Indice

Capitolo 1

Intro

Gli automi a pila sono automi che, oltre ad avere un controllo a stati finiti, hanno una memoria arbitrariamente grande organizzata – appunto – pila; cioè una memoria a cui si può accedere solo all'elemento più in cima. Il modello di cui tratteremo principalmente è quello non-deterministico e one-way sul nastro di input. Infatti due risultati che mostreremo sono che:

- la versione two-way è più potente (Sezione 2.7);
- il modello deterministico è – a differenza degli FSA – meno potente di quello nondeterministico (Sezione **??**).

Definizione 1 (Automa a pila)**.** 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].

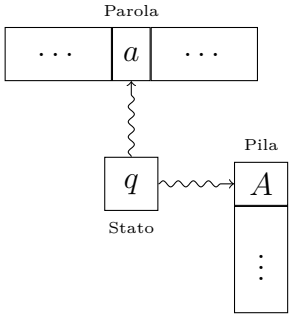
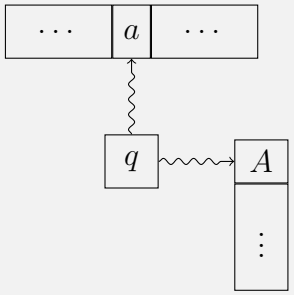


Figura 1.1: Rappresentazione delle varie parti di un PDA

Supponiamo di essere nello stato



e che la funzione δ sia così definita

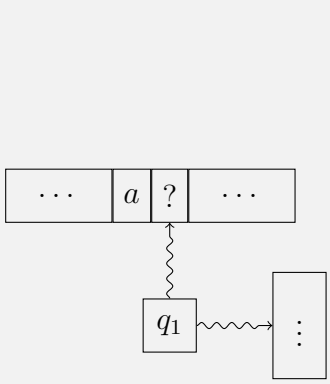
$$\delta(q, a, A) = \{(q_1, \varepsilon), (q_2, BCC)\}$$

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

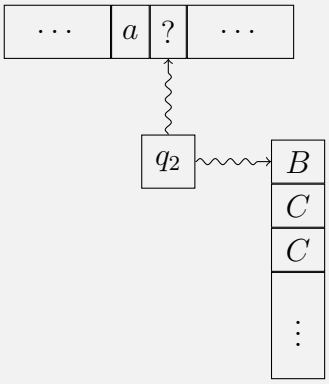
[2] Scriviamo $\Sigma \cup \{\varepsilon\}$ perché sono contemplate mosse in base allo stato dell'automa che modificano la pila senza leggere un simbolo in input.

[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.

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

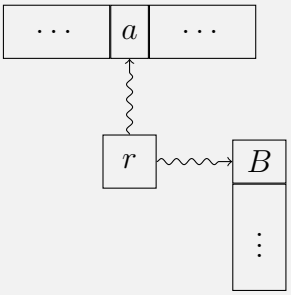


(a) Lo stato per (q_1, ε)



(b) Lo stato per (q_2, BCC)

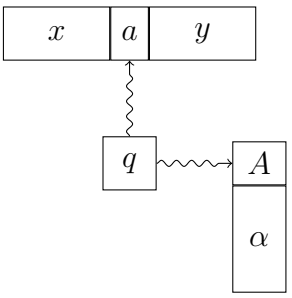
Inoltre potremmo anche avere ε-mosse, ad esempio $\delta(q, \varepsilon, A) = \{(r, B)\}$, porterebbe l'automa nello stato



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

$$(q, ay, A\alpha) \vdash (q_1, y, \alpha)$$
$$(q, ay, A\alpha) \vdash (q_2, y, BCC\alpha)$$
$$(q, ay, A\alpha) \vdash (r, ay, B\alpha)$$

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.

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

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

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

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.

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

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. ■

Capitolo 2

Grammatiche di tipo 2

Una grammatica è formata da quattro elementi:

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

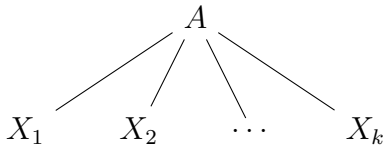
e nello specifico, in quelle di tipo 2 le produzioni hanno la forma

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

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

$$\begin{array}{c} A \\ | \\ \varepsilon \end{array}$$

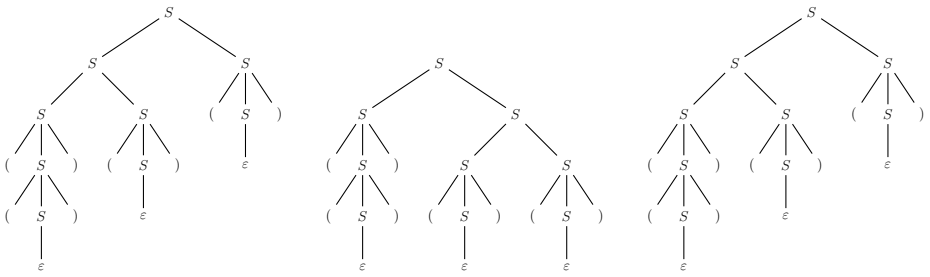
solo se $A \to \varepsilon \in P$.

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

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

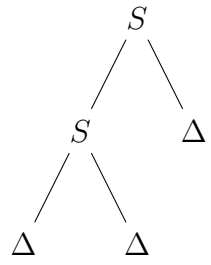
prendiamo ora la stringa $w = (())()()$ e scriviamone la derivazione



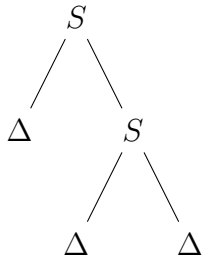
$$
\begin{aligned}
S &\Rightarrow SS \\
&\Rightarrow SSS \\
&\Rightarrow (S)SS \\
&\Rightarrow (S)(S)S \\
&\Rightarrow (S)(S)(S) \\
&\Rightarrow ((S))(S)(S) \\
&\Rightarrow ((S))(S)() \\
&\Rightarrow ((S))()() \\
&\Rightarrow (())()()
\end{aligned}
\qquad
\begin{aligned}
S &\Rightarrow SS \\
&\Rightarrow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

Figura 2.1: 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.

Definizione 2. 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.

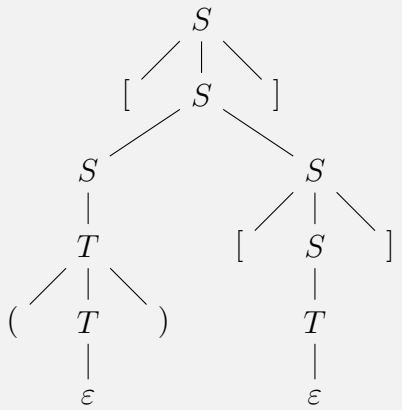
Se nella grammatica di sopra vorremmo anche le quadre, senza precedenze, questa è facilmente

$$S \to \varepsilon$$
$$S \to (S)$$
$$S \to [S]$$
$$S \to SS$$

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

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

e vediamo un albero di derivazione di esempio



2.1 Equivalenza tra grammatiche di tipo 2 ad automi a pila

Mostriamo ora l'equivalenza tra le grammatiche di tipo 2 e gli automi a pila.

