Indice

Capitolo 1

Intro

Gli automi a pila sono automi che oltre ad avere un controllo a stati finiti hanno una memoria arbitrariamente grande, ma organizzata a pila; cioè una memoria a cui si può accedere solo all'elemento più in cima. Questo modello è one-way sul nastro di input, e dimostreremo che la versione two-way è più potente.

Durante il corso analizzeremo principalmente il modello nondeterministico, infatti un altro risultato che dimostriamo è che il modello deterministico è – a differenza degli FSA – meno potente di quello nondeterministico.

Un automa a pila è una tupla

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$$

dove

- Γ è l'alfabeto della pila o alfabeto di lavoro
- δ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale dell'automa
- $Z_0 \in \Gamma$ è lo stato iniziale della pila
- F è un insieme di stati finali

La funzione di transizione dipende da tre cose: dallo stato corrente, dal simbolo dell'input corrente e dal simbolo in cima alla pila

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathrm{PF}(Q \times \Gamma^*)$$

La funzione di transizione contemporaneamente cambia lo stato dell'automa e rimpiazza il simbolo in cima alla pila con una stringa di stati della pila[1] [2] [3].

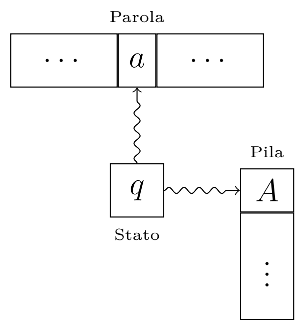
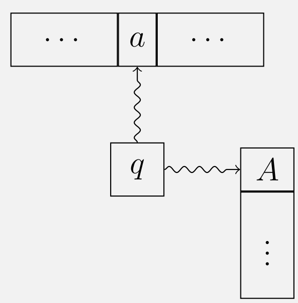


Figura 1.1: Rappresentazione delle varie parti di un PDA

Supponiamo di essere nello stato



e che la funzione δ sia così definita

$$\delta(q, a, A) = \{(q_1, \varepsilon), (q_2, BCC)\}$$

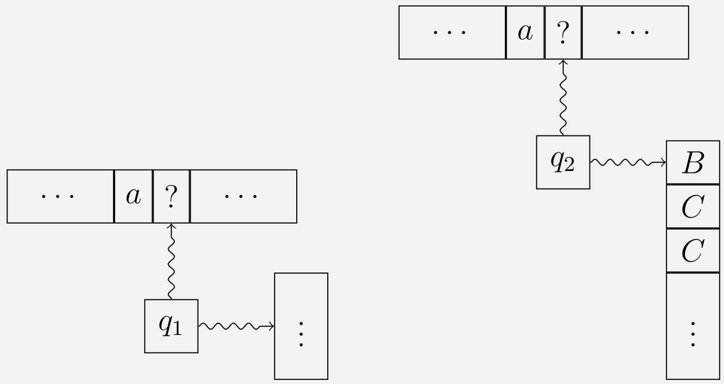
L'applicazione delle due alternative porterebbe l'automa nei seguenti stati

---

[1] $PF(-)$ sta per le parti finite, infatti se utilizzassimo $2^{Q \times \Gamma^*}$ potremmo avere programmi infiniti, visto che Γ^* è un insieme infinito.

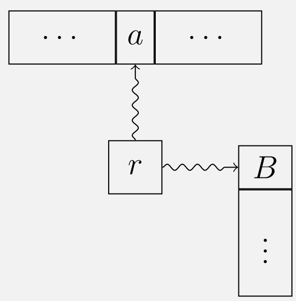
[2] Scriviamo $\Sigma \cup \{\varepsilon\}$ perché sono contemplate mosse in base allo stato dell'automa che modificano la pila senza leggere un simbolo in input.

[3] Per convenzione la stringa di stati viene messa sulla pila da destra a sinistra, quindi il simbolo più a sinistra sarà in cima alla pila.



(a) Lo stato per (q_1, ε) (b) Lo stato per (q_2, BCC)

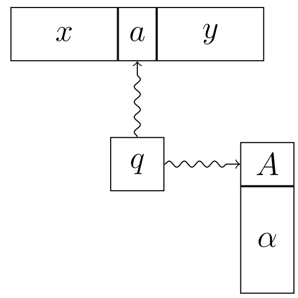
Inoltre potremmo anche avere ε-mosse, ad esempio $\delta(q, \varepsilon, A) = \{(r, B)\}$, porterebbe l'automa nello stato



1.1 Definizioni

Ci riferiremo agli automi a pila come PDA (Push Down Automaton) e assumeremo che siano sempre nondeterministici, a meno che diversamente specificato.

Chiameremo lo stato complessivo dell'automa a pila la sua **configurazione**, questa verrà rappresentata compattamente come la tripla dello stato corrente, la porzione di input ancora da leggere, e il contenuto della pila. Quindi



è rappresentato dalla configurazione

$$(q, ay, A\alpha)$$

con $q \in Q, a \in \Sigma \cup \{\varepsilon\}, y \in \Sigma^*, A \in \Gamma, \alpha \in \Gamma^*$.

Una mossa, scritto $q \vdash p$, indica che da una configurazione q posso passare ad un'altra p. Ad esempio nell'Esempio 1 abbiamo che

$$\begin{aligned}
(q, ay, A\alpha) &\vdash (q_1, y, \alpha) \\
(q, ay, A\alpha) &\vdash (q_2, y, BCC\alpha) \\
(q, ay, A\alpha) &\vdash (r, ay, B\alpha)
\end{aligned}$$

Più rigorosamente, sia $(q, ay, Z\alpha)$ la configurazione corrente, con $Z \in \Gamma$ e M l'automa a pila, diciamo che

$$(q, ay, Z\alpha) \underset{M}{\vdash} (p, y, \beta\alpha)$$

sse $(p, \beta) \in \delta(q, a, Z)$ dove $q, p \in Q$, $y \in \Sigma^*$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$ e $\alpha, \beta \in \Gamma^*$. Se l'automa è ovvio dal contesto possiamo ometterlo da $\underset{M}{\vdash}$ e scrivere solo $\vdash$.

Da una configurazione C' arrivo ad una configurazione C'' in un certo numero di mosse – scritto

$$C' \overset{*}{\underset{M}{\vdash}} C''$$

sse esistono $C_0, \ldots, C_k$ con $C_0 = C'$ e $C_k = C''$ e $\forall i \in 1, \ldots, k$ $C_{i-1} \underset{M}{\vdash} C_i$.

La configurazione iniziale di un automa su input $w \in \Sigma^*$ è

$$(q_0, w, Z_0)$$

Per accettare possiamo dare alcune diverse definizioni di configurazione accettante:

- una volta finito l'input mi trovo in uno stato $q \in F$ e la pila può essere una stringa qualunque, questa è detta *accettazione per stati finali*[4] , ed indichiamo il linguaggio accettato per stati finali dall'automa a pila M come

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \varepsilon, \gamma), q \in F, \gamma \in \Gamma^*\}$$

- è ragionevole pensare che tutto quello che viene messo sulla pila debba anche essere tolto, questa è detta *accettazione per pila vuota* per cui si deve arrivare alla fine dell'input ed aver svuotato l'intera pila, ignorando lo stato. Il linguaggio accettato per pila vuota dall'automa M lo indichiamo come

$$N(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \varepsilon, \varepsilon), q \in Q\}$$

  In questo caso ovviamente si può omettere F dalla definizione dell'automa.

- si può pensare di richiedere entrambe le precedenti, come vedremo più avanti queste tre nozioni sono equivalenti nel caso nondeterministico (Sezione 1.2).

Nota 1. *Questa cosa la vedremo meglio, ma visto che la pila è la struttura fondamentale per la ricorsione, i linguaggi CF sono i linguaggi regolari a cui è stata aggiunta la ricorsione.*

[4]Siccome sono accettate le ε mosse può esserci il caso in arriviamo alla fine dell'input con uno stato non finale, e si può fare una ε-mossa ed arrivare ad uno stato finale.

