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 e alle inferenze ricorsive.
- Ci possono essere diversi alberi sintattici per la stessa stringa
- Idealmente ci dovrebbe essere solo un albero sintattico (la "vera" struttura), cioè il linguaggio dovrebbe essere non ambiguo.
- Sfortunatamente, non sempre possiamo rimuovere l'ambiguità.

Costruzione di un albero sintattico

Sia $G = (V, T, P, S)$ una CFG. Un albero è un **albero sintattico** per G se:

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

$$X_1 X_2 \ldots X_k$$

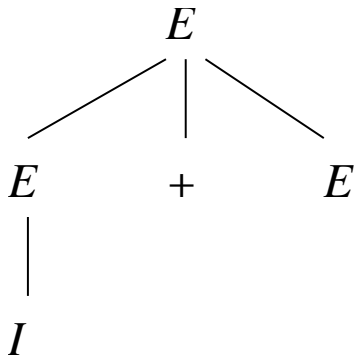
allora $A \to X_1 X_2 \ldots X_k \in P$.

Esempio

Nella grammatica

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

il seguente è un albero sintattico:



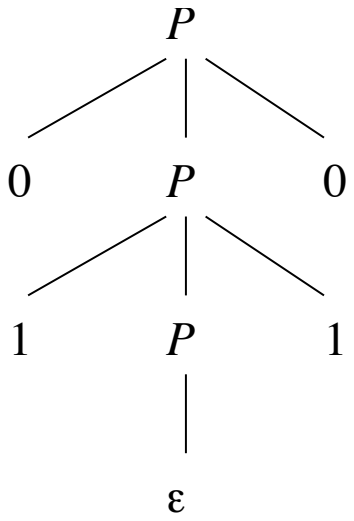
Questo albero sintattico mostra la derivazione $E \overset{*}{\Rightarrow} I + E$

Esempio

Nella grammatica

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

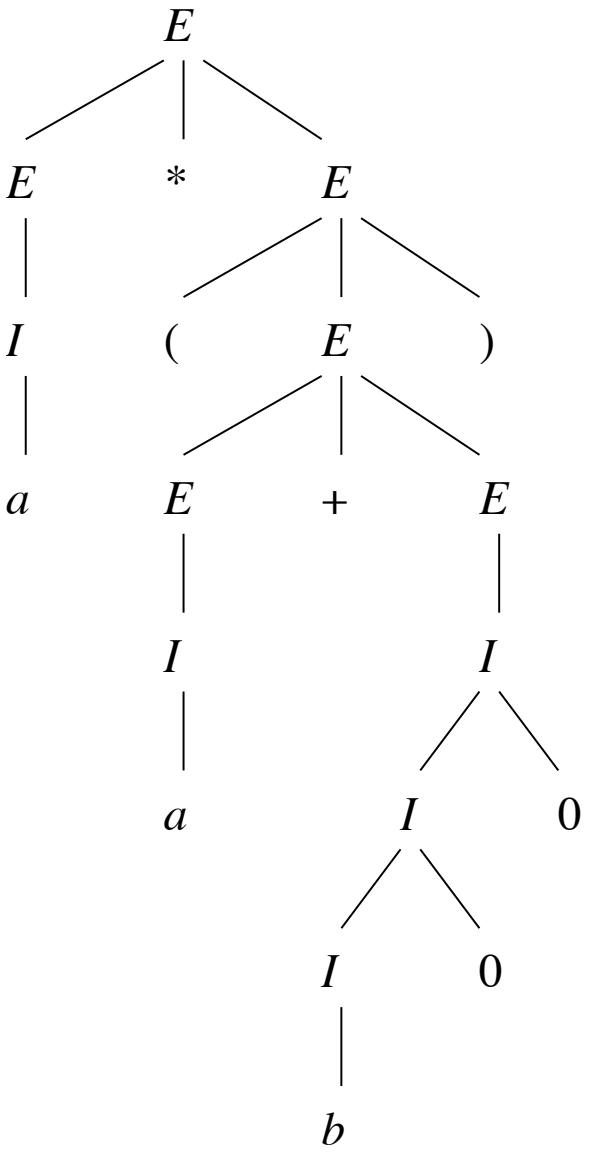
il seguente è un albero sintattico:



Il prodotto di un albero sintattico

- Il **prodotto** di un albero sintattico è la stringa di foglie da sinistra a destra.
- Sono importanti quegli alberi sintattici dove:
  1. Il prodotto è una stringa terminale.
  2. La radice è etichettata dal simbolo iniziale.
- L'insieme dei prodotti di questi alberi sintattici è il linguaggio della grammatica.

