# Proprieta' dei linguaggi liberi da contesto

# Proprieta' di CFL

- **Semplificazione** di una CFG: se un linguaggio e' un CFL, ha una grammatica di una forma speciale.
- **Pumping Lemma per CFL**: simile ai linguaggi regolari.
- **Proprieta' di chiusura**: 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' indecidibile.

# Forma normale di Chomsky

Una CGF $G = (V, T, P, S)$ e' in *forma normale di Chomsky (CNF)* sse tutte le produzioni $P$ hanno una delle seguenti forme

- $A \rightarrow BC$ dove $A, B, C \in V$ sono variabili
- $A \rightarrow a$ dove $A \in V$ e' una variabile e $a \in T$ e' un terminale.

# Forma normale di Chomsky

Si puo' dimostrare che ogni linguaggio $L$ CFL tale che $L \neq \emptyset$ e $\epsilon \notin L$ ha una grammatica CNF. Per ottenere una CNF si parte da una grammatica $G = (V, T, P, S)$, che genera $L$ (e.g. $L(G) = L$), e si applicano una serie di trasformazioni.
In particolare,

1. Eliminare i **simboli inutili**, quelli che non appaiono in nessuna derivazione $S \stackrel{*}{\Rightarrow} w$, per simbolo iniziale $S$ e terminale $w$.
2. Eliminare le produzioni $\epsilon$, della forma $A \rightarrow \epsilon$.
3. Eliminare le **produzioni unitarie**, cioe' produzioni della forma $A \rightarrow B$, dove $A$ e $B$ sono variabili.
4. Eliminare le produzioni con piu' di due nonterminali sulla destra.

# Simboli inutili

Data una grammatica $G = (V, T, P, S)$ diciamo che un simbolo $X$ e' **utile** sse esiste una derivazione

$$S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w$$

dove $w \in T^*$ e' una stringa di terminali.
**Nota**: un simbolo $X$ (terminale o variabile) inutile, puo' essere eliminato dalla grammatica, senza modificare il linguaggio che viene generato.
Per eliminarli bisogna notare che: un simbolo $X$ e' utile sse e ' **generatore** e **raggiungibile**:

1. $X$ e' raggiungibile sse $S \stackrel{*}{\Rightarrow} \alpha X \beta$
2. $X$ e' generatore sse $X \stackrel{*}{\Rightarrow} w$ per una stringa di terminali $w$.

# Eliminazione dei simboli inutili

Da una grammatica $G = (V, T, P, S)$

1. eliminiamo tutti i simboli non generatori e le produzioni in cui compiaono
2. eliminiamo i simboli non raggiungibili e le produzioni in cui compiaono.

La grammatica $G'$ ottenuta non contiene simboli inutili e $L(G') = L(G)$.

# Esempio

Data la grammatica $G = (\{S, A, B\}, \{a, b\}, P, S)$ e $P$

$$S \to AB|a \qquad A \to b$$

1. tutti i simboli (terminali e non terminali) tranne $B$ sono generatori. Otteniamo $G' = (\{S, A\}, \{a, b\}, P', S)$ e $P'$

$$S \to a \qquad A \to b$$

2. in $G'$ $A$ e' non raggiungibile. Otteniamo $G'' = (\{S\}, \{a, b\}, P'', S)$ e $P''$

$$S \to a$$

# Algoritmo Induttivo

Data una grammatica $G = (V, T, P, S)$ si possono definire algoritmi induttivi per calcolare i simboli generatori e quelli raggiungibili.

1. **generatori**:
   - **caso base** : $T$ sono generatori
   - **caso induttivo**: per ogni $A \in V$ $A$ e' generatore se esiste una regola $A \rightarrow \alpha \in P$ tale che ogni simbolo che occorre in $\alpha$ e' generatore
2. **raggiungibili**:
   - **caso base** : $S$ e' raggiungibile
   - **caso induttivo**: se $A$ e' raggiungibile, per ogni regola $A \rightarrow \alpha \in P$ ogni simbolo che occorre in $\alpha$ e' raggiungibile

# Eliminazione delle $\epsilon$-produzioni

Data una grammatica $G = (V, T, P, S)$ tale che $\epsilon \notin L(G)$ vogliamo ottenere una grammatica $G' = (V', T', P', S')$ che

1. non contiene $\epsilon$-produzioni, $A \rightarrow \epsilon \notin P'$
2. e' equivalente a $G$, $L(G') = L(G)$.

