Linguaggi Liberi da Contesto

Nicola Fanizzi
Corso di Linguaggi di Programmazione

Dipartimento di Informatica
Università degli Studi di Bari

1 maggio 2014

Sommario

Sommario

Grammatiche e Linguaggi Liberi da Contesto

Una grammatica $G = (\Sigma, V, S, P)$ è una **grammatica libera da contesto** sse le sue produzioni sono tutte del tipo:

$$A \longrightarrow \omega$$

con $A \in V$ e $\omega \in (V \cup \Sigma)^*$
Il linguaggio $\mathbf{L}(G)$ si dice **linguaggio libero da contesto**

- nome: in una derivazione, ogni non terminale può essere sostituito con una parte destra di una sua produzione, *indipendentemente* dal contesto della forma di frase in cui si trova
- sostituzione sempre valida
- appartiene a questa categoria la maggior parte dei *linguaggi di programmazione*

Osservazioni

- La classe dei linguaggi liberi è *inclusa* in quella dei linguaggi contestuali

- Una grammatica libera considera produzioni dove il contesto destro e quello sinistro siano *vuoti* ($= \epsilon$)
- *Eccezione* una grammatica libera ammette ogni tipo di ϵ*-regola* ossia ogni produzione del tipo

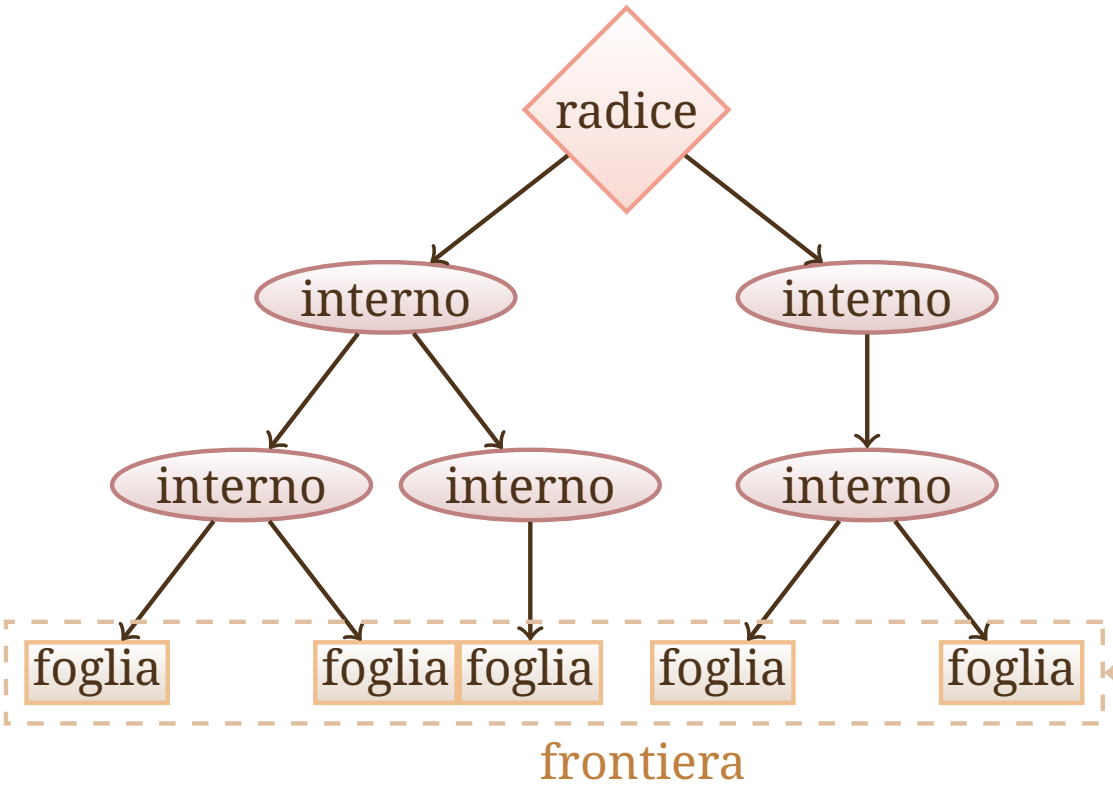
$$A \longrightarrow \epsilon \qquad \text{con } A \neq S$$

vietate dalle grammatiche contestuali

Alberi

Le derivazioni di una grammatica libera possono essere rappresentate graficamente attraverso *alberi*

Albero: grafo orientato, aciclico, connesso tale che ogni nodo ha al più un arco entrante

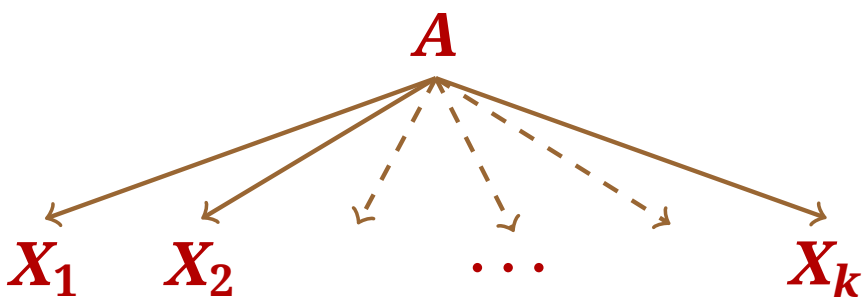


- la **lunghezza** di un cammino dalla radice ad una foglia è data dal numero di nodi incontrati
- l'**altezza** dell'albero è data dalla lunghezza del cammino più lungo
- la sequenza delle foglie (lette da sinistra verso destra) si definisce **frontiera** dell'albero

Alberi di Derivazione I

Data $G = (\Sigma, V, S, P)$ libera e $w \in \Sigma^*$ tale che $S \stackrel{*}{\Longrightarrow} w$, un albero di derivazione di w, denotato, T_w è così formato:

1. radice $\leftarrow S$
2. nodi interni $\leftarrow$ simboli di V
3. foglie $\leftarrow$ simboli di $\Sigma \cup V \cup \{\epsilon\}$ (o solo Σ se totalmente espanso)
4. se $\exists A \longrightarrow X_1 X_2 \cdots X_k \in P$ allora un nodo interno A può essere padre dei nodi figli $X_1, X_2, \ldots, X_k$



5. w si ottiene da T_w leggendo le foglie da sinistra a destra

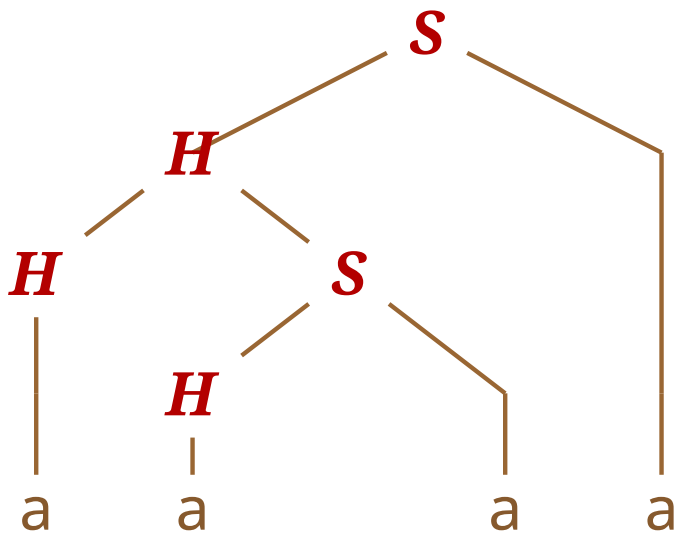
Alberi di Derivazione II

Osservazioni

Un albero di derivazione non impone *alcun criterio di scelta* del non terminale da espandere tramite una regola di produzione per ottenere la sequenza corrispondente ad una derivazione

- data una derivazione
  esiste *uno ed un solo* albero che la rappresenta
- dato un albero di derivazione
  esistono *più* derivazioni possibili
  a seconda dell'ordine scelto per l'applicazione delle produzioni

Esempio Data $G = (\{a\}, \{S, H\}, S, P)$ libera, con
$P = \{S \xrightarrow{1} Ha, \quad H \xrightarrow{2} HS, \quad H \xrightarrow{3} a\}$



La stringa $aaaa \in \mathbf{L}(G)$ si deriva sia tramite:

$$S \underset{1}{\Longrightarrow} Ha \underset{2}{\Longrightarrow} HSa \underset{3}{\Longrightarrow} aSa \underset{1}{\Longrightarrow} aHaa \underset{3}{\Longrightarrow} aaaa$$

sia con:

$$S \underset{1}{\Longrightarrow} Ha \underset{2}{\Longrightarrow} HSa \underset{1}{\Longrightarrow} HHaa \underset{3}{\Longrightarrow} Haaa \underset{3}{\Longrightarrow} aaaa$$

Derivazioni Canoniche I

Data una grammatica $G = (\Sigma, V, S, P)$ si dirà che la derivazione

$$S \Longrightarrow \omega_1 \Longrightarrow \omega_2 \Longrightarrow \cdots \Longrightarrow \omega_n = w \in \Sigma^*$$

dove $\omega_i = \alpha_i A \beta_i$ e $\omega_{i+1} = \alpha_i \phi_i \beta_i$, con $i = 1, \ldots, n-1$

è una **derivazione canonica sinistra** (risp. **canonica destra**) sse per ogni $i = 1, \ldots, n-1$ risulta:

$$\alpha_i \in \Sigma^* \qquad (\text{risp. } \beta_i \in \Sigma^*)$$

Talvolta queste derivazioni vengono denotate con $\underset{lm}{\Longrightarrow}$ e $\underset{rm}{\Longrightarrow}$, rispettivamente

Derivazioni Canoniche II

Esempio Si consideri la grammatica con produzioni:

$$P = \left\{ \begin{array}{l} S \longrightarrow 0B \mid 1A \\ A \longrightarrow 0 \mid 0S \mid 1AA \\ B \longrightarrow 1 \mid 1S \mid 0BB \end{array} \right\}$$

Derivazione sinistra di 001101:

$\underline{S} \Longrightarrow 0\underline{B} \Longrightarrow 00\underline{BB} \Longrightarrow 001\underline{S}B \Longrightarrow 0011\underline{A}B \Longrightarrow 00110\underline{B} \Longrightarrow 001101$

Derivazione destra di 001101:

$\underline{S} \Longrightarrow 0\underline{B} \Longrightarrow 00\underline{BB} \Longrightarrow 00\underline{B}1 \Longrightarrow 001\underline{S}1 \Longrightarrow 0011\underline{A}1 \Longrightarrow 001101$

ma anche:

$\underline{S} \Longrightarrow 0\underline{B} \Longrightarrow 00\underline{BB} \Longrightarrow 00B1\underline{S} \Longrightarrow 00B10\underline{B} \Longrightarrow 00\underline{B}101 \Longrightarrow 001101$

Derivazioni Canoniche III

$G = (\{a, +, *\}, \{E, I\}, E, P)$ con
$P = \{E \longrightarrow E{+}E | E{*}E | I,\ I \longrightarrow a\}$
Derivazione sinistra di a+a:

$$E \Longrightarrow E{+}E \Longrightarrow I{+}E \Longrightarrow a{+}E \Longrightarrow a{+}I \Longrightarrow a{+}a$$

Derivazione destra di a+a:

$$E \Longrightarrow E{+}E \Longrightarrow E{+}I \Longrightarrow E{+}a \Longrightarrow I{+}a \Longrightarrow a{+}a$$