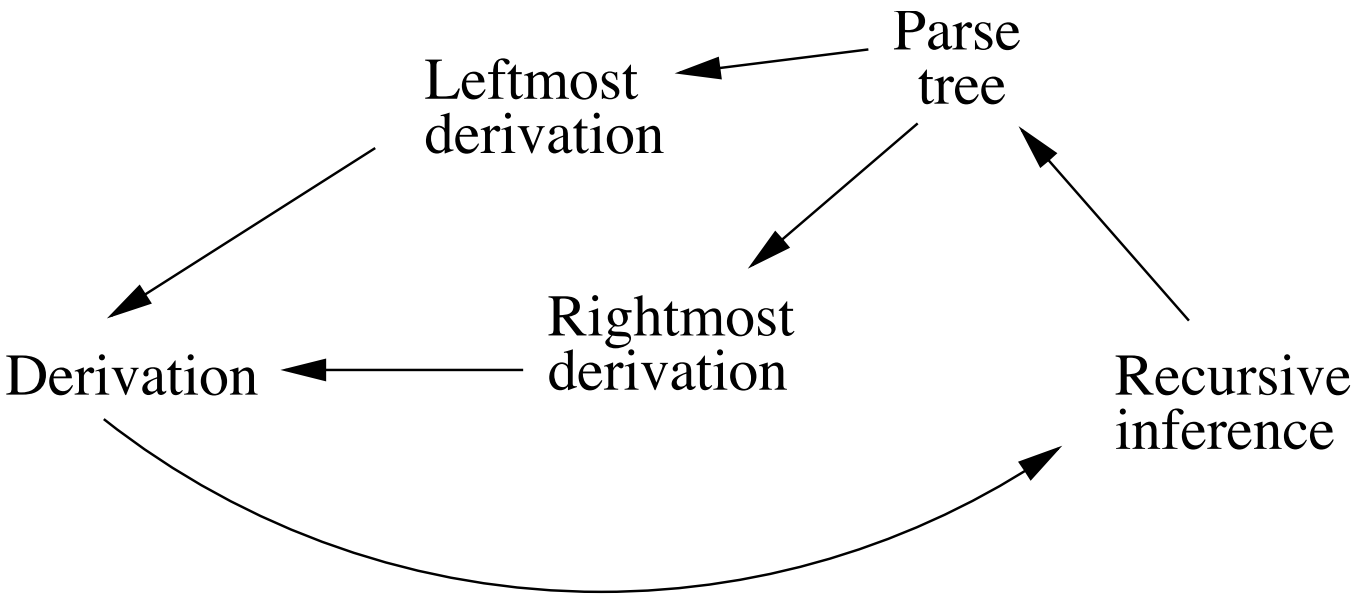
Esempio



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

1. Possiamo determinare per inferenza ricorsiva che w è nel linguaggio di A
2. $A \stackrel{*}{\Rightarrow} w$
3. $A \underset{lm}{\stackrel{*}{\Rightarrow}} w$, e $A \underset{rm}{\stackrel{*}{\Rightarrow}} w$
4. C'è un albero sintattico di G con radice A e prodotto w.

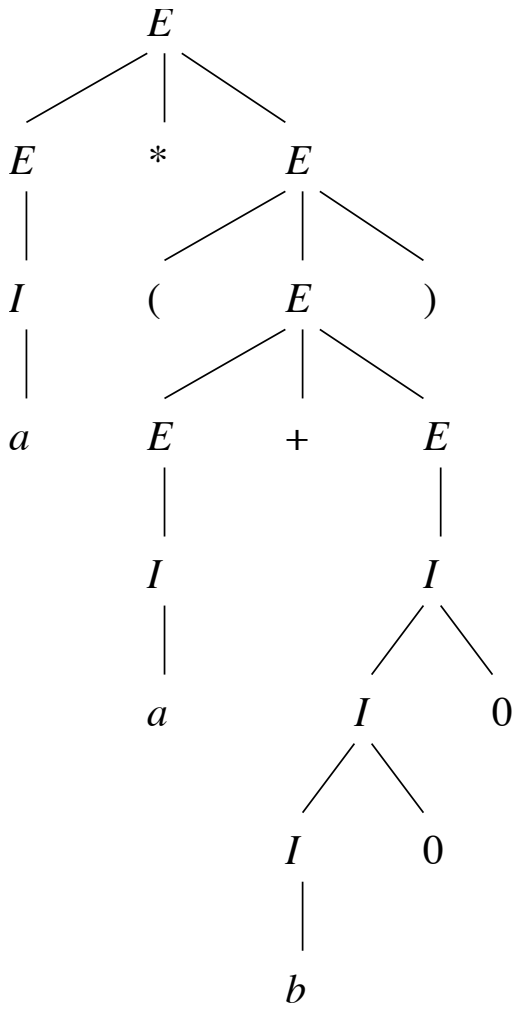
Per provare l'equivalenza, usiamo il seguente piano.



- Dalle inferenze ricorsive agli alberi: partiamo dall'ultimo passo e costruiamo sottoalberi, poi passiamo ai passi precedenti fino al primo.
- Dagli alberi alle derivazioni a sinistra: seguo l'albero da sinistra a destra.
- Dalle derivazioni alle inferenze ricorsive: vado all'indietro nella derivazione.

Esempio

Costruiamo la derivazione a sinistra per l'albero



Supponiamo di aver induttivamente costruito la derivazione a sinistra

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

corrispondente al sottoalbero più a sinistra, e la derivazione 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 più 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:

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

Ambiguità in Grammatiche e Linguaggi

Nella grammatica

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

$\cdots$

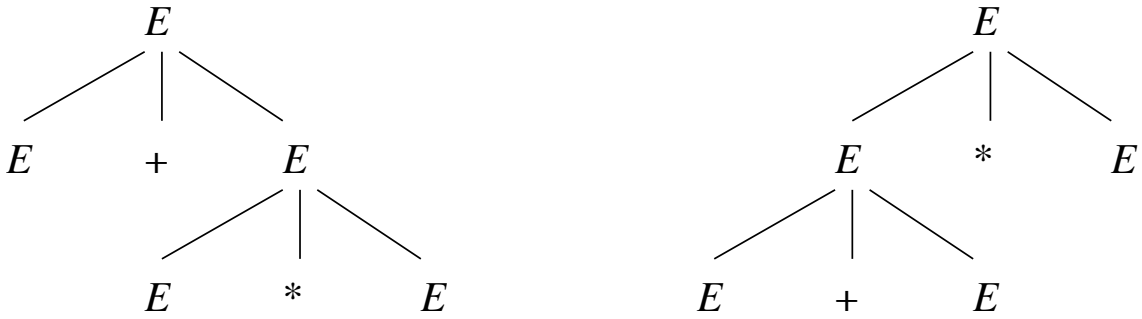
la forma sentenziale $E + E * E$ ha due derivazioni:

$$E \Rightarrow E + E \Rightarrow E + E * E$$

e

$$E \Rightarrow E * E \Rightarrow E + E * E$$

Questo ci dà due alberi sintattici:



