

LINGUAGGI CONTEXT FREE

Corso di Fondamenti di Informatica - modulo I

Giorgio Gambosi

a.a. 2023-2024

Equivalenza tra ASF, RG e RE

La derivazione di una stringa generata da una grammatica di tipo 2 può essere rappresentata mediante una struttura ad albero. Tali alberi vengono chiamati **alberi di derivazione**, o **alberi sintattici**.

In un albero sintattico, ad ogni nodo interno è associato un simbolo non-terminale e ad ogni foglia è associato un simbolo terminale. Per ogni produzione del tipo $S \rightarrow aSbA$ che viene applicata nel processo di derivazione, il nodo interno etichettato con S avrà nell'albero quattro figli etichettati con a, S, b, A

Data la grammatica G avente le produzioni

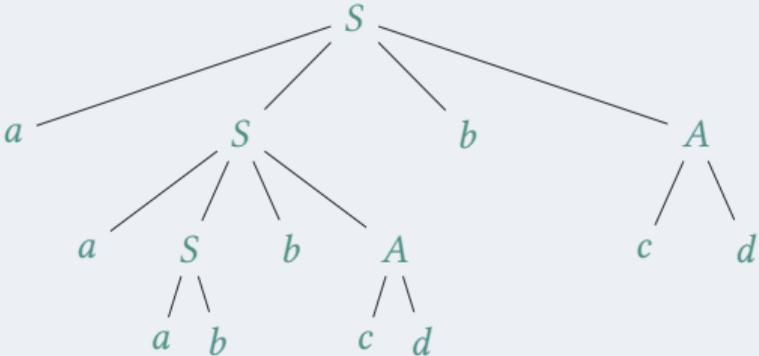
$$S \longrightarrow aSbA \mid ab$$

$$A \longrightarrow cAd \mid cd$$

la stringa $aaabbcdbcd \in L(\mathcal{G})$ può essere così derivata:

$$\begin{aligned} S &\Longrightarrow aSbA \Longrightarrow aaSbAbA \Longrightarrow aaabbAbA \\ &\Longrightarrow aaabbcdbA \Longrightarrow aaabbcdbcd. \end{aligned}$$

Albero sintattico



In questa rappresentazione non si mantiene traccia dell'ordine con cui le produzioni sono state applicate. Ad un unico albero possono corrispondere diverse derivazioni.

Vantaggio: un albero di derivazione fornisce una descrizione sintetica della struttura sintattica della stringa, indipendentemente dall'ordine con cui le produzioni sono state applicate.

Al fine di studiare alcune proprietà dei linguaggi generati da queste grammatiche, è utile considerare grammatiche “ristrette”, comprendenti soltanto produzioni con struttura particolare.

È importante dimostrare che i linguaggi non contestuali possono essere generati mediante tali tipi di grammatiche.

Una grammatica G è in forma ridotta se

1. non contiene ε -produzioni (se non, eventualmente, in corrispondenza all'assioma, ed in tal caso l'assioma non compare mai al lato destro di una produzione),
2. non contiene **produzioni unitarie**, cioè produzioni del tipo

$$A \rightarrow B, \text{ con } A, B \in V_N,$$

3. non contiene **simboli inutili**, cioè simboli che non compaiono in nessuna derivazione di una stringa di soli terminali.

Trasformazione di una grammatica $\mathcal{G} = \langle V_T, V_N, P, S \rangle$ di tipo 2 in una grammatica equivalente in forma ridotta mediante sequenza di passi.

1. A partire da \mathcal{G} , derivazione di \mathcal{G}_1 di tipo 2 senza ε -produzioni tale che $L(\mathcal{G}_1) = L(\mathcal{G}) - \{\varepsilon\}$.
2. A partire da \mathcal{G}_1 , derivazione di \mathcal{G}_2 di tipo 2 senza ε -produzioni e senza produzioni unitarie tale che $L(\mathcal{G}_2) = L(\mathcal{G}_1)$.
3. A partire da \mathcal{G}_2 , derivazione di \mathcal{G}_3 di tipo 2 senza ε -produzioni, senza produzioni unitarie e senza simboli inutili tale che $L(\mathcal{G}_3) = L(\mathcal{G}_2)$.
4. La grammatica \mathcal{G}_4 , di tipo 2, equivalente a \mathcal{G} coincide con \mathcal{G}_3 se $\varepsilon \notin L(\mathcal{G})$; altrimenti, \mathcal{G}_4 è ottenuta da \mathcal{G}_3 introducendo un nuovo assioma ed un opportuno insieme di produzioni su tale simbolo.

Teorema

Data una grammatica $G = \langle V_T, V_N, P, S \rangle$ il cui insieme di produzioni P comprende soltanto produzioni di tipo non contestuale e produzioni vuote, esiste una grammatica non contestuale G' tale che $L(G') = L(G) - \{\varepsilon\}$.

Passo 1

Determinazione dell'insieme $N \subseteq V_N$ dei simboli che si annullano, cioè i non terminali da cui è possibile derivare ε in G .

Costruzione di una sequenza $N_0, N_1, \dots, N_k = N$ di sottoinsiemi di V_N , con $N_0 = \{A \in V_N \mid A \rightarrow \varepsilon \in P\}$ e N_{i+1} derivato da N_i :

$$N_{i+1} = N_i \cup \{B \in V_N \mid (B \rightarrow \beta \in P) \wedge (\beta \in N_i^+)\}.$$

La costruzione termina quando $N_{k+1} = N_k, k \geq 0$.

$\varepsilon \in L(G)$ se e solo se $S \in N$.

Passo 1

input grammatica $G = \langle V_T, V_N, P, S \rangle$;
output insieme $N \subseteq V_N$ dei simboli che si annullano;
 $N := \{A \in V_N \mid A \rightarrow \varepsilon \in P\}$;
repeat
 $\hat{N} := N$;
 $N := \hat{N} \cup \{B \in V_N \mid (B \rightarrow \beta \in P) \wedge (\beta \in \hat{N}^+)\}$
until $N = \hat{N}$

Costruzione dell'insieme P' delle produzioni di G' :

- ⊙ Si esamina ciascuna produzione $A \rightarrow \alpha$ di P , con l'esclusione delle ε -produzioni
 - Se nessun simbolo di α è annullabile: $A \rightarrow \alpha$ è inserita in P'
 - Altrimenti α contiene $k > 0$ simboli che si annullano: sono inserite in P' tutte le possibili produzioni ottenute da $A \rightarrow \alpha$ eliminando da α uno dei sottoinsiemi di simboli che si annullano

Passo 1

input grammatica $G = \langle V_T, V_N, P, S \rangle$, insieme $N \subseteq V_N$ dei simboli che si annullano;

output insieme P' delle produzioni di G' ;

$P' := \emptyset$;

for each $A \rightarrow \alpha \in P$ con $\alpha \neq \varepsilon$ **do**

 sia $\alpha = Z_1, \dots, Z_t$;

$J := \{i \mid Z_i \in N\}$;

for each $J' \in 2^J$ **do**

if $J' \neq \{1, \dots, t\}$ **then**

 sia β la stringa ottenuta eliminando da α ogni Z_i
 con $i \in J'$;

$P' := P' \cup \{A \rightarrow \beta\}$

Nel caso in cui $L(\mathcal{G})$ contiene ε , si può ottenere da \mathcal{G}' una grammatica equivalente a \mathcal{G} tramite la semplice introduzione di una ε -produzione sull'assioma di \mathcal{G}' .