La strategia di basa sull'eliminazione delle variabili **annullabili**. Una variabile $A \in V$ e' annullabile se $A \stackrel{*}{\Rightarrow} \epsilon$ (la variabile $A$ genera la stringa vuota)

# Algoritmo per i simboli annullabili

Data una grammatica $G = (V, T, P, S)$ il seguente algoritmo induttivo permette di determinare tutte e sole le variabili annullabili

- **caso base** : $A \in V$ e' annullabile se $A \rightarrow \in P$
- **caso induttivo**: se $B \rightarrow C_1 \dots C_n \in P$ e, per ogni $i \in \{1, \dots, n\}$, $C_i$ e' annullabile , allora $B$ e' annullabile

# Algoritmo per eliminare le $\epsilon$-produzioni

Sia $G = (V, T, P, S)$ tale che $\epsilon \notin L(G)$.

- calcoliamo tutte le variabili annullabili
- definiamo la grammatica $G' = (V, T, P', S)$ dove $P'$ e' ottenuto da $P$ nel seguente modo. Per ogni $A \to C_1 \dots C_k \in P$ con $k \geq 1$, supponiamo che $m$ dei $k$ simboli siano annullabili.
  Allora $P'$ conterra' $2^k$ diverse versioni della regola, corrispondenti a tutte le combinazioni ottenute da $C_1 \dots C_k$, considerando per ogni $C_i$ annullabile il caso che ci sia o non ci sia.

**Nota** $P'$ e' ottenuto da $P$ eliminando le produzioni del tipo $A \to \epsilon$ , e propagando l'effetto dell'eliminazione della produzione in ogni regola che contiene $A$ nel corpo. Quindi $L(G) = L(G')$.

# Esempio

Consideriamo la grammatica $G = (\{S, A, B\}, \{a, b\}, P, S)$ dove $P$ e'

$$S \rightarrow AB$$
$$A \rightarrow aAA|\epsilon$$
$$B \rightarrow bBB|\epsilon$$

Notiamo che tutte e tre le variabili sono annullabili. Quindi otteniamo le produzioni $P'$

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

**Nota**: per il simbolo iniziale le combinazioni sono solo tre, la quarta $A$ e $B$ assenti darebbe $\epsilon$.

# Eliminazione delle Produzioni Unitarie

Una produzione *unitaria* e' una produzione della forma $A \longrightarrow B$, dove sia $A$ che $B$ sono variabili.
La grammatica delle espressioni

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

Ha varie produzioni unitarie $F \longrightarrow I$, $T \longrightarrow F$, $E \longrightarrow T$.

# Algoritmo per l'eliminazione delle Produzioni Unitarie

1. Una produzione unitaria $A \longrightarrow B$ puo' essere espansa con $A \longrightarrow \alpha$, per ogni regola $B \longrightarrow \alpha \in P$.
2. **Problema**: cicli di produzioni unitarie $A \longrightarrow B$, $B \longrightarrow C$ e $C \longrightarrow A$.
3. Semplice algoritmo induttivo permette di derivare tutte le *coppie unitarie* $(A, B)$ tali che $A \stackrel{*}{\Rightarrow} B$ solo mediante produzioni unitarie
4. avendo calcolato le coppie possiamo poi espandere le regole come detto sopra eliminando le produzioni unitarie

Questa modifica delle regole non modifica il linguaggio generato.

# Esempio

La grammatica delle espressioni modificata, espandendo le produzioni unitarie

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

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

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

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

# Forma Normale di Chomsky

1. con le traformazioni viste fino ad adesso sappiamo trovare una grammatica equivalente a quella di partenza, che non ha simboli inutili, non ha $\epsilon$-produzioni, non ha produzioni unitarie
2. per ottenere CNF dalla forma (1) bisogna avere un metodo per trasformare le produzioni
   1. i corpi delle produzioni di lunghezza 1 vanno gia' bene, possiamo avere solo un terminale (le produzioni unitarie sono state eliminate)
   2. i corpi delle produzioni di lunghezza 2 devono contenere solo variabili
   3. i corpi di lunghezza maggiore di due vanno scomposti utilizzando una cascata di produzioni, ciascuna con un corpo fatto di due variabili

# Trattamento dei terminali

Consideriamo tutti i terminale $a \in T$ che appare nel corpo di una produzione di lunghezza due (e.g. $A \rightarrow aX$ o $A \rightarrow Xa$ simmetricamente) o piu'. Per ognuno introduciamo una nuova variabile $A_a$ e modifichiamo le produzioni come segue

1. aggiungiamo la produzione $A_a \rightarrow a$
2. per ogni regola $A \rightarrow \alpha$ dove $a$ compare in $\alpha$ rimpiazziamo $a$ con $A_a$.