L'esistenza di **varie derivazioni** di per sé non è pericolosa, è l'esistenza di **vari alberi sintattici** che rovina la grammatica.
Esempio: Nella stessa grammatica

5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

la stringa $a + b$ ha varie derivazioni:

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

e

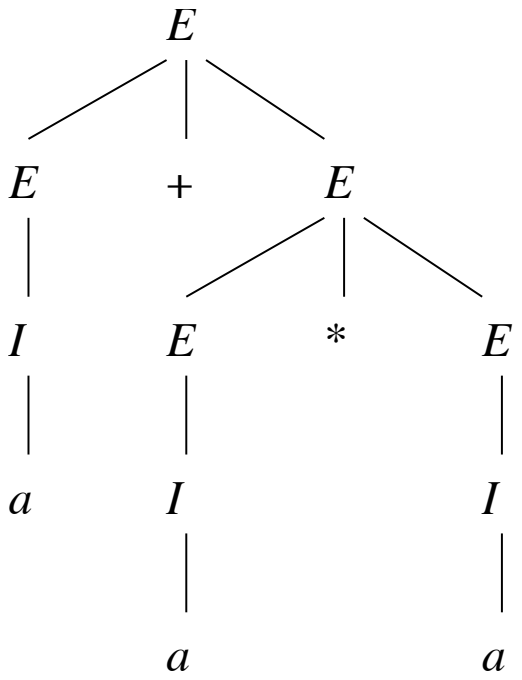
$$E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b$$

Però il loro albero sintattico è lo stesso, e la struttura di $a + b$ è quindi non ambigua.

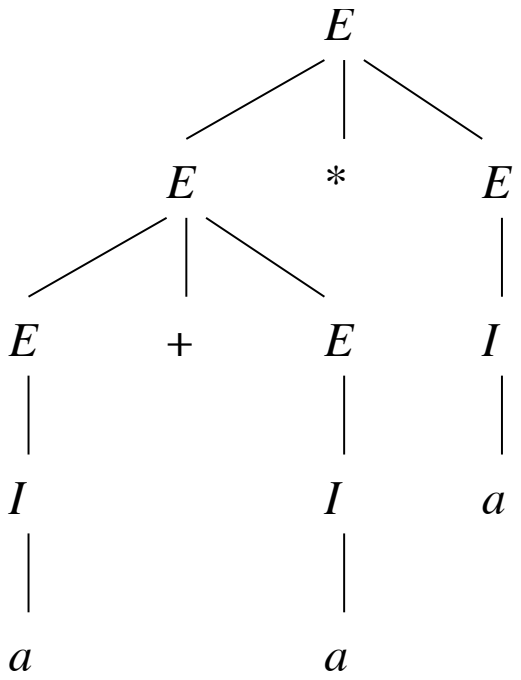
Definizione: Sia $G = (V, T, P, S)$ una CFG. Diciamo che G è **ambigua** se esiste una stringa in T^* che ha più di un albero sintattico.

Se ogni stringa in $L(G)$ ha al più un albero sintattico, G è detta **non ambigua**.

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



(a)



(b)

Rimuovere l'ambiguità dalle grammatiche

- Buone notizie: a volte possiamo rimuovere l'ambiguità
- Cattive notizie: non c'è nessun algoritmo per farlo in modo sistematico
- Ancora cattive notizie: alcuni CFL hanno solo CFG ambigue
- Studiamo la grammatica

$$E \to I \mid E + E \mid E * E \mid (E)$$

$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

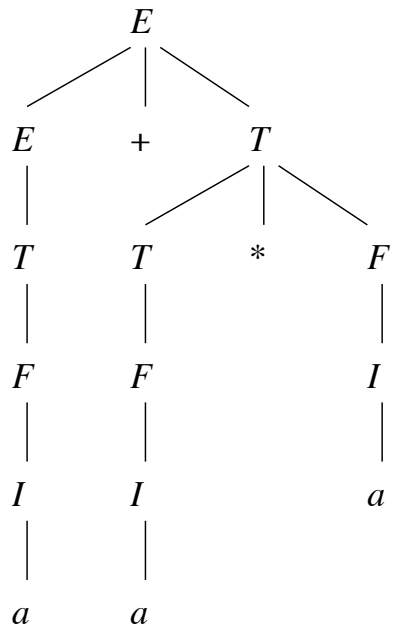
  - Non c'è precedenza tra * e +
  - Non c'è raggruppamento di sequenze di operatori: $E + E + E$ è inteso come $E + (E + E)$ o come $(E + E) + E$?

Soluzione: Introduciamo più variabili, ognuna che rappresenta espressioni con lo stesso grado di "forza di legamento"

1. Un *fattore* è un'espressione che non può essere spezzata da un * o un $+$ adiacente. I nostri fattori sono:
   1. Identificatori
   2. Un'espressione racchiusa tra parentesi.
2. Un *termine* è un'espressione che non può essere spezzata da un $+$. Ad esempio, $a * b$ può essere spezzata da $a1*$ o $*a1$. Non può essere spezzata da $+$, perché ad esempio $a1 + a * b$ è (secondo le regole di precedenza) lo stesso di $a1 + (a * b)$, e $a * b + a1$ è lo stesso di $(a * b) + a1$.
3. Il resto sono *espressioni*, cioè possono essere spezzate con * o $+$.