Consideriamo la grammatica $G = \langle \{a, b\}, \{S, A, B\}, P, S \rangle$, le cui produzioni P sono:

$$S \longrightarrow A \mid SSa$$

$$A \longrightarrow B \mid Ab \mid \varepsilon$$

$$B \longrightarrow S \mid ab \mid aA.$$

Sequenza di insiemi di simboli annullabili:

$$N_0 = \{A\}$$

$$N_1 = \{S, A\}$$

$$N_2 = \{S, A, B\}$$

$$N_3 = \{S, A, B\} = N_2 = N$$

Produzioni P' :

$$S \longrightarrow A \mid SSa \mid Sa \mid a \mid \varepsilon$$

$$A \longrightarrow B \mid Ab \mid b$$

$$B \longrightarrow S \mid ab \mid aA \mid a.$$

Teorema

Per ogni grammatica \mathcal{G} di tipo 2 senza ε -produzioni, esiste sempre una grammatica \mathcal{G}' di tipo 2 senza ε -produzioni, priva di produzioni unitarie ed equivalente a \mathcal{G} .

Sia, per ogni $A \in V_N$, $U(A)$, il sottoinsieme di $V_N - \{A\}$ comprendente tutti i non terminali derivabili da A applicando una sequenza di produzioni unitarie,

$$U(A) = \{B \in V_N - \{A\} \mid A \xRightarrow{*} B\}$$

Data la grammatica $\mathcal{G} = \langle V_T, V_N, P, S \rangle$, P' è costruito:

- ⊙ inserendo dapprima in P' tutte le produzioni non unitarie in P
- ⊙ inserendo in P' , per ogni non terminale A e per ogni $B \in U(A)$, la produzione $A \rightarrow \beta$ se e solo se in P esiste una produzione non unitaria $B \rightarrow \beta$

Passo 2

input Grammatica CF $G = \langle V_T, V_N, P, S \rangle$ priva di ε -produzioni;

output Grammatica CF $G' = \langle V_T, V_N, P', S \rangle$ priva di ε -produzioni e di produzioni unitarie equivalente a G ;

$P' := \{A \rightarrow \alpha \in P \mid \alpha \notin V_N\}$;

for each $A \in V_N$ **do**

$P' := P' \cup \{A \rightarrow \beta \mid B \rightarrow \beta \in P \wedge B \in U(A) \wedge \beta \notin V_N\}$

Costruire un algoritmo che, data una grammatica \mathcal{G} di tipo 2 senza ε -produzioni e dato un non terminale A della grammatica, determini l'insieme $U(A)$

Passo iniziale: inserisci in $U(A)$ tutti i simboli B tali che $A \rightarrow B$

Passo iterativo: per ogni simbolo $B \in U(A)$, inserisci in $U(A)$ tutti i simboli C tali che $B \rightarrow C$;
termina se nessun nuovo simbolo è stato inserito in $U(A)$

Teorema

Per ogni grammatica $\mathcal{G} = \langle V_T, V_N, P, S \rangle$ di tipo 2 senza ε -produzioni e senza produzioni unitarie, esiste sempre una grammatica \mathcal{G}' di tipo 2 senza ε -produzioni, priva di produzioni unitarie e di simboli inutili ed equivalente a \mathcal{G} .

Affinché un simbolo $A \in V_N$ non sia inutile, è necessario che nella grammatica \mathcal{G} si abbia che:

- A sia un simbolo **fecondo**, vale a dire che da esso siano generabili stringhe di terminali, cioè $\exists w \in V_T^+$ tale che $A \xRightarrow{*} w$;
- A sia generabile dall'assioma in produzioni che non contengano simboli non fecondi, cioè $S \xRightarrow{*} \alpha A \beta$ con $\alpha, \beta \in (V_T \cup V_N)^*$ e, per ogni $B \in V_N$ in α o β , valga la proprietà precedente.

Equivalentemente, un simbolo $A \in V_N$ non è inutile se esiste una derivazione

$$S \xRightarrow{*} \alpha A \beta \xRightarrow{*} w \in V_T^+.$$

Un non terminale A è fecondo se e solo se vale una delle due condizioni seguenti:

1. esiste $w \in V_T^+$ tale che $A \rightarrow w \in P$;
2. esiste $\alpha \in (V_N \cup V_T)^*$ tale che $A \rightarrow \alpha \in P$ e tutti i simboli non terminali in α sono fecondi.

Passo 3

input Grammatica non contestuale $G = \langle V_T, V_N, P, S \rangle$,
priva di ε -produzioni e di produzioni unitarie;

output Grammatica non contestuale $\hat{G} = \langle V_T, \hat{V}_N, \hat{P}, S \rangle$,
priva di ε -produzioni, di produzioni unitarie e di simboli
non fecondi, equivalente a G ;

$F := \emptyset$;

while $\exists A \in V_N - F$ per cui $\exists A \rightarrow \alpha \in P$, con $\alpha \in (F \cup V_T)^*$ **do**

$F := F \cup \{A\}$;

$\hat{P} := \{A \rightarrow \alpha \mid A \rightarrow \alpha \in P, A \in F, \alpha \in (F \cup V_T)^*\}$

È necessario verificare che i simboli rimasti siano generabili a partire dall'assioma.

Ciò può essere effettuato in modo iterativo, osservando che A è generabile a partire da S se vale una delle due condizioni seguenti:

1. esistono $\alpha, \beta \in (F \cup V_T)^*$ tali che $S \rightarrow \alpha A \beta \in \hat{P}$;
2. esistono $\alpha, \beta \in (F \cup V_T)^*$ e $B \in F$, generabile a partire da S , tali che $B \rightarrow \alpha A \beta \in \hat{P}$.