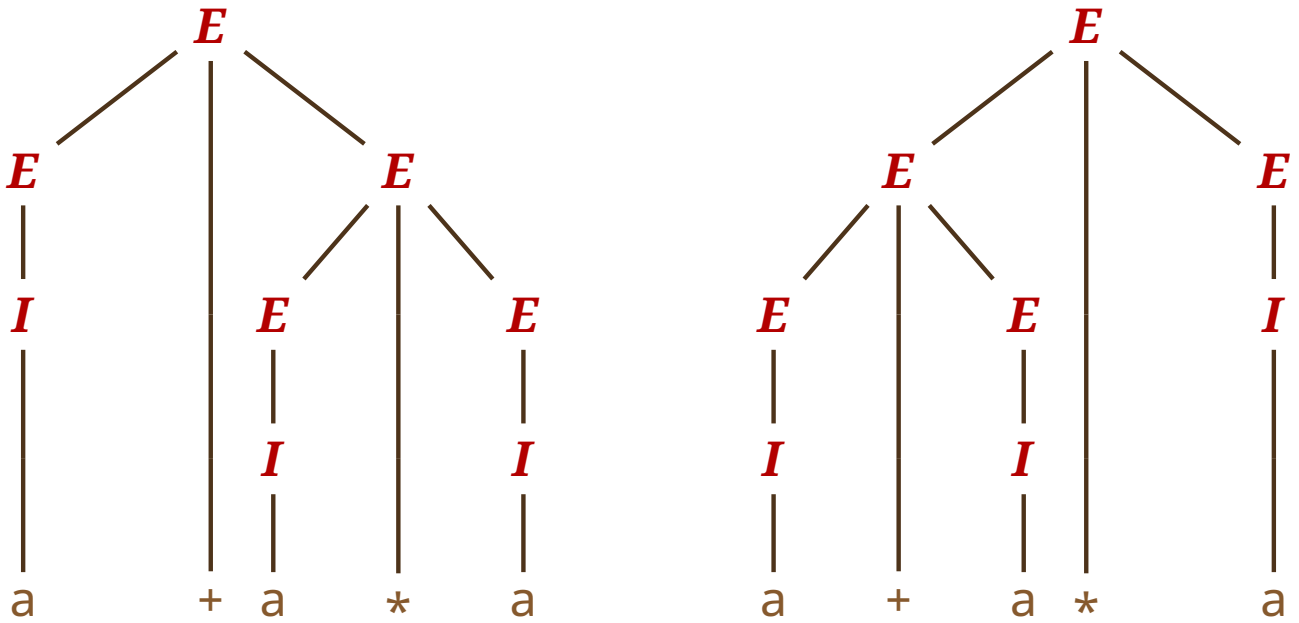
Derivazione non canonica di a+a:

$$E \Longrightarrow E{+}E \Longrightarrow E{+}I \Longrightarrow I{+}I \Longrightarrow a{+}I \Longrightarrow a{+}a$$

Ambiguità I

Una grammatica G libera si dice **ambigua** sse esiste $w \in \mathbf{L}(G)$ generabile attraverso almeno due alberi di derivazione differenti ovvero sse w ammette almeno due derivazioni sinistre (o destre)

Esempio $G = (\{a, +, *\}, \{E, I\}, E, P)$ ambigua con
$P = \{E \longrightarrow E+E|E*E|I,\ I \longrightarrow a\}$
Ad es. $w = a+a*a$ ottenibile mediante 2 alberi:



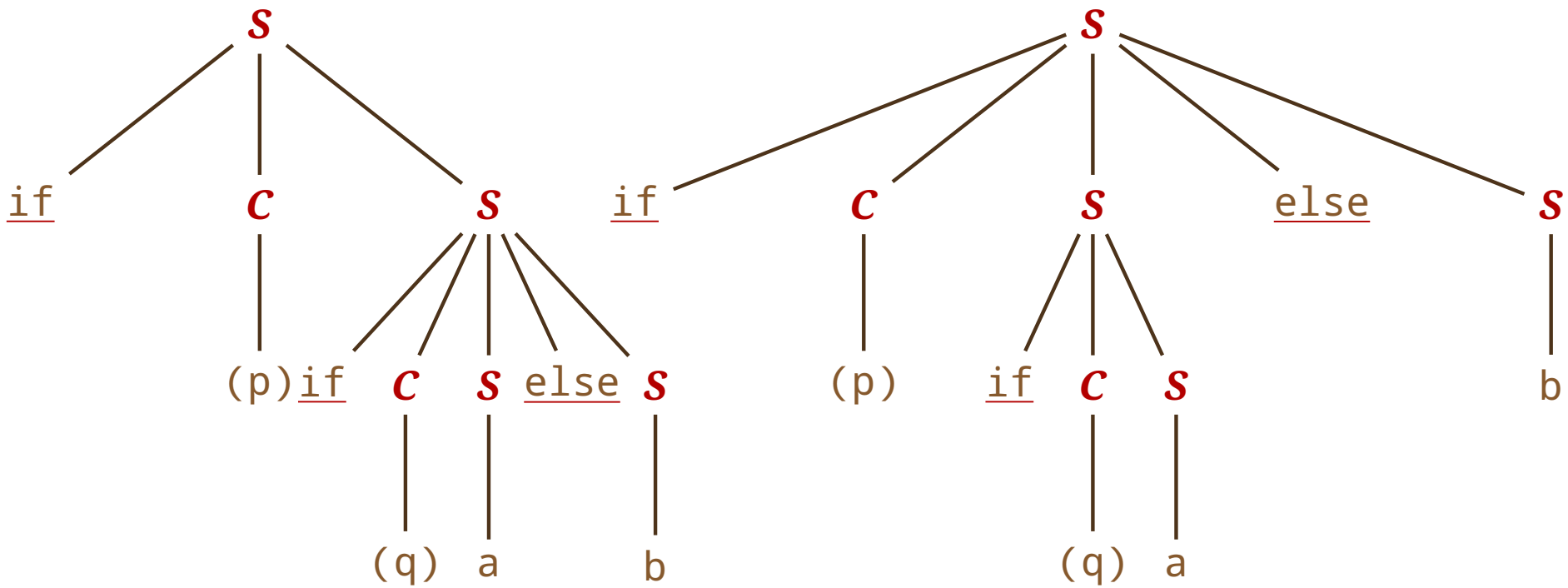
Ambiguità II

Esempio $G = (\Sigma, V, S, P)$ con $\Sigma = \{\underline{\texttt{if}}, \underline{\texttt{else}}, \texttt{(}, \texttt{)}, \texttt{a}, \texttt{b}, \texttt{p}, \texttt{q}\}$

$V = \{S, C\}$ $P = \{\ S \longrightarrow \underline{\texttt{if}}\ C\ S\ \underline{\texttt{else}}\ S \mid \underline{\texttt{if}}\ C\ S \mid \texttt{a} \mid \texttt{b},$

$C \longrightarrow \texttt{(p)} \mid \texttt{(q)}\ \}$

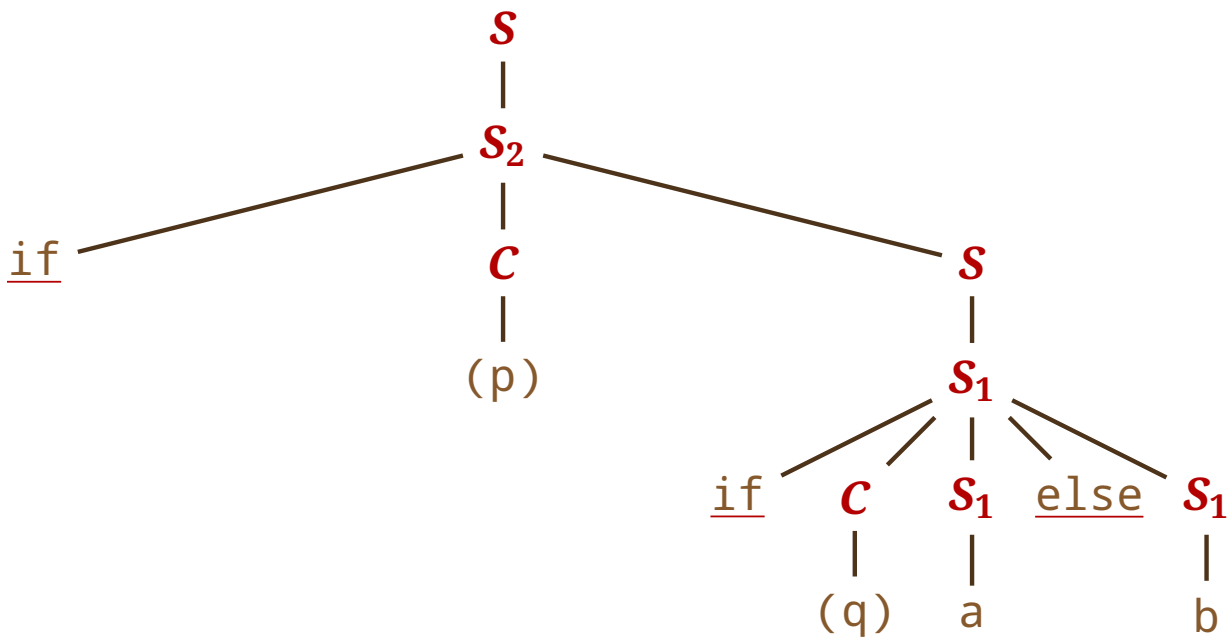
Si consideri la stringa $\underline{\texttt{if}}\ \texttt{(p)}\ \underline{\texttt{if}}\ \texttt{(q)}\ \texttt{a}\ \underline{\texttt{else}}\ \texttt{b}$



Ambiguità III

$G' = (\Sigma, V \cup \{S_1, S_2\}, S, P')$ non ambigua,
si adotta la convenzione di associare ogni `else` alla `if` più vicina:

$$P' = \left\{ \begin{array}{lll} S & \longrightarrow & S_1 \mid S_2 \\ S_1 & \longrightarrow & \texttt{if}\ C\ S_1\ \texttt{else}\ S_1 \mid \texttt{a} \mid \texttt{b} \\ S_2 & \longrightarrow & \texttt{if}\ C\ S \mid \texttt{if}\ C\ S_1\ \texttt{else}\ S_2 \\ C & \longrightarrow & \texttt{(p)} \mid \texttt{(q)} \end{array} \right\}$$



Ambiguità IV

Teorema

Il problema del riconoscimento dell'ambiguità di un linguaggio libero non è decidibile.