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 contenuto della pila in quel punto restituisce la forma sentenziale durante la derivazione. Questo corrisponde a

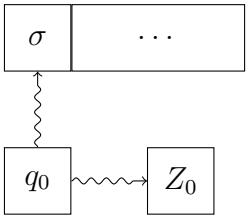
$$\begin{aligned}S &\Rightarrow TU\\ &\Rightarrow aTbU\\ &\Rightarrow abU\\ &\Rightarrow abbUa\\ &\Rightarrow abbbUaa\\ &\Rightarrow abbbaa\end{aligned}$$

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

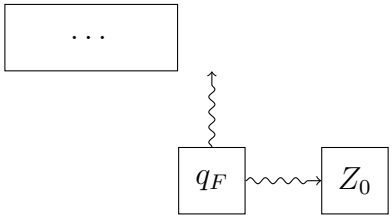
Mostriamo ora il lato opposto dell'equivalenza, per fare questo però Per la

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.

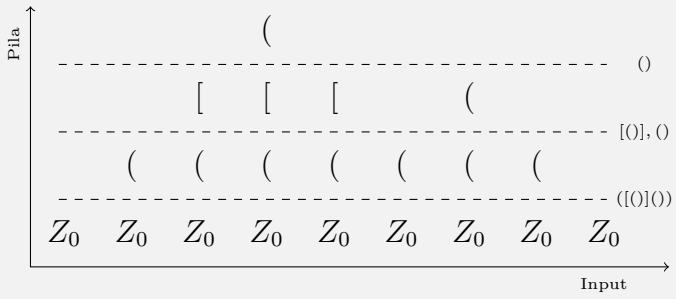
In questa forma

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

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.

Da un automa a pila costruiamo una grammatica di tipo 2. ■

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.

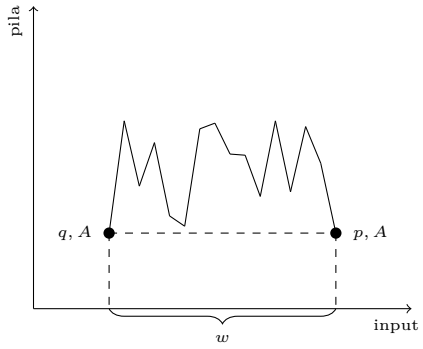


Figura 2.2: La computazione rappresentata dal simbolo $[qAp]$

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

    $$\forall q \in Q, A \in \Gamma \mid [qAq] \to \varepsilon$$

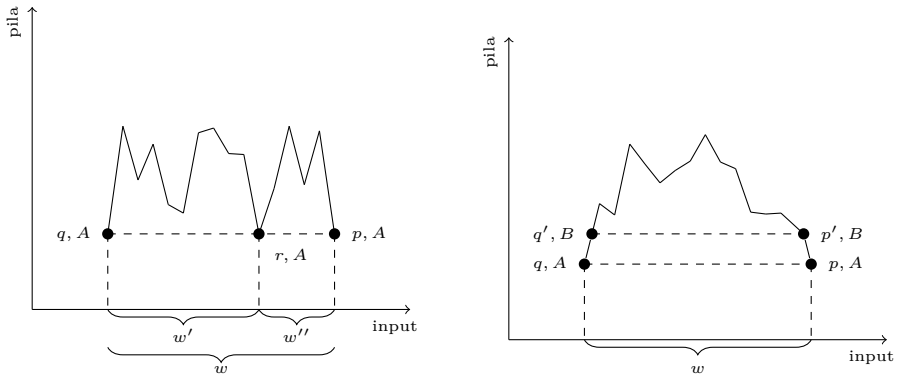
  - caso $0'$: il secondo caso più semplice è quello in qui si è nello stato q, si consuma un carattere o nessuno, e questo ci porta nello stato p; cioè $(p, -) \in \delta(q, a, A)$ con $a \in \Sigma \cup \{\varepsilon\}$. Questo si traduce nella produzione

    $$[qAp] \to a, \qquad a \in \Sigma \cup \{\varepsilon\}$$

    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

$$\forall q, q', p, p' \in Q, A, B \in \Gamma \mid (q', \text{push}(B)) \in \delta(q, \varepsilon, A) \land (p, \text{pop}) \in \delta(p', \varepsilon, B) \Rightarrow [qAp] \to [q'Bp']$$

  - 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

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

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.

2.2 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.

2.2.1 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

$$A \to aBB$$
$$A \to b$$
$$B \to bB$$
$$B \to b$$

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.

2.2.2 Forma normale di Chomsky

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

$$A \to BC \qquad A, B, C \in V$$
$$A \to a \qquad A \in V, a \in \Sigma$$

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 $A \to X_1 X_2 \dots X_k$ e $X_1, X_2, \dots, X_k$ sono tutti cancellabili.

   Questo può essere definito come una chiusura dove

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

e

$$C_i = C_{i-1} \cup \{A \mid \exists A \to X_1 X_2 \dots X_k \text{ con } \forall i \in 1, \dots, k \mid X_i \in C_{i-1}\}$$

Visto che

$$C_0 \subseteq C_1 \subseteq \dots \subseteq V$$

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 $A \to X_1 X_2 \dots X_k$ 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 $1 \leq i_1 < i_2 < \dots < i_j \leq k$ 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

$$A \to CD$$
$$A \to CE$$
$$A \to DE$$
$$A \to C$$
$$A \to D$$
$$A \to E$$

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 $\exists S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w \in \Sigma^*$. 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

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

2.3 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.