**Nota** Applicando la tecnica per tutti i terminali coinvolti otterremo solo variabili nei corpi delle produzioni di lunghezza due

# Modifica dei corpi delle produzioni di lunghezza maggiore di due

Sia $A \to B_1 \dots B_k$ una produzione con $k > 2$. Introduciamo $k-2$ nuove variabili $C_1, \dots, C_{k-2}$.
Rimpiazziamo la produzione con

$$
\begin{aligned}
&A \to B_1 C_1 \\
&C_1 \to B_2 C_2 \\
&\dots \\
&\dots \\
&C_{k-2} \to B_{k-1} B_k
\end{aligned}
$$

**Nota** otteniamo produzioni in cui i corpi di lunghezza maggiore di due sono stati eliminati, e quelli di lunghezza due contengono esattamente due variabili

# Esempio

Consideriamo la grammatica $G = (\{S, A, B\}, \{a, b\}, P, S)$ dove $P$ e'

$$S \rightarrow bA|aB$$
$$A \rightarrow bAA|aS|a$$
$$B \rightarrow aBB|bS|b$$

**Nota**: non ci sono $\epsilon$-transizioni, non ci sono produzioni unitarie, e neanche simboli inutili

# Prima Trasformazione

Introducendo non terminali $C_a$ e $C_b$ nuovi per i terminali $a$ e $b$ otteniamo

$$
\begin{aligned}
S &\rightarrow C_bA|C_aB \\
A &\rightarrow C_bAA|C_aS|a \\
B &\rightarrow C_aBB|C_bS|b \\
C_a &\rightarrow a \\
C_b &\rightarrow b
\end{aligned}
$$

**Nota**: $C_a$ e $C_b$ rimpiazzano $a$ e $b$ in tutti corpi delle produzioni di lunghezza maggiore di due.

# Seconda Trasformazione

Modificando le produzioni $A \longrightarrow C_bAA$ e $B \longrightarrow C_aBB$ otteniamo

$$
\begin{aligned}
S &\longrightarrow C_bA|C_aB \\
A &\longrightarrow C_bD_1|C_aS|a \\
B &\longrightarrow C_aD_2|C_bS|b \\
C_a &\longrightarrow a \\
C_b &\longrightarrow b \\
D_1 &\longrightarrow AA \\
D_2 &\longrightarrow BB
\end{aligned}
$$

**Nota**: Abbiamo ottenuto una grammatica equivalente CNF senza simboli inutili

# Pumping lemma per CFL

- **Informalmente**:
  In ogni stringa sufficientemente lunga di un CFL si possono trovare due sottostringhe vicine che e' 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 $| z | \geq n$, possiamo scrivere $z = uvwxy$ con le seguenti condizioni:
  1. $| vwx | \leq n$
  2. $vx \neq \epsilon$
  3. per ogni $i \geq 0$, $uv^i wx^i y \in L$.

# 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 $\mid vwx \mid \leq n$ e $v$ e $x$ non entrambi vuote, $vwx$ non puo' contenere sia 0 che 2 perche' 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 piu' 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 $L$.

# Proprieta' di chiusura dei CFL

- alcune delle proprieta' di chiusura dei linguaggi regolari valgono anche per i CFL.
- i CFL sono chiusi rispetto all'unione, alla concatenazione, alla chiusura di Kleene, rispetto alla sostituzione ed all'omomorfismo inverso

# Sostituzioni

Siano $\Sigma, \Delta$ degli alfabeti. Una *sostituzione* e' una funzione

$$f : \Sigma \rightarrow 2^{\Delta^*}$$

La sostituzione associa, ad ogni simbolo di $\Sigma$, un insieme di stringhe di $\Delta$, ovvero un linguaggio su $\Delta$.
Data una sostituzione $f : \Sigma \rightarrow 2^{\Delta^*}$ definiamo

- l'applicazione di $f$ ad una stringa $w \in \Sigma^*$ in modo induttivo

$$f(\epsilon) = \epsilon$$

$$f(xa) = f(x).f(a)$$

- l'applicazione di $f$ ad un linguaggio $L \subseteq \Sigma^*$,

$$f(L) = \{f(w) \mid w \in L\}$$

# Chiusura rispetto alla sostituzione

**Teorema**: Sia $L$ un CFL su $\Sigma$ e $f : \Sigma \rightarrow 2^{\Delta^*}$ una sostituzione tale che, per ogni $a \in \Sigma$, $f(a) = L_a$ e' un CFL. Allora $f(L)$ e' un CFL.