In pratica: soluzioni con aggiunta di livelli di precedenza, parentesi e associatività

Esempio modificando la precedente grammatica ambigua: $G = (\{\mathtt{a}, +, *, (,)\}, \{E, I, T, F\}, E, P)$ non ambigua con $P = \{E \longrightarrow T | E+E,\ T \longrightarrow F | T*T,\ F \longrightarrow I | (E),\ I \longrightarrow \mathtt{a}\}$

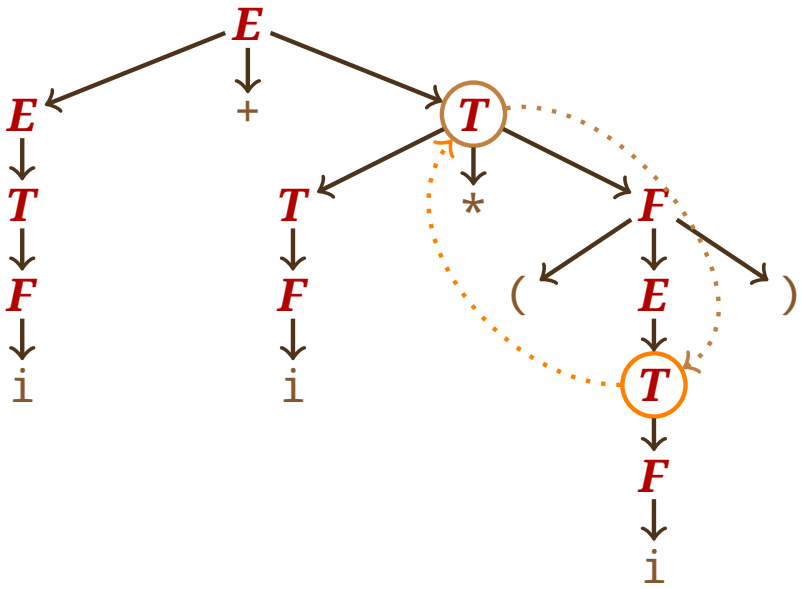
Un linguaggio libero L si dice **inerentemente ambiguo** sse ogni grammatica che lo genera è ambigua

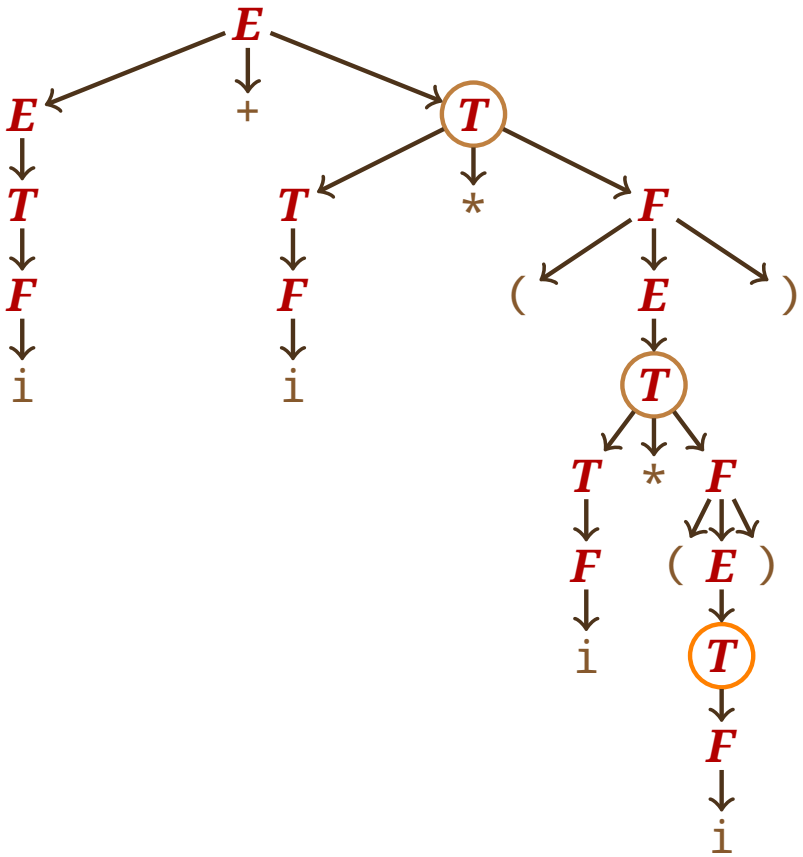
Esempio $L = \{a^i b^j c^k \mid i, j, k > 0,\ (i = j) \vee (j = k)\}$

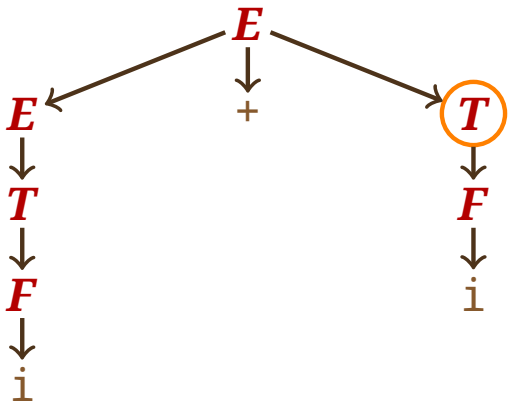
Sommario

Principio di sostituzione I

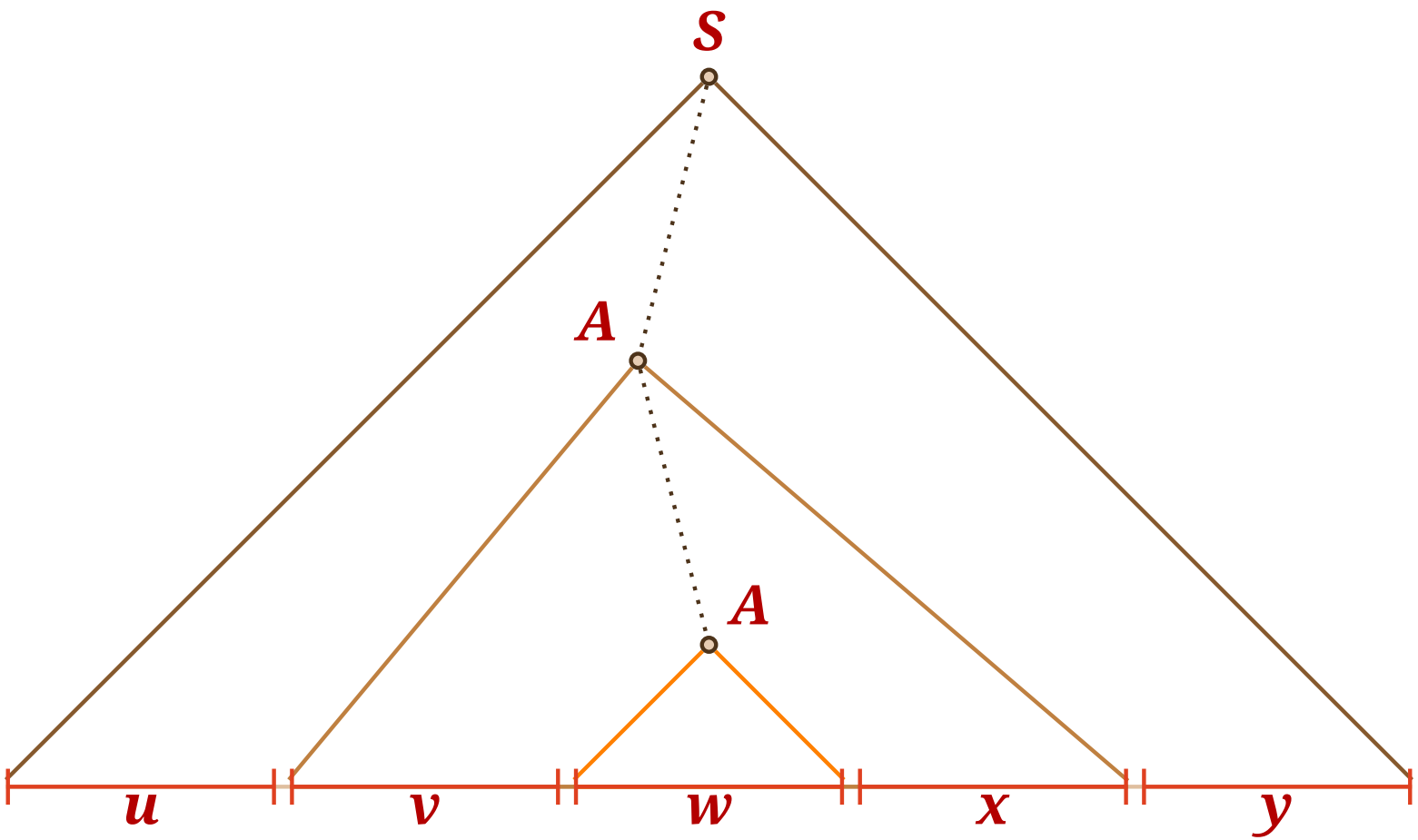
$$P = \{E \to T|E{+}T,\ T \to F|T{*}F,\ F \to \mathtt{i}|(E)\}$$







Principio di sostituzione II

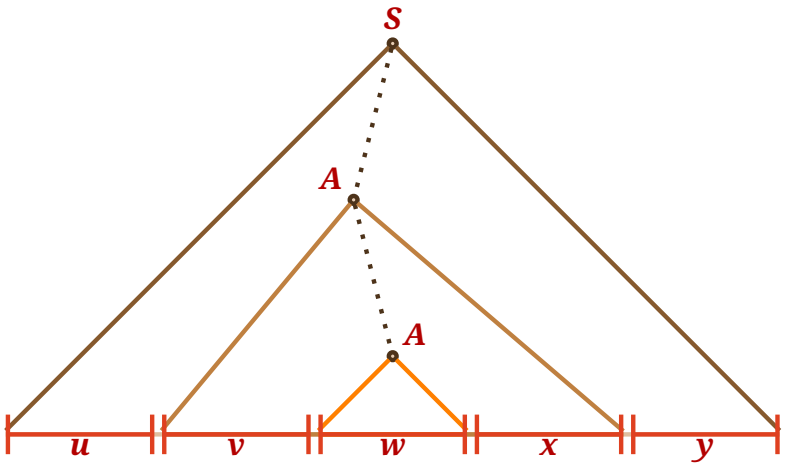


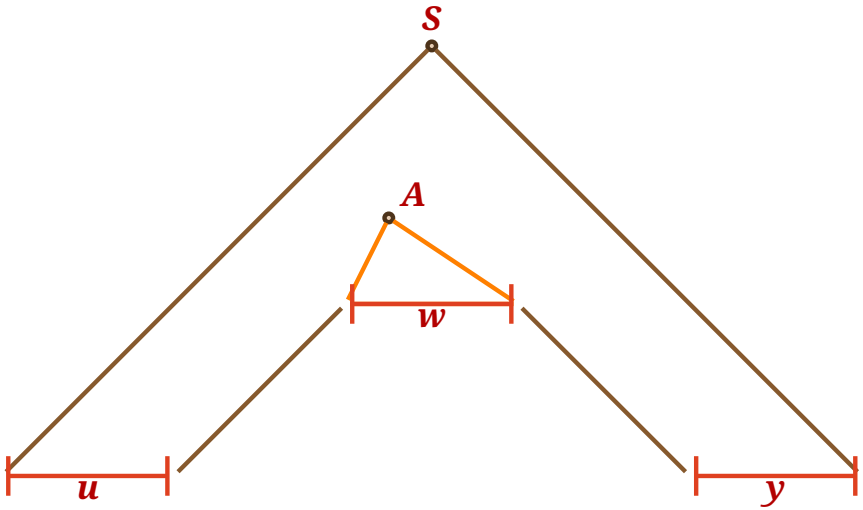
Principio di sostituzione III

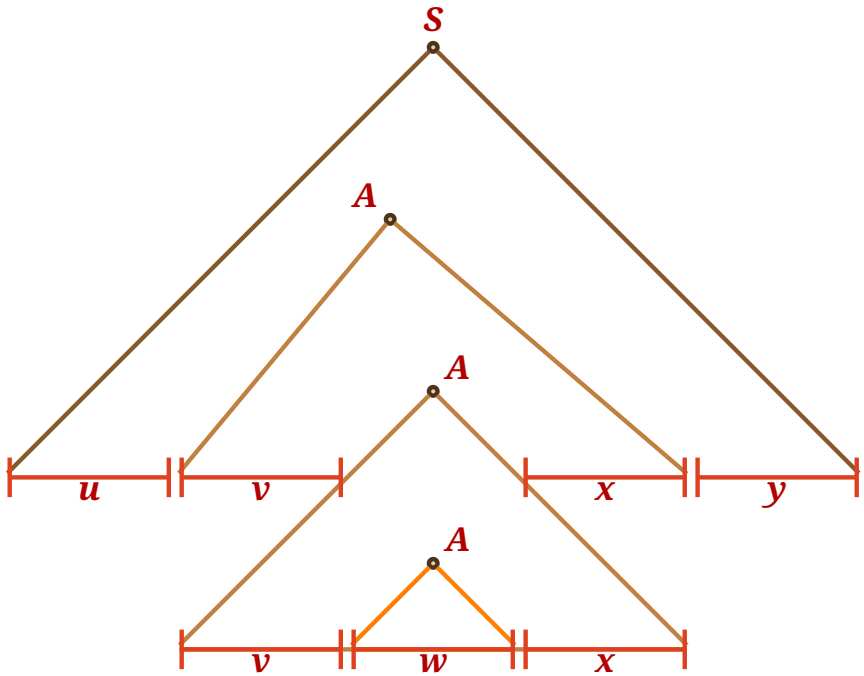
- Sottoalberi si possono sostituire con sottoalberi di pari radice
- Generalizzazione: se si incontra almeno due volte un NT A nell'albero di derivazione di z:
  - il sottoalbero interno con radice A genera w
    quello esterno genera vwx
  - sostituendo l'albero esterno con quello interno si ottiene un albero valido per uwy
  - invece, sostituendo interno con quello esterno si ottiene un albero per $uvvwxxy$ cioè uv^2wx^2y
  - iterando questa sostituzione possono derivare le stringhe in