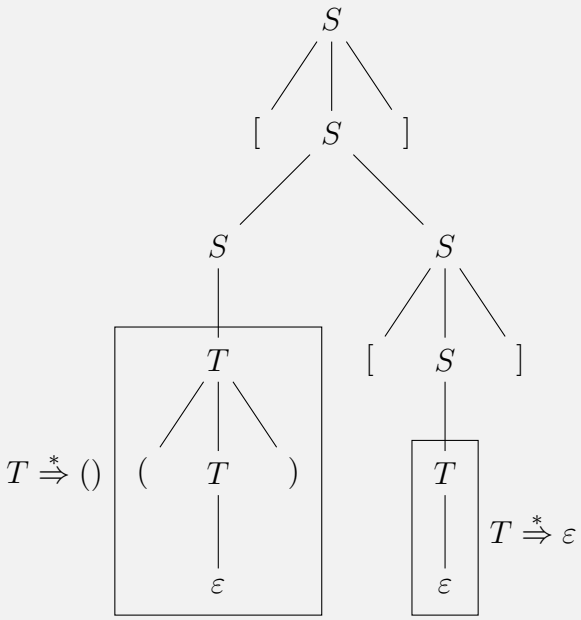
Prendiamo la grammatica

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

e una derivazione

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

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



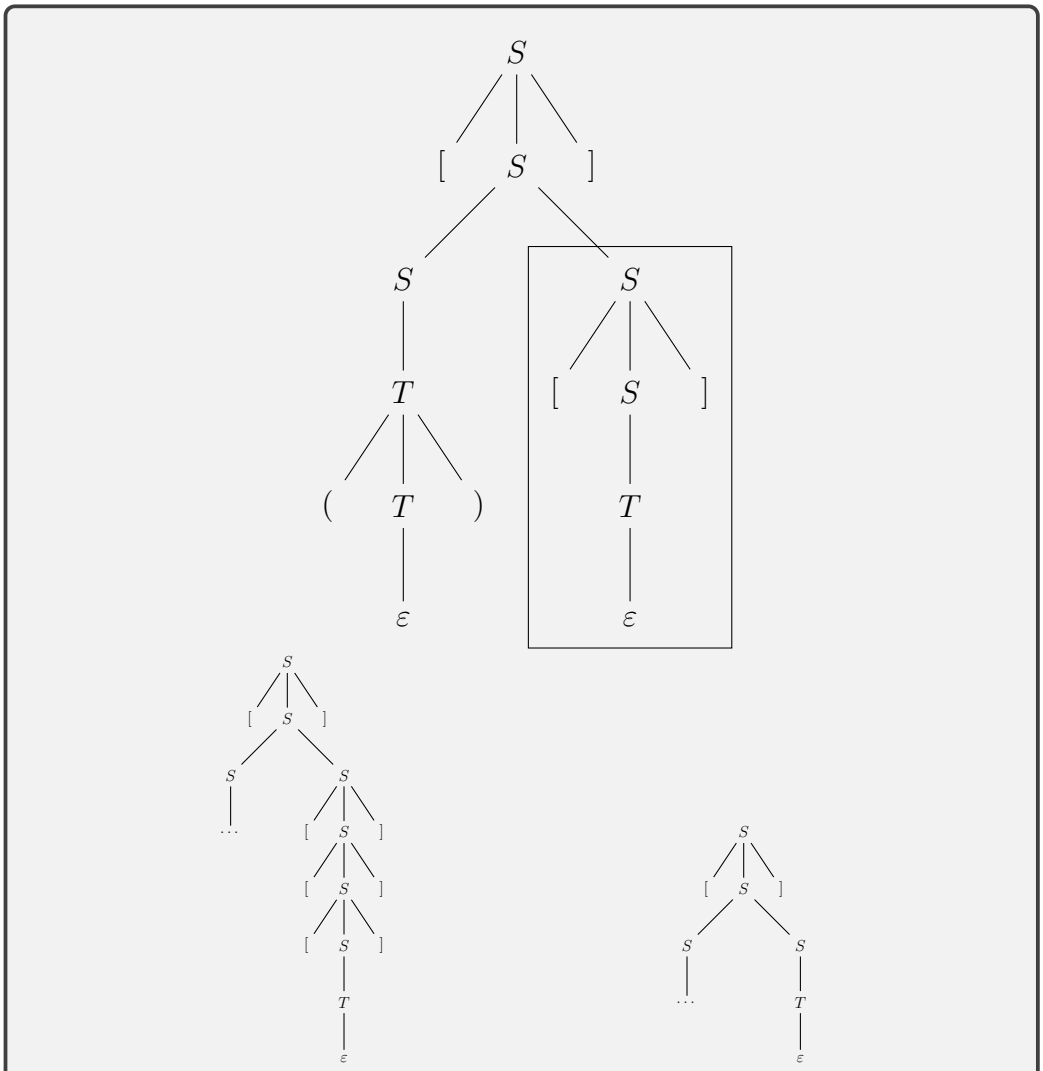
sugli alberi di derivazione si possono fare operazione di sostituzione di sottoalberi, ad esempio nell'albero di sopra sostituenzo il sottoalbero 2 con il sottoalbero 1 otteniamo l'albero di derivazione per [()[()]]. In generale quello di sopra è un particolare albero che è rappresentato dalla derivazione

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

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.