**Prova** L'idea e' di estendere la tecnica di prova presentata per i linguaggi regolari, adattandola alle grammatiche.
Consideriamo le grammatiche corrispondenti a $L$ e $L_a$ per ogni $a \in \Sigma$. Avremo

$$G = (V, \Sigma, P, S)$$

dove $L(G) = L$.
Inoltre, per ogni $a \in \Sigma$ avremo

$$G_a = (V_a, T_a, P_a, S_a)$$

dove $L(G_a) = L_a$.

# Chiusura rispetto alla sostituzione

**Nota**: Dato che la scelta dei nomi delle variabili e' libera, prendiamo grammatiche che hanno variabili tutte disgiunte

$$V \cap V_a = \emptyset \quad V_a \cap V_b = \emptyset$$

In questo modo possiamo costruire la seguente grammatica per l'immagine di $L$, $f(L)$.

$$G' = (V', T', P', S)$$

- $V' = V \cup \bigcup_{a \in \Sigma} V_a$ e' l'unione di tutte le variabili
- il simbolo iniziale e' $S$
- $T' = \bigcup_{a \in \Sigma} T_a$ i terminali sono l'unione dei terminali per ogni simbolo $a \in Sigma$ ($\subseteq \Delta$).
- $P' = \bigcup_{a \in \Sigma} P_a \cup P''$ dove

$$P'' = \{A \to \alpha[S_a/a] \mid A \to \alpha \in P\}$$

# Correttezza della Costruzione

Dobbiamo fare vedere che $f(L) = L(G')$, ovvero che per ogni $w \in \Delta^*$

$$w \in f(L) \text{ sse } S \stackrel{*}{\Rightarrow}_{G'} w$$

**Se:** Se $w \in f(L)$ significa che esiste $x = a_1, \ldots, a_n \in L$ tale che $f(L) = w$. Inoltre, deve essere

$$w = w_1, \ldots, w_n$$

dove $w_i = f(a_i)$ per ogni $i \in \{1, \ldots, n\}$.
Dato che $x \in L$ esiste una derivazione di $x$ da $S$ nella grammatica $G$,

$$S \stackrel{*}{\Rightarrow}_G x = a_1, \ldots, a_n$$

Allora per definizione delle produzioni di $G'$ avremo una derivazione analoga in $G'$ del tipo

$$S \stackrel{*}{\Rightarrow}_{G'} S_{a_1}, \ldots, S_{a_n}$$

dove $S_{a_i}$ e' il simbolo iniziale della grammatica $G_{a_i}$.

# Correttezza della Costruzione

Notiamo che la stringa $w_i = f(a_i)$ appartiene al linguaggio $L_{a_i}$ ed e' quindi derivabile dal simbolo iniziale $S_{a_i}$ nella grammatica $G_{a_i}$

$$S_{a_i} \stackrel{*}{\Rightarrow}_{G_a} w_i$$

$$S_{a_i} \stackrel{*}{\Rightarrow}_{G'} w_i$$

Dato che le produzioni di $G'$ includono quelle di $G_{a_i}$.
Combinando queste derivazioni in $G'$ avremo

$$S \stackrel{*}{\Rightarrow}_{G'} S_{a_1}, \ldots, S_{a_n} \stackrel{*}{\Rightarrow}_{G'} w_1 S_{a_2}, \ldots, S_{a_n} \ldots \stackrel{*}{\Rightarrow}_{G'} w_1 \ldots w_n = w$$