Passo 3

input Grammatica non contestuale $\hat{G} = \langle V_T, F, \hat{P}, S \rangle$ priva di ε -produzioni, di produzioni unitarie e di simboli non fecondi;

output Grammatica non contestuale $G' = \langle V_T, V'_N, P', S \rangle$ priva di ε -produzioni, produzioni unitarie, simboli non fecondi, simboli non generabili da S , equivalente a \hat{G} ;

$NG := F - \{S\}$;

for each $A \in F - NG$ **do**

for each $\alpha \in (V_T \cup F)^*$ tale che $A \rightarrow \alpha \in \hat{P}$ **do**

for each $B \in NG$ che appare in α **do**

$NG := NG - \{B\}$;

$V'_N := F - NG$;

$P' := \{A \rightarrow \alpha \mid A \rightarrow \alpha \in \hat{P}, A \in V'_N, \alpha \in (V'_N \cup V_T)^*\}$

Passo 3

Al fine di eliminare i simboli inutili (non fecondi e non generabili da S) è necessario applicare i due algoritmi nell'ordine dato: eliminare prima i simboli non generabili e poi quelli non fecondi può far sì che non tutti i simboli inutili vengano rimossi dalla grammatica.

Infatti, si consideri la grammatica

$$\begin{array}{l} S \longrightarrow AB \mid a \\ A \longrightarrow a. \end{array}$$

Procedendo prima all'eliminazione dei simboli non derivabili dall'assioma e poi all'eliminazione di quelli non fecondi, otterremo le seguenti grammatiche:

$$\begin{array}{l} S \longrightarrow AB \mid a \\ A \longrightarrow a \end{array} \quad \text{e successivamente} \quad \begin{array}{l} S \longrightarrow a \\ A \longrightarrow a. \end{array}$$

che non è in forma ridotta.

Se invece si procede come indicato sopra si ottengono le due grammatiche

$$\begin{array}{l} S \rightarrow a \\ A \rightarrow a \end{array} \quad \text{e successivamente} \quad S \rightarrow a.$$

Una grammatica $\mathcal{G} = \langle V_T, V_N, P, S \rangle$ può essere estesa in una grammatica $\mathcal{G}' = \langle V_T, V'_N, P', S' \rangle$ che generi anche la stringa vuota ε nel modo seguente:

1. $V'_N = V_N \cup \{T\}$, dove $T \notin V_N$;
2. $P' = P \cup \{T \rightarrow \varepsilon\} \cup \{T \rightarrow \alpha \mid S \rightarrow \alpha \in P\}$;
3. $S' = T$.

- 1 $S \rightarrow aUVb \mid TZ$
- 2 $Z \rightarrow aZ$
- 3 $U \rightarrow bU \mid b$
- 4 $V \rightarrow W$
- 5 $V \rightarrow aY$
- 6 $Y \rightarrow bY \mid b$
- 7 $W \rightarrow cWd \mid cd$
- 8 $T \rightarrow tT \mid tz.$

- ⊙ L'eliminazione delle produzioni unitarie porta ad escludere la produzione 4 e ad aggiungere una terza produzione alla 1.
- ⊙ L'eliminazione di simboli non fecondi porta ad escludere la produzione 2 e la seconda produzione della 1.
- ⊙ L'eliminazione dei simboli non raggiungibili porta infine ad escludere la produzione 8.

Si ottiene quindi la grammatica

$$\begin{aligned} S &\longrightarrow aUVb \mid aUWb \\ U &\longrightarrow bU \mid b \\ V &\longrightarrow aY \\ Y &\longrightarrow bY \mid b \\ W &\longrightarrow cWd \mid cd. \end{aligned}$$

Trasformare la grammatica seguente in una grammatica equivalente in forma ridotta.

$$S \rightarrow H \mid Z$$

$$H \rightarrow A \mid \varepsilon$$

$$Z \rightarrow bZb$$

$$A \rightarrow bbABa \mid a$$

$$B \rightarrow cB \mid BZY \mid \varepsilon$$

$$Y \rightarrow Yb \mid b.$$

Una grammatica di tipo 2 si dice in **Forma Normale di Chomsky** se tutte le sue produzioni sono del tipo $A \rightarrow BC$ o del tipo $A \rightarrow a$, con $A, B, C \in V_N$ ed $a \in V_T$.

Teorema

Data una grammatica G non contestuale tale che $\varepsilon \notin L(G)$, esiste una grammatica equivalente in CNF.

Come mostrato, è possibile derivare una grammatica G' in forma ridotta equivalente a G : in particolare, G' non ha produzioni unitarie.

Da G' , è possibile derivare una grammatica G'' in CNF, equivalente ad essa

Forma normale di Chomsky

Sia $A \rightarrow \zeta_{i_1} \dots \zeta_{i_n}$ una produzione di G' non in CNF. Si possono verificare due casi:

⊙ $n \geq 3$ e $\zeta_{i_j} \in V_N, j = 1, \dots, n$.

In tal caso, introduciamo $n - 2$ nuovi simboli non terminali Z_1, \dots, Z_{n-2} e sostituiamo la produzione $A \rightarrow \zeta_{i_1} \dots \zeta_{i_n}$ con le produzioni

$$\begin{aligned} A &\rightarrow \zeta_{i_1} Z_1 \\ Z_1 &\rightarrow \zeta_{i_2} Z_2 \\ &\dots \\ Z_{n-2} &\rightarrow \zeta_{i_{n-1}} \zeta_{i_n}. \end{aligned}$$

⊙ $n \geq 2$ e $\zeta_j \in V_T$ per qualche $j \in \{1, \dots, n\}$.

In tal caso per ciascun $\zeta_j \in V_T$ introduciamo un nuovo non terminale \bar{Z}_{i_j} , sostituiamo \bar{Z}_{i_j} a ζ_j nella produzione considerata e aggiungiamo la produzione $\bar{Z}_{i_j} \rightarrow \zeta_j$. Così facendo o abbiamo messo in CNF la produzione considerata (se $n = 2$) o ci siamo ricondotti al caso precedente (se $n \geq 3$).

Si consideri la grammatica di tipo 2 che genera il linguaggio $\{a^n b^n \mid n \geq 1\}$ con le produzioni

$$S \rightarrow aSb$$

$$S \rightarrow ab$$

La grammatica è in forma ridotta.

Forma normale di Chomsky

Grammatica in CNF equivalente:

⊙ $V_N = \{S, Z_1, \bar{Z}_1, \bar{Z}_2, \bar{Z}_3, \bar{Z}_4\}$

⊙ P :

$$S \rightarrow \bar{Z}_1 Z_1$$

$$Z_1 \rightarrow S \bar{Z}_2$$

$$S \rightarrow \bar{Z}_3 \bar{Z}_4$$

$$\bar{Z}_1 \rightarrow a$$

$$\bar{Z}_2 \rightarrow b$$

$$\bar{Z}_3 \rightarrow a$$

$$\bar{Z}_4 \rightarrow b$$

Forma normale di Greibach

Una grammatica di tipo 2 si dice in **Forma Normale di Greibach** (GNF) se tutte le sue produzioni sono del tipo $A \rightarrow a\beta$, con $A \in V_N$, $a \in V_T$, $\beta \in V_N^*$.