2.3.1 Pumping lemma

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}*.*

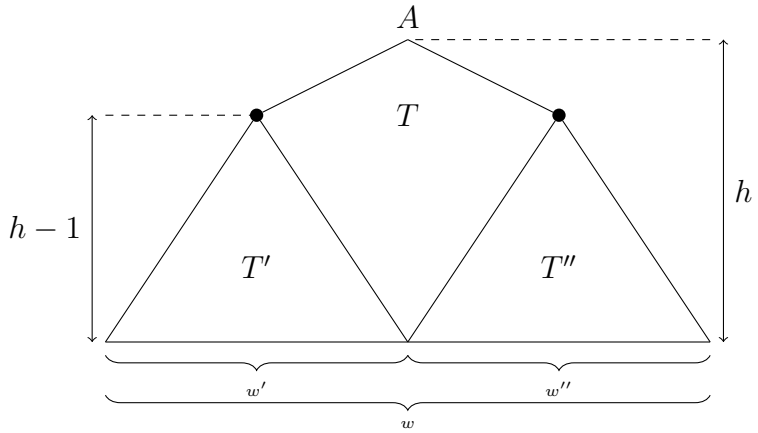
$$|w| \leq 2^{h-1}$$

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

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

■

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*

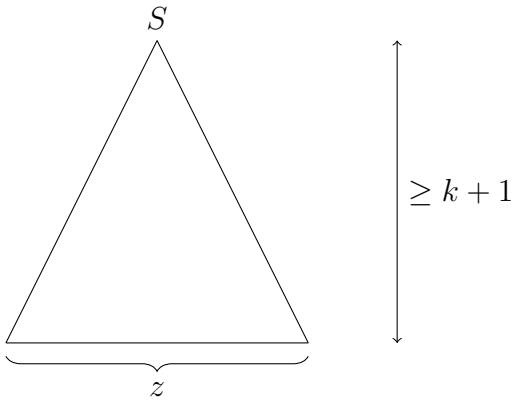
$$z = uvwxy$$

tali che

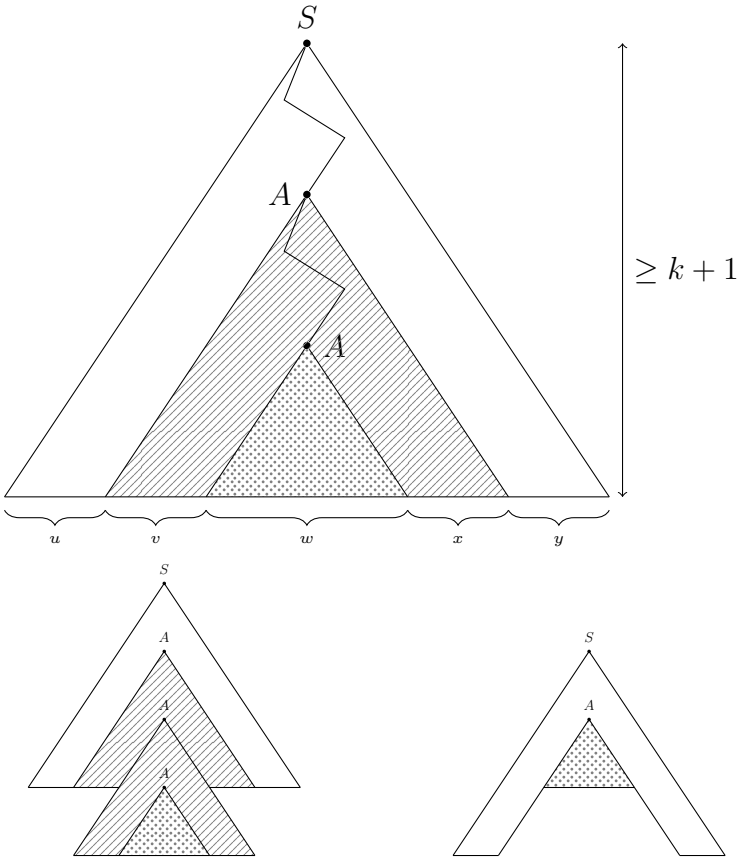
1. $|vwx| \leq N$
2. $vx \neq \varepsilon$
3. $\forall i \geq 0 \mid uv^i wx^i y \in L$

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

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

è facile vedere che

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} \ldots \overset{*}{\Rightarrow} uv^iAx^iy \overset{*}{\Rightarrow} uv^iwx^iy$$

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.

2.3.2 Lemma di Odgen

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

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

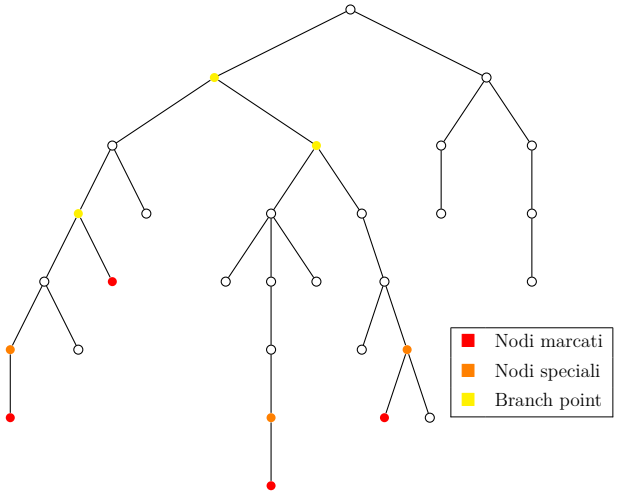
- $vwx \in a^+b^+$, questo a sua volta si divide in vari sottocasi

  - $v \in a^+b^+$, 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

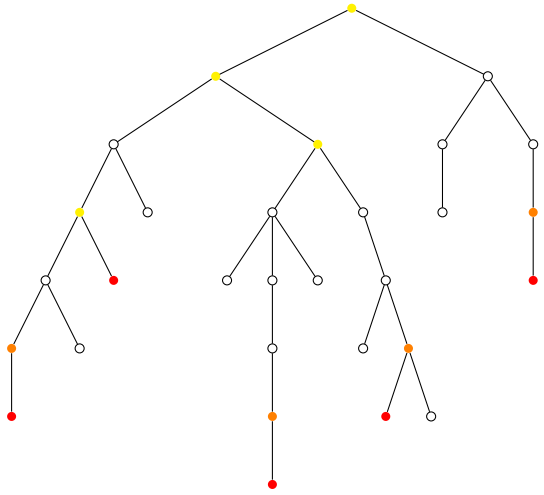
      $$v = a^h, b = b^h$$

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 $A \to BC$, 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

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

2.4 Ambiguità

Prendiamo il linguaggio

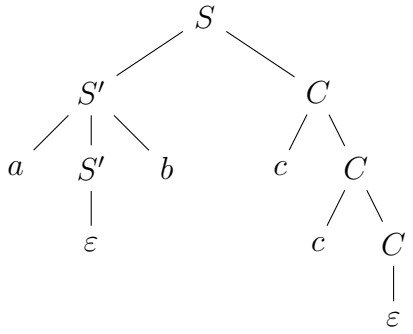
$$\mathcal{L} = \{a^p b^q c^r \mid p = q \vee q = r\}$$

sappiamo gli automi per $a^n b^n c^m$ e per $a^m b^n c^n$, quindi nondeterministicamente all'inizio scegliamo quale strada prendere nell'automa. Lo stesso si può fare con una grammatica

$$S \to S'C \mid AS''$$

dove $S' \to aS'b \mid \varepsilon$ genera $a^n b^n$ e $S'' \to bS''c \mid \varepsilon$ genera $b^n c^n$ e

$$C \to cC \mid \varepsilon$$
$$A \to aA \mid \varepsilon$$



vediamo ora i casi particolari dei due alberi

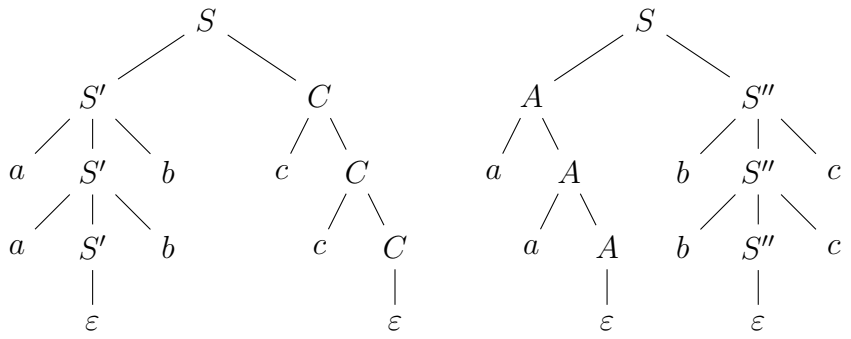


Figura 2.6: Alberi per la stringa $a^2b^2c^2$

Entrambi gli alberi rappresentano derivazioni leftmost che generano la stessa stringa $a^2b^2c^2$.

Una grammatica viene detta *ambigua* se esiste almeno una stringa che può essere generata in due modi diversi.

Definizione 3 (Grado di ambiguità)**.** Chiamiamo il *grado di ambiguità* di una stringa $w \in \Sigma^*$ in G è uguale al numero di alberi di derivazione di w in G. Ed il grado di ambiguità di G è il massimo grado di ambiguità generato dalle stringhe della grammatica.