# Correttezza della Costruzione

Dobbiamo fare vedere che $f(L) = L(G')$, ovvero che per ogni $w \in \Delta^*$

$$w \in f(L) \text{ sse } S \stackrel{*}{\Rightarrow}_{G'} w$$

**Solo Se:** Abbiamo $S \stackrel{*}{\Rightarrow}_{G'} w$.
Osserviamo la forma delle produzioni di $G'$ e sfruttiamo il fatto che le variabili sono distinte e che i terminali $T'$ della stringa $w$ sono generati solo ed esclusivamente dalle produzioni delle grammatiche $G_a$ per $a \in \Sigma$.
Allora la derivazione deve essere scomponibile nel seguente modo

$$S \stackrel{*}{\Rightarrow}_{G'} S_{a_1} \dots S_{a_n}$$
$$S_{a_i} \stackrel{*}{\Rightarrow}_{G'} w_i$$

dove $w = w_1 \dots w_n$ per ogni $i \in \{1, \dots, n\}$.
Inoltre ogni $w_i \in L(G_{a_i})$, quindi $w_i \in f(a_i)$, per ogni $i \in \{1, \dots, n\}$.

# Correttezza della Costruzione

Inoltre, per definizione di $G'$ la parte della derivazione $S \stackrel{*}{\Rightarrow}_{G'} S_{a_1}, \ldots, S_{a_n}$ coinvolge solo produzioni della grammatica $G$.
Quindi avremo una derivazione corrispondente in G

$$S \stackrel{*}{\Rightarrow}_{G} a_1, \ldots, a_n$$

Questo dimostra che

$$a_1 \ldots a_n \in L(G)$$

Inoltre

$$f(a_1 \ldots a_n) = f(a_1) \ldots f(a_n) = w_1 \ldots w_n = w \in f(L).$$

# Esempio

Consideriamo i seguenti linguaggi

$L_1 = \{0^n 1^n \mid n \geq 1\}$
$L_2 = \{ww^r \mid w \in \{0,2\}^*\}$
$L_3 = \{w \mid w \in \{a,b\}^*$ w contiene lo stesso numero di $a$ e di $b\}$

Grammatiche per $L_1$, $L_2$ e $L_3$

$$S_1 \rightarrow 01|0S_1 1$$
$$S_2 \rightarrow \epsilon|0S_2 0|2S_2 2$$
$$S_3 \rightarrow \epsilon|aS_3 bS_3|bS_3 aS_3$$

# Sostituzione

Definiamo una sostituzione sull'alfabeto di $L_3$, $\{a, b\}$,

$$f(a) = L_1 \quad f(b) = L_2$$

La grammatica per $f(L_3)$ ottenuta ha simbolo iniziale $S_3$ e produzioni

$$\begin{aligned} S_1 &\to 01|0S_11 \\ S_2 &\to \epsilon|0S_20|2S_22 \\ S_3 &\to \epsilon|S_1S_3S_2S_3|S_2S_3S_1S_3 \end{aligned}$$

Nelle produzioni di $L_3$ (di $S_3$) al posto di $a$ viene sostituito $S_1$ e al posto di $b$ viene sostituto $S - 2$.

# Proprieta' di chiusura dei CFL

**Teorema:** I CFL sono chiusi sotto unione, concatenazione e chiusura di Kleene.

**Prova** Usiamo delle opportune sostituzioni, sfruttando la proprieta' dimostrate in precedenza. Siano $L_1$ ed $L_2$ due CFL.

- **Unione** $L_1 \cup L_2$ e' il linguaggio $s(L)$ per $L = \{1, 2\}$ e $s(1) = L_1$ e $s(2) = L_2$.
- **Concatenazione** $L_1 L_2$ e' il linguaggio $s(L)$ per $L = \{12\}$ e $s(1) = L_1$ e $s(2) = L_2$.
- **Chiusura di Kleene** $L_1^*$ e' il linguaggio $s(L)$ per $L = \{1\}^*$ e $s(1) = L_1$.