Usiamo F per i fattori, T per i termini, e E per le espressioni. Consideriamo la seguente grammatica e l'unico albero sintattico per $a + a * a$

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



Perché la nuova grammatica non è ambigua?

- Un fattore è o un identificatore o (E), per qualche espressione E.
- L'unico albero sintattico per una sequenza

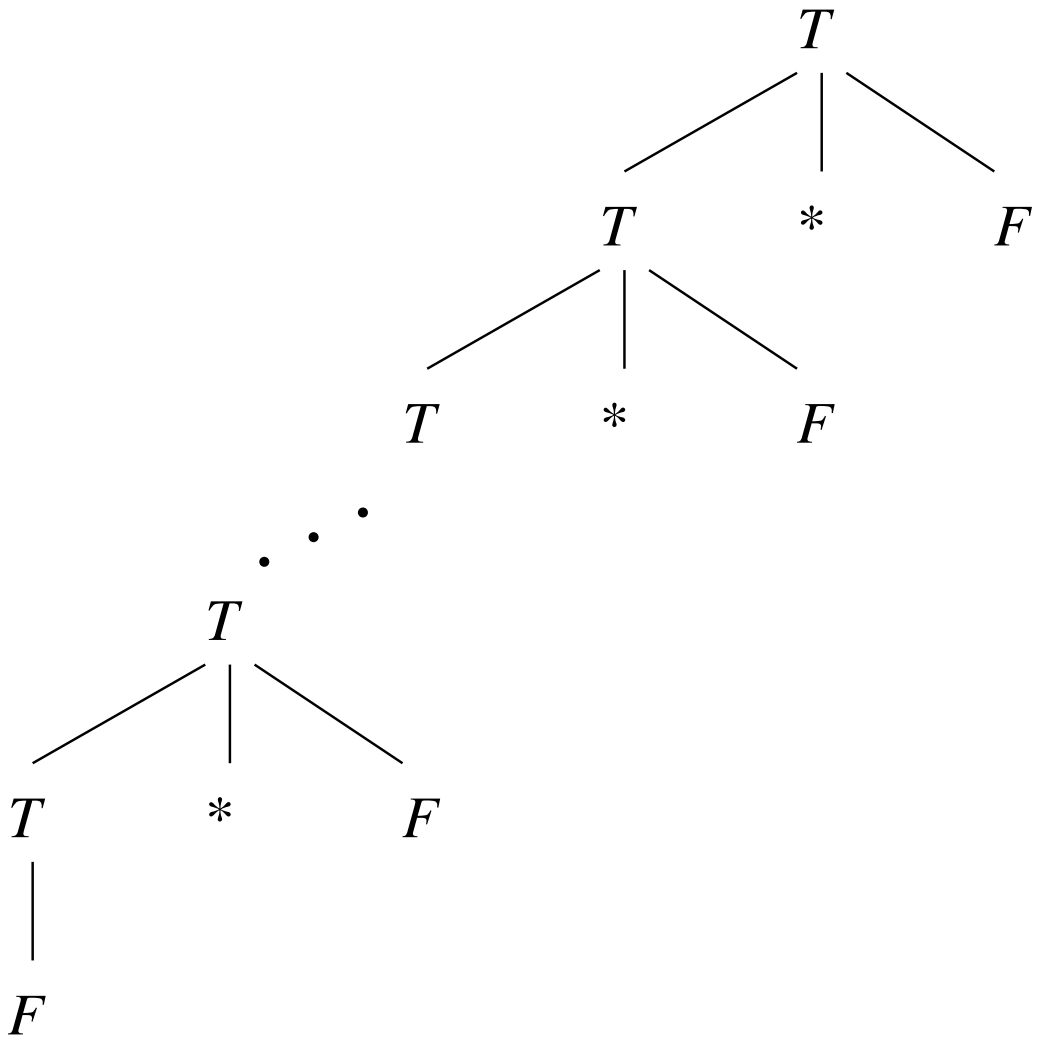
$$f_1 * f_2 * \cdots * f_{n-1} * f_n$$

di fattori è quello che dà $f_1 * f_2 * \cdots * f_{n-1}$ come termine e f_n come fattore, come nell'albero del prossimo lucido.