$$\{uv^nwx^ny \mid n \geq 0\}$$

Principio di sostituzione IV







Relazione altezza albero – lunghezza stringa I

Il **fattore di ramificazione** per una grammatica libera $G = (\Sigma, V, S, P)$ è dato da:

$$\max\{|\omega| \in \mathbb{N} \mid A \longrightarrow \omega \in P\}$$

Proposizione

Data una grammatica libera $G = (\Sigma, V, S, P)$ con fattore di ramificazione m, sia $w \in \mathsf{L}(G)$.
Se $h \in \mathbb{N}$ è l'altezza di T_w, allora $|w| \leq m^h$

Dim. Per induzione sull'altezza h dell'albero di T_w:

Relazione altezza albero – lunghezza stringa II

$h = 1$ In tal caso, der. diretta con una $S \longrightarrow \omega$.
Ma $|\omega| \leq m = m^1$

$h > 1$ Si suppone che la tesi valga per alberi di altezza pari al più a h, si deve dimostrare per T_w di altezza $h + 1$.

Sia $S \longrightarrow \omega$, dove $\alpha = X_1 X_2 \cdots X_k$
con $X_i \in \Sigma \cup V$ in ω, $i = 1, \ldots, k \leq m$ la prod. che determina il livello più alto dell'albero T_w

Ogni X_i può essere, a sua volta, radice di T_i di altezza $\leq h$.
Quindi, per ipotesi di induzione, i T_i generano k stringhe w_i tali che $|w_i| \leq m^h$.

Pertanto:
$|w| = \sum_{i=1}^{k} |w_i| \leq \sum_{i=1}^{k} m^h = k \cdot m^h \leq m \cdot m^h = m^{h+1}$

Pumping Lemma per Linguaggi Liberi I

Teorema (uvwxy)

Sia L un linguaggio libero da contesto.
Esiste una costante p, dipendente solo da L, tale che
ogni $z \in L$ con $|z| > p$ può essere scritta come $uvwxy$ in modo che:

1. $|vwx| \leq p$
2. *al più uno tra v e x è la parola vuota ($vx \neq \epsilon$)*
3. $\forall i \in \mathbb{N}: \; uv^i wx^i y \in L$

Dim. Sia G una grammatica che genera L
Siano $m = \max\{|\omega| \mid A \longrightarrow \omega \in P\}$ e $k = |V|$

Posto $p = m^{k+1}$, consideriamo $z \in L$ tale che $|z| > p$

Pumping Lemma per Linguaggi Liberi II

Per il lemma: $|z| > p = m^{k+1}$ allora ogni albero di derivazione per z ha un'altezza maggiore di $k+1$

$k = |V|$ implica che in un cammino dell'albero ci sia una multipla occorrenza di un NT, sia esso $A \in V$

Possiamo identificare 2 occorrenze:

1) la prima compare ad una distanza dalle foglie $\leq k+1$ sia vwx la stringa derivata dal suo sottoalbero
2) dalla seconda occorrenza di A deriva la sottostringa w

Pumping Lemma per Linguaggi Liberi III

1. Dal Lemma risulta: $|vwx| \leq m^{k+1} = p$
2. Per assurdo, se fosse $v = \epsilon = x$, la sostituzione del sottoalbero superiore con quello inferiore non provoca nessun cambiamento.

   Ma, in tal caso, esiste un cammino di lunghezza $\leq k + 1$ e quindi si avrebbe un albero per z di altezza $\leq k + 1$. Contraddizione.
3. Applicando il principio di sostituzione a $z = uvwxy$ sostituiamo il sottoalbero inferiore con quello superiore, ottenendo:

   $$uwy = uv^0wx^0y$$

   Con la sostituzione inversa: uv^2wx^2y e ripetendo $i - 1$ volte:

   $$uv^iwx^iy$$

Pumping Lemma per Linguaggi Liberi IV

Osservazioni

- Dato un linguaggio generato da una grammatica non libera non si può escludere che esista una grammatica libera che lo generi
- Se un linguaggio infinito non rispetta il *Pumping Lemma* dei linguaggi liberi non può essere generato da una grammatica libera
- Quindi questo teorema fornisce una condizione *necessaria* ma non sufficiente perché un linguaggio sia libero
  - Si utilizza per dimostrare che un linguaggio non è libero

Esercizi – Pumping Lemma I

Dimostrare che i seguenti linguaggi non sono liberi da contesto:

1. $\{a^t \mid t \text{ primo}\}$
2. $\{a^n b^n c^n \mid n > 0\}$
3. $\{a^{n^2} \mid n \geq 0\}$
4. $\{a^i b^j \mid i = 2^j,\ i, j \geq 0\}$
5. $\{a^n b^m \mid n > 2^m,\ n, m \geq 0\}$
6. $\{a^k b^r \mid k > 0,\ r > k^2\}$
7. $\{a^n b^m c^p \mid 1 \leq n \leq m \leq p\}$
8. $\{a^{2^n} \mid n \geq 1\}$
9. $\{a^i b^j c^i d^j \mid i, j \geq 1\}$
10. $\{ww \mid w \in \{0, 1\}^+\}$

Esercizi – Pumping Lemma II

Esercizio 1. Dimostrare che $L = \{a^t \mid t \text{ primo}\} \notin \mathcal{L}_2$

- Dato p del P.L., sia t primo tale che $t > p$
- Si può scegliere quindi $z = a^t \in L$
- Sia $z = uvwxy$, con $v = a^k$ e $x = a^h$, essendo $k + h > 0$ (poiché $vx \neq \epsilon$)
- Si consideri $z_{(t+1)} = uv^{(t+1)}wx^{(t+1)}y$

$$|z_{(t+1)}| = |z| + |v^t| + |x^t| = t + (k+h)t = t(1+k+h)$$

- Ma $t(1+k+h)$ non è primo, quindi $z_{(t+1)} \notin L$

Contraddizione. Quindi $L \notin \mathcal{L}_2$.

Esercizi – Pumping Lemma III

Esercizio 2. Dimostrare che $L = \{a^n b^n c^n \mid n > 0\}$ non è libero

- Supponiamo che L sia libero.
  Allora vale il P.L.: per un certo $p \in \mathbb{N}$ fissato si consideri
  $z = uvwxy = a^p b^p c^p \in L$: $|z| = 3p > p$.
- Per il P.L.: $|vwx| \leq p$ e $z_i = uv^i wx^i y \in L$, $\forall i \in \mathbb{N}$ con $vx \neq \epsilon$
- Per costruzione, vwx non può contenere assieme sia a iniziali sia c finali, essendo $|vwx| \leq p$
- Quindi 2 casi:
  1. nessuna c: quindi $z_0 = uv^0 wx^0 y = uwy \notin L$
     avendo perso delle a e b in v e x ma non le c di z
  2. nessuna a: analogamente, $uwy \notin L$
     avendo meno b e c delle a rimaste inalterate (sono p).

  (nessuna b, ma solo a e c, impossibile)

Quindi L non può essere libero

Esercizi – Pumping Lemma IV

Esercizio 3. Dimostrare che $L = \{a^{n^2} \mid n \geq 0\}$ non è libero

- Supponiamo che $L = \{\epsilon, a, aaaa, a^9, a^{16}, \ldots\}$ sia libero.
- Valendo il *Pumping Lemma* per un certo $p \in \mathbb{N}$.
- si considera $z = uvwxy = a^{p^2} \in L$ tale che $|z| = p^2 > p$
- Anche $z_2 = uv^2wx^2y \in L$ (per la 3. del Lemma)
- Ma si osservi la catena di maggiorazioni:
  $|z_2| = |z| + |vx| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$
- Quindi: $|z_2| < (p+1)^2$. Ma $|z_2| = |z| + |vx| > |z| = p^2$
- Perciò $p^2 < |z_2| < (p+1)^2$ lunghezza compresa tra due quadrati consecutivi di n.ri naturali, ciò implica che $z_2 \notin L$.

Assurdo, quindi L non è libero

Esercizi – Pumping Lemma V

Esercizio 5. Dimostrare che $\{a^n b^m \mid n > 2^m,\ n, m \geq 0\} \notin \mathcal{L}_2$

- Se vale il *Pumping Lemma* per un certo $p \in \mathbb{N}$ si considera $z = uvwxy = a^{(2^p+s)} b^p \in L$ tale che $|z| = 2^p + s + p > p$
- Caso: $v = a^h$ e $x = b^k$.
  - Allora $|z_i| = |uv^i wx^i y| = |z| + (i-1)h + (i-1)k = 2^p + s + p + (i-1)h + (i-1)k$.
  - Se $\exists i:\ 2^p + s + (i-1)h \leq p + (i-1)k$ allora $z_i \notin L$
  - La diseq. si può semplificare $2^p + (i-1)h \leq (i-1)k$ da cui: $i(k-h) \geq 2^p - (k-h)$ ossia $i \geq 2^p/(k-h) - 1$.
    Se $k > h$ allora basta considerare $i = 2^p \geq 2^p/(k-h) - 1$
- Altri casi, per esercizio.

Quindi L non è libero

Chiusura rispetto all'unione I

Teorema (chiusura unione)

La classe dei linguaggi liberi $\mathcal{L}_2$ è chiusa rispetto all'unione.

Dim.