Non sempre si può trovare una grammatica non ambigua per un linguaggio. Esistono linguaggi detti *inerentemente ambigui*, cioè per cui ogni grammatica di tipo 2 che li genera sarà ambigua.

Utilizzando il lemma di Ogden mostreremo che il linguaggio L è interentemente ambiguo. Mostriamo prima una versione alternativa del lemma

Lemma 6 (Lemma di Ogden)**.** *Sia G una grammatica CF e sia $L = L(G)$. Esiste una costante N tale che per ogni $z \in L$ con almeno N posizioni marcate possiamo decomporre z in 5 parti*

$$z = uvwxy$$

tali che

1. *vwx contiene al più N posizioni marcate*
2. *vx contiene almeno una posizione marcata*
3. *esiste una variabile $A \in V$ tale che $S \overset{*}{\Rightarrow} uAy$, e $A \overset{*}{\Rightarrow} vAx$ e $A \overset{*}{\Rightarrow} w$. E dunque per ogni $i \geq 0$ $uv^i wx^i y \in L$.*

Noi il teorema di Ogden lo abbiamo dimostrato utilizzando grammatiche in CNF, si può fare anche per grammatiche generiche, ma è più tedioso.

Ambiguità di L. Noi vogliamo dimostrare che

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

è inerentemente ambiguo.

Sia G una qualunque grammatica CF per $\mathcal{L}$. Sia N la costante del lemma di Ogden e $m = \max(N, 3)$ il massimo tra N e 3. Prendiamo

$$z = a^m b^m c^{m+m!} \in \mathcal{L}$$

e marchiamo tutte le a.

Prendiamo la stringaα ottenuta ponendo $i = 2$, cioè la stringa $\alpha = uv^2wx^2y \in \mathcal{L}$. Contiamo le b in α

$$\begin{aligned}
\#_b(\alpha) &= \underbrace{\#_b(z)}_{m} + \underbrace{\#_b(vx)}_{\leq m} \\
&\leq 2m \\
&< m + m! && \text{(Visto che } m \text{ è almeno 3)} \\
&\leq \#_c(\alpha) && \text{(Le } c \text{ rimaste sono almeno quelle che c'erano prima)}
\end{aligned}$$

e visto che $uv^2wx^2y \in \mathcal{L}$ allora $\#_b(\alpha) = \#_a(\alpha)$ e per la proprietà 2 del lemma di Ogden $\#_a(vx) = \#_b(vx) \geq 1$. Inoltre affinché $uv^2wx^2y \in \mathcal{L}$ sia ancora in $\mathcal{L}$ deve valere che

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

con $1 \leq l \leq m$. Altrimenti ripetendo perderemmo la struttura.

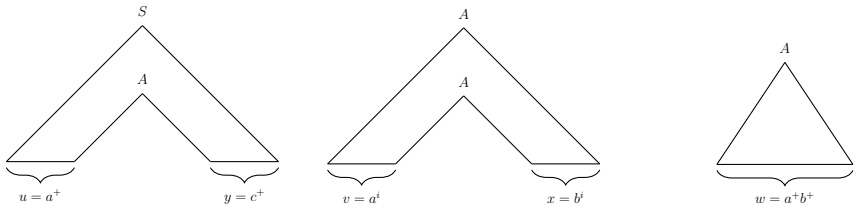
Prendiamo ora

$$uv^iwx^iz = a^{m+(i-1)l}b^{m+(i-1)l}c^{m+m!} \in \mathcal{L}$$

scegliendo $i = 1 + \frac{m!}{l}$ otteniamo la stringa

$$\beta = a^{m+m!}b^{m+m!}c^{m+m!}$$

che appartiene al nostro linguaggio.



Nella stringa β la maggior parte delle b è ottenuta replicando il secondo sottoalbero.

Partiamo invece da una stringa

$$z' = a^{m+m!}b^mc^m$$

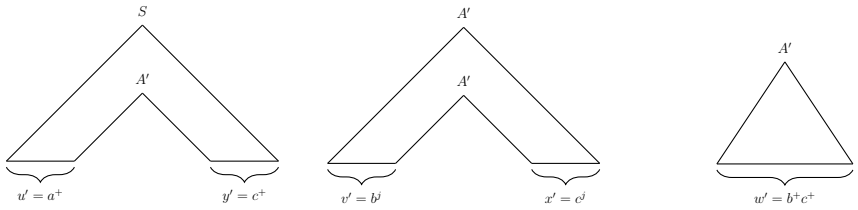
e marco tutte le c. Scomponiamola in

$$u'v'w'x'y'$$

e utilizzando i ragionamenti di prima mostriamo che $v' = b^j$ e $x' = c^j$ e ripetendo i volte v' e x' otteniamo

$$a^{m+m!}b^{m+(i-1)j}c^{m+(i-1)j}$$

e scegliendo $i = 1 + \frac{m!}{j}$ come prima otteniamo β. Come prima possiamo immaginare alberi di forma

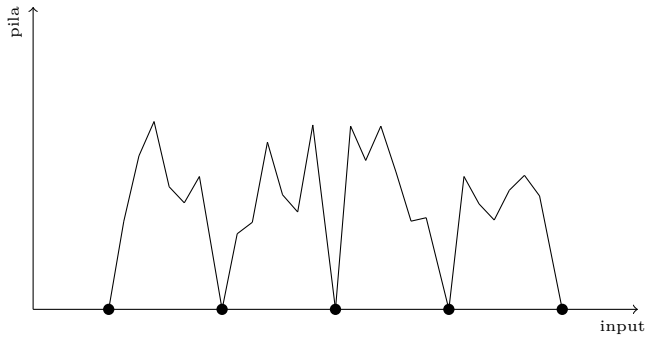


Qui ancora la maggior parte delle b è generata dal secondo albero. Ma visto che l'albero di A genera anche a e l'albero di A' genera anche c i due sono per forza diversi. ■

Possiamo dare una definizione analoga di ambiguità per gli automi a pila.

Definizione 4. Un PDA è ambiguo se esiste una stringa con due computazioni accettanti differenti.

È facile vedere attraverso la trasformazione da grammatica ad automa che il numero di computazioni diverse è uguale al numero di alberi leftmost diversi, visto che si simulava la derivazione leftmost. Dal lato opposto è più difficile da mostrare e soprattutto la trasformazione non preserva il grado di ambiguità.



Si può vedere dal grafico infatti che la pila può essere scomposta in vari modi equivalenti (1(2(34)), (12)(34), (((1)2)3)4, e così via). Quindi parlare di ambiguità negli automi a pila e nelle grammatiche di tipo 2 è equivalente. E di conseguenza un linguaggio inerentemente ambiguo avrà sia grammatica che automa a pila ambigui.

Perché un automa a pila ammetta ambiguità questo deve per forza essere nondeterministico. Quindi

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

necessita il nondeterminismo.