Definiamo il linguaggio

$$\mathcal{L} = \{a^n b^n \mid n \geq 1\}$$

possiamo usare la pila per contare il numero di a.

$$\delta(q_0, a, Z_0) = \{(q_0, A)\}$$
$$\delta(q_0, a, A) = \{(q_0, AA)\}$$
$$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$$

E vale che data questa δ

$$\mathcal{L} = N(M)$$

Questo caso particolare di automa a pila in cui utilizziamo in simbolo solo (cioè A, oltre a Z_0) è detto *automa a contatore*.
Definiamo alternativamente

$$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$$
$$\delta(q_0, a, A) = \{(q_0, AA)\}$$
$$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, \varepsilon, Z_0) = \{(q_F, \varepsilon)\}$$

con $F = \{q_F\}$, e vale che con questa δ

$$\mathcal{L} = L(M)$$

Vediamo ora il caso di sopra, ma in cui

$$\mathcal{L} = \{a^n b^n \mid n \geq 0\}$$

possiamo usare la pila per contare il numero di a.

$$\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}$$
$$\delta(q_0, a, Z_0) = \{(q_0, A)\}$$
$$\delta(q_0, a, A) = \{(q_0, AA)\}$$
$$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$$

E vale che data questa δ in cui si può direttamente accettare dallo stato q_0

$$\mathcal{L} = N(M)$$

L'introduzione della prima regola è problematica, perché con input non vuoto permette di svuotare la pila da Z_0, bloccando la continuazione dell'automa, quindi abbiamo introdotto il nondeterminismo tra le due

Definiamo ora l'automa a pila deterministico. Questo in ogni configurazione permette una singola scelta:

- sono vietate configurazioni che ammettono una mossa e una ε-mossa, quindi $\forall q \in Q, z \in \Gamma$ se $\delta(q, \varepsilon, z) \neq \varnothing$ allora $\forall a \in \Sigma\ \delta(q, a, Z) = \varnothing$

- per ogni tripletta q, a, Z è ammessa al massimo una mossa, quindi

$$\forall q \in Q, z \in \Gamma, a \in \Sigma \cup \{\varepsilon\} \mid |\delta(q, a, Z)| \leq 1$$

1.2 Equivalenza tra le due nozioni di accettazione nel modello nondeterministico

Dimostriamo ora che le due nozioni di PDA che abbiamo visto nella Sezione 1.1 sono equivalenti.

Da stati finali a pila vuota. Dato un automa $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ e supponiamo che $L = L(M)$ sia il linguaggio accettato per stati finali. Definiamo l'automa

$$M' = (Q \cup \{q_0', q_e\}, \Sigma, \Gamma \cup \{X\}, \delta', q_0', X, \varnothing)$$

con $q_0', q_e \notin Q$ e $X \notin \Gamma$, vogliamo che $L = N(M')$.

Ad alto livello quando M arriva in uno stato finale, M' si sposta nello stato q_e in cui inizia a svuotare la pila. Infatti la e di q_e sta per "empty".

Definiamo ora δ':

1. prima di tutto
$$\delta'(q_0', \varepsilon, X) = \{(q_0, Z_0X)\}$$
questo serve solo ad infilare X in fondo alla pila. La X è necessaria per evitare che se l'automa iniziale M svuota la pila si accetti la stringa.

2. per ogni altra cosa M' si può comportare come M:
$$\forall q \in Q, a \in \Sigma \cup \{\varepsilon\}, z \in \Gamma \mid \delta(q, a, Z) \subseteq \delta'(q, a, Z)$$

3. ogni qualvolta M entra in uno stato finale M' può – enfasi su può – iniziare a svuotare l'intera pila:
$$\forall q \in F, z \in \Gamma \cup \{X\} \mid (q_e, \varepsilon) \in \delta'(q, \varepsilon, Z)$$

4. una volta entrato nello stato di svuotamento, continua a svuotare:

$$\forall z \in \Gamma \cup \{X\} \mid \delta'(q_e, \varepsilon, Z) = \{(q_e, \varepsilon)\}$$

Questo necessariamente introduce nondeterminismo, infatti l'automa M potrebbe entrare in uno stato finale prima di essere arrivato alla fine della stringa. Ed anche se l'automa di partenza è deterministico il punto 3 potrebbe in ogni caso introdurre nondeterminismo.

Supponiamo di avere un automa deterministico che accetta la stringa w a pila vuota, allora ogni stringa che ha w come prefisso non può essere accettata, perché il prefisso w svuoterebbe la pila e un automa con pila vuota non può andare a avanti. Quindi il nondeterminismo è in un certo senso necessario per automi a pila che accettano con pila vuota. ■

Da pila vuota a stati finali. Dato un automa $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$ che accetta per pila vuota il linguaggio $L = N(M)$, vogliamo creare un automa che accetti per stati finali. Sia questo

$$M' = (Q \cup \{q_0', q_F\}, \Sigma, \Gamma \cup \{X\}, \delta', q_0', X, F = \{q_F\})$$

con $q_0', q_F \notin Q, X \notin \Gamma$.

Definiamo ora δ':

- come prima inizialmente infiliamo X in fondo alla pila:

$$\delta'(q_0', \varepsilon, X) = \{(q_0, Z_0X)\}$$

  X serve a riconoscere quando la pila è vuota.

- a questo punto copiamo tutte le mosse di M, per cui

$$\forall q \in Q, a \in \Sigma \cup \{\varepsilon\}, Z \in \Gamma \mid \delta'(q, a, Z) = \delta(q, a, Z)$$

- nel momento in cui M svuota la prima, M' si trova X sulla pila, a questo punto può entrare in uno stato finale

$$\forall q \in Q \mid \delta'(q, \varepsilon, X) = \{(q_F, \varepsilon)\}$$

Supponendo che M sia deterministico, M' rimane deterministico – la trasformazione preserva il determinismo. ■

1.3 Grammatiche di tipo 2

Una grammatica è formata da quattro elementi:

$$G = \langle V, \Sigma, P, S \rangle$$

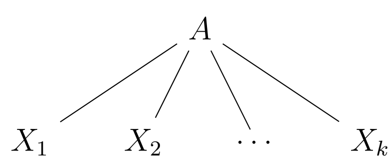
e nello specifico, in quelle di tipo 2 le produzioni hanno la forma

$$A \to \alpha \qquad A \in V, \alpha \in (V \cup \Sigma)^*$$

Una rappresentazione utile per le derivazioni di linguaggi CF sono gli alberi, ad esempio data $w \in L(G)$, allora $S \overset{*}{\Rightarrow} w$. Questa derivazione io la possono rappresentare come un albero di derivazione, o albero di parsing, o ancora parse tree. Questo è un albero

- con radice etichettata con il simbolo iniziale della grammatica
- le foglie da sinistra a destra sono w
- i nodi possono essere di tre tipi:
  - variabili, per i nodi interni
  - terminali, per le foglie
  - ε la parola vuota, in casi speciali per le foglie

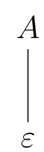
Dato un nodo



rappresenta l'applicazione della regola di produzione

$$A \to X_1 X_2 \dots X_k \in P \qquad A \in V, \forall i \in 1, \dots, k \mid X_i \in V \cup \Sigma$$

All'ultimo livello possiamo avere nodi



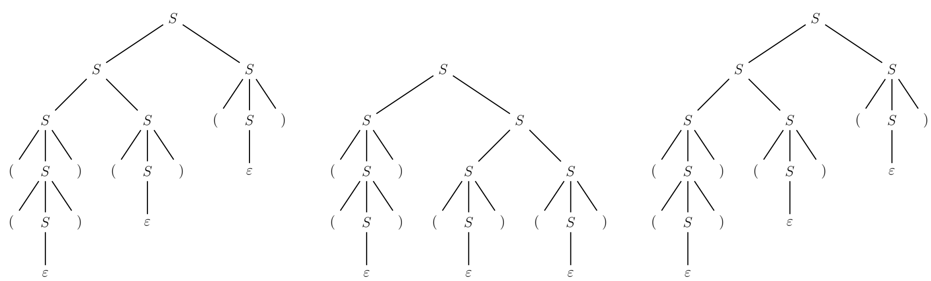
solo se $A \to \varepsilon \in P$.

Abbiamo detto che gli automi a pila riconoscono linguaggi con ricorsione, dove questa nell'automa si esprime nella memoria a pila, nelle grammatiche si esprime nella struttura ad albero.