- Un'espressione è una sequenza

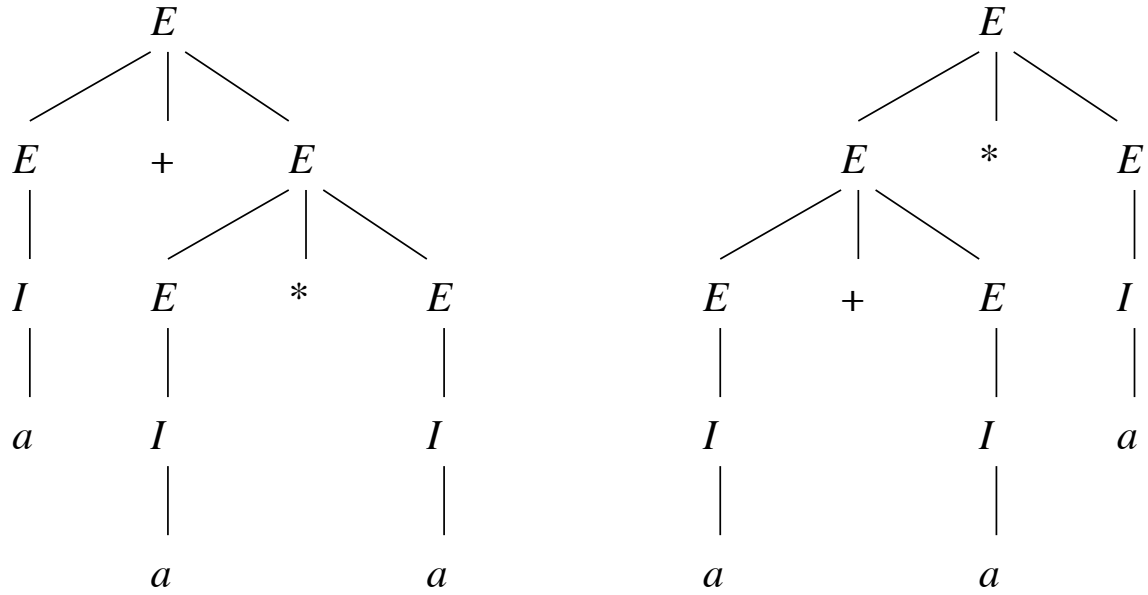
$$t_1 + t_2 + \cdots + t_{n-1} + t_n$$

di termini t_i. Può essere solo raggruppata con $t_1 + t_2 + \cdots + t_{n-1}$ come un'espressione e t_n come un termine.


T
T * F
T * F
T * F
F

Derivazioni a sinistra e ambiguità

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



(a) (b)

danno luogo a due derivazioni:

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

In generale:

- Un albero sintattico, ma molte derivazioni
- Molte derivazioni **a sinistra** implica molti alberi sintattici.
- Molte derivazioni **a destra** implica molti alberi sintattici.

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.

Dimostrazione:

- *(Solo se.)* Se due alberi sintattici sono diversi, hanno un nodo dove sono state usate due diverse produzioni: $A \to X_1 X_2 \cdots X_k$ e $B \to Y_1 Y_2 \cdots Y_m$. Le corrispondenti derivazioni a sinistra useranno queste diverse produzioni e quindi saranno distinte.
- *(Se.)* Per come costruiamo un albero da una derivazione, è chiaro che due derivazioni distinte generano due alberi distinti.

Ambiguità inerente

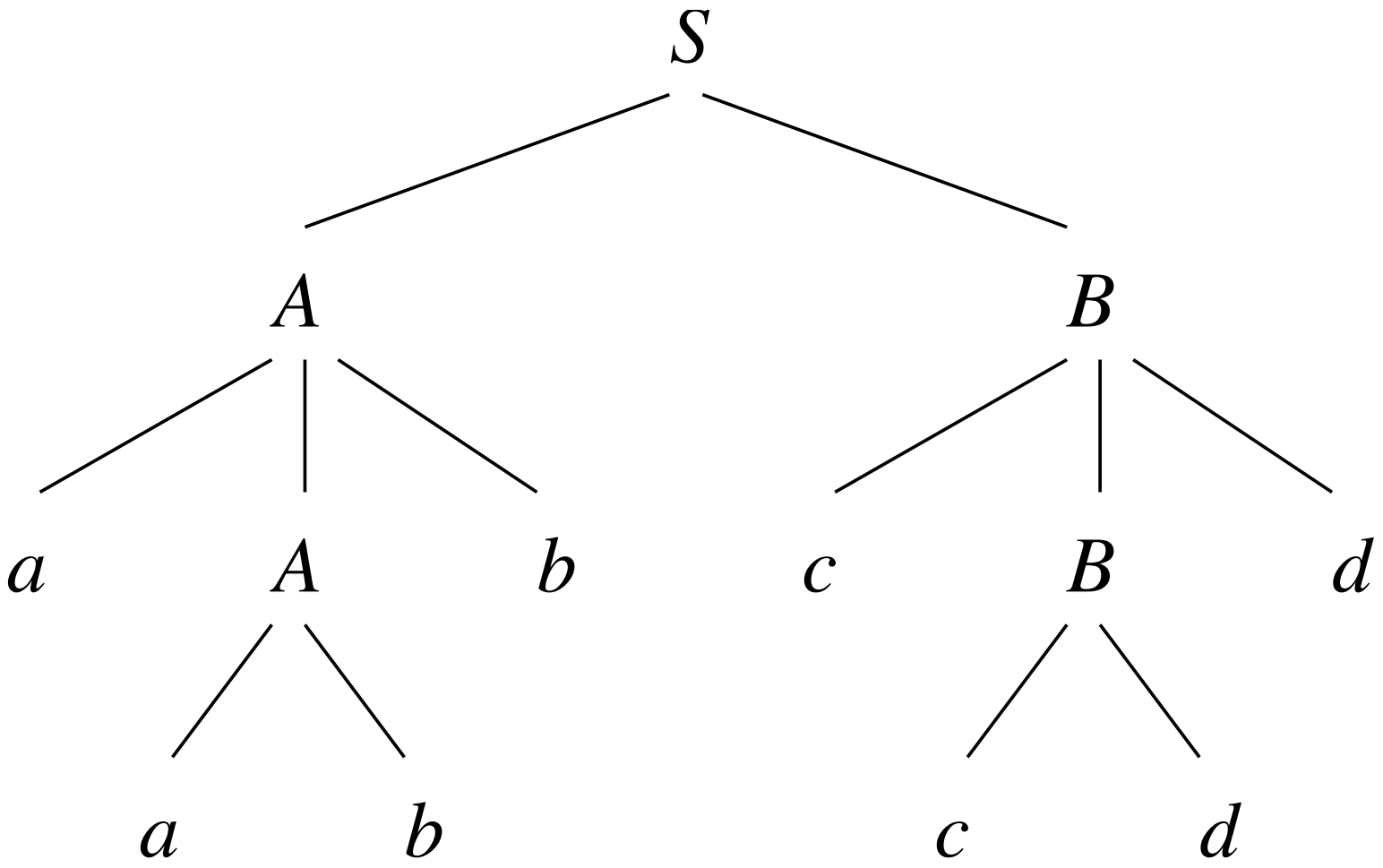
Un CFL L è **inerentemente ambiguo** se **tutte** le grammatiche per L sono ambigue.
Esempio: Consideriamo $L =$

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

Una grammatica per L è

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

Guardiamo la struttura sintattica della stringa *aabbccdd*.