Richiamiamo la definizione di automa a pila deterministico:

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

è deterministico se ad ogni passo possiamo fare una singola scelta, cioè sse

- $\forall q \in Q, A \in \Gamma \mid \delta(q, \varepsilon, A) \neq \varnothing \Rightarrow \forall a \in \Sigma \delta(q, a, A) = \varnothing$
- $\forall q \in Q, A \in \Gamma, a \in \Sigma \cup \{\varepsilon\} \mid |\delta(q, a, A)| = 1$

mostriamo ora che i due criteri di accettazione sono diversi per automi deterministici, specificamente la accettazione per pila vuota è più debole. Supponiamo infatti di avere un automa a pila che accetta per pila vuota il linguaggio regolare $(aa)^+$. Siccome l'automa deve accettare aa dopo questa avrà la pila vuota. Ma da una pila vuota non può più continuare, quindi non può accettare $aaaa$ ad esempio. Quindi i linguaggi che accetta sono tutti quelli le cui stringhe non hanno come prefissi altre stringhe del linguaggio, perché nel riconoscere il prefisso svuoterà la pila e non potrà più andare avanti. Questi in teoria dei codici sono detti codici prefissi e contengono alcuni regolari e alcuni context free.

Di solito si ovvia a questo utilizzando un simbolo finale non nel linguaggio, ad esempio $(aa)^+\#$, in modo tale che la pila non si svuoti completamente prima di arrivare al marcatore finale.

Quindi da ora in poi quando parliamo di DCFL, cioè i linguaggi CF deterministici, intendiamo i linguaggi riconosciuti per stati finali. Questi contengono per forza i linguaggi regolari, perché semplicemente possiamo non utilizzare la pila. Vale che

$$\text{Reg} \subset \text{DCFL} \subset \text{CFL}$$

perché $\{a^p b^q c^r \mid p = q \vee q = r\} \in \text{CFL}$ ma $\notin \text{DCFL}$.

Abbiamo visto che l'ambiguità necessità del nondeterminismo. Possiamo chiederci l'inverso, cioè se un linguaggio necessita del nondeterminismo allora questo è per forza ambiguo. Un linguaggio che necessita del nondeterminismo

è quello dei palindromi (non ancora dimostrato), o – per semplicità – dei palindromi pari

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

L'automa deve "scommettere" su quando è arrivato a metà , quindi il non-determinismo è necessario, ma la metà è una sola, qiundi non è ambiguo. È anche facile vederlo dalla grammatica che è

$$S \to aSa \mid bSb \mid \varepsilon$$

Quindi il nondeterminismo non implica la ambiguità.

Non studieremo il determinismo per le grammatiche perché questo è più strano e ammette miriadi di classi diverse.

2.5 Operazioni e chiusura con i linguaggi CF

Tabella 2.1: Chiusura delle opreazioni

Operazione	CFL?	DCFL?
Unione	Sì	No
Intersezione	No	No
Intersezione con un regolare	Sì	Sì
Complemento	No	Sì
Prodotto	Sì	No
Chiusura di Kleene	Sì	No
Morfismo	Sì	No
Sostituzione	Sì	No
Reversal[1]	Sì	No
Shuffle[1]	No	No

[1] Risultati non mostrati a lezione

Unione Date due grammatiche $G' = \langle V', \Sigma, P', S' \rangle$ e $G'' = \langle V'', \Sigma, P'', S'' \rangle$ con $V' \cap V'' = \varnothing$, definiamo la unione delle due come

$$G = \langle V' \cup V'' \cup \{S\}, \Sigma, P' \cup P'' \cup \{S \to S', S \to S''\}, S \rangle$$

e nel caso degli automi possiamo fare la scelta all'inizio. La unione è chiusa per i CFL e non chiusa per i CDFL, infatti dati $L' = \{a^n b^n c^m\} \in \text{DCFL}$ e $L'' = \{a^m b^n c^n\} \in \text{DCFL}$ la loro unione $L' \cup L'' = \{a^p b^q c^r \mid p = q \vee q = r\}$ è ambigua, quindi nondeterministica.

Intersezione La intersezione non è chiusa, infatti $L' \cap L'' = \{a^n b^n c^n\}$ che abbiamo dimostrato che non è CF. Inoltre non possiamo utilizzare il metodo dei FSA di far andare in parallelo i due automi, qui i due due automi interferirebbero nel loro utilizzo della pila.

Intersezione con un regolare Possiamo usare il metodo degli FSA di far andare in parallelo i due automi, quindi incorporo nel controllo del PDA l'FSA.

Complemento Se il complemento fosse chiuso, potremmo ottenere una intersezione chiusa attraverso l'unione e De Morgan, quindi il complemento deve per forza non essere chiuso. Il complemento per i deterministici invece è chiuso.

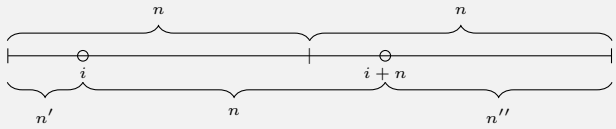
Esibiamo ora un controesempio che mostra che il complemento non è chiuso rispetto ai CF. Prendiamo il linguaggio

$$L = \{x \in \{a, b\}^* \mid \forall w : x \neq ww\} \in \text{CF}$$

il complemento di questo è

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

che come abbiamo già mostrato in passato non è context free.
Per completezza mostriamo anche che L sia effettivamente CF costruendo un automa a pila che lo riconosce. Questo – molto brevemente – scommette sulla posizione dei due caratteri diversi.



con $n' + n'' = n$ e $w_i \neq w_{i+n}$. Più precisamete l'automa segue le seguenti fasi

1. sulla pila viene contato n'.
2. si sceglie nondeterministicamente un terminale σ e si iniziano a togliere i n' simboli accumulati finché non si svuota la pila.
3. a questo punto si ricomincia a contare sulla pila fino a che non si sceglie un secondo terminale ρ che deve essere differente.
4. si ricomincia a togliere i simboli n'' dalla pila consumando l'input
5. deve valere che alla fine dell'input sia vuota anche la pila e viceversa, affinché $n' + n'' = n$

Lemma 7. *I DCFL sono chiusi rispetto al complemento.*

Uno dei problemi con gli automi a pila è che questi possono non terminare. Infatti utillizzando le ε-mosse l'automa può continuare a far oscillare la pila o far crescere la pila all'infinito. Visto però che gli stati e i simboli della pila sono finiti, data una configurazione di superficie (*surface configuration*, o lo stato e il simbolo in cima alla pila) capire se ci sarà un ciclo infinito è decidibile, e consiste banalmente nel controllare se lo stesso stato e simbolo di pila si ripetono senza consumare input.

Inoltre visto che sono ammesse le ε-mosse, in un automa che accetta per stato finale, può accadere che una volta che l'input è finito l'automa continui a fare mosse, e addirittura può finire in un ciclo infinito. In ogni caso se c'è almeno uno stato finale tra quelli che visita dopo la fine dell'input, allora l'input è accettato.