Si osservi come una grammatica di tipo 3 corrisponda al caso in cui $|\beta| \leq 1$

Trasformazione in forma normale di Greibach

Lemma (Sostituzione)

Sia G una grammatica di tipo 2 le cui produzioni includono

$$\begin{aligned} A &\longrightarrow \alpha_1 B \alpha_2 \\ B &\longrightarrow \beta_1 \mid \dots \mid \beta_n, \end{aligned}$$

($\alpha_1, \alpha_2 \in V^*$) e in cui non compaiono altre B -produzioni oltre a quelle indicate. La grammatica G' in cui la produzione $A \longrightarrow \alpha_1 B \alpha_2$ è stata sostituita dalla produzione

$$A \longrightarrow \alpha_1 \beta_1 \alpha_2 \mid \dots \mid \alpha_1 \beta_n \alpha_2$$

è equivalente alla grammatica G .

Trasformazione in forma normale di Greibach

Lemma (Eliminazione ricursione sinistra)

Sia data una grammatica G con ricursione sinistra sul non terminale A e sia

$$A \longrightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n,$$

l'insieme dell A -produzioni in \mathcal{G} , dove nessuna delle stringhe β_i inizia per A . La grammatica G' in cui le A -produzioni in G sono state sostituite dalle produzioni:

$$\begin{aligned} A &\longrightarrow \beta_1 A' \mid \dots \mid \beta_n A' \mid \beta_1 \dots \mid \beta_n \\ A' &\longrightarrow \alpha_1 A' \mid \dots \mid \alpha_m A' \mid \alpha_1 \dots \mid \alpha_m \end{aligned}$$

è equivalente a G e non presenta ricursione sinistra rispetto al non terminale A .

Teorema

Ogni linguaggio non contestuale L tale che $\varepsilon \notin L$ può essere generato da una grammatica di tipo 2 in GNF.

Trasformazione in forma normale di Greibach

Si assuma che G sia una grammatica CF in CNF che generai L .

La derivazione di G' da G avviene applicando iterativamente i due lemmi precedenti, a partire da un ordinamento arbitrario A_1, \dots, A_n tra i non terminali di G .

Fase 1

⊙ per k da 2 a n

• per j da 1 a $k - 1$

- Applica il Lemma di sostituzione ad ogni produzione del tipo $A_k \rightarrow A_j\alpha$
- Applica il Lemma di eliminazione della ricorsione sinistra ad ogni produzione del tipo $A_k \rightarrow A_k\alpha$

Trasformazione in forma normale di Greibach

Siano B_1, \dots, B_l i non terminali aggiunti. A questo punto le produzioni sono tutte di uno tra i tipi:

- ⊙ (a) $A_k \rightarrow A_j \gamma$ con $j > k, \gamma \in (V_N \cup \{B_1, \dots, B_l\})^*$
- ⊙ (b) $A_k \rightarrow a \gamma$ con $a \in V_T, \gamma \in (V_N \cup \{B_1, \dots, B_l\})^*$
- ⊙ (c) $B_k \rightarrow \gamma$ con $\gamma \in V_N \cdot (V_N \cup \{B_1, \dots, B_l\})^*$

Inoltre, le A_k -produzioni sono:

- ⊙ se $k = n$ tutte del tipo (b)
- ⊙ se $k < n$ del tipo (b) o del tipo (a), con $j \leq n$

Fase 2

- ⊙ per h da $n - 1$ a 1
 - per j da n a h
 - Applica il Lemma di sostituzione ad ogni produzione del tipo $A_h \rightarrow A_j \gamma$

A questo punto le produzioni sono tutte del tipo (b) o (c)

Fase 3

- ⊙ per i da 1 a l
 - per j da 1 a m
 - Applica il Lemma di sostituzione ad ogni produzione del tipo $B_i \rightarrow A_j \gamma$

A questo punto le produzioni sono tutte del tipo (b)

Data una grammatica avente le produzioni

$$S \rightarrow AB \mid b$$

$$A \rightarrow b \mid BS$$

$$B \rightarrow a \mid BA \mid AS,$$

consideriamo in modo arbitrario l'ordinamento S, A, B tra i non terminali

Fase 1.

Sostituiamo alla produzione $B \rightarrow AS$ la coppia di produzioni $B \rightarrow bS \mid BSS$:

$$S \rightarrow AB \mid b$$

$$A \rightarrow b \mid BS$$

$$B \rightarrow a \mid bS \mid BA \mid BSS$$

Fase 1.

Eliminiamo la ricursione sinistra nelle B -produzioni, ottenendo

$$S \rightarrow AB \mid b$$

$$A \rightarrow b \mid BS$$

$$B \rightarrow a \mid bS \mid aB' \mid bSB'$$

$$B' \rightarrow A \mid SS \mid AB' \mid SSB'.$$

Fase 2.

Sostituiamo alla produzione $A \rightarrow BS$ le produzioni $A \rightarrow aS \mid bSS \mid aB'S \mid bSB'S$ ottenendo

$$\begin{aligned} S &\rightarrow AB \mid b \\ A &\rightarrow b \mid aS \mid bSS \mid aB'S \mid bSB'S \\ B &\rightarrow a \mid bS \mid aB' \mid bSB' \\ B' &\rightarrow A \mid SS \mid AB' \mid SSB'. \end{aligned}$$

Fase 2.

Sostituiamo alla produzione $S \rightarrow AB$ le produzioni $S \rightarrow aSB \mid bSSB \mid aB'SB \mid bSB'SB \mid bB$ ottenendo

$$\begin{aligned} S &\rightarrow aSB \mid bSSB \mid aB'SB \mid bSB'SB \mid bB \mid b \\ A &\rightarrow b \mid aS \mid bSS \mid aB'S \mid bSB'S \\ B &\rightarrow a \mid bS \mid aB' \mid bSB' \\ B' &\rightarrow A \mid SS \mid AB' \mid SSB'. \end{aligned}$$

Esempio

Fase 3.

Sostituiamo nelle B' -produzioni ottenendo

$$\begin{aligned} S &\longrightarrow aSB \mid bSSB \mid aB'SB \mid bSB'SB \mid bB \mid b \\ A &\longrightarrow b \mid aS \mid bSS \mid aB'S \mid bSB'S \\ B &\longrightarrow a \mid bS \mid aB' \mid bSB' \\ B' &\longrightarrow aS \mid bSS \mid aB'S \mid bSB'S \mid b \\ &\quad asBS \mid bSSBS \mid aB'SBS \mid bSB'SBS \mid bSB \mid bS \mid \\ &\quad aSB' \mid bSSB' \mid aB'SB' \mid bSB'SB' \mid bB' \mid \\ &\quad aSBSB' \mid bSSBSB' \mid aB'SBSB' \mid \\ &\quad bSB'SBSB' \mid bBSB' \mid bSB'. \end{aligned}$$

Sia data la seguente grammatica:

$$S \rightarrow AbA \mid b$$

$$A \rightarrow SaS \mid a.$$

Derivare una grammatica in GNF equivalente ad essa.