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

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

Può essere provato che **ogni** grammatica per L si comporta come questa. Il linguaggio L è quindi inerentemente ambiguo.

Linguaggi regolari e grammatiche

- Un linguaggio regolare è anche libero da contesto.
- Da una espressione regolare, o da un automa, si può ottenere una grammatica che genera lo stesso linguaggio.

Da espressione regolare a grammatica

Per induzione sulla struttura della espressione regolare:

- se $E = a$, allora produzione $S \to a$
- se $E = \epsilon$, allora produzione $S \to \epsilon$
- se $E = F + G$, allora produzione $S \to F \mid G$
- se $E = FG$, allora produzione $S \to FG$
- se $E = F^*$, allora produzione $S \to FS \mid \epsilon$

Esempio

Espressione regolare: $0^*1(0+1)^*$
Grammatica:

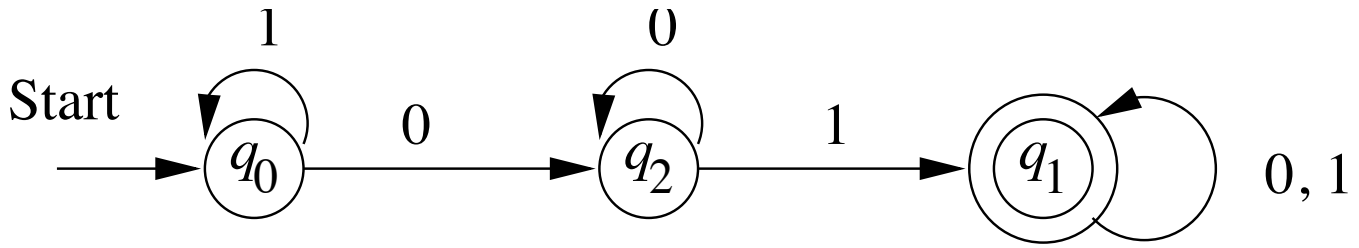
$$S \to ABC$$
$$A \to 0A \mid \epsilon$$
$$B \to 1$$
$$C \to DC \mid \epsilon$$
$$D \to 0 \mid 1$$

Da automa a grammatica

- Un simbolo non-terminale per ogni stato.
- Simbolo iniziale $=$ stato iniziale.
- Per ogni transizione da stato s a stato p con simbolo a, produzione $S \rightarrow aP$.
- Se p stato finale, allora produzione $P \rightarrow \epsilon$

Esempio

Automa:



Grammatica:

$$Q_0 \to 1Q_0 \mid 0Q_2$$
$$Q_2 \to 0Q_2 \mid 1Q_1$$
$$Q_1 \to 0Q_1 \mid 1Q_1 \mid \epsilon$$

La stringa 1101 è accettata dall'automa. Nella grammatica, ha la derivazione:

$$Q_0 \Rightarrow 1Q_0 \Rightarrow 11Q_0 \Rightarrow 110Q_2 \Rightarrow 1101Q_1 \Rightarrow 1101$$

Esercizi su minimizzazione e sulle grammatiche

- Sia L il linguaggio dato dall'intersezione dei seguenti linguaggi:
  - $L_1 = \{w \in \{0,1\}^* |$ in w ci siano almeno due 0 consecutivi$\}$
  - $L_2 = \{w \in \{0,1\}^* |$ in w non ci siano due 1 consecutivi$\}$

  Costruire l'automa D per riconoscere L e minimizzarlo se necessario.
- Ideare la grammatica libera per generare i seguenti linguaggi:
  - $\{0^n 1^n \ n \geq 1\}$
  - L'insieme di tutte le stringhe in $\{0,1\}^*$ tali che il numero di 0 sia il doppio del numero di 1.
  - L'insieme delle stringhe di parentesi bilanciate.

Proprietà di CFL

- **Pumping Lemma per CFL**: simile ai linguaggi regolari.
- **Proprietà di chiusura**: alcune delle proprietà di chiusura dei linguaggi regolari valgono anche per i CFL.
- **Proprietà di decisione**: possiamo controllare l'appartenenza e l'essere vuoto, ma, per esempio, l'equivalenza di CFL è indecidibile.

Forma normale di Chomsky

Ogni CFL (senza ϵ) è 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 è un simbolo terminale. Questa è detta **forma normale di Chomsky (CNF)**. È possibile ottenerla. In particolare è necessario

1. eliminare i **simboli inutili**, quelli che non appaiono in nessuna derivazione $S \overset{*}{\Rightarrow} w$, per simbolo iniziale S e terminale w.
2. eliminare le produzioni ϵ, della forma $A \to \epsilon$.
3. eliminare le **produzioni unità**, cioè produzioni della forma $A \to B$, dove A e B sono variabili.
4. eliminare le produzioni con più di due nonterminali sulla destra.