Lemma 8. *Supponiamo che*

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

sia un automa a pila deterministico. Costruiamo

$$M' = \langle Q', \Sigma, \Gamma', \delta', q_0', X_0, F' \rangle$$

tale che

$$L(M) = L(M')$$

ed M' scandisce sempre l'intero input.

Dimostrazione. Definiamo l'automa M' tale che

- $Q' = Q \cup \{q_0', d, f\}$, con d detto *stato trappola*
- $\Gamma' = \Gamma \cup \{X_0\}$
- $F' = F \cup \{f\}$

e definiamo le mosse che fa l'automa

- $\delta'(q_0', \varepsilon, X_0) = (q_0, Z_0 X_0)$, cioè – come in altre costruzioni – "infiliamo" in fondo alla pila un nostro simbolo. Questo è necessario perché la prossima mossa potrebbe non essere definita perché si è svuotata la pila.
- mostriamo ora alcune regole particolari che portano allo stato trappola
  - se in una certa configurazione di superficie dell'automa M non esiste una prossima mossa definita, cioè se

    $$\delta(q, a, X) = \delta(q, \varepsilon, X) = \varnothing, \qquad q \in Q, a \in \Sigma, X \in \Gamma$$

    allora nell'automa M' finisco nello stato trappola

    $$\delta'(q, a, X) = (d, X)$$

  - se nell'automa M la pila si è svuotata e non potrei pià fare mosse nell'automa M' finisco nello stato trappola

    $$\delta'(q, a, X_0) = (d, X_0), \qquad q \in Q', a \in \Sigma$$

  - nello stato trappola consumo l'input e rimango nello stato trappola

    $$\delta'(d, a, X) = (d, X), \qquad a \in \Sigma, X \in \Gamma'$$

- se da (q, X), con $q \in Q$ e $X \in \Gamma$ è possibile un loop di ε-mosse

$$\delta'(q, \varepsilon, X) = \begin{cases} (d, X) & \text{se il loop non visita stati finali} \\ (f, X) & \text{altrimenti} \end{cases}$$

- se sono nello stato finale f e posso ancora leggere input, allora non sono alla fine quindi mi sposto nello stato trappola

$$\delta'(f, a, X) = (d, X), \qquad a \in \Sigma, X \in \Gamma'$$

- in tutti gli altri casi $\delta'(q, a, X) = \delta(q, a, X)$ con $q \in Q, a \in \Sigma \cup \{\varepsilon\}, X \in \Gamma$

■

Automa per il complemento. Sia

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

un DPDA che accetta L e scandisce sempre l'intero input. Costruiamo l'automa per il complemento

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

con

- $Q' = Q \times \{y, n, A\}$, dove y, n, A servono a ricordare se nell'ultima sequenza di ε-mosse ho visto uno stato finale, rispettivamente:
  - y indica che nell'attuale stato della sequenza di ε-mosse è stato visitato uno stato finale.
  - n indica che nell'attuale stato della sequenza di ε-mosse non è stato visitato uno stato finale.
  - A indica che non ho visitato uno stato finale e non posso fare alcuna mossa.
- $F' = Q \times \{A\}$, cioè tutti gli stati in cui non posso procedere e non ho visitato uno stato finale

Definiamo ora δ' come

- se $\delta(q, \varepsilon, X) = (p, \gamma)$ allora
  - $\delta'([q, y], \varepsilon, X) = ([p, y], \gamma)$, quindi se in passato ho visitato uno stato finale, continuo a ricordare che ho visitato uno stato finale.
  - se non ho ancora visto uno stato finale, e il prossimo lo è, cambio n a y
$$\delta'([q, n], \varepsilon, X) = \begin{cases} ([p, n], \gamma) & \text{se } p \notin F \\ ([p, y], \gamma) & \text{se } p \in F \end{cases}$$
- se $\delta(q, a, X) = (p, \gamma)$ con $a \in \Sigma$, allora
  - se ero in uno stato y e consumo dell'input e finisco in uno stato non finale, cambio y a n
$$\delta'([q, y], a, X) = \begin{cases} ([p, n], \gamma) & \text{se } p \notin F \\ ([p, y], \gamma) & \text{se } p \in F \end{cases}$$
  - alternativametne se ero in uno stato n spezzo la mossa in due parti
$$\delta'([q, n], \varepsilon, X) = ([q, A], X)$$
$$\delta'([q, A], a, X) = \begin{cases} ([p, n], \gamma) & \text{se } p \notin F \\ ([p, y], \gamma) & \text{se } p \in F \end{cases}$$
    cioè prima di consumare dell'input assumo di aver finito la stringa senza esser passato per stati finali nella sequenza di ε-mosse. Se c'è ancora in input trasformo lo A in y o n.

Infine definiamo
$$q_0' = \begin{cases} [q_0, n] & \text{se } q_0 \notin F \\ [q_0, y] & \text{se } q_0 \in F \end{cases}$$

■

Dalla costruzione di sopra si vede che le ε-mosse sono molto fastidiose. Come abbiamo visto attraverso la trasformazione in GNF nel caso di PDA si può fare a meno delle ε-mosse a patto di sacrificare la parola vuota. Lo stesso non vale nel caso di DPDA, e senza ε-mosse otteniamo un modello meno potente.

Dai linguaggi

$$L' = \{a^i b^j c^k \mid i = j\}$$
$$L'' = \{a^i b^j c^k \mid j = k\}$$

Definiamo il linguaggio

$$L_0 = L' \cup dL'' \in DCFL$$

in base a se la stringa inizia con una d o no sappiamo se dobbiamo riconoscere una parola di L' o di L''.

Prodotto Non deterministicamente quando un automa arriva in fondo faccio partire l'automa successivo. Questo si può fare anche in termini di grammatiche, date $G' = \langle V', \Sigma, P', S' \rangle$ e $G'' = \langle V'', \Sigma, P'', S'' \rangle$ con $V' \cap V'' = \varnothing$ creiamo

$$G = \langle V' \cup V'' \cup \{S\}, \Sigma, P' \cup P'' \cup \{S \to S'S''\}, S \rangle$$

Per il caso deterministico utilizziamo L_0 di prima e definiamo

$$L' = \{\varepsilon, d\} \cdot L_0 = \{d^s a^i b^j c^k \mid 0 \leq s \leq 2\}$$

con

$$\begin{cases} s = 0 & i = j \\ s = 2 & j = k \\ s = 1 & i = j \vee j = k \end{cases} \}$$

Per mostrare che è ambiguo prendiamo

$$L' \cap da^* b^* c^* = \{da^i b^j c^k \mid i = j \vee j = k\} \notin \text{DCFL}$$

con $da^* b^* c^*$ regolare, visto che i deterministici sono chiusi rispetto all'intersezione con regolari, L' non può essere deterministico. Inoltre $\{\varepsilon, d\}$ è finito, quindi non solo i linguaggi deterministici non sono chiusi rispetto al prodotto, ma non sono chiusi neanche rispetto al prodotto a sinistra con linguaggi finiti.