Non contestualità

Pumping lemma

Teorema

Sia $L \subseteq V_T^*$ un linguaggio non contestuale. Esiste allora una costante n tale che se $z \in L$ e $|z| \geq n$ allora esistono 5 stringhe $u, v, w, x, y \in V_T^*$ tali che

- i) $uvwxy = z$
- ii) $|vx| \geq 1$
- iii) $|vwx| \leq n$
- iv) $\forall i \geq 0 \ uv^iwx^iy \in L.$

Pumping lemma: interpretazione come gioco a due

Se L è context free, Alice vince sempre questo gioco con Bob:

1. Alice fissa un intero $n > 0$ opportuno
2. Bob sceglie una stringa $z \in L$ con $|z| > n$
3. Alice divide z in cinque parti $uvwxy$ con $|vwx| \leq n$ e $|vx| \geq 1$
4. Bob sceglie un intero $i \geq 0$
5. Alice mostra a Bob che $uv^iwx^iy \in L$

Grammatica $G = \langle V_T, V_N, P, S \rangle$ in CNF che genera $L = L(G)$ e sia $k = |V_N|$ il numero di simboli non terminali in G .

Qualunque albero sintattico $A(\sigma)$ relativo ad una stringa $\sigma \in V_T^*$ derivata in G sarà tale da avere tutti i nodi interni (corrispondenti a simboli non terminali) di grado 2, eccetto quelli aventi foglie dell'albero come figli, che hanno grado 1.

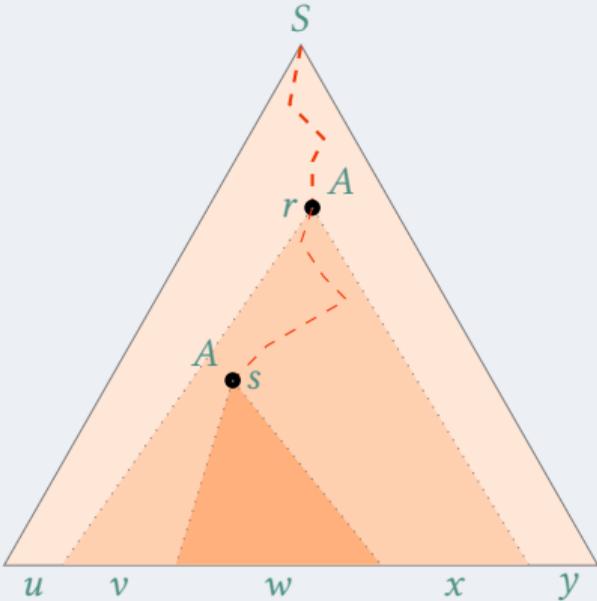
- ⊙ Se h è l'altezza di A (numero massimo di archi, e anche numero massimo di nodi interni, in un cammino dalla radice ad una foglia), il massimo numero di foglie $l(h)$ è dato dal caso in cui l'albero è completo (i nodi interni hanno due figli, eccetto i padri di foglie, che ne hanno uno). Si può facilmente verificare che in tal caso abbiamo $l(h) = 2^{h-1}$, in quanto $l(1) = 1 = 2^0$ e $l(h+1) = 2 \cdot l(h) = 2 \cdot 2^{h-1} = 2^h$
- ⊙ Se l'albero sintattico $A(\sigma)$ relativo alla stringa $\sigma \in L$ ha altezza $h(\sigma)$, la lunghezza di σ è allora $|\sigma| \leq 2^{h(\sigma)-1}$, e quindi $h(\sigma) \geq 1 + \log_2 |\sigma|$
- ⊙ Se σ è una stringa sufficientemente lunga (in questo caso, $|\sigma| > 2^{|V_N|-1}$), ne risulta che $h(\sigma) \geq 1 + \log_2 |\sigma| > |V_N|$
- ⊙ Quindi, se $|\sigma| > 2^{|V_N|-1}$ esiste almeno un cammino c dalla radice ad una foglia di $A(\sigma)$ che attraversa almeno $|V_N| + 1$ nodi interni

- ⊙ I nodi interni di $A(\sigma)$ sono etichettati da simboli non terminali (le parti sinistre della produzioni nella derivazione di σ)
- ⊙ Dato che i simboli non terminali sono $|V_N|$ mentre i nodi interni in c sono più di $|V_N|$, deve esistere (per il pigeonhole principle) un simbolo non terminale A che compare in due diversi nodi di c
- ⊙ Di questi due nodi, indichiamo con r il nodo più vicino alla radice e con s il nodo associato ad A più vicino alla foglia
- ⊙ Indichiamo con $r(\sigma)$ e $s(\sigma)$ le sottostringhe di σ corrispondenti alle foglie dei due sottoalberi $R(\sigma), S(\sigma)$ di $A(\sigma)$ aventi radice r e s
- ⊙ Dato che s è un discendente di r , necessariamente $s(\sigma)$ è una sottostringa di $r(\sigma)$, per cui esistono due sottostringhe di v, x di σ tali che $r(\sigma) = v \cdot s(\sigma) \cdot x$

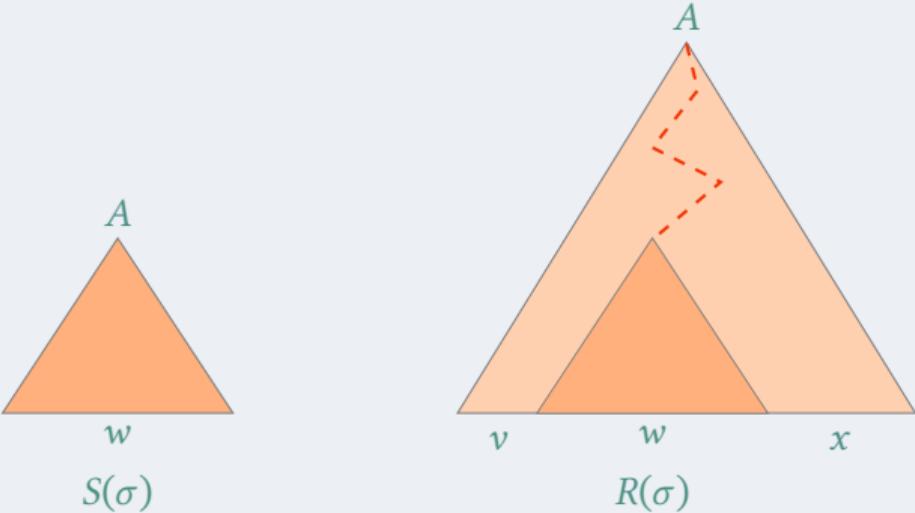
- ⊙ La grammatica considerata è in CNF, per cui non sono presenti produzioni unitarie (a parte quelle relative alle foglie): di conseguenza, non può essere $s(\sigma) = r(\sigma)$, e quindi $|vx| > 1$
- ⊙ Senza perdere generalità, possiamo assumere che $r(\sigma)$ sia il nodo in c più vicino alle foglie per il quale c'è un nodo sottostante $s(\sigma)$ associato allo stesso non terminale: quindi, il cammino più lungo da $r(\sigma)$ ad una foglia attraversa al più $|V_N| + 1$ nodi interni (esso stesso incluso).
- ⊙ Dalle osservazioni precedenti, ne deriva che $r(\sigma)$ ha lunghezza al più $2^{|V_N|+1-1} = 2^{|V_N|}$