Esempio

Iniziamo dalla grammatica

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

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

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

$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Aggiungiamo le produzioni

$A \to a, B \to b, Z \to 0, O \to 1$

$P \to +, M \to *, L \to (, R \to)$

e otteniamo la grammatica

$E \to EPT \mid TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$T \to TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$F \to LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$I \to a \mid b \mid IA \mid IB \mid IZ \mid IO$

$A \to a, B \to b, Z \to 0, O \to 1$

$P \to +, M \to *, L \to (, R \to)$

Per eliminare parti destre con più di 2 nonterminali, 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 CFN finale è

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

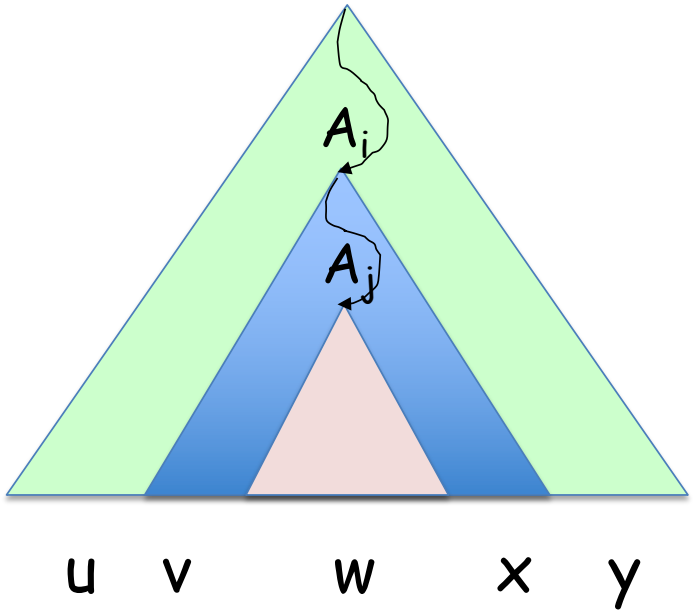
Pumping lemma per CFL

- **Informalmente**:
  In ogni stringa sufficientemente lunga di un CFL si possono trovare due sottostringhe brevi e vicine che è possibile eliminare o ripetere (insieme), ottenendo sempre stringhe del linguaggio.
- **Formalmente**:
  Sia L un CFL. Esiste una costante n tale che, se $z \in L$ e $\mid z \mid \geq n$, possiamo scrivere $z = uvwxy$ con le seguenti condizioni:
  1. $\mid vwx \mid \leq n$
  2. $vx \neq \epsilon$ (almeno una delle due è diversa da ϵ)
  3. per ogni $i \geq 0$, $uv^i wx^i y \in L$.

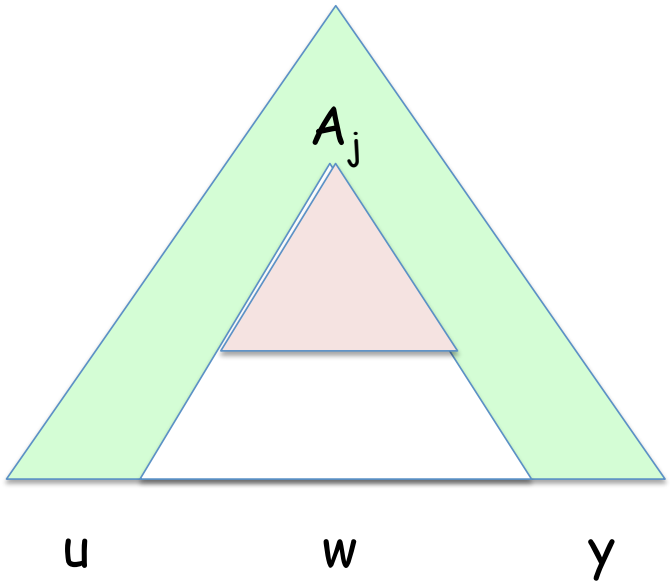
Dimostrazione informale

- Se la stringa w è sufficientemente lunga, l'albero sintattico che produce $w = uvwxy$ ha un simbolo non terminale che si ripete in un cammino dalla radice ad una foglia. Supponiamo $A_i = A_j$.
- Allora può essere individuato il sottoalbero con radice A_j (in rosa nella figura) e chiamato w il suo prodotto.
- Inoltre, dato il sottoalbero con radice A_i (in azzurro nella figura), chiamiamo vwx il suo prodotto.
- Dato che $A_i = A_j$, possiamo rimpiazzare il sottoalbero di A_i con quello di A_j, ottenendo quindi uwy ($i = 0$), che deve ancora appartenere a L.
- Oppure possiamo rimpiazzare il sottoalbero di A_j con quello di A_i, ottenendo $uvvwxxy$ ($i = 2$), ancora generata da L, e così via per ogni i.