Definiamo la grammatica per le parentesi correttamente bilanciate

$$\begin{aligned} S &\to \varepsilon \\ S &\to (S) \\ S &\to SS \end{aligned}$$

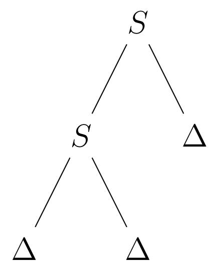
prendiamo ora la stringa $w = (())()()$ e scriviamone la derivazione



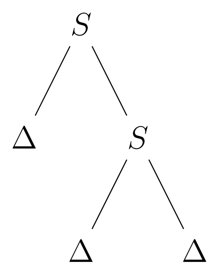
$$\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow SSS \\ &\Rightarrow (S)SS \\ &\Rightarrow (S)(S)S \\ &\Rightarrow (S)(S)(S) \\ &\Rightarrow ((S))(S)(S) \\ &\Rightarrow ((S))(S)() \\ &\Rightarrow ((S))()() \\ &\Rightarrow (())()() \end{aligned}
\qquad
\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow (S)S \\ &\Rightarrow ((S))S \\ &\Rightarrow (())S \\ &\Rightarrow (())SS \\ &\Rightarrow (())(S)S \\ &\Rightarrow (())()S \\ &\Rightarrow (())()(S) \\ &\Rightarrow (())()() \end{aligned}
\qquad
\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow SSS \\ &\Rightarrow (S)SS \\ &\Rightarrow ((S))SS \\ &\Rightarrow (())SS \\ &\Rightarrow (())(S)S \\ &\Rightarrow (())()S \\ &\Rightarrow (())()(S) \\ &\Rightarrow (())()() \end{aligned}$$

Figura 1.3: Tre derivazioni diverse per la stringa $(())()()$ e gli alberi corrispondenti

Possiamo vedere che una stessa stringa ammette diverse derivazioni, ma non tutte queste portano allo stesso albero. Infatti la prima e la terza derivazione utilizzano le stesse sostituzioni, solo in ordine diverso, e quindi generano alberi uguali; mentre nel secondo albero applichiamo derivazioni diverse. Nella prima e nella terza derivazione abbiamo una struttura



mentre la seconda ha una struttura



Per evitare derivazioni multiple si utilizza un criterio detto di *derivazione leftmost*: una derivazione è lefmost se ogni volta che si fa una sostituzione sostituisco sempre la variabile più a sinistra della forma sentenziale.

Proposizione 1. *Esiste una corrispondenza uno a uno tra derivazioni leftmost e alberi di derivazione.*

La seconda e la terza derivazioni dell'esempio di sopra sono due derivazioni leftmost diverse.

Diciamo che una grammatica è *ambigua* se c'è una stringa che ammette almeno due alberi di derivazione – o derivazioni leftmost – diversi.

Nell'esempio di sopra si può vedere anche che ogni sottoalbero è una sequenza bilanciate di parentesi.

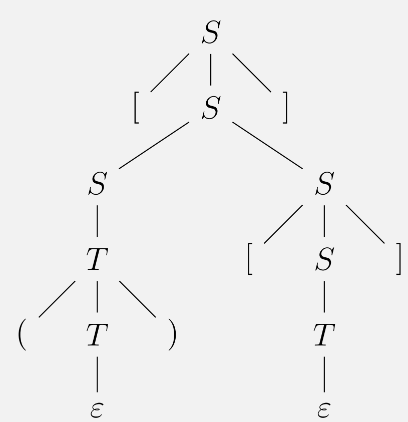
Se nella grammatica di sopra vorremmo anche le quadre, senza precedenze, questa è facilmente

$$
\begin{aligned}
S &\to \varepsilon \\
S &\to (S) \\
S &\to [S] \\
S &\to SS
\end{aligned}
$$

Ma se si chiede che le quadre non possano stare all'interno delle tonde, diventa necessario suddividere le variabili in due livelli

$$
\begin{aligned}
S &\to T \\
S &\to [S] \\
S &\to SS \\
T &\to \varepsilon \\
T &\to (T) \\
T &\to TT
\end{aligned}
$$

e vediamo un albero di derivazione di esempio



1.4 Equivalenza tra grammatiche di tipo 2 ad automi a pila

1.4.1 Da una grammatica che genera un linguaggio generiamo un automa che riconosce lo stesso

Data una grammatica

$$G = \langle V, \Sigma, P, S \rangle$$

di tipo 2, vogliamo costruire

$$M = \langle Q, \Sigma, \Gamma, \delta, q, Z_0, \varnothing \rangle$$

che accetta per pila vuota, con

- Q formato da un solo stato $\{q\}$
- $\Gamma = \Sigma \cup V$
- $Z_0 = S$

e δ definito come

- se $A \to \alpha \in P$ allora $(q, \alpha) \in \delta(q, \varepsilon, A)$
- $\forall \sigma \in \Sigma$, $\delta(q, \sigma, \sigma) = \{(q, \varepsilon)\}$, cioè si consuma il simbolo in cima alla pila

Si può dimostrare che il linguaggio generato dalla grammatica $L(G)$ è uguale al linguaggio accettato dall'automa per pila vuota $N(M)$.

Prendiamo

$$G = \langle \{S, T, U\}, \{a, b\}, P, S \rangle$$

con P definito

$$\begin{aligned} S &\to TU \\ T &\to aTb \mid \varepsilon \\ U &\to bUa \mid \varepsilon \end{aligned}$$

questo genera

$$L = \{a^n b^{n+m} a^m \mid n \geq 0, m \geq 0\}$$

Scriviamo le transizioni dell'automa corrispondente

$$M = \langle \{q\}, \{a, b\}, \{S, T, U, a, b\}, \delta, q, S, \varnothing \rangle$$

con δ definito come

$$\begin{aligned}
\delta(q, \varepsilon, S) &= \{(q, TU)\} \\
\delta(q, \varepsilon, T) &= \{(q, aTb), (q, \varepsilon)\} \\
\delta(q, \varepsilon, U) &= \{(q, bUa), (q, \varepsilon)\} \\
\delta(q, a, a) &= \{(q, \varepsilon)\} \\
\delta(q, b, b) &= \{(q, \varepsilon)\}
\end{aligned}$$

Prendendo per esempio $w = abbbaa$, vediamo come viene accettata nondeterministicamente

$$
\begin{aligned}
(q, abbbaa, S) &\vdash (q, abbbaa, TU) \\
&\vdash (q, abbbaa, TU) \\
&\vdash (q, abbbaa, aTbU) \\
&\vdash (q, bbbaa, TbU) \\
&\vdash (q, bbbaa, bU) \\
&\vdash (q, bbaa, U) \\
&\vdash (q, bbaa, bUa) \\
&\vdash (q, baa, Ua) \\
&\vdash (q, baa, bUaa) \\
&\vdash (q, aa, Uaa) \\
&\vdash (q, aa, aa) \\
&\vdash (q, a, a) \\
&\vdash (q, \varepsilon, \varepsilon)
\end{aligned}
$$

Leggere la i terminali consumati fino a un certo punto e il contentuto della pila in quel punto restituisce la forma sentenziale durante la derivazione. Questo corrisponde a

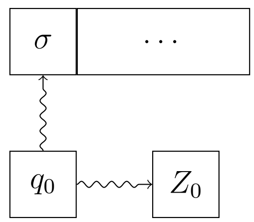
$$
\begin{aligned}
S &\Rightarrow TU \\
&\Rightarrow aTbU \\
&\Rightarrow abU \\
&\Rightarrow abbUa \\
&\Rightarrow abbbUaa \\
&\Rightarrow abbbaa
\end{aligned}
$$

L'automa a pila tenta di simulare il processo di derivazione leftmost della stringa.

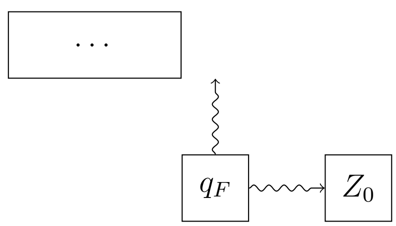
1.4.2 Da un automa a pila costruiamo una grammatica di tipo 2

Per la dimostrazione useremo una variazione degli automi a pila che non ne cambia la potenza computazionale. In questa forma normale

- all'inizio la pila contiene un simbolo speciale Z_0 che viene mai rimosso e non viene mai aggiunto



- alla fine l'input è stato letto completamente, la pila contiene solo Z_0 e lo stato è finale.



- le mosse sulla pila possono essere solo
  - push di un simbolo
  - pop di un simbolo
  - pila invariata

  quindi il pop non è più implicito

- se una mossa legge un simbolo da input, allora non modifica la pila. Cioè le mosse che manipolano la pila sono seperate da quelle che manipolano l'input.

