L e CFL, allora lo e' anche L^R.

Prova: Supponiamo che L sia generato da $G = (V, T, P, S)$. Costruiamo $G^R = (V, T, P^R, S)$, dove

$$P^R = \{A \to \alpha^R : A \to \alpha \in P\}$$

Si mostra per induzione sulla lunghezza delle derivazioni in G e in G^R che $(L(G))^R = L(G^R)$.

I CFL non sono chiusi rispetto all'intersezione

Sia $L_1 = \{0^n 1^n 2^i : n \geq 1, i \geq 1\}$. Allora L_1 e' CFL con grammatica

$$
\begin{array}{c}
S \to AB \\
A \to 0A1|01 \\
B \to 2B|2
\end{array}
$$

Inoltre, $L_2 = \{0^i 1^n 2^n : n \geq 1, i \geq 1\}$ e' CFL con grammatica

$$
\begin{array}{c}
S \to AB \\
A \to 0A|0 \\
B \to 1B2|12
\end{array}
$$

Invece, $L_1 \cap L_2 = \{0^n 1^n 2^n : n \geq 1\}$ non e' CFL (dimostrazione tramite Pumping Lemma per esercizio).

Operazioni su liberi e regolari

Teorema 7.27: Se L e' CFL, e R e' regolare, allora $L \cap R$ e' CFL.

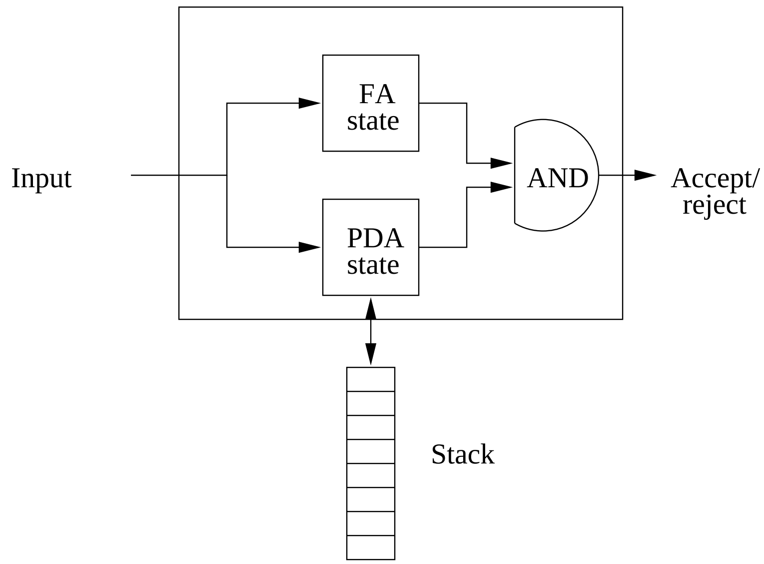
Prova: Sia L accettato dal PDA

$$P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$$

per stato finale, e sia R accettato dal DFA

$$A = (Q_A, \Sigma, \delta_A, q_A, F_A)$$

Costruiremo un PDA per $L \cap R$ secondo la figura



Formalmente, definiamo

$$P' = (Q_P \times Q_A, \Sigma, \Gamma, \delta, (q_P, q_A), Z_0, F_P \times F_A)$$

dove

$$\delta((q,p), a, X) = \{((r, \hat{\delta}_A(p,a)), \gamma) : (r,\gamma) \in \delta_P(q,a,X)\}$$

Possiamo provare per induzione su $\overset{*}{\vdash}$ che

$$(q_P, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \gamma) \text{ in } P$$

se e solo se

$$((q_P, q_A), w, Z_0) \overset{*}{\vdash} \big((q, \hat{\delta}(q_A, w)), \epsilon, \gamma\big) \text{ in } P'$$

Teorema 7.29: Siano L, L_1, L_2 CFL e R regolare. Allora

1. $L \setminus R$ e' CFL
2. $\bar{L}$ non e' necessariamente CFL
3. $L_1 \setminus L_2$ non e' necessariamente CFL

Prova:

1. $\bar{R}$ e' regolare, $L \cap \bar{R}$ e' CFL, e $L \cap \bar{R} = L \setminus R$.
2. Se $\bar{L}$ fosse sempre CFL, seguirebbe che

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

   sarebbe sempre CFL.

3. Notare che Σ^* e' CFL, quindi se $L_1 \setminus L_2$ fosse sempre CFL, allora lo sarebbe sempre anche $\Sigma^* \setminus L = \bar{L}$.

Proprieta' di decisione per CFL

Analizzeremo i seguenti problemi decidibili:

- Verificare se $L(G) \neq \emptyset$, per una CFG G
- Verificare se $w \in L(G)$, per una stringa w ed una CFG G

E elencheremo alcuni **problemi indecidibili**

Verificare se un CFL e' vuoto

$L(G)$ e' non-vuoto se il simbolo iniziale S e' generante

Una implementazione naive del calcolo dei simboli generanti di G richiede tempo $O(n^2)$

Ottimizzando le strutture dati di appoggio può essere calcolato in tempo $O(n)$.

$$\boxed{w \in L(G)\text{?}}$$

Tecnica inefficiente:

Supponiamo che G sia in CNF, e che la stringa w abbia lunghezza $|w| = n$. Visto che il suo albero sintattico e' binario, ci sono $2n-1$ nodi interni

Basta quindi generare *tutti* gli alberi sintattici di G con $2n-1$ nodi interni, e poi controllare se almeno uno genera w

Numero degli alberi/etichettature possibili (complessita' algoritmo) quindi esponenziale in n.

Problemi indecidibili per CFL

I seguenti problemi sono indecidibili:

1. Una data CFG G e' ambigua?

2. Un dato CFL L e' inerentemente ambigua?

3. L'intersezione di due CFL e' vuota?

4. Due CFL sono uguali?

5. Un CFL e' universale (cioe' uguale a Σ^*)?