Albero sintattico nel pumping lemma

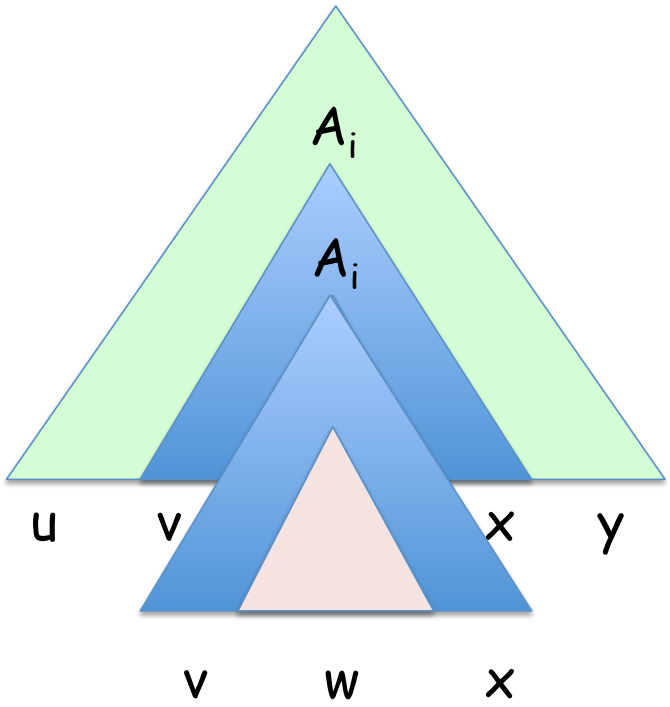


Pumping up and down



i = 0

i = 2



Dimostrazione formale

- Sia G una grammatica libera in Forma Normale di Chomsky, t.c. $L(G) = L \setminus \{\epsilon\}$, e m il numero dei non terminali in G. Scegliamo n come 2^m e supponiamo che z sia lunga almeno n.
- Un albero sintattico il cui cammino più lungo sia lungo m, o meno, ha un prodotto di lunghezza $2^{m-1} = n/2$, o meno.
- Un albero siffatto non può avere z come prodotto, in quanto troppo lunga. Quindi ogni albero con prodotto z deve avere un cammino di lunghezza almeno $k \geq m + 1$.
- Essendoci non più di m non terminali, almeno due simboli nel cammino lungo più di $m + 1$ devono ripetersi. Supponiamo $A_i = A_j$ (where $k - m \leq i < j < k$).
- Allora possiamo "pompare" l'albero come uv^0wx^0y o come uv^2wx^2y, e in generale come uv^iwx^iy, $i \geq 0$.
- Dato che il cammino più lungo nel sottoalbero A_i è al più $m + 1$, abbiamo che $|vwx| \leq 2^m = n$.

Esempio

- Consideriamo $L = \{0^n 1^n 2^n \mid n \geq 1\}$.
- Dato un n generico, scegliamo $z = 0^n 1^n 2^n$.
- Comunque noi spezziamo z in $uvwxy$, con $|\ vwx\ | \leq n$ e v e x non entrambi vuote, vwx non può contenere sia 0 che 2 perché l'ultimo 0 e il primo 2 sono lontani n+1 posti.
- Ci sono i seguenti casi:
  - vwx non contiene 2. Allora vx ha solo 0 e 1. Quindi uwy, che dovrebbe essere in L, ha n 2, ma meno di n 0 o 1.
  - vwx non contiene 0. Analogo.

Esempi

- I CFL non sanno abbinare coppie con lo stesso numero di simboli, se le coppie sono intrecciate.
  - Esempio: $L = \{0^i 1^j 2^i 3^j \mid i, j \geq 1\}$.
  - Dato n, scegliamo $z = 0^n 1^n 2^n 3^n$. Quindi vwx contiene un solo simbolo o due simboli. In ogni caso, le stringhe generate non sono in L.
- I CFL non sanno abbinare due stringhe di lunghezza arbitraria, se sono su un alfabeto di più di un simbolo.
  - Esempio: $L = \{ww \mid w \in \{0, 1\}^*\}$.
  - Dato n, scegliamo $z = 0^n 1^n 0^n 1^n$. Comunque la scomponiamo, non otteniamo stringhe di L.

Proprietà di chiusura dei CFL

Teorema 7.24: I CFL sono chiusi sotto

- unione,
- concatenazione,
- chiusura di Kleene e chiusura positiva $+$

Basta mettere insieme le grammatiche:

- per l'unione: $S \to A \mid B$
- per la concatenazione: $S \to AB$
- per la chiusura di Kleene: $S \to SA \mid \epsilon$
- per la chiusura positiva: $S \to SA \mid A$

Chiusura rispetto all'inversione

Teorema: Se L e CF, allora lo è anche 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 sotto l'intersezione

Sia $L_1 = \{0^n 1^n 2^i : n \geq 1, i \geq 1\}$. Allora L_1 è libero da contesto, con grammatica

$$S \to AB$$
$$A \to 0A1|01$$
$$B \to 2B|2$$

Inoltre, $L_2 = \{0^i 1^n 2^n : n \geq 1, i \geq 1\}$ è libero da contesto, con grammatica

$$S \to AB$$
$$A \to 0A|0$$
$$B \to 1B2|12$$

Invece, $L_1 \cap L_2 = \{0^n 1^n 2^n : n \geq 1\}$ non è CF.

Intersezione tra CFL e linguaggi regolari

Si può dimostrare il seguente teorema.

Teorema 7.27: Se L è CF, e R è regolare, allora $L \cap R$ è CF.

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

1. $L \setminus R$ è CF
2. $\bar{L}$ non è necessariamente CF
3. $L_1 \setminus L_2$ non è necessariamente CF

Prova:

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

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

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

Problemi indecidibili per linguaggi liberi da contesto

I seguenti problemi sono indecidibili (cioè non esiste nessun algoritmo che possa risolverli):

1. Data G, è ambigua?
2. È un dato CFL inerentemente ambiguo?
3. É l'intersezione di due CFL vuota?
4. Dati due CFL, sono uguali?
5. Dato un CFL, è uguale a Σ^*?

Alcuni esercizi di riepilogo

1. Dimostrare che $L = \{0^n \ n = k^2, k > 0\}$ non è regolare.
2. Applicando il Pumping Lemma a un linguaggio regolare "vince l'avversario" e non si può finire la dimostrazione. Dove fallisce nei casi $L = \emptyset$ e $L = \{00, 11\}$.
3. Scrivere la grammatica libera per generare il linguaggio:

$$\{a^i b^j c^k | i \neq j \text{ o } j \neq k\}$$

4. Data la gramatica libera G definita dalle produzioni $S \rightarrow aS|aSbS|\epsilon$: (i) dimostrare che è ambigua; (ii) trovare una grammatica per lo stesso linguaggio che non lo sia.
5. Dimostrare che $L = \{a^i b^j c^k | i < j < k\}$ non è libero.