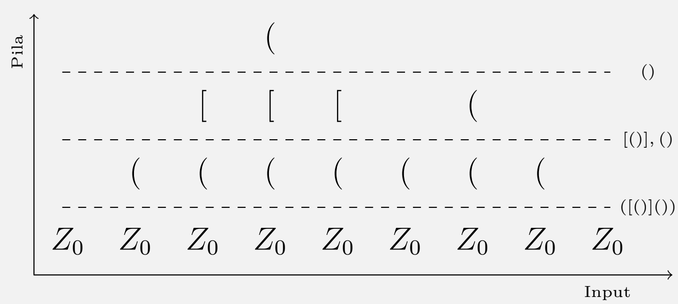
In questa forma

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \{-, \text{pop}, a \in \Gamma | \text{push}(A)\}}$$

ed abbiamo che le mosse possono avere le seguenti forme

- mosse di lettura: $(p, -) \in \delta(q, a, A) \qquad a \in \Sigma \cup \{\varepsilon\}$
- pop: $(p, \text{pop}) \in \delta(q, \varepsilon, A)$
- push: $(p, \text{push}(B)) \in \delta(q, \varepsilon, A)$
- mosse che lasciano la pila invariata: $(p, -) \in \delta(q, \varepsilon, A)$

Ad esempio se avessimo una sequenza di parentesi ([()]()), la pila contiene inizialmente Z_0



questo disegno mostra la natura ricorsiva degli automi. Infatti visto che la pila di questo automa non può mai scendere sotto il suo livello iniziale, tutte le evoluzioni definite dalle linee tratteggiate definiscono parole valide del linguaggio.

Nota 2. *Gli automi che abbiamo visto fino ad ora possono essere simulati da questa versione normalizzata, scomponendo una mossa una pop ed una serie di push utilizzando degli stati ausiliari.*

Ripetiamo la versione di automa a pila semplificato vista a lezione scorsa, in questo per riuscire ad accettare dobbiamo arrivare in uno stato finale con solo Z_0 lo stato finale sulla pila.

Dobbiamo trovare un modo di trasformare un automa a pila come definito nella lezione scorsa, in una grammatica. Questa grammatica ha nonterminali della forma $[qAp]$ con $q, p \in Q$ e $A \in V$, e rappresenta:

- q è lo stato in cui si inizia
- p è lo stato in cui si finisce
- e A è il simbolo in cima alla pila all'inizio e alla fine della computazione.

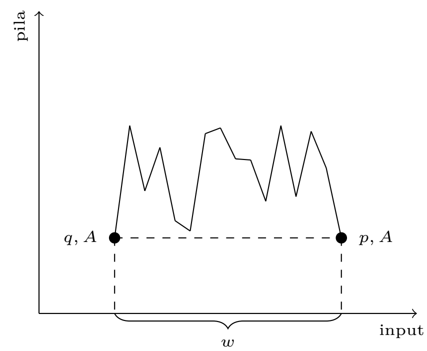


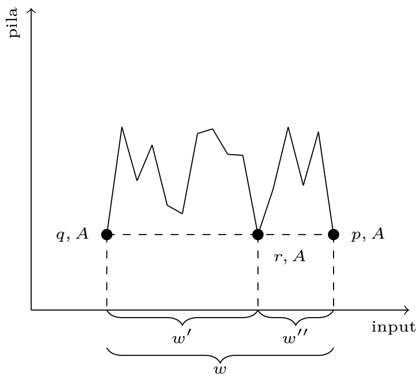
Figura 1.4: La computazione rappresentata dal simbolo $[qAp]$

Infatti negli automi come li abbiamo definiti, vale la proprietà per cui ???

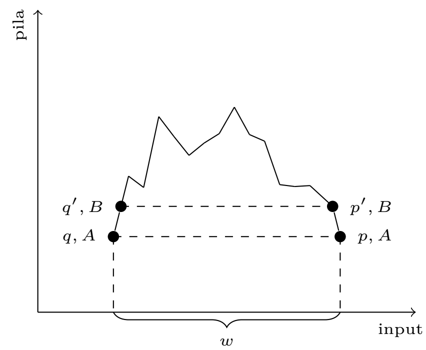
Definiamo ora le regole di produzione della grammatica induttivamente come le stringhe riconosciute dalla computazione $[qAp]$:

- base: abbiamo due casi
  - caso 0: il caso più semplice è $[qAq]$, qui l'unica parola riconosciuta è ε, quindi è necessaria la regola
  $$[qAq] \to \varepsilon$$
  Quindi creo tutte le produzioni della forma
  $$\forall q \in Q, A \in \Gamma \mid [qAq] \to \varepsilon$$
  - caso $0'$: il secondo caso più semplice è quello in qui si è nello stato q, si consuma un carattere o nessuno, e questo ci porta nello stato p; cioè $(p, -) \in \delta(q, a, A)$ con $a \in \Sigma \cup \{\varepsilon\}$. Questo si traduce nella produzione
  $$[qAp] \to a, \qquad a \in \Sigma \cup \{\varepsilon\}$$
  Quindi creo tutte le produzioni della forma
  $$\forall q, p \in Q, A \in \Gamma, a \in \Sigma \cup \{\varepsilon\} \mid [qAp] \to a$$

- passo: si distinguono due casi



(a) Caso 2



(b) Caso 1

  - caso 1: nei passi intermedi (tranne l'ultimo) la pila è sempre strettamente più alta di quando si è iniziato, questo si traduce in
  $$[qAp] \to [q'Bp']$$
  con $(q', \text{push}(B)) \in \delta(q, \varepsilon, A)$ e $(p, \text{pop}) \in \delta(p', \varepsilon, B)$. Quindi
  $$\begin{aligned} &\forall q, q', p, p' \in Q, A, B \in \Gamma \\ &\qquad \mid (q', \text{push}(B)) \in \delta(q, \varepsilon, A) \wedge (p, \text{pop}) \in \delta(p', \varepsilon, B) \\ &\qquad\qquad \Rightarrow [qAp] \to [q'Bp'] \end{aligned}$$

  - caso 2: la computazione $[qAp]$ svuota la pila fino alla A inizia e poi continua, allora possiamo scomporre la computazione in due parti, quindi
  $$\forall q, p, r \in Q, A \in \Gamma \mid [qAp] \to [qAr][rAp]$$

Si può dimostrare che

Lemma 1. $\forall q, p \in Q, A \in \Gamma, w \in \Sigma^* \mid [qAp] \overset{*}{\Rightarrow} w$ *sse l'automa M in una configurazione con A in cima alla pila, stato q, dopo aver letto w raggiunge una configurazione in cui il contentuto della pila è lo stesso dell'inizio, lo stato è p e nei passi intermedi la pila non scende mai sotto il livello iniziale.*

Quindi durante una computazione quello che c'è sotto al simbolo in cima alla pina all'inizio della computazione non è rilevante.

Per finire di costruire la grammatica manca di definire l'assioma. Prima di tutto si può notare che visto che l'automa parte nello stato q_0 con Z_0 e basta sulla pila, una stringa w può essere generata solo dalle triple

$$[q_0 Z_0 q_F] \overset{*}{\Rightarrow} w$$

con q_0 iniziale e q_F finale. Quindi definiamo l'insieme dei nonterminali della grammatica V come l'insieme di tutte le triple definite induttivamente sopra unito ad un nuovo nonterminale S tale che

$$\forall q_F \in F \mid S \to [q_0 Z_0 q_F] \in P$$

e questo S così definito è il simbolo iniziale.

1.5 Forme normali per le grammatiche di tipo 2

Si può vedere che tutte le produzioni generate dalla traduzione da automa a pila a grammatica sono di pochi tipi: variabile a terminale, variabile a variabile e variabile a coppia di variabili. Da questo fatto e dal fatto che i linguaggi riconosciuti dagli automi a pila sono esattamente quelli generati dalle grammatiche di tipo 2, ci rendiamo conto che possiamo restringere di molto il tipo di forma che il lato destro di una produzione di una grammatica CF può assumere. Nel caso di sopra appunto da variabile a terminale, da variabile a variabile e da variabile a coppia di variabili.

Vediamo ora due forme normali. Ogni grammatica può essere trasformata in una di queste forme normali a patto di sacrificare la parola vuota.

1.5.1 Forma normale di Greibach

In una grammatica in FNG (Forma Normale di Greibach) tutte le produzioi sono della forma

$$A \to aB_1 \dots B_k, \qquad a \in \Sigma, A, B_1, \dots, B_k \in V, k \geq 0$$

Supponiamo di avere la gramamtica

$$
\begin{aligned}
A &\to aBB \\
A &\to b \\
B &\to bB \\
B &\to b
\end{aligned}
$$

e di aver fatto la trasformazione in automa a pila. In questa forma normale la pila avrà in cima sempre un terminale e quindi si può avere un simbolo di lookahead e scegliere più precisamente la prossima produzione da utilizzare, anche se non si toglie il nondeterminismo (v. $B \to bB$ e $B \to b$). Un altro vantaggio di avere sempre un terminale in cima alla pila è che in questo tipo di automa si possono eliminare le ε-mosse.