Siano $L_1, L_2 \in \mathcal{L}_2$, generati, risp., da $G_1 = (\Sigma, V_1, S_1, P_1)$ e $G_2 = (\Sigma, V_2, S_2, P_2)$ con $V_1 \cap V_2 = \emptyset$

Si costruisce $G = (\Sigma, V, S, P)$ con

- $V = V_1 \cup V_2 \cup \{S\}$,
- $S \notin V_1 \cup V_2$
- $P = \{S \longrightarrow S_1 | S_2\} \cup P_1 \cup P_2$

[Nota: questo vale anche per $L_1, L_2 \in \mathcal{L}_i,\ i = 0, 1$]

Chiusura rispetto al prodotto

Teorema (chiusura prodotto)

La classe dei linguaggi liberi $\mathcal{L}_2$ è chiusa rispetto al prodotto

Dim.
Date $G_1 = (\Sigma, V_1, S_1, P_1)$ e $G_2 = (\Sigma, V_2, S_2, P_2)$ libere tali che $\mathsf{L}(G_i) = L_i, \quad i = 1, 2$

si costruisce $G = (\Sigma, V, S, P)$ con

$$P = \{S \longrightarrow S_1 \cdot S_2\} \cup P_1 \cup P_2$$

Si osservi che questo metodo vale per solo la classe $\mathcal{L}_2$
(per $\mathcal{L}_{0,1}$ occorre evitare l'*interferenza dei contesti*)

Chiusura rispetto all'iterazione

Teorema (chiusura iterazione)

La classe dei linguaggi liberi $\mathcal{L}_2$ è chiusa rispetto all'iterazione

Dim.

Per $L_1 \in \mathcal{L}_2$, considerata $G_1 = (\Sigma, V_1, S_1, P_1)$, costruiamo $G = (\Sigma, V_1 \cup \{S\}, S, P)$ dove:

$$P = \{S \longrightarrow \epsilon,\ S \longrightarrow S_1 S\} \cup P_1$$

Anche questo metodo vale per solo la classe $\mathcal{L}_2$ (per $\mathcal{L}_{0,1}$ occorre evitare l'*interferenza dei contesti*)

Altre proprietà di chiusura

Proposizione

La classe dei linguaggi liberi $\mathcal{L}_2$ non è chiusa rispetto al complemento e all'intersezione

Dim. si considerino i linguaggi
$L_1 = \{a^n b^n c^k \mid n, k > 0\}$ e
$L_2 = \{a^n b^k c^k \mid n, k > 0\}$
la cui intersezione è $L_1 \cap L_2 = \{a^k b^k c^k \mid k > 0\} \in \mathcal{L}_1 \setminus \mathcal{L}_2$

Tuttavia:

Teorema

Se L è libero e L_R è regolare allora $L \cap L_R$ è libero

Chiusura rispetto alla sostituzione I

Si definisce una **sostituzione** su Σ come $s : \Sigma \to \wp(T^*)$ tale che:

$$a \in \Sigma \mapsto s(a) = L_a \subseteq T^*$$

Questa funzione si può estendere alle stringhe, $s : \Sigma^* \to \wp(T^*)$ con

$$w = a_1 a_2 \cdots a_n \in \Sigma^* \mapsto s(w) = s(a_1)s(a_2)\cdots s(a_n)$$

ed ai linguaggi: $s : \wp(\Sigma^*) \to \wp(T^*)$ con

$$L \subseteq \Sigma^* \mapsto s(L) = \bigcup_{w \in L} s(w)$$

Chiusura rispetto alla sostituzione II

Esempio Siano $s(0) = \{a^n b^n \mid n > 0\}$ e $s(1) = \{aa, bb\}$.
Sia $w = 01$, allora
$s(w) = s(0)s(1) = \{w \mid w = a^n b^n aa \vee w = a^n b^n bb,\ n > 0\}$
Sia $L = \mathbf{L}(0^*)$, allora $s(L) = s((0)^*) = [s(0)]^*$, quindi
$s(L) = \{a^{n_1} b^{n_1} a^{n_2} b^{n_2} \ldots a^{n_k} b^{n_k} \mid k \geq 0\}$

Teorema (chiusura sostituzione)

La classe dei linguaggi liberi è chiusa rispetto alla sostituzione.

Da ciò discendono anche le chiusure rispetto ad unione, prodotto e chiusure positiva e transitiva e riflessiva ($^+$ e *) ed anche:

Teorema (chiusura omomorfismo)

La classe dei linguaggi liberi è chiusa rispetto ad omomorfismo (ed anche all'omomorfismo inverso)

Chiusura rispetto alla riflessione

- Una parola $w \in \Sigma^*$ si dice **palindroma** se $w = w^R$

Proposizione

Sia $w \in \Sigma^*$. Allora w è palindroma sse $\exists v \in \Sigma^* \; \exists s \in \Sigma \cup \{\epsilon\}$

$$w = vsv^R$$

Teorema (chiusura rispetto alla riflessione)

La classe dei linguaggi liberi è chiusa rispetto a riflessione.

Esercizi – proprietà di chiusura I

1. Dimostrare che $L = \{a^n b^n c^m \mid n, m > 0\}$ è libero
2. Dati $L_1 = \{a^n b^n \mid n \geq 0\}$ e $L_2 = \{a\}^* \cdot \{bb\}^*$, dimostrare che $L = L_1 \cap L_2$ è libero
3. Utilizzare la proprietà di chiusura di $\mathcal{L}_2$ rispetto a $\cup$ per dimostrare che i seguenti linguaggi sono liberi:
   i. $L = \{a^i b^k \mid i \neq k,\ i, k \geq 0\}$
   ii. $L = \{w \in \{a, b\}^* \mid w = w^R\}$
   iii. $L = \{a, b\}^* \setminus \{a^i b^i \mid i \geq 0\}$
4. Usando la proprietà di chiusura dell'intersezione con ling. regolari, dimostrare che sono liberi:
   i. $\{a^n b^n \mid n \neq 7\}$
   ii. $\{0^n 1^n \mid n = 3k + 1,\ k \geq 0\}$
   iii. $\{w \in \{a, b\}^* \mid w \textit{ palindroma},\ |w| \neq 3k,\ k \geq 0\}$
5. Analogamente, dimostrare che non sono liberi:
   i. $\{a^n b^m c^k \mid k \geq m \geq n \geq 0\}$
   ii. $\{w \in \{a, b, c\}^* \mid n_a(w) = n_b(w) = n_c(w)\}$

Esercizi – proprietà di chiusura II

Esercizio 1. Dimostrare che $L = \{a^n b^n c^m \mid n, m > 0\}$ è libero e trovare una grammatica che lo generi

---

L può essere inteso come prodotto di linguaggi:
$L_1 = \{a^n b^n \mid n > 0\}$ libero e
$L_2 = \{c^m \mid m > 0\} = \{c\}^+ = \{c\}^* \setminus \{\epsilon\}$ lineare destro
(e quindi anche libero $\mathcal{L}_3 \subsetneq \mathcal{L}_2$)
$L = L_1 \cdot L_2$ deve essere libero per la chiusura

- $G_1 = (\{a, b\}, \{S_1\}, S_1, P_1)$ con $P_1 = \{S_1 \longrightarrow aS_1b|ab\}$
- $G_2 = (\{c\}, \{S_2\}, S_2, P_2)$ con $P_2 = \{S_2 \longrightarrow cS_2|c\}$

Quindi: $G = (\Sigma, V, S, P)$
$\Sigma = \{a, b\} \cup \{c\} = \{a, b, c\}$ $\quad V = V_1 \cup V_2 \cup \{S\} = \{S, S_1, S_2\}$

$$P = \{S \longrightarrow S_1S_2\} \cup P_1 \cup P_2 =$$
$$= \{S \longrightarrow S_1S_2,\ S_1 \longrightarrow aS_1b|ab,\ S_2 \longrightarrow cS_2|c\}$$

Esercizi – proprietà di chiusura III

Esercizio 3. Utilizzare la proprietà di chiusura di $\mathcal{L}_2$ rispetto a $\cup$ per dimostrare che i seguenti linguaggi sono liberi:

i. $L = \{a^i b^k \mid i \neq k,\ i, k \geq 0\}$
ii. $L = \{w \in \{a, b\}^* \mid w = w^R\}$
iii. $L = \{a, b\}^* \setminus \{a^i b^i \mid i \geq 0\}$

---

1. Riscrivendo L come segue $\{a^i b^k \mid i < k\} \cup \{a^i b^k \mid i > k\}$ ...
2. Si osservi che $w \in L \Leftrightarrow w = szs$, e inoltre $w = uu^R$ oppure $w = usu^R$, con $s \in \Sigma$ e $z \in L, u \in \Sigma^*$ ...
3. Scomponiamo il linguaggio (in modo ricorsivo) come segue

$$\begin{aligned} L = &\{wa \mid w \in \Sigma^*\} \cup \\ &\{bw \mid w \in \Sigma^*\} \cup \\ &\{sws \mid w \in \Sigma^*,\ s \in \Sigma\} \cup \\ &\{azb \mid z \in L\} \end{aligned}$$

Sommario

Automi con Memoria

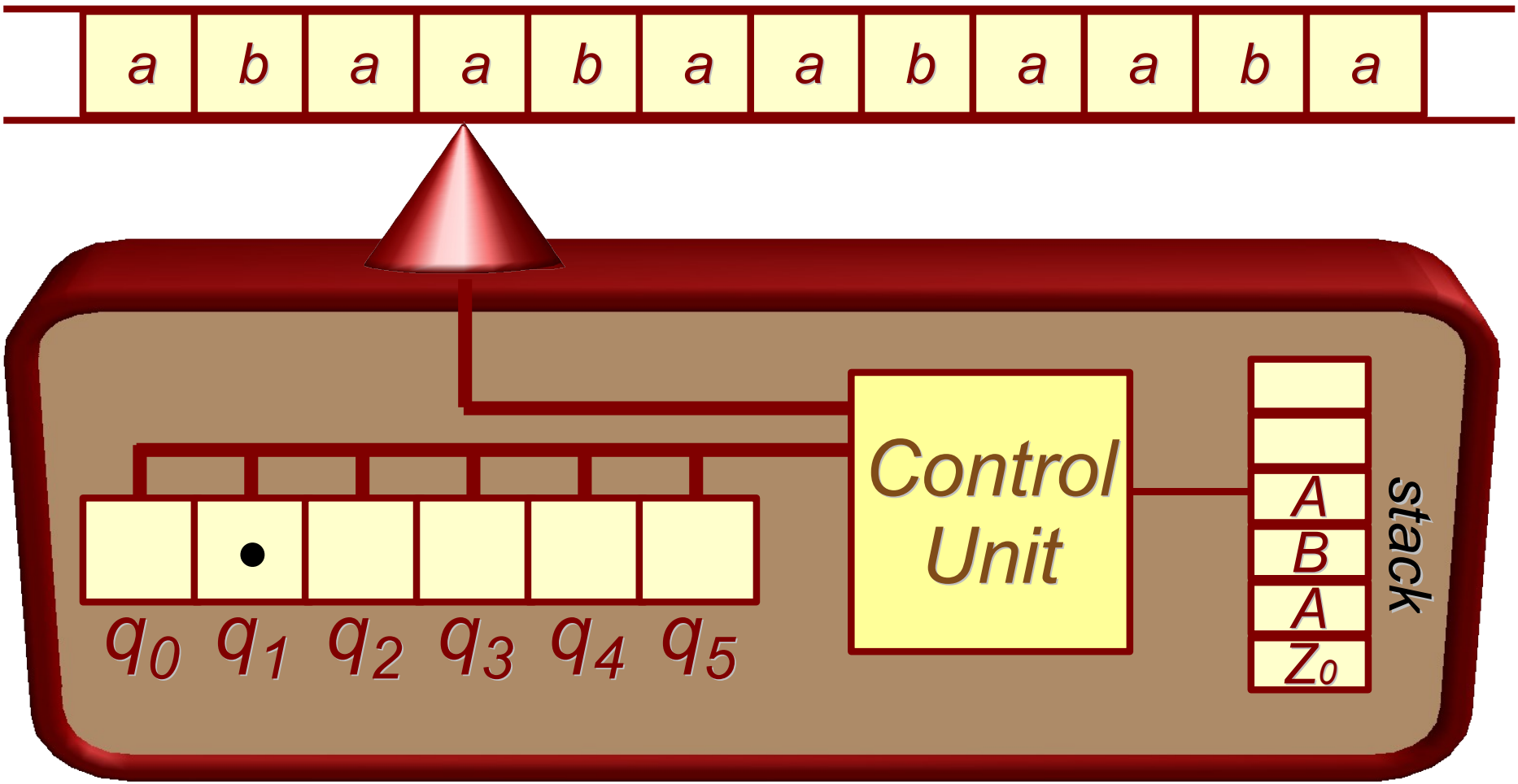
Per *riconoscere* linguaggi più complessi rispetto a quelli di $\mathcal{L}_{\text{df}}$ occorrono altri strumenti più potenti:
Dato un alfabeto finito Σ di ingresso:

nastro di ingresso: contiene i simboli dell'alfabeto di ingresso Σ; su un simbolo insiste una *testina di lettura*

memoria ausiliaria: ha capacità virtualmente illimitata ed un proprio *alfabeto di memoria* (o *di lavoro*)
Se l'organizzazione della memoria è a stack, si definirà l'automa risultante *automa a pila* (o *automa push-down*, *PDA*)

unità di controllo: controlla le transizioni dell'automa, in base al contenuto della memoria e del simbolo letto dalla testina sul nastro.