Dimostrazione

Poniamo $s(\sigma) = w$ e quindi $r(\sigma) = vwx$.



Dimostrazione



- ⊙ Gli alberi $R(\sigma)$ e $S(\sigma)$ possono essere sostituiti (avendo radice corrispondente allo stesso non terminale) l'uno all'altro all'interno di un qualunque albero sintattico
- ⊙ Quindi, anche la stringa uwy è generata dalla grammatica (sostituendo, in $A(\sigma)$, $S(\sigma)$ a $R(\sigma)$)
- ⊙ Mediante la sostituzione opposta, anche la stringa $uvvwxy$ risulta generabile.

La proprietà mostrata fornisce soltanto una condizione necessaria di un linguaggio context free: non può essere utilizzata per mostrare la contestualità di un linguaggio, ma solo per dimostrarne la non contestualità.

L contestuale \implies pumping lemma verificato

pumping lemma non verificato $\implies L$ non contestuale

Pumping lemma: utilizzo come gioco a due

Se Alice vince sempre questo gioco con Bob, allora L non è CF

1. Bob sceglie un intero $n > 0$
2. Alice sceglie una stringa $z \in L$ con $|z| > n$
3. Bob divide z in cinque parti $uvwxy$ con $|vwx| \leq n$ e $|vx| \geq 1$
4. Alice sceglie un intero $i \geq 0$
5. Alice mostra a Bob che $uv^iwx^iy \notin L$

Esempio

$L = \{a^k b^k c^k \mid k > 0\}$ non è CF

1. Bob sceglie un intero $n > 0$
2. Alice sceglie la stringa $a^n b^n c^n \in L$
3. Bob divide z in cinque parti $uvwxy$ con $|vwx| \leq n$ e $|vx| \geq 1$. vwx o è una sequenza di occorrenze dello stesso simbolo (ad esempio a^h , $h > 0$) o è composta di due sottosequenze di stessi simboli (ad esempio $a^r b^s$, $r, s > 0$). Quindi, almeno uno dei simboli a, b, c non compare in vwx e quindi né in v né in x
4. Alice sceglie $i = 2$
5. Alice mostra a Bob che $uv^2wx^2y \notin L$ in quanto almeno un simbolo ha aumentato il numero di occorrenze ed almeno un altro simbolo ha un numero di occorrenze invariato

Proprietà dei linguaggi context free

Chiusura dei linguaggi CF: intersezione

Il linguaggio $L = \{a^n b^n c^n \mid n \geq 1\}$ non è context free.

Del resto, $L_1 = \{a^n b^n c^m \mid n, m \geq 1\}$ e $L_2 = \{a^m b^n c^n \mid n, m \geq 1\}$ sono contestuali

Ma $L = L_1 \cap L_2$, da cui deriva che la classe dei linguaggi CF non è chiusa rispetto all'intersezione

Dati due linguaggi context free $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$, siano $\mathcal{G}_1 = \langle \Sigma_1, V_{N1}, P_1, S_1 \rangle$ e $\mathcal{G}_2 = \langle \Sigma_2, V_{N2}, P_2, S_2 \rangle$ due grammatiche di tipo 2 tali che $L_1 = L(\mathcal{G}_1)$ e $L_2 = L(\mathcal{G}_2)$.

Il linguaggio $L = L_1 \cup L_2$ potrà allora essere generato dalla grammatica di tipo 2 $\mathcal{G} = \langle \Sigma_1 \cup \Sigma_2, V_{N1} \cup V_{N2} \cup \{S\}, P, S \rangle$, dove $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}$.

Chiusura dei linguaggi CF: concatenazione

Dati due linguaggi context free $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$, siano $\mathcal{G}_1 = \langle \Sigma_1, V_{N1}, P_1, S_1 \rangle$ e $\mathcal{G}_2 = \langle \Sigma_2, V_{N2}, P_2, S_2 \rangle$ due grammatiche di tipo 2 tali che $L_1 = L(\mathcal{G}_1)$ e $L_2 = L(\mathcal{G}_2)$.

Mostriamo che il linguaggio $L = L_1 \circ L_2$ è generato dalla grammatica di tipo 2 definita come $\mathcal{G} = \langle \Sigma_1 \cup \Sigma_2, V_{N1} \cup V_{N2} \cup \{S\}, P, S \rangle$, dove $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}$.

Chiusura dei linguaggi CF: iterazione

Dato un linguaggio context free $L \subseteq \Sigma^*$, sia $\mathcal{G} = \langle \Sigma, V_N, P, S \rangle$ una grammatica di tipo 2 tale che $L = L(\mathcal{G})$.

Il linguaggio $L' = L^*$ è allora generato dalla grammatica di tipo 2 $\mathcal{G}' = \langle \Sigma, V_N \cup \{S'\}, P', S' \rangle$, dove $P' = P \cup \{S' \rightarrow SS' \mid \varepsilon\}$.

Chiusura dei linguaggi CF: complemento

La classe dei linguaggi CF non è chiusa rispetto al complemento.

Infatti, se così fosse, avremmo che dati due qualunque linguaggi CF L_1, L_2 , il linguaggio $L = \overline{\overline{L_1} \cup \overline{L_2}}$ sarebbe CF anch'esso. Ma $L = L_1 \cap L_2$ e quindi ne risulterebbe la chiusura rispetto all'intersezione, che non sussiste.

Decidibilità di predicati su linguaggi CF

Data una grammatica G di tipo 2 è decidibile stabilire se $L(G) = \emptyset$.

Assumiamo che $\mathcal{G} = \langle V_T, V_N, P, S \rangle$ sia in CNF.

Per il pumping lemma, se esiste una stringa $z = uvwxy \in L(\mathcal{G})$ con $|z| \geq 2^{|V_N|}$, allora esiste una stringa $z' = uwy \in L(\mathcal{G})$ con $|z'| < 2^{|V_N|}$. Quindi, se il linguaggio non è vuoto, esiste una stringa in esso di lunghezza minore di $2^{|V_N|}$.