1.5.2 Forma normale di Chomsky

Nella FNC (Forma Normale di Chomsky) ci sono solo due tipi di regole

$$
\begin{aligned}
&A \to BC && A, B, C \in V \\
&A \to a && A \in V, a \in \Sigma
\end{aligned}
$$

Questa genera alberi di derivazione binari, salvo sulle foglie; ed è comoda per studiare alcune proprietà combinatorie.

Trasformazione in FNC

Data una grammatica genererica G eseguiamo i seguenti passi (l'ordine è importante) per trasformarla in FNC:

1. eliminazione delle ε-produzioni: diciamo che una variabile A è *cancellabile* sse $A \overset{*}{\Rightarrow} \varepsilon$. Induttivamente A è cancellabile se

   - banalmente $A \to \varepsilon$
   - o se $A \to X_1 X_2 \ldots X_k$ e $X_1, X_2, \ldots, X_k$ sono tutti cancellabili.

   Questo può essere definito come una chiusura dove

   $$C_0 = \{A \mid A \to \varepsilon\}$$

e

$$C_i = C_{i-1} \cup \{A \mid \exists A \to X_1X_2\dots X_k \text{ con } \forall i \in 1,\dots,k \mid X_i \in C_{i-1}\}$$

Visto che

$$C_0 \subseteq C_1 \subseteq \cdots \subseteq V$$

e V è finito, allora esiste un i tale che $C_i = C_{i-1}$.

Ora sia C l'insisme delle variabili cancellabili, costruiamo una grammatica $G' = \langle V, \Sigma, P', S\rangle$ con P' costituito da tutte le produzioni di P eccetto le ε produzioni e per ogni produzione $A \to X_1X_2\dots X_k$ con $X_1, X_2, \dots, X_k \in V \cup \Sigma$ aggiungo a P' le produzioni $A \to X_{i_1}X_{i_2}\dots X_{i_j}$ tali che $1 \leq i_1 < i_2 < \cdots < i_j \leq k$ e per $\forall X_l \notin X_{i_1}, \dots, X_{i_j} \mid X_l \in C$ e $j \geq 1$.

Supponiamo di avere nella gramamtica G che vogliamo trasformare la produzione

$$A \to BCaD$$

e che l'insieme delle variabili cancellabili è $C = \{C, D\}$.
Nella mia gramamtica G' simulo la cancellazione di C e D aggiungendo le produzioni

$$A \to BaD$$
$$A \to BCa$$
$$A \to Ba$$

Supponiamo di avere nella gramamtica G che vogliamo trasformare la produzone

$$A \to CDE$$

e che l'insieme delle variabili cancellabili è $C = \{C, D, E\}$.
Nella mia gramamtica G' simulo la cancellazione di C e D, quindi aggiungo le produzioni

$$\begin{aligned} A &\to CD \\ A &\to CE \\ A &\to DE \\ A &\to C \\ A &\to D \\ A &\to E \end{aligned}$$

Visto che le produzioni da aggiungere sotto tutti i sottoinsiemi delle variabili cancellabili di un lato destro meno l'insieme vuoto, vengono aggiunge nel caso peggiore un numero esponenziale di produzioni.

2. eliminazione delle produzioni unitarie: una produzione unitaria è una produzione della forma

$$A \to B, \qquad A, B \in V$$

Costruiamo similmente a prima l'insieme di tutte le coppie di variabili X, Y tali per cui $X \overset{+}{\Rightarrow} Y$, cioè per cui vale Abbiamo quindi

$$X \to A_1 \to \cdots \to Y$$

questo processo infatti le catene sono di lunghezza al più $|V|$ senza contenere cicli.

Nella nuova grammatica tolgo tutte le produzioni unitarie e se $X \to \cdots \to Y \to \alpha$ e $\alpha \in \Sigma$ oppure $|\alpha| > 1$, allora aggiungo la produzione $X \to \alpha$.

3. eliminazione simboli inutili: $X \in V \cup \Sigma$ è utile sse $\exists S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w \in \Sigma^*$. Questi sono eliminati utilizzando algoritmi sui grafi (chiusura bottom up, chiusura top down, non lo ha spiegato ma ci sono negli appunti del Santini).

4. eliminazione dei terminali: in tutte le produzioni $A \to \alpha$ con $|\alpha| > 1$ si introducono nonterminali per ogni terminale.

   Supponiamo di avere le produzioni

   $$\begin{aligned} A &\to AaabC \\ A &\to bC \\ A &\to bb \end{aligned}$$

   introduciamo i nonterminali X_a e X_b e le regole

   $$\begin{aligned} A &\to AX_aX_aX_bC \\ A &\to X_bC \\ A &\to X_bX_b \\ X_a &\to a \\ X_b &\to b \end{aligned}$$

5. binarizzazione delle produzioni: per ogni produzione $A \to B_1B_2 \dots B_k$ con $k > 2$, si introducono delle produzioni intermedie

   $$\begin{aligned} A &\to B_1Z_1 \\ Z_1 &\to B_2Z_2 \\ &\vdots \\ Z_{k-2} &\to B_{k-1}B_k \end{aligned}$$

Date le produzioni

$$
\begin{aligned}
S &\to aB \\
S &\to bA \\
A &\to a \\
A &\to aS \\
A &\to bAA \\
B &\to b \\
B &\to bS \\
B &\to aBB
\end{aligned}
$$

questa è già priva di ε-produzioni, produzioni unitarie e tutti i simboli sono utili.
Ora eliminiamo i terminali e otteniamo

$$
\begin{aligned}
S &\to X_aB \\
S &\to X_bA \\
A &\to a \\
A &\to X_aS \\
A &\to X_bAA \\
B &\to b \\
B &\to X_bS \\
B &\to X_aBB \\
X_a &\to a \\
X_b &\to b
\end{aligned}
$$

ed ora binarizziamo le produzioni

$$
\begin{aligned}
S &\to X_a B \\
S &\to X_b A \\
A &\to a \\
A &\to X_a S \\
A &\to X_b E_1 \\
E &\to AA \\
B &\to b \\
B &\to X_b S \\
B &\to X_a E_2 \\
E_2 &\to BB \\
X_a &\to a \\
X_b &\to b
\end{aligned}
$$

1.6 Appartenenza ai Context Free di un linguaggio

Dato un linguaggio ci possiamo chiedere se questo sia CF. Ad esempio

$$L = \{a^l b^k c^j \mid k = j\} \overset{?}{\in} \text{CF}$$

Un linguaggio è CF se possiamo costruire una grammatica di tipo 2 o un automa a pila. Ad esempio per il linguaggio di sopra possiamo consumare tutte le a e controllare che il numero di b e di c sia uguale con una pila.

Prendiamo invece

$$L = \{a^i b^k c^j \mid i = j = j\} \overset{?}{\in} \text{CF}$$

L'automa per il linguaggio di prima non può essere adattato a questo linguaggio.

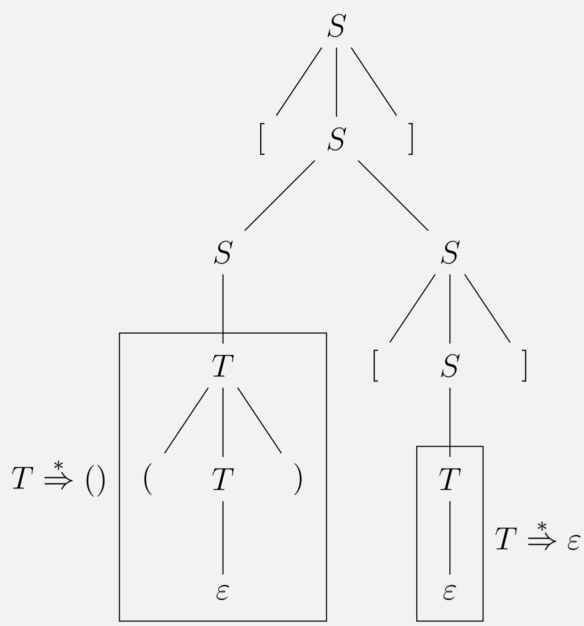
Prendiamo la grammatica

$$S \to [S] \mid SS \mid T$$
$$T \to (T) \mid TT \mid \varepsilon$$

e una derivazione

$$S \overset{*}{\Rightarrow} [()[]]$$

Ora un albero di derivazione che possiamo fare per questa stringa è



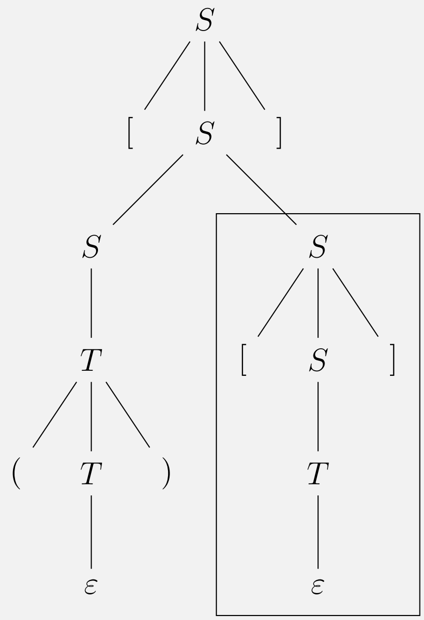
sugli alberi di derivazione si possono fare operazione di sostituzione di sottoalberi, ad esempio nell'albero di sopra sostituenzo il sottoalbero 2 con il sottoalbero 1 otteniamo l'albero di derivazione per $[()[()]]$. In generale quello di sopra è un particolare albero che è rappresentato dalla derivazione