Modello di PDA



Automi a pila I

Un *automa a pila* è costituito da una n-pla:

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

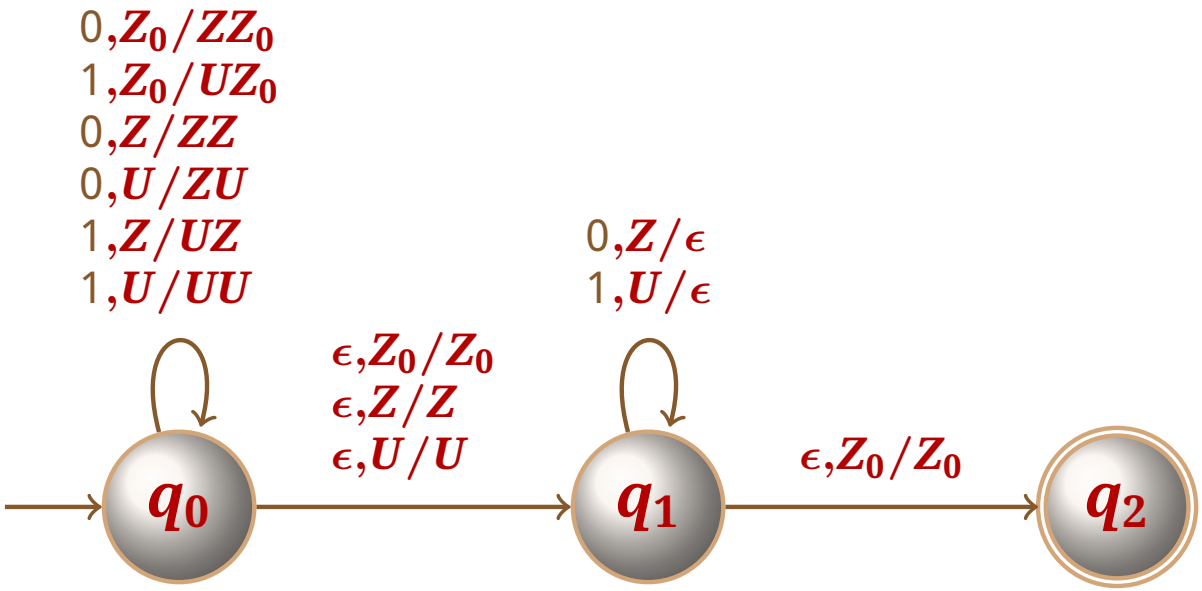
- Q insieme finito e non vuoto degli *stati*
- Σ insieme finito e non vuoto detto *alfabeto di ingresso*
- Γ insieme finito e non vuoto detto *alfabeto della pila*
- δ funzione di transizione:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \longrightarrow \wp(Q \times \Gamma^*)$$

  scritta anche come $(q', \sigma) \in \delta(q, s, Z)$ ovvero (q, s, Z, q', σ)
- $q_0 \in Q$ è lo *stato iniziale*
- $Z_0 \in \Gamma$ è il *simbolo iniziale* della pila
- $F \subseteq Q$ è l'insieme degli *stati finali*

Automi a pila II

Diagramma di transizione:



Configurazioni Istantanee e transizioni I

Per descrivere lo *stato globale* di un PDA in ogni istante, una *configurazione istantanea* (*ID*) è una terna

$$(q, w, \sigma) \in Q \times \Sigma^* \times \Gamma^*$$

- $q \in Q$ — *stato corrente* dell'unità di controllo
- $w = a_1 a_2 \cdots a_n \in \Sigma^*$ — sottostringa da esaminare sul nastro (testina posizionata su a_1)
- $\sigma = Z_1 Z_2 \cdots Z_m$ — contenuto della pila (con Z_1 in cima)

configurazione iniziale (q_0, w, Z_0)

configurazione finale (q, ϵ, σ) con $q \in F$, $\sigma \in \Gamma^*$

Configurazioni Istantanee e transizioni II

Sia M nello stato q, sia a letto sul nastro e A in cima alla pila:

1. se δ contiene $(q, a, A, q', A_1 \cdots A_k)$
   ossia se $(q', A_1 \cdots A_k) \in \delta(q, a, A)$
   M *può* operare la transizione:

$$(q, aw, A\sigma) \vdash (q', w, A_1 \cdots A_k\sigma)$$

2. se δ è descritta dalla n-pla $(q, \epsilon, A, q', A_1 \cdots A_k)$
   ossia se $(q', A_1 \cdots A_k) \in \delta(q, \epsilon, A)$
   allora M *può* operare la transizione:

$$(q, w, A\sigma) \vdash (q', w, A_1 \cdots A_k\sigma)$$

Condizioni di accettazione per PDA

$w \in \Sigma^*$ è **accettata** dal PDA M *in condizione di stato finale* sse:

$$(q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \sigma) \qquad q \in F,\ \sigma \in \Gamma^*$$

linguaggio accettato da M *in condizione di stato finale*:

$$\mathsf{L}(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \sigma) \text{ con } q \in F,\ \sigma \in \Gamma^*\}$$

$w \in \Sigma^*$ è **accettata** dal PDA M *in condizione di pila vuota* sse:

$$(q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon) \qquad q \in Q$$

linguaggio accettato da M *in condizione di pila vuota*:

$$\mathsf{L}(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon) \text{ con } q \in Q\}$$

Linguaggi accettati da PDA

Teorema

La classe dei linguaggi accettati da PDA in condizione di pila vuota coincide con la classe dei linguaggi accettati in condizione di stato finale

Dim. (cenni)

Se si raggiunge uno stato finale e la pila non è vuota occorre ripulire la pila dai simboli restanti senza cambiare stato

Se la pila è vuota bisogna obbligare l'automa a transitare in uno stato finale senza modificare la situazione della pila

PDA – Esempio #1 / I

$$L = \{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\}$$

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

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- $P = \{S \longrightarrow ab|ba|SS|aSb|bSa\}$

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

- $Q = \{q_0\}$
- $\Gamma = \{Z_0, A, B\}$
- $F = \emptyset$

pila	stato	a	b
A	q_0	AA	ϵ
B	q_0	ϵ	BB
Z_0	q_0	AZ_0	BZ_0

PDA – Esempio #1 / II

Programma per PDA (è omesso lo stato q_0):

1. (a, Z_0, AZ_0)
2. (b, Z_0, BZ_0)
3. (a, A, AA)
4. (b, B, BB)
5. (a, B, ϵ)
6. (b, A, ϵ)
7. $(\epsilon, Z_0, \epsilon)$ ϵ-regola per cancellare il fondo della pila

PDA – Esempio #1 / III

pila	stato	a	b
A	q_0	AA	ϵ
B	q_0	ϵ	BB
Z_0	q_0	AZ_0	BZ_0

stringa abba

$(q_0, \text{abba}, Z_0) \vdash (q_0, \text{bba}, AZ_0) \vdash (q_0, \text{ba}, Z_0) \vdash (q_0, \text{a}, BZ_0) \vdash (q_0, \epsilon, Z_0) \vdash (q_0, \epsilon, \epsilon)$

la stringa è accettata.

---

stringa aababa

$(q_0, \text{aababa}, Z_0) \vdash (q_0, \text{ababa}, AZ_0) \vdash (q_0, \text{baba}, AAZ_0) \vdash (q_0, \text{aba}, AZ_0) \vdash (q_0, \text{ba}, AAZ_0) \vdash (q_0, \text{a}, AZ_0) \vdash (q_0, \epsilon, AAZ_0)$

la stringa non è accettata.

PDA – Esempio #2 / I

$$L = \{wcw^R \mid w \in \{a,b\}^*\}$$

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

- $\Sigma = \{a, b, c\}$
- $V = \{S\}$
- $P = \{S \longrightarrow c|aSa|bSb\}$

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

- $Q = \{q_0, q_1\}$
  - q_0: lettura
  - q_1: match
- $\Gamma = \{Z_0, A, B\}$
- $F = \emptyset$

PDA – Esempio #2 / II

pila	stato	a	b	c
A	q_0	(q_0, AA)	(q_0, BA)	(q_1, A)
A	q_1	(q_1, ϵ)	-	-
B	q_0	(q_0, AB)	(q_0, BB)	(q_1, B)
B	q_1	-	(q_1, ϵ)	-
Z_0	q_0	(q_0, AZ_0)	(q_0, BZ_0)	(q_1, Z_0)
Z_0	q_1	-	-	-

Osservazione

c segnala il cambiamento di stato (centro del palindromo): in q_1 si comincia a svuotare la pila riempita nello stato q_0

PDA – Esempio #2 / III

1. $(q_0, \mathtt{a}, Z_0, q_0, AZ_0)$
2. $(q_0, \mathtt{a}, A, q_0, AA)$
3. $(q_0, \mathtt{a}, B, q_0, AB)$
4. $(q_0, \mathtt{b}, Z_0, q_0, BZ_0)$
5. $(q_0, \mathtt{b}, A, q_0, BA)$
6. $(q_0, \mathtt{b}, B, q_0, BB)$
7. $(q_0, \mathtt{c}, Z_0, q_1, Z_0)$
8. $(q_0, \mathtt{c}, A, q_1, A)$
9. $(q_0, \mathtt{c}, B, q_1, B)$
10. $(q_1, \mathtt{a}, A, q_1, \epsilon)$
11. $(q_1, \mathtt{b}, B, q_1, \epsilon)$
12. $(q_1, \epsilon, Z_0, q_1, \epsilon)$ ϵ-regola

Programma per PDA:

- la 1. la 2. e la 3. si possono riassumere con:
  $(q_0, \mathtt{a}, Z, q_0, AZ)\ \forall Z \in \Gamma$
- la 4. la 5. e la 6. si possono riassumere con:
  $(q_0, \mathtt{b}, Z, q_0, BZ)\ \forall Z \in \Gamma$
- la 7. la 8. e la 9. si possono riassumere con:
  $(q_0, \mathtt{c}, Z, q_1, Z)\ \forall Z \in \Gamma$