Decidibilità di predicati su linguaggi CF

In una grammatica in CNF ogni applicazione di una produzione o incrementa di uno la lunghezza della forma di frase (se la produzione è del tipo $A \rightarrow BC$) o sostituisce un terminale a un non terminale (se è del tipo $A \rightarrow a$). Quindi, una stringa di lunghezza 2^k è generata da una derivazione di lunghezza $2^{k+1} - 1$

Per verificare se esiste una stringa di lunghezza minore di 2^{N+1} generabile, è sufficiente considerare tutte le derivazioni di lunghezza minore di $2^{N+1} - 1$ che sono, al più

$$\sum_{k=1}^{2^{N+1}-2} |P|^k = \frac{|P|^{2^{N+1}-1} - 1}{|P| - 1} = O(2^{2^{N+1}})$$

Un metodo più efficiente consiste nel portare la grammatica in forma ridotta, verificando se esistono simboli fecondi. Condizione necessaria e sufficiente affinché il linguaggio sia vuoto è che la grammatica non abbia simboli fecondi.

Decidibilità di predicati su linguaggi CF

Data una grammatica G di tipo 2 è decidibile stabilire se $L(G)$ è infinito.

Assumiamo che $\mathcal{G} = \langle V_T, V_N, P, S \rangle$ sia in CNF.

Per il pumping lemma, se esiste una stringa $z = uvwxy \in L(\mathcal{G})$ con $|z| \geq 2^{|V_N|}$, allora esistono infinite stringhe $z_i = uv^iwx^i y \in L(\mathcal{G})$, con $i \geq 0$, e almeno una di queste ha lunghezza $|z'| < 2^{|V_N|+1}$

Quindi, se il linguaggio è infinito, esiste una stringa in esso di lunghezza $z \in [2^{|V_N|}, 2^{|V_N|+1} - 1]$, che sarà derivata (in una grammatica in CNF) da una derivazione di lunghezza compresa tra $2^{|V_N|+1} - 1$ e $2^{|V_N|+2} - 3$

Decidibilità di predicati su linguaggi CF

È possibile allora considerare tutte le derivazioni di lunghezza compresa tra $2^{V_N+1} - 1$ e $2^{V_N+2} - 3$ che sono al più

$$\sum_{k=2^{V_N+1}-1}^{2^{V_N+2}-3} |P|^k = \frac{|P|^{2^{V_N+2}-2} - |P|^{2^{V_N+1}-1}}{|P| - 1} = O(2^{2^{V_N+2}})$$

e verificare se qualcuna di esse dà origine ad una stringa di terminali.

Metodo più efficiente: verificare la ciclicità del grafo orientato $G = (N, A)$ derivato dalla grammatica in CNF che genera L , ponendo $N = V_N$ e introducendo, per ogni produzione $B \rightarrow CD$, gli archi $\langle B, C \rangle$ e $\langle B, D \rangle$

Una grammatica G si dice **ambigua** se esiste una stringa x in $L(G)$ derivabile con due diversi alberi sintattici.

L'albero sintattico di una stringa corrisponde in qualche modo al significato della stringa stessa, quindi l'univocità di questo albero è importante per comprendere senza ambiguità tale significato

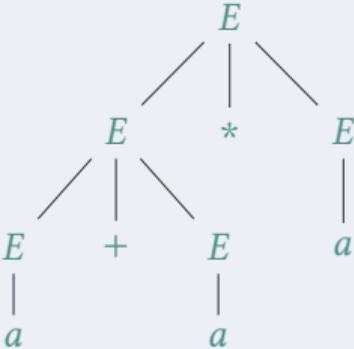
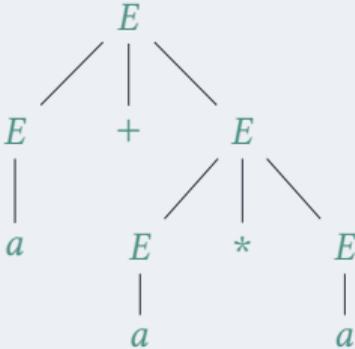
Si consideri la grammatica

$$E \longrightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid a.$$

Essa genera tutte le espressioni aritmetiche sulla variabile a , ma come si vede facilmente la stessa espressione può essere derivata con alberi di derivazione diversi.

Ambiguità

Ad esempio la stringa $a + a * a$ può venire derivata mediante due diversi alberi.



Si consideri la grammatica

$$E \longrightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid a.$$

Essa genera tutte le espressioni aritmetiche sulla variabile a , ma come si vede facilmente la stessa espressione può essere derivata con alberi di derivazione diversi.

Eliminazione dell'ambiguità:

- ⊙ Introduzione di parentesi
- ⊙ Precedenza tra operatori

Parentesi:

$$E \longrightarrow (E + E) \mid (E - E) \mid (E * E) \mid (E/E) \mid (E) \mid a.$$

I due diversi alberi di derivazione che davano origine alla stessa stringa, danno ora origine alle due stringhe

$$(a + (a * a))$$

$$((a + a) * a).$$

Precedenza tra operatori:

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow (E) \mid a$$

La grammatica rappresenta nella sua struttura le relazioni di precedenza definite tra gli operatori (nell'ordine non decrescente +, -, *, /) e in tal modo consente di utilizzare le parentesi soltanto quando strettamente necessario.

Riconoscimento: Data una grammatica \mathcal{G} non contestuale, \mathcal{G} è ambigua?

Il problema è **indecidibile** nel caso di CFG: non esiste quindi nessun algoritmo di decisione che, data una CFG, restituisca T se la grammatica è ambigua e F altrimenti.

Indecidibilità dimostrata mediante **riduzione** da un altro problema di decisione \mathcal{P} , che si sa essere indecidibile.