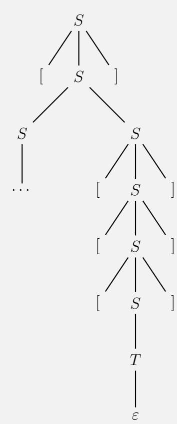
$$A \overset{*}{\Rightarrow} vAx, \qquad v, x \in \Sigma^*$$

questi sono interessanti perché possiamo ???. Ad esempio nell'albero prima abbiamo

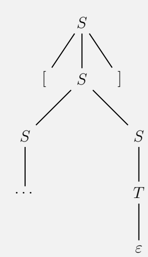
$$S \overset{*}{\Rightarrow} [()S]$$

possiamo vedere che possiamo sia accorciare la derivazione





(a) Allungamento (uv^2wx^2z)



(b) Accorciamento (uwz)

In questo modo possiamo generare infinite stringhe.

Dal fatto che le grammatiche possono essere convertite in FNC (a patto di sacrificare la parola vuota), lavoreremo con grammatiche in FNC per semplificare la dimostrazione del pumping lemma.

Definiamo la *profondità* (o altezza) di un albero come il più lungo cammino dalla radice ad una foglia.

Lemma 2. *Sia*

$$G = \langle V, \Sigma, P, S\rangle$$

una grammatica in FNC e sia $T : A \overset{*}{\Rightarrow} w \in \Sigma^*$ *un albero di derivazione di altezza* h*. Allora la lunghezza di* w *è minore o uguale a* 2^{h-1}.

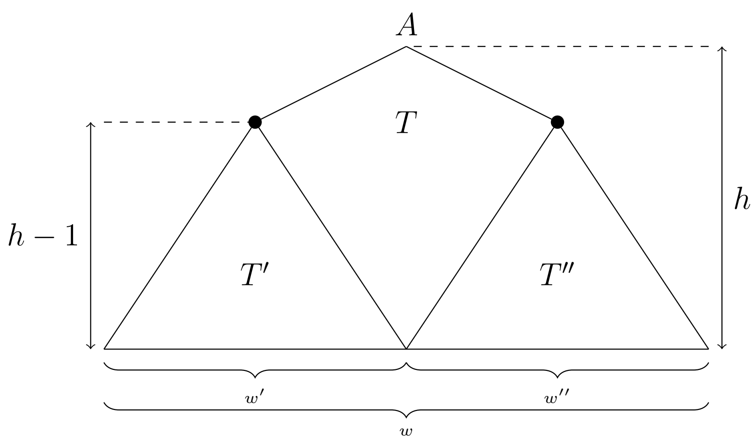
$$|w| \leq 2^{h-1}$$

Dimostrazione. Procediamo per induzione su h:

- per $h = 1$: per forza l'albero deve rappresentare una produzione della forma $A \to a \in \Sigma$, quindi $w = a$ e

$$|w| = 1 = 2^0 = 2^{1-1}$$

- la produzione applicata alla radice deve per forza essere della forma $A \to BC$, quindi l'albero si divide in due sottoalberi, un albero T' : $B \overset{*}{\Rightarrow} w'$ e un albero T'' : $C \overset{*}{\Rightarrow} w''$.



Questi due hanno altezza minore o uguale ad $h - 1$. Ora applicando l'ipotesi induttiva

$$|w'| \leq 2^{h-2}$$
$$|w''| \leq 2^{h-2}$$

e

$$|w| = |w'| + |w''| \leq 2^{h-2} + 2^{h-2} = 2^{h-1}$$

$\blacksquare$

Lemma 3 (Pumping lemma per CFL)**.** *Sia L un linguagio CF allora $\exists N > 0$ tale che $\forall z \in L$ con $|z| \geq N$, questa può essere scomposta*

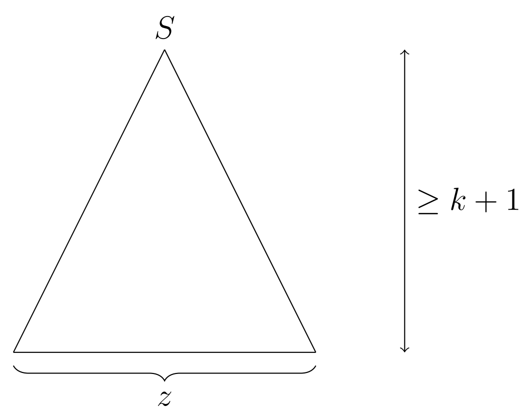
$$z = uvwxy$$

tali che

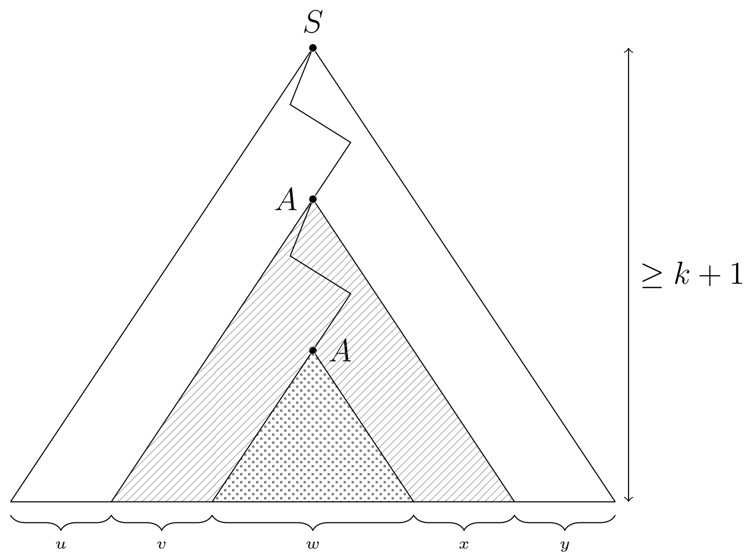
1. $|vwx| \leq N$
2. $vx \neq \varepsilon$
3. $\forall i \geq 0 \mid uv^i wx^i y \in L$

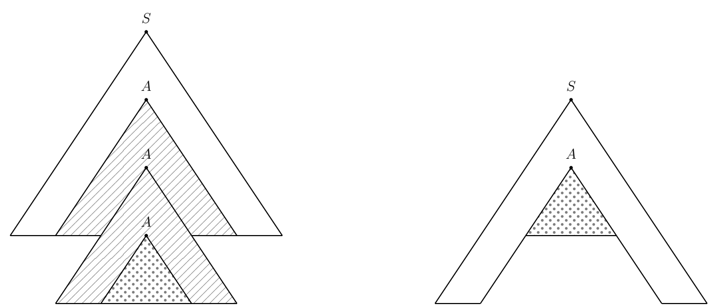
Dimostrazione. Sia $G = \langle V, \Sigma, P, S \rangle$ una grammatica in FNC per $L \setminus \{\varepsilon\}$. Sia $k = |V|$ e definiamo $N = 2^k$.

Sia $z \in L$ con $|z| \geq N$, allora ha un albero di derivazione $T : S \overset{*}{\Rightarrow} z$



visto che la lunghezza di z è maggiore di 2^k, allora dal lemma precedente abbiamo che l'altezza di T è almeno $k+1$, quindi esiste un cammino dalle foglie alle radici da $k+1$ archi, quindi $k+2$ nodi. Visto che l'ultimo nodo è un terminale, durante questo cammino incontreremo $k+1$ non terminali, e quindi almeno un non terminale si ripeterà in questo percorso. Sia A questo non terminale.