# Chiusura rispetto all'inversione

**Teorema:** Se $L$ e CF, allora lo e' 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 $L_1$ e' libero da contesto, con grammatica

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

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

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

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

# Intersezione tra CFL e linguaggi regolari

**Teorema 7.27:** Se $L$ e' CFL, e $R$ e' regolare, allora $L \cap R$ e' CF.
**Prova:** Sia $L$ accettato dal PDA

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

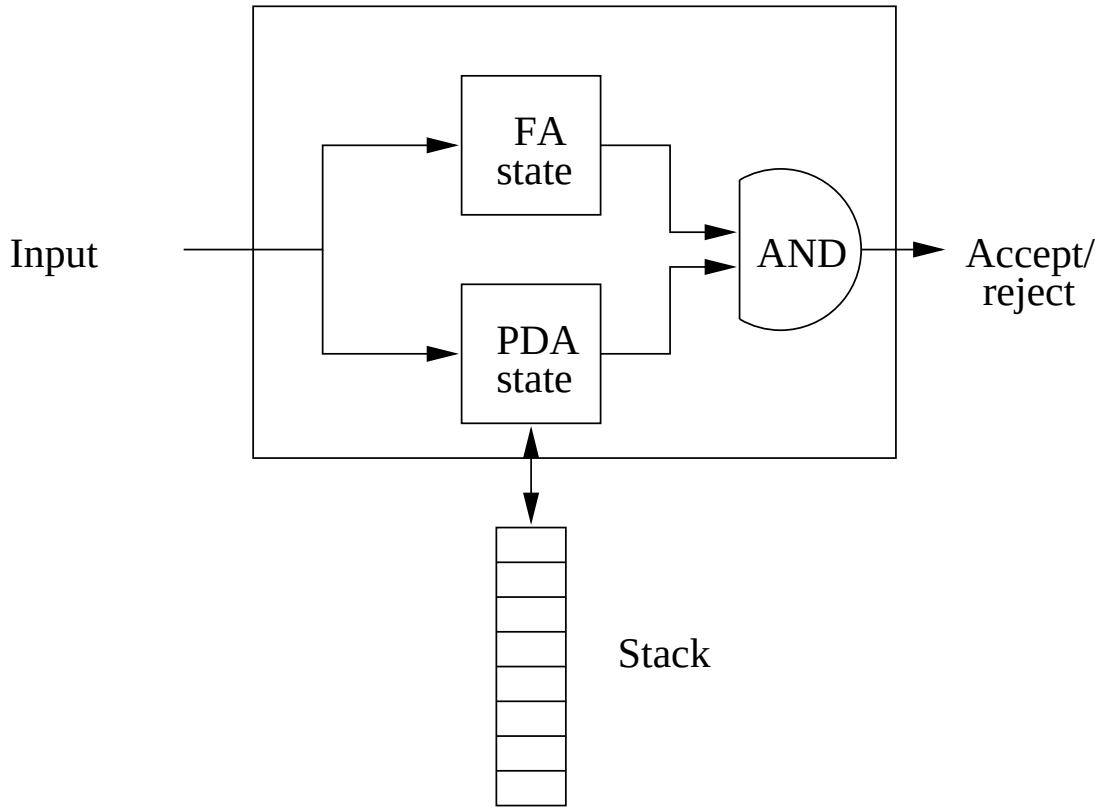
per stato finale, e sia R accettato dall'automa a stati finiti

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

Generalizziamo la costruzione vista per i linguaggi regolari, costruendo un PDA che esegue in parallelo $P$ ed $A$.

# Intuitivamente

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




Gli stati del PDA sono coppie di stati dei due automi, i due automi si muovono in parallelo e accettano sse entrambi arrivano in uno stato finale.

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

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

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

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

**Prova:**

1. $\bar{R}$ e' regolare, $L \cap \bar{R}$ e' 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 $\Sigma^*$ e' CF, quindi se $L_1 \setminus L_2$ fosse sempre CF, allora lo sarebbe sempre anche $\Sigma^* \setminus L = \bar{L}$.

# Omomorfismo

Siano $\Sigma, \Delta$ degli alfabeti. Un *omomorfismo* e' una funzione

$$h : \Sigma \rightarrow \Delta^*$$

- Un omomorfismo e' una sostituzione che associa, ad ogni simbolo di $\Sigma$, esattamente una stringa $\Delta^*$ (invece che un linguaggio)
- Dato che una omomorfismo e' un tipo particolare di sostituzione, i CFL sono *chiusi per omomorfismo*

# Omomorfismo Inverso

Siano $\Sigma, \Delta$ alfabeti e $h : \Sigma \rightarrow \Delta^*$ un omomorfismo. Dato un linguaggio $L \subseteq \Delta^*$, definamo la **controimmagine** di $L$ rispetto ad $h$ come

$$h^{-1}(L) = \{w \mid h(w) \in L\}$$

Ovviamente, $h^{-1}(L) \subseteq \Sigma^*$ e' insieme delle stringhe che sono immagine di qualche stringa di $L$ tramite $h$.
**Teorema**: Sia $L$ un CFL su $\Delta$ e $h : \Sigma \rightarrow \Delta^*$ un omomorfismo. Allora $h^{-1}(L)$ e' un CFL.