Schema generale di dimostrazione:

- ⊙ si vuole mostrare che il problema \mathcal{P}_1 è indecidibile
- ⊙ si individua un altro problema \mathcal{P}_0 che si sa essere indecidibile
- ⊙ si definisce un algoritmo \mathcal{A} che trasforma ogni istanza I_0 di \mathcal{P}_0 in una istanza $I_1 = \mathcal{A}(I_0)$ di \mathcal{P}_1
- ⊙ si mostra che l'istanza I_1 è positiva per \mathcal{P}_1 se e solo I_0 è positiva per \mathcal{P}_0
- ⊙ si conclude che \mathcal{P}_1 è indecidibile: se così non fosse avremmo un algoritmo che decide \mathcal{P}_0 , in quanto potremmo trasformare, per mezzo di \mathcal{A} , ogni sua istanza in una istanza corrispondente di \mathcal{P}_1 che potremmo, per ipotesi, risolvere

Nel nostro caso:

- ⊙ \mathcal{P}_1 è il problema di determinare, data una grammatica CF (istanza del problema), se essa è ambigua
- ⊙ \mathcal{P}_0 è **PCP** (Problema delle Corrispondenze di Post):
 - data una istanza del problema, composta da:
 - un alfabeto Σ
 - due sequenze di k parole $X = x_1, \dots, x_k$ e $Y = y_1, \dots, y_k$ costruite su Σ
 - ci si chiede se esiste una sequenza di $m \geq 1$ interi i_1, i_2, \dots, i_m in $[1, \dots, k]$ tale che risulti

$$x_{i_1} x_{i_2} \dots x_{i_m} = y_{i_1} y_{i_2} \dots y_{i_m}$$

Esempio di PCP

- ⊙ Consideriamo le due sequenze 1, 10111, 10 e 111, 10, 0 costruite sull'alfabeto $\{0, 1\}$
- ⊙ si può verificare che la sequenza di interi 2, 1, 1, 3 costituisce una soluzione alla istanza di PCP considerata.
- ⊙ infatti, si ottiene in un caso la sequenza $10111 \cdot 1 \cdot 1 \cdot 10 = 101111110$ e nell'altro la stessa sequenza $10 \cdot 111 \cdot 111 \cdot 0 = 101111110$

PCP è indecidibile (dimostrazione per riduzione dal **Problema della fermata**)

⊙ Sia $A = x_1, \dots, x_k$ e $B = y_1, \dots, y_k$ una istanza (generica) di PCP su un alfabeto Σ

⊙ Consideriamo

- l'alfabeto $\Sigma \cup \{a_1, a_2, \dots, a_k\}$, con $a_i \notin \Sigma, i = 1, \dots, k$

- il linguaggio $L' = L_A \cup L_B$ definito su Σ , in cui:

- $L_A = \{x_{i_1} x_{i_2} \dots x_{i_m} a_{i_m} a_{i_{m-1}} \dots a_{i_1} \mid m \geq 1\}$

- $L_B = \{y_{i_1} y_{i_2} \dots y_{i_m} a_{i_m} a_{i_{m-1}} \dots a_{i_1} \mid m \geq 1\}$.

- la relativa grammatica CF

$$\mathcal{G}' = \langle \{S, S_A, S_B\}, \Sigma \cup \{a_1, \dots, a_k\}, P, S \rangle,$$

con produzioni P , per $i = 1, \dots, k$:

$$S \longrightarrow S_A \mid S_B$$

$$S_A \longrightarrow x_1 S_A a_1 \mid \dots \mid x_k S_A a_k \mid x_1 a_1 \mid \dots \mid x_k a_k$$

$$S_B \longrightarrow y_1 S_B a_1 \mid \dots \mid y_k S_B a_k \mid y_1 a_1 \mid \dots \mid y_k a_k$$

Data l'istanza $([1, 10111, 10], [111, 10, 0])$, la corrispondente grammatica sarà data da:

$$S \rightarrow A \mid B$$

$$A \rightarrow 1Aa \mid 10111Ab \mid 10Ac \mid 1a \mid 10111b \mid 10c$$

$$B \rightarrow 111Ba \mid 10Bb \mid 0Bc \mid 111a \mid 10b \mid 0c$$

Equivalenza tra istanze

Se l'istanza (A, B) di PCP ha soluzione allora \mathcal{G}' è ambigua.

- ⊙ Sia i_1, \dots, i_m una soluzione di PCP, tale che quindi $x_{i_1} \cdots x_{i_m} a_{i_m} \cdots a_{i_1} = y_{i_1} \cdots y_{i_m} a_{i_m} \cdots a_{i_1} = \sigma$.
- ⊙ La stringa σ appartiene a L' e ammette due distinti alberi sintattici, corrispondi il primo alla derivazione

$$S \Longrightarrow S_A \Longrightarrow x_{i_1} S_A a_{i_1} \Longrightarrow x_{i_1} x_{i_2} S_A a_{i_2} a_{i_1} \xRightarrow{*} x_{i_1} \cdots x_{i_m} a_{i_m} \cdots a_{i_1},$$

e il secondo alla derivazione

$$S \Longrightarrow S_B \Longrightarrow y_{i_1} S_B a_{i_1} \xRightarrow{*} y_{i_1} \cdots y_{i_m} a_{i_m} \cdots a_{i_1} = x_{i_1} \cdots x_{i_m} a_{i_m} \cdots a_{i_1}.$$

- ⊙ \mathcal{G}' risulta dunque ambigua

Equivalenza tra istanze

Se \mathcal{G}' è ambigua allora l'istanza (A, B) di PCP ha soluzione.

- ⊙ Sia z una stringa di L' che ammette due distinti alberi sintattici
- ⊙ Per definizione di L' , deve essere $z = wa_{i_m} \cdots a_{i_1}$ per un qualche $m \geq 1$
- ⊙ Inoltre, per definizione di L' , z deve appartenere ad almeno uno tra L_A e L_B : assumiamo, senza perdere generalità, che $z \in L_A$
- ⊙ Allora, deve essere $w = x_{i_1} \cdots x_{i_m}$, e la produzione iniziale della derivazione deve essere $S \rightarrow S_A$
- ⊙ Ma per definizione di \mathcal{G}' , l'altro modo di derivare z non può che prevedere come prima produzione $S \rightarrow S_B$, per cui $w = y_{i_1} \cdots y_{i_m}$
- ⊙ Ne deriva che i_1, \dots, i_m è una soluzione dell'istanza (A, B) di PCP

Come osservato, la sequenza 2,1,1,3 costituisce una soluzione dell'istanza (positiva, quindi) di PCP.

Corrispondentemente, la stringa 101111110caab può essere ottenuta dalle due derivazioni:

$$S \rightarrow 10111Ab \rightarrow 101111Aab \rightarrow 1011111Aaab \rightarrow 101111110caab$$

$$S \rightarrow 10Bb \rightarrow 10111Babb \rightarrow 10111111Baab \rightarrow 101111110caab$$

- ⊙ La trasformazione definita deriva quindi da una istanza di PCP una grammatica CF che è ambigua se e solo se l'istanza ha soluzione
- ⊙ Se avessimo un algoritmo che determina se una grammatica CF è ambigua, allora potremmo determinare se una istanza di PCP ha soluzione
- ⊙ Ma un algoritmo che determina se una istanza di PCP ha soluzione non esiste
- ⊙ Quindi, non esiste un algoritmo che determina se una grammatica CF è ambigua

Esistenza di grammatica equivalente non ambigua: Un linguaggio di tipo 2 si dice **inerentemente ambiguo** se tutte le grammatiche che lo generano sono ambigue.

Anche il problema dell'inerente ambiguità di un linguaggio è indecidibile.