PDA – Esempio #3 / I

$$L = \{ww^R \mid w \in \{a,b\}^*\}$$

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

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- $P = \{S \longrightarrow \epsilon | aSa | bSb\}$

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

- $Q = \{q_0, q_1\}$
  - q_0: lettura
  - q_1: match
- $\Gamma = \{Z_0, A, B\}$
- $F = \emptyset$

PDA – Esempio #3 / II

pila	stato	a	b
A	q_0	$\{(q_0, AA), (q_1, \epsilon)\}$	(q_0, BA)
A	q_1	(q_1, ϵ)	-
B	q_0	(q_0, AB)	$\{(q_0, BB), (q_1, \epsilon)\}$
B	q_1	-	(q_1, ϵ)
Z_0	q_0	(q_0, AZ_0)	(q_0, BZ_0)
Z_0	q_1	-	-

automa non deterministico con due possibilità, trovandosi in q_0

- leggere il prossimo simbolo dal nastro
- passare in q_1

PDA – Esempio #3 / III

Programma per PDA:

1. $(q_0, \mathrm{a}, Z, q_0, AZ)\ \forall Z \in \Gamma$
2. $(q_0, \mathrm{b}, Z, q_0, BZ)$
3. $(q_0, \epsilon, Z, q_1, Z)$ (invece della ϵ-regola precedente)
4. $(q_1, \mathrm{a}, A, q_1, \epsilon)$
5. $(q_1, \mathrm{b}, B, q_1, \epsilon)$
6. $(q_1, \epsilon, Z_0, q_1, \epsilon)$

PDA – Esempio #3 / IV

pila	stato	a	b
A	q_0	$\{(q_0, AA), (q_1, \epsilon)\}$	(q_0, BA)
A	q_1	(q_1, ϵ)	-
B	q_0	(q_0, AB)	$\{(q_0, BB), (q_1, \epsilon)\}$
B	q_1	-	(q_1, ϵ)
Z_0	q_0	(q_0, AZ_0)	(q_0, BZ_0)
Z_0	q_1	-	-

Riconoscimento della stringa abba

$(q_0, \text{abba}, Z_0) \vdash (q_0, \text{bba}, AZ_0) \vdash (q_0, \text{ba}, BAZ_0)$?

scelta $\overset{2}{\vdash} (q_0, \text{a}, BBAZ_0) \vdash (q_0, \epsilon, ABBAZ_0)$ blocco. ↺

scelta $\overset{5}{\vdash} (q_1, a, AZ_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_1, \epsilon, \epsilon)$

La stringa è accettata.

PDA e determinismo

Osservazioni

- $L_1 = \{wcw^R \mid w \in \{a,b\}^*\}$
  accettato da un PDA deterministico
- $L_2 = \{ww^R \mid w \in \{a,b\}^*\}$
  accettato da un PDA non deterministico

si può dimostrare che

$$\nexists M \in \text{PDA deterministico tale che } \mathbf{L}(M) = L_2$$

pertanto la classe dei linguaggi riconosciuta dai PDA deterministici è inclusa strettamente in quella dei linguaggi riconosciuti da PDA non deterministici

Sommario

Forme Normali I

Esempio Si consideri
$G = (\Sigma, V, S, P)$

- $\Sigma = \{0, 1, 2\}$
- $V = \{S, A, B\}$
- $P = \{S \longrightarrow 0SAB|1,\ A \longrightarrow 1A|1,\ B \longrightarrow 2B|2\}$

Data la forma delle produzioni,
la lettura del primo simbolo (terminale) può essere usata in modo *predittivo* per decidere il resto della stringa che si dovrà derivare

Forme Normali II

Costruiamo l'automa a pila equivalente:
$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

- $Q = \{q_0\}$
- $\Gamma = \{S, A, B\}$ con $Z_0 = S$
- $F = \emptyset$

pila	stato	0	1	2
S	q_0	(q_0, SAB)	(q_0, ϵ)	
A	q_0		$\{(q_0, A), (q_0, \epsilon)\}$	
B	q_0			$\{(q_0, B), (q_0, \epsilon)\}$

Forme Normali III

Se le produzioni della grammatica sono in una certa forma si può definire facilmente un PDA mediante il seguente risultato:

Teorema

Data $G = (\Sigma, V, S, P)$ *libera con produzioni della forma*

$$A \longrightarrow b\sigma$$

con $b \in \Sigma, \sigma \in V^*$, *esiste un PDA* M, *tale che* $\mathsf{L}(G) = \mathsf{L}(M)$.

Dim. Si consideri $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ con

- $Q = \{q_0\}$
- $\Gamma = V$
- $Z_0 = S$
- $F = \emptyset$
- $\forall A \longrightarrow b\alpha \in P\colon\ (q_0, \alpha) \in \delta(q_0, b, A)$

Forma Normale di Chomsky

Una grammatica libera $G = (\Sigma, V, S, P)$
è in **forma normale di Chomsky**
se ogni produzione è di uno dei tipi seguenti:

1. $S \longrightarrow \epsilon$
2. $A \longrightarrow BC$
   con $A \in V$ e $\begin{cases} B, C \in V \setminus \{S\} & \text{se } S \longrightarrow \epsilon \in P \\ B, C \in V & \text{altrimenti} \end{cases}$
3. $A \longrightarrow a$
   con $A \in V, a \in \Sigma$

Forma Normale NLR

Una grammatica libera $G = (\Sigma, V, S, P)$
è in forma **priva di ricorsioni sinistre** (*NLR*, *No Left Recursion*)
se non ha produzioni del tipo:

$$A \longrightarrow A\beta$$

dove $A \in V$, $\beta \in (V \cup \Sigma)^*$

Forma Normale di Greibach

Una grammatica libera $G = (\Sigma, V, S, P)$
è in **forma normale di Greibach** (*GNF*, *Greibach Normal Form*)
se ogni produzione è del tipo:

1. $S \longrightarrow \epsilon$
2. $A \longrightarrow a\sigma$
   dove $A \in V, \quad a \in \Sigma, \quad \sigma \in V^*$

Passaggio alle Forme Normali

Teorema

Sia G una grammatica libera.
Allora esistono G_i, $i = 1, 2, 3$ equivalenti a G, tali che

- G_1 *è in forma normale di Chomsky*
- G_2 *è in forma normale di Greibach*
- G_3 *è in forma normale NLR priva di ricorsioni sinistre*

Trasformazioni e riduzioni

Tramite opportuni algoritmi,[1] è possibile trasformare una grammatica libera nella equivalente grammatica **ridotta** eliminando

- le ϵ-**produzioni**
- le produzioni **unitarie** (*chain-rules*) $A \longrightarrow B$
- i simboli **inutili** X:
  non generativi $X \not\Longrightarrow^{*} w \in \Sigma^{*}$ o
  non raggiungibili $S \not\Longrightarrow^{*} \alpha X \beta$

[1] Ad es. cfr. [Ausiello *et al.*,(2003)][Hopcroft *et al.*,(2009)].

Sostituzione

Considereremo nel seguito linguaggi privi di ϵ
(si può sempre aggiungere facendola produrre da S)

Teorema (sostituzione)

Sia $G = (\Sigma, V, S, P)$ *libera.*
Se $\exists A \longrightarrow \sigma B \delta \in P$ *con* $\sigma, \delta \in (\Sigma \cup V)^*$
e $\exists B \neq A$: $B \longrightarrow \omega_1 | \cdots | \omega_n \in P$
allora si può costruire $\hat{G} = (\Sigma, V, S, \hat{P})$ *equivalente con:*

$$\hat{P} = P \setminus \{A \longrightarrow \sigma B \delta\} \cup \{A \longrightarrow \sigma \omega_1 \delta | \cdots | \sigma \omega_n \delta\}$$

Esempio $A \longrightarrow 0|00A|01B2$ e $B \longrightarrow 011A|1$
Sostituendo B in A: $A \longrightarrow 0|00A|01011A2|0112$ e $B \longrightarrow 011A|1$

Rimozione delle produzioni inutili I

Esempio $S \longrightarrow 0S1|\epsilon|A$ e $A \longrightarrow 0A$
$S \longrightarrow A$ è inutile perché A non è *generatore* di stringhe terminali

Esempio $S \longrightarrow A,\ A \longrightarrow 0A|\epsilon,\ B \longrightarrow 1A$
B inutile perché non è *raggiungibile* da S pur generando (via A) stringhe di terminali

Definizione (non-terminali utili)

Data $G = (\Sigma, V, S, P)$ libera. Si dirà che $A \in V$ è **utile** sse

$$\exists w \in \mathbf{L}(G):\ S \stackrel{*}{\Longrightarrow} \sigma A \delta \stackrel{*}{\Longrightarrow} w$$

con $\sigma, \delta \in (\Sigma \cup V)^*$.
Si dirà **inutile** un non-terminale che non è utile.

Rimozione delle produzioni inutili II

Esempio $S \longrightarrow 0S|A|C,\ A \longrightarrow 0,\ B \longrightarrow 00,\ C \longrightarrow 0C1$

NT utili perché generatori di stringhe terminali:
A per $A \longrightarrow 0$,
B per $B \longrightarrow 00$,
S per $S \Longrightarrow A \Longrightarrow 0$

Rimuovendo C: $S \longrightarrow 0S|A,\ A \longrightarrow 0,\ B \longrightarrow 00$

Studiamo le dipendenze con un *grafo* di raggiungibilità:

due NT C e D sono collegati da un arco sse $\exists(C \longrightarrow \sigma D\delta) \in P$

Quindi eliminando B rimangono: $S \longrightarrow 0S|A,\ A \longrightarrow 0$

Rimozione delle produzioni inutili III

Teorema (eliminazione non-terminali e prod. inutili)

Sia $G = (\Sigma, V, S, P)$ libera. Esiste $\hat{G} = (\Sigma, \hat{V}, S, \hat{P})$ libera equivalente senza non-terminali e produzioni inutili

Dim.

1. data G, costruire $G' = (\Sigma, V', S, P')$:
   1. $V' \leftarrow \emptyset$
   2. **esegui**
      - **per ogni** $A \in V$ tale che $A \longrightarrow \omega \in P$ con $\omega \in (\Sigma \cup V')^*$:
        $V' \leftarrow V' \cup \{A\}$

      fintantoché V' sia stato modificato
   3. P' prende da P le prod. con simboli in $\Sigma \cup V'$
2. data G', costruire $\hat{G}$:
   1. Si costruisce il grafo delle dipendenze di G'
   2. Si trovano le var. non raggiungibili da S
   3. Si eliminano le loro produzioni

Rimozione delle ϵ-produzioni I

Una produzione

$$A \longrightarrow \epsilon$$

si dice ϵ-**produzione**

Spesso si possono rimuovere
Esempio $S \longrightarrow 0N1,\ N \longrightarrow 0N1|\epsilon$ per generare $\{0^n1^n \mid n > 0\}$
Si rimuove la ϵ-produzione per sostituzione:
$S \longrightarrow 0N1|01,\ N \longrightarrow 0N1|01$

Un non-terminale A tale che

$$A \stackrel{*}{\Longrightarrow} \epsilon$$

si dice **annullabile**

Rimozione delle ϵ-produzioni II

Teorema (eliminazione ϵ-produzioni)

Sia $G = (\Sigma, V, S, P)$ libera tale che $\epsilon \notin G$.
Allora esiste $\hat{G} = (\Sigma, \hat{V}, S, \hat{P})$ libera equivalente senza ϵ-produzioni

Dim.