Ricordiamo che nel caso dei linguaggi regolari la simulazione costruiva un DFA $B$ per $h^{-1}(L)$ a partire dal DFA $A$ che riconosce $L$.

- $A$ e $B$ hanno gli stessi stati, ma alfabeti differenti $A$ legge simboli di $\Delta$ e $B$ legge simboli di $\Sigma$
- La funzione di transizione di $B$ simula sequenze di mosse di $A$
- In particolare, l' automa $B$ nello stato $q$, leggendo il simbolo $a \in \Sigma$, si muove nello stato in cui si muoverebbe $A$, leggendo la stringa $h(a)$.

La simulazione e' un po' piu' complicata per i CFL dato che $A$ e' un PDA, bisogna tenere conto della pila.
Bisogna che l'automa $B$ memorizzi $h(a)$ in un *buffer* dopo avere letto $a$, i simboli nel buffer verranno passati all'automa $A$ uno alla volta. Quando il buffer e' vuoto allora $B$ puo' leggere il prossimo input.

# Prova

Consideriamo il PDA $A$ che riconosce $L$ per stato finale,

$$A = (Q, \Delta, \Gamma, \delta, q_0, Z_0, F)$$

Costruiamo un automa $B$ sull'alfabeto $\Sigma$ che riconosce $h^{-1}(L)$,

$$B = (Q', \Sigma, \Gamma, \gamma, (q_0, \epsilon), Z_0, F \times \{\epsilon\})$$

- $Q'$ e' l'insieme di coppie $(q, x)$ tali che $q \in Q$ e' uno stato di $A$ ed $x \in \Gamma^*$ e' un suffisso di una stringa in $h(a)$, per qualche $a \in \Sigma$ (il buffer)
- lo stato iniziale e' $(q_0, \epsilon)$ il buffer e' vuoto
- gli stati finali sono coppie e' $(q, \epsilon)$ dove $q \in F$ e il buffer e' vuoto
- la funzione di transizione $\gamma$ e' definita in base a $\delta$ come segue

# Le mosse dell'automa $B$

Definiamo $\gamma : Q' \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q' \times \Gamma^*}$ dove

- $\gamma((q, \epsilon), a, X) = \{((q, h(a)), X)\}$ se il buffer e' vuoto allora l'automa legge $a$ e memorizza la sua immagine $h(a)$ nel buffer
- per ogni $b \in \Delta$ e $X \in \Gamma$ se $(p, \alpha) \in \delta(q, b, X)$ allora

$$((p, x), \alpha)) \in \gamma((q, bx), \epsilon, X)$$

L'automa $A$ da $q$, leggendo $a$, e con pila $X$ si muove in $p$ e modifica la pila con $\alpha$.
In tal caso l'automa $B$ consuma $b$ nel buffer e si muove nello stato $(p, x)$. La mossa e' una $\epsilon$-transizione, la pila di $B$ viene modificata in modo analogo ad $A$.

# Proprieta' della Costruzione

Per induzione sulla lunghezza di $w \in \Sigma^*$, si puo' provare che

$$(q_0, h(w), Z_0) \vdash_A^* (p, \epsilon, \gamma) \text{ sse } ((q_0, \epsilon), w, Z_0) \vdash_B^* ((p, \epsilon), \epsilon, \gamma)$$

Basta fare vedere che una sequenza di mosse di $B$ ha per definizione la seguente forma: (a) lettura di un simbolo $a$ e memorizzazione di $h(a)$ nel buffer; (b) sequenza di mosse $\epsilon$ che simulano $A$ su $h(a)$ fino ad arrivare al buffer vuoto.
Di consequenza, $h(w) \in L(A)$ sse $p \in F$ sse $(p, \epsilon)$ accetta sse $w \in L(B)$.

# Problemi indecidibili per linguaggi liberi da contesto

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

1. Data $G$, e' ambigua?
2. E' un dato CFL inerentemente ambiguo?
3. E' l'intersezione di due CFL vuota?
4. Dati due CFL, sono equivalenti?
5. Dato un CFL, e' uguale a $\Sigma^*$?