(a) Allungamento (b) Accorciamento

Per questo non terminale A vale

$$A \stackrel{*}{\Rightarrow} w$$
$$A \stackrel{*}{\Rightarrow} vAx$$
$$S \stackrel{*}{\Rightarrow} uAy$$

è facile vedere che

$$S \stackrel{*}{\Rightarrow} uAy \stackrel{*}{\Rightarrow} uvAxy \stackrel{*}{\Rightarrow} \dots \stackrel{*}{\Rightarrow} uv^iAx^iy \stackrel{*}{\Rightarrow} uv^iwx^iy$$

quindi abbiamo dimostrato il punto 3.

La produzione centrale di A deve essere per forza della forma $A \to BC$, supponiamo che C sia il non terminale sul percorso più lungo che genera

wx, allora visto che siamo in FNC e non possiamo generare la parola vuota, allora per forza B genera qualcosa diverso da ε, quindi abbiamo dimostrato il punto 2.

L'altezza della parte dell'albero che genera vwx è al massimo $k+1$, cioè il numero massimo di nodi che possiamo vedere prima di trovare una ripetizione. Quindi utilizzando ancora il lemma di sopra, $|vwx| \leq N$. ■

Riprendiamo il linguaggio di prima

$$L = \{a^n b^n c^n \mid c \geq 0\}$$

mostriamo che non soddifsa il pumping lemma.
Supponiamo per assurdo che L sia CF e mostriamo che non può esistere una costante N per cui valga il pumping lemma. Sia N la costante di L, prendiamo

$$z = a^N b^N c^N = uvwxy$$

Visto che per la prima condizione $|vwx| \leq N$, vwx potrà contenere solo due dei tre simboli, più precisamente $vwx \in a^*b^*$ o $vwx \in b^*c^*$. Supponiamo che $vwx \in a^*b^*$, prendiamo $i = 0$ e la stringa $z' = uwy$, questa per la condizione 3 dovrebbe essere in L. Calcoliamo ora le occorrenze dei simboli in z':

$$\begin{aligned} \#_c(z') &= N \\ \#_a(z') + \#_b(z') &= 2N - (\#_a(vx) + \#_b(vx)) \\ &\leq 2N \qquad \text{Per la condizione 2 } (\#_a(vx) + \#_b(vx)) \geq 1 \end{aligned}$$

e quindi $z' = a^k b^j c^N \notin L$ con $k, j < N$, quindi abbiamo un assurdo.

Prendiamo il linguaggio

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

questo non è CF.
Mostriamolo ancora attraverso il pumping lemma. Sia N la costante del pumping lemma, scegliamo la stringa

$$z = a^N b^N a^N b^N \in L = uvwxy$$

Utilizznado ancora la condizione 1 abbiamo due casi

- $vwx \in a^*b^*$, prendiamo la stringa $z' = uwx$, questa dovrebbe essere in L per la condizione 3. Questa è $z' = a^N b^{N'} a^{N''} b^N$, con $N' \leq N, N'' \leq N$, ora possono essere diminuite solo le a, solo le b o entrambe, ma in ogni caso $z' \notin L$.

- $vwx \in b^*a^*$, questo a sua volta dsi divide in due casi, in base al fatto che vwx sia nella prima o nella seconda parte della stringa. Prendendo ancora $i = 0$, $z' = a^{N'} b^{N''} a^N b^N$, con $N' \leq N$ e $N'' \leq N$, con ancora almeno uno tra N' e N'' minore o uguale a N.

Prendiamo il linguaggio

$$L = \{a^h b^j a^k \mid j = \max(h, k)\}$$

supponiamo sia CF e chiamiamo N la costante del pumping lemma. Prendiamo la stringa

$$z = a^N b^N a^N \in L = uvwxy$$

Anche qui ci sono due casi

- $vwx \in a^*b^*$, sappiamo che $vx \neq \varepsilon$, distinguiamo tre casi
  - $vx \in a^+$, prendendo $i = 2$, otteniamo $z' = a^{N'} b^N a^N$, con $N' > N$, che non fa parte di L
  - $vx \in b^+$, prendendo $i = 0$, otteniamo $z' = a^N b^{N'} a^N$, con $N' < N$, che non fa parte di L
  - $vwx \in a^+b^+$, prendendo $i = 0$, otteniamo $z' = a^{N'} b^{N''} a^N$, con $N'' < N$, che non fa parte di L
- $vwx \in b^*a^*$, questo caso è simmetrco al precedente

Prendiamo il linguaggio

$$L = \{a^n b^n c^l \mid k \neq n\}$$

è un linguaggio che rispetta il pumping lemma, ma non è CF.

Bisogna trovare un i tale per cui

$$m + (i-1)(l+r)$$

per cui la somma non sia un numero primo. Scegliendo $i = m+1$, allora

$$m + m(l+r) = m(l+r+1)$$

non è primo.

Nota 3. *Se l'alfabeto è di una lettera sola, non c'è differenza tra regolari e context free.*

Sia

$$\mathcal{L} = \{a^n b^n c^k \mid k \neq n\}$$

Intuitivamente non è CF, infatti posso usare una pila per confrontare le a e le b, ma una volta fatto questo ho perso l'informazione su n.
Mostriamolo con il pumping lemma. Scegliamo fissiamo la costante N e una scegliamo una stringa

$$z = a^m b^m c^j = uvwxy \in \mathcal{L}\text{t.c.}|z| \geq N$$

quindi $2m + j \geq N$.
Analizziamo la composizione di vwx:

- $vwx \in a^+$, questo è facilmente risolvibile mostrando che se si aumentano o diminuiscono le a il loro numero diventa diverso da quello delle b
- $vwx \in b^+$, è analogo al caso di sopra
- $vwx \in c^+$, se $j = 1$ allora facilmente possiamo fissare una i tale che rende il numero delle c uguale a m Alternativamente

possiamo mostrare un caso non valido anche con la stringa

$$z = a^{N+N!}b^{N+N!}c^{N}$$

se assumiamo che $|vx| = k$, in

$$uv^iwx^iy = a^{N+N!}b^{N+N!}c^{N+k(i-1)}$$

e $0 < k \leq N$, vogliamo che $k(i-1) = N!$, quindi scegliamo $i - 1 = \frac{N!}{k}$, cioè

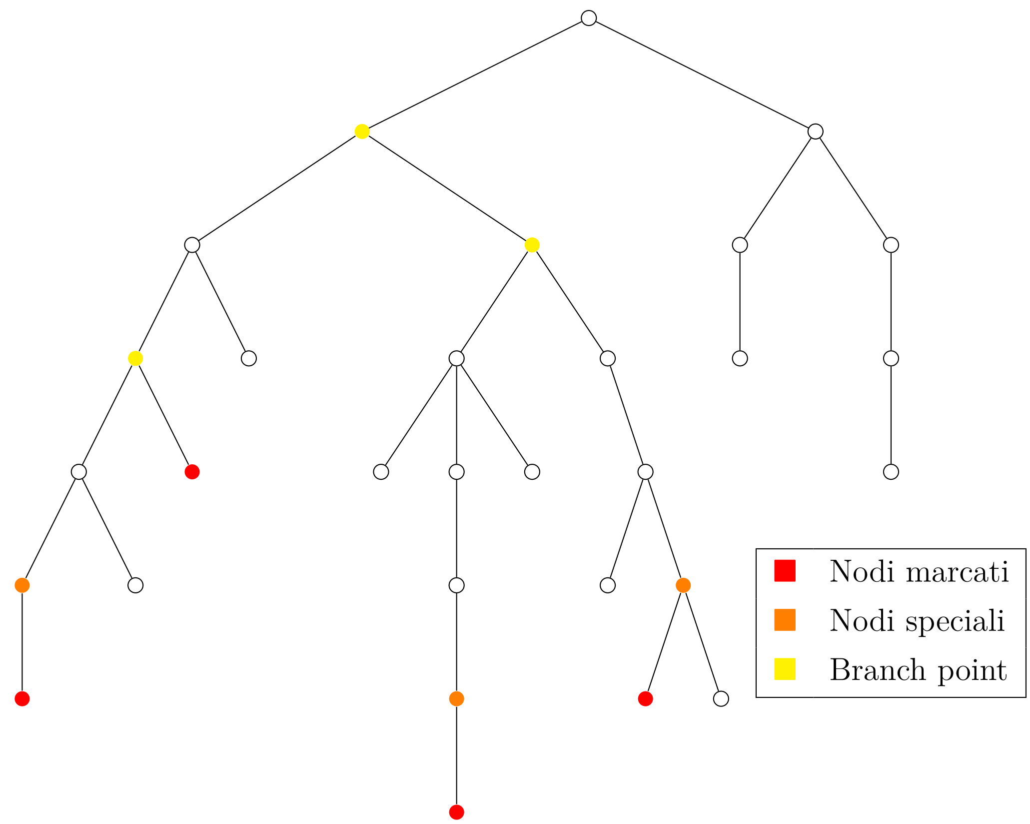
$$i = 1 + \frac{N!}{k}$$

- $vwx \in a^+b^+$, questo a sua volta si divide in vari sottocasi
  - $v \in a^+b^+$, cioè il confine tra a e b cade in v, è facile mostrare che v^i sarebbe una stringa composta da a seguite da b seguite ancora da a e così via
  - $w \in a^+b^+$, allora $v \in a^*$ e $x \in b^*$ abbiamo ancora altri casi
    - se v e x sono di dimensione diversa è facile
    - se v e x sono di dimensione uguale, cioè