1. trovare $V_N \subseteq V$ insieme dei non-terminali annullabili in G
   1. **per ogni** produzione $A \longrightarrow \epsilon \in P$:
      $V_N \leftarrow V_N \cup \{A\}$
   2. **ripeti**
      - **per ogni** produzione $B \longrightarrow A_1 \cdots A_k \in P$ con $A_1, \ldots, A_k \in V_N$:
        $V_N \leftarrow V_N \cup \{B\}$

      fino a quando non ci sono più modifiche
2. **per ogni** $p = A \longrightarrow s_1 \cdots s_k \in P$ con $s_1, \ldots, s_k \in V \cup \Sigma$:
   $\hat{P} \leftarrow \hat{P} \cup \{p\} \cup sostNull(p, V_N)$

Rimozione delle ϵ-produzioni III

Osservazione $sostNull(p, V_N)$ restituisce l'insieme delle produzioni ottenute sostituendo le variabili annullabili di V_N in p con ϵ in tutte le combinazioni possibili

- (tranne $A \longrightarrow \epsilon$, nel caso le s_i fossero tutte annullabili)

Esempio data G con produzioni:
$S \longrightarrow AB0C,\ A \longrightarrow BC,\ B \longrightarrow 1|\epsilon,\ C \longrightarrow D|\epsilon,\ D \longrightarrow 2$
dall'algoritmo $V_N = \{A, B, C\}$,
per cui si trasforma in:
$S \longrightarrow AB0C|B0C|A0C|AB0|0C|B0|A0|0$
$A \longrightarrow BC|B|C$
$B \longrightarrow 1$
$C \longrightarrow D$
$D \longrightarrow 2$

Rimozione delle produzioni unitarie I

Una produzione

$$A \longrightarrow B$$

si dice **produzione unitaria**

Si possono rimuovere utilizzando il teorema di sostituzione

Teorema (eliminazione produzioni unitarie)

Sia $G = (\Sigma, V, S, P)$ libera tale priva di senza ϵ-produzioni. Allora esiste $\hat{G} = (\Sigma, \hat{V}, S, \hat{P})$ libera equivalente senza produzioni unitarie

Rimozione delle produzioni unitarie II

Dim. le produzioni $A \longrightarrow A$ si possono eliminare direttamente.

- $\hat{P} \leftarrow \{p \in P \mid p \text{ prod. non unitaria}\}$
- Costruire il grafo delle dipendenze tra NT limitato alle prod. unitarie
  arco da C a D sse $C \longrightarrow D \in P$
- **Per ogni** $\langle A, B \rangle$, tali che $A \stackrel{*}{\Longrightarrow} B$ con $A \neq B$ secondo il grafo:
  - Selezionare le prod. di B in $\hat{P}$:

$$B \longrightarrow \omega_1 | \omega_2 | \cdots | \omega_m \in \hat{P}$$

  - $\hat{P} \leftarrow \hat{P} \cup \{A \longrightarrow \omega_1 | \omega_2 | \cdots | \omega_m\}$

Rimozione delle produzioni unitarie III

Esempio date le prod. $S \longrightarrow A0|B,\ A \longrightarrow 0|12|B,\ B \longrightarrow A|11$

$$S \to B \circlearrowright A$$

da cui: $S \overset{*}{\Longrightarrow} A,\ S \overset{*}{\Longrightarrow} B,\ A \overset{*}{\Longrightarrow} B,\ B \overset{*}{\Longrightarrow} A$

Quindi si inizializza con: $\{S \longrightarrow A0,\ A \longrightarrow 0|12,\ B \longrightarrow 11\}$

Aggiungendo: $S \longrightarrow \overbrace{0}^{A} | \overbrace{12}^{A} | \overbrace{11}^{B},\ A \longrightarrow \overbrace{11}^{B},\ B \longrightarrow \overbrace{0}^{A} | \overbrace{12}^{A}$
si ottiene:

$$S \longrightarrow A0|0|12|11,\ A \longrightarrow 0|12|11,\ B \longrightarrow 11|0|12$$

con B inutile

Rimozione delle produzioni unitarie IV

Osservazioni

- applicare le sostituzioni direttamente alle singole prod. unitarie non funzionerebbe nel caso di dipendenze *cicliche*
  - per esercizio, provare con la gramm. dell'es. precedente
- su alcuni testi si parla di chiusura della rel. di raggiungibilità tramite prod. unitarie U_G
  - $\langle A, B\rangle \in U_G$ sse $A \longrightarrow B$
  - chiusura
    $U_G^* = \{\langle A, B\rangle \mid A \longrightarrow B\} \cup \{\langle A, C\rangle \mid \langle A, B\rangle \in U_G^* \wedge B \longrightarrow C \in P\}$
  - quindi $\langle A, B\rangle \in U_G^*$ sse $A \stackrel{*}{\Longrightarrow} B$

Applicazione delle trasformazioni

Dato che una trasformazione può generare problemi dell'altro tipo, l'ordine consigliato è:

1. Rimozione delle ϵ-produzioni
2. Rimozione produzioni inutili
3. Rimozione delle produzioni unitarie

altre trasformazioni:

- Eliminazione ricorsione sinistra
- Fattorizzazione

Teoremi di Equivalenza

Attraverso le trasformazioni trattate in precedenza

Un linguaggio libero
è sempre generabile da una grammatica libera in GNF

Da ciò discendono:

Teorema

Ogni linguaggio libero è riconosciuto da un automa a pila

Teorema

Ogni linguaggio accettato da un automa a pila è libero

Predicati Decidibili

Sfruttando le argomentazioni del pumping lemma si può determinare, mediante opportuni algoritmi, se una grammatica libera genera un linguaggio *infinito*, *vuoto* o *finito* ma non vuoto [Ausiello *et al.*,(2003)]:

Teorema

Data una grammatica $\boldsymbol{G}$ *di tipo 2 è decidibile stabilire se* $\mathbf{L}(\boldsymbol{G})$ *è infinito.*

Teorema

Data una grammatica $\boldsymbol{G}$ *di tipo 2 è decidibile stabilire se* $\mathbf{L}(\boldsymbol{G}) = \emptyset$.

quindi, per esclusione:

Teorema

Data una grammatica $\boldsymbol{G}$ *di tipo 2 è decidibile stabilire se* $\mathbf{L}(\boldsymbol{G})$ *è finito ma non vuoto.*

Appartenenza ad un linguaggio libero

Teorema

Data una grammatica G di tipo 2 ed una stringa sul suo alfabeto $w \in \Sigma^*$ è decidibile stabilire se $w \in \mathbf{L}(G)$

È questo il lavoro svolto da ogni *parser* per un dato ling. di programmazione: controllare se una stringa (prog. sorgente) è corretta rispetto alle regole della grammatica.

Parsing: Algoritmo CYK I

Data $G = (\Sigma, V, S, P)$ libera in *CNF*, algoritmo di COCKE-YOUNGER-KASAMI per decidere se $z = a_1 \cdots a_n \in \mathbf{L}(G)$

Passi Consideriamo le $v_{i,j} = a_i \cdots a_j$, con $|v_{i,j}| = j - i + 1$:

1. Per ogni $v_{i,i}$: $|v_{i,i}| = 1$ (quindi $v_{i,i} = a_i \in \Sigma$)
   si trova l'insieme $X_{i,i} \in \wp(V)$ dei NT A tali che $A \longrightarrow a_i \in P$
2. Per ogni $v_{i,i+1}$: $|v_{i,i+1}| = 2$
   si trova $X_{i,i+1} \in \wp(V)$ degli A tali che $A \stackrel{*}{\Longrightarrow} v_{i,i+1}$

$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$

n-1. Per le 2 sottostringhe $v_{1,n-1}$ e $v_{2,n}$, $|v_{1,n-1}| = |v_{2,n}| = n - 1$
   si trova $X_{1,n-1} \in \wp(V)$ 8risp. $X_{2,n} \in \wp(V)$) degli A tali che
   $A \stackrel{*}{\Longrightarrow} v_{1,n-1}$ (risp. $A \stackrel{*}{\Longrightarrow} v_{2,n}$)

n. Per $z = v_{1,n}$: $|v_{1,n}| = n$
   si trova $X_{1,n} \in \wp(V)$ degli A tali che $A \stackrel{*}{\Longrightarrow} v_{1,n}$

Parsing: Algoritmo CYK II

$z \in \mathsf{L}(G)$ se $S \in X_{1,n}$

	1	2	3	$\dots$	$n-1$	n
1	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$\cdots$	$X_{1,n-1}$	$X_{1,n}$
2		$X_{2,2}$	$X_{2,3}$	$\cdots$	$X_{2,n-1}$	$X_{2,n}$
3			$X_{3,3}$	$\cdots$	$X_{3,n-1}$	$X_{3,n}$
				$\ddots$	$\vdots$	$\vdots$
$n-1$					$X_{n-1,n-1}$	$X_{n-1,n}$
n						$X_{n,n}$

Parsing: Algoritmo CYK III

Input $G = (\Sigma, V, S, P)$: grammatica
$z = x_1 \cdots x_n$: stringa
Output d: boolean
Var Σ: matrice $n \times n$ triangolare a valori in $\wp(V)$

for each $(i,j) \in \{1,\ldots,n\} \times \{1,\ldots,n\}$ **do** $X_{i,j} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
  for each $A \in V$ **do**
    if $\exists (A \longrightarrow x_i) \in P$ **then** $X_{i,j} \leftarrow X_{i,j} \cup \{A\}$
for $s \leftarrow 2$ **to** n **do**
  for $i \leftarrow 1$ **to** $n - s + 1$ **do**
    for $k \leftarrow i$ **to** $i + s - 2$ **do**
      if $\exists B \in X_{i,k} \wedge \exists C \in X_{k+1,i+s-1} \wedge \exists (A \longrightarrow BC) \in P$
      then $X_{i,i+s-1} \leftarrow X_{i,i+s-1} \cup \{A\}$
return $d \leftarrow (S \in X_{1,n})$

Parsing: Algoritmo CYK IV

Esempio Controllare se $G = (\Sigma, V, S, P)$ con produzioni CNF:

$$\begin{aligned} P = \{ & S \longrightarrow AT \mid AB, \\ & T \longrightarrow XB, \\ & X \longrightarrow AT \mid AB, \\ & A \longrightarrow \mathrm{a}, \\ & B \longrightarrow \mathrm{b} \} \end{aligned}$$

possa generare aaabbb

	1	2	3	4	5	6
1	$\{A\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{S, X\}$
2		$\{A\}$	$\emptyset$	$\emptyset$	$\{S, X\}$	$\{T\}$
3			$\{A\}$	$\{S, X\}$	$\{T\}$	$\emptyset$
4				$\{B\}$	$\emptyset$	$\emptyset$
5					$\{B\}$	$\emptyset$
6						$\{B\}$

Riferimenti

- Ausiello G.; D'Amore F.; Gambosi G. 2003.
  Linguaggi, Modelli, Complessità.
  FrancoAngeli.
- Cohen D. I. 1996.
  Introduction to Computer Theory.
  Wiley.
- Hopcroft J. E.; Motwani R.; Ullman J. D. 2009.
  Automi, Linguaggi e Calcolabilità.
  Pearson Italia, 3a edizione.
- Linz P. 2012.
  An Introduction to Formal Languages and Automata.
  Jones & Bartlett, 5a edizione.
- Moll R. N.; Arbib M. A.; Kfoury A. J. 1988.
  An introduction to formal language theory.
  Springer.
- Sipser 2005.
  Introduction to the theory of computation.
  Thomson, 2a edizione.
- Sudkamp 2006.
  Languages and Machines.
  Addison-Wesley, 3a edizione.