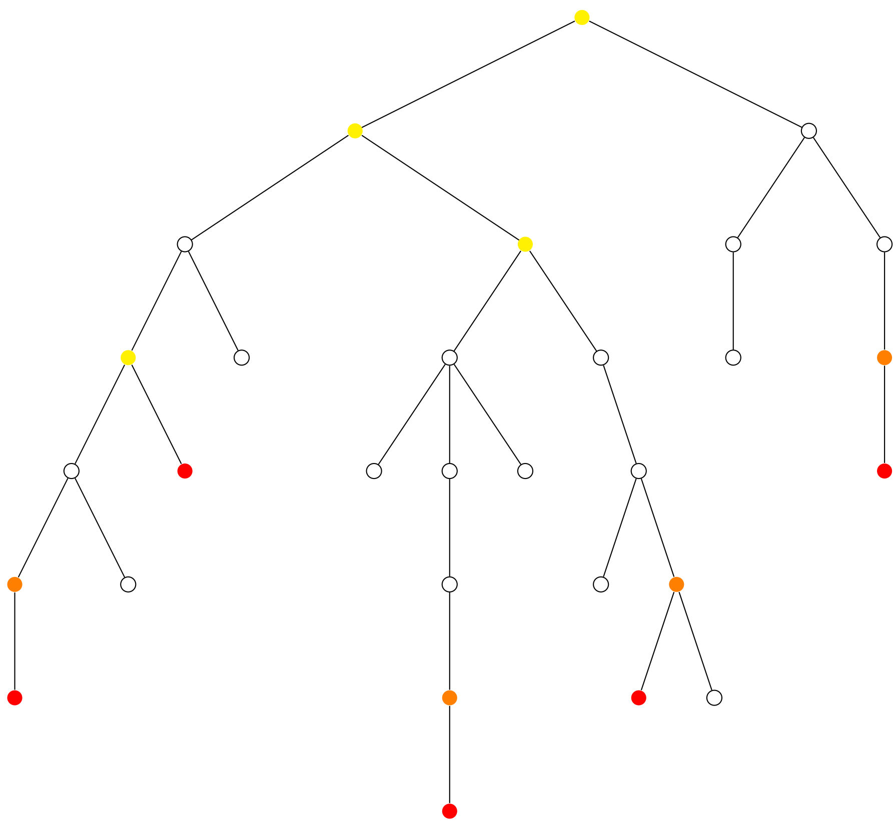
$$v = a^h, b = b^h$$

In questo esempio il pumping lemma non si può applicare.

Sia T un albero con alcune foglie marcate. E definiamo dei nodi interni speciali detti *branch point* definiti come nodi che hanno almeno due figli marcati o due figli con discendenti marcati. Definiamo i *nodi speciali* come tutti i branch point e i nodi che hanno almeno un figlio marcato.



supponiamo di marcare un altro nodo



Lemma 4. *Sia $G = \langle V, \Sigma, P, S \rangle$ una grammatica in FNC e prendiamo un albero di derivazione*

$$A \overset{*}{\Rightarrow} w, \qquad A \in V, w \in \Sigma^*$$

e w contiene alcune posizioni marcate.

Se il numero massimo di nodi speciali su un cammino dalla radice alle foglie è minore o uguale a k, allora il numero di posizioni marcate in w è $\leq 2^k - 1$.

Questo è una generalizzazione del lemma della lezione precedente, nell'altro supponevamo marcata l'intera stringa.

Dimostrazione. Procediamo per induzione su k

- $k = 1$, quindi in ogni cammino dalla radice ad una foglia esiste al massimo un nodo speciale. Supponiamo che questo nodo speciale sia un branch point ma questo non può valere visto che in FNC l'unica produzione che può generare un terminale è della forma $C \Rightarrow a$, quindi anche questi due dovrebbero essere nodi speciali. Quindi esiste una singola foglia marcata, infatti se ne esistesse più di una, allora il loro nodo in comune sarebbe un branch point.

- supponiamo che sia vero per valori $< k$, e mostriamo che è vero per k. Fermiamoci al primo nodo speciale dalla radice, e che questo sia un branch point, per ipotesi induttiva nell'abero di sinistra e di destra ci saranno al massimo 2^{k-2} posizioni marcate. Inoltre z_0 e z_3 non possono avere posizioni marcate, altrimenti il branch point sarebbe più in alto. Quindi $\leq 2^{k-2} + 2^{k-2} = 2^{k-1}$.

■

Lemma 5 (Lemma di Ogden)**.** *Sia $L \in$ CF, allora esiste una costante N tale che per ogni $z \in L$ sono marcate N posizioni e possiamo scomporre z tale che*

$$z = uvwxy$$

tale che

- *vx contiene almeno una posizione marcata*
- *vwx contiene al più N posizioni marcate*
- *per ogni $i > 0$, $uv^iwx^iy \in L$*

Quindi il pumping lemma è un caso speciale di questo in cui ogni posizione è marcata.

Dimostrazione. Sia $G = \langle V, \Sigma, S, P \rangle$ una grammatica in FNC, definiamo $k = |V|$ e fissiamo $N = 2^k$. Prendiamo una stringa $z \in L$ con almeno N nodi marcati. Prendiamo il cammino dalle foglie alla radice che contiene il maggior numero di nodi speciali. In base al lemma di prima, il massimo numero di nodi speciali sul cammino è $\geq k + 1$. Percorrendo il cammino e leggendo le variabili che compaiono sui nodi speciali, sia questa A, troveremo almeno una ripetizione. I sottoalberi di questa coppia di definisce le nostre parti u, v, w, x e y.

Visto che la prima occorrenza di A sarà un branch point, questa sarà una produzione del tipo $A \rightarrow BC$, uno dei due rami potrerà alla seconda A e sicuramente il secondo porterà ad una foglia marcata, quindi in v e x c'è almeno un branch point. ■

Con questo nuovo lemma proviamo a dimostrare l'esempio di prima

Sia

$$L = \{a^n b^n c^k \mid k \neq n\}$$

con N costante per L. La stringa con cui si può arrivare ad un assurdo è

$$z = a^N b^N c^{N+N!} = uvwxy$$

con tutte le a marchiate.
Visto che vx deve contenere almeno una posizione marcata, allora questo deve avere almeno una a al suo interno. Questo ci restringe ai tre casi

- $vwx \in a^+$, il numero di a cresce, ma non il numero di b
- $vwx \in a^+b^+$, si divide in alcuni sottocasi in base a dove il confine tra le a e b cade
  - $v \in a^+b^+$, ripetere v fa sì che la stringa perda la struttura e porta ad avere a dopo le b
  - analogo a sopra se il confine si trova su x
  - $v \in a^+$ e $x \in b^+$, supponiamo più precisamente che

    $$v = a^l, x = b^r$$

    * se $l \neq r$ è ovvio che la stringa non sia valida, anche solo per $i = 0$ cancellerei un numero diverso di a e di b
    * se $l = r$, la stringa che otterremmo ripetendo i volte sarebbe

      $$a^{N+l(i-1)} b^{N+l(r-1)} c^{N+N!}$$

      se si scegle $i = 1 + \frac{N!}{l}$ allora otteniamo che le a e le b sono uguali al numero di c e $N!$ è sicuramente divisibile da l.
- $vwx \in a^+b^Nc^+$,
  - se v e x contengono tipi di lettere diverse, come prima ripetendo si perde la struttura
  - per il resto si tratta in maniera analogo a il secondo caso

Sia

$$\mathcal{L} = \{a^p b^q c^r \mid p = q \vee q = r\}$$

e

$$\mathcal{L} = \{a^p b^q c^r \mid p = q \otimes q = r\}$$