Si può mostrare però che i DCFL sono chiusi rispetto al prodotto a destra con regolari, cioè dati $L \in$ DCFL e $R \in$ Reg il prodotto $L \cdot R \in$ DCFL. Questa costruzione è simile al prodotto per gli automi a stati finiti.

Chiusura di Kleene – star Questo costruzione è molto simile al prodotto. Data la grammatica $G' = \langle V', \Sigma, P', S' \rangle$, costruiamo

$$G = \langle V' \cup \{S\}, \Sigma, P' \cup \{S \to \varepsilon, S \to S'S\}, S \rangle$$

Nel caso dei deterministici prendiamo ancora L_0 e analizziamo

$$L_0^* \cap da^*b^*c^* = d(L_1 \cup L_2)$$

infatti questo ha stringhe della forma

$$da^ib^jc^k$$

tali che

- dL_2, per cui $j = k$
- o il prodotto $d \in L_2$ per $a^ib^jc^j \in L_1$, per cui $i = j$

ma questo abbiamo visto che non è deterministico, quindi L_0^* non è chiuso rispetto alla star.

Morfismo Per ogni terminale $a \in \Sigma$, lo sostituisco con una stringa $w \in \Delta^*$ utilizzando una funzione

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

Per questo basta sostituire i terminali in ogni produzione.

Nel caso dei deterministici prendiamo

$$L' = \{a^ib^jc^k \mid i = j\}$$
$$L'' = \{a^ib^jc^k \mid j = k\}$$

e sappiamo che $L', L'' \in$ DCFL, e che $L' \cup L'' \notin$ DCFL. Mentre vale che $L_0 = L' \cup dL'' \in$ DCFL. Prendiamo il morfismo

$$h(\sigma) = \begin{cases} \sigma & \text{se } \sigma \neq d \\ \varepsilon & \text{altrimenti} \end{cases}$$

vale che $h(L_0) = L' \cup L'' \notin$ DCFL, quindi i determinisici non sono chiusi rispetto al morfismo.

Sostituzione Ad ogni terminale associamo un linguaggio, utilizzando una funzione

$$s : \Sigma \to 2^{\Delta^*}$$

Se $L \in$ CFL e $\forall a \in \Sigma \mid s(a) \in$ CFL, allora $s(L) \in$ CFL. Molto ad alto livello ad ogni terminale corrisponde una grammatica, nelle produzioni sostituiamo ai terminali l'assioma della grammatica.

Visto che il morfismo è un caso particolare della sostituzione, i determinisitici non sono chiusi rispetto alla sostituzione.

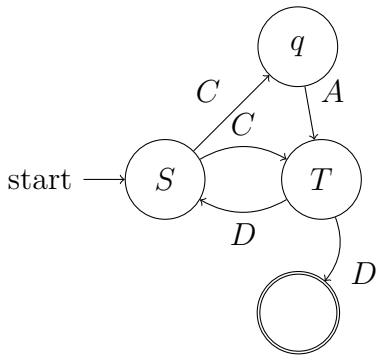
Data la grammatica

$$G = \langle V = \{S, T, A, B\}, \Sigma = \{a, b, c, d\}, S, P \rangle$$

con P definito come

$$\begin{aligned} S &\to cAT \mid cT \\ T &\to dS \mid d \\ A &\to BaA \mid Ba \\ B &\to bB \mid \varepsilon \end{aligned}$$

$$\begin{aligned} S &\to cAT \mid cT \\ T &\to AS \mid d \end{aligned}$$



Proviamo a fare la conversione da automa a linguaggio regolare:

$$\begin{cases} S &= cQ + cT \\ Q &= AT \\ T &= dS + d \end{cases}$$

sostituendo la terza nella prima e la seconda abbiamo

$$\begin{cases} S &= cQ + cdS + cd \\ Q &= AdS + Ad \\ T &= dS + d \end{cases}$$

e poi sostituiamo la seconda nella prima

$$\begin{cases} S &= cAdS + cAd + cdS + cd \\ Q &= AdS + Ad \\ T &= dS + d \end{cases}$$

e semplificando

$$\begin{cases} S &= (cAd + cd)S + cAd + cd \\ Q &= AdS + Ad \\ T &= dS + d \end{cases}$$

che corrisponde a $S = (cAd + cd)^*(cAd + c) = (cAd + cd)^+$.

Alla regola $A \to BaA \mid Ba$ corrisponde il linguaggio regolare $A \overset{*}{\Rightarrow} (Ba)^+$, e a $B \to bB \mid \varepsilon$ corrisponde ovviamente $B \overset{*}{\Rightarrow} b^*$. Quindi sostituendo nel regolare di A abbiamo $A \overset{*}{\Rightarrow} (b^*a)^+$, e sostituendo in $S \overset{*}{\Rightarrow} (c(b^*a)^+d + cd)^+$. Quindi $L(G) = (c(b^*a)^*d)^+$.

Teorema 1 (Teorema di Chomsky-Schutzenberger)**.** *Sia Ω_k un alfabeto di k tipi di parentesi*

$$\Omega_k = \{(_1, (_2, \ldots, (_k,)_1,)_2, \ldots,)_k\}$$

$D_k \subseteq \Omega_k^*$ è detto un linguaggio di Dyck, se è il linguaggio delle parentesi bilanciate correttamente. Sia $L \subseteq \Sigma^*$ un linguaggio CF, allora

$$\exists k > 0 \mid h : \Omega_k \to \Sigma \text{ morfismo } eR \subseteq \Omega_k \text{ regolare}$$

tale che $L = h(D_k \cap R)$.

Non dimostrato.

Prendiamo il caso banale $L = D_1$, allora $k = 1$ e $h = \mathrm{id}$ e $R = \Omega_1^*$.

Prendiamo il caso

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

allora $k = 1$ e

$$h = \begin{cases} (= a \\ ) = b \end{cases}$$

e infine $R = (^*)^*$.

Prendiamo il caso

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

allora prendiamo $k = 2$ e

$$h = \begin{cases} (= a \\ a =) \\ [= b \\ ] = b \end{cases}$$

e infine il filtro $R = ((+ [)^*(] +))^*$.

Prendiamo il caso

$$L = \{x \mid\mid x \in \{a, b\}^* \wedge x = x^R\}$$

allora prendiamo $k = 4$ e similmente a prima definiamo h

$$h = \begin{cases} \langle = a \\ \rangle = \varepsilon \\ \lfloor = b \\ \rfloor = \varepsilon \\ (= a \\ a =) \\ [= b \\ ] = b \end{cases}$$

e infine il filtro $R = ((+ [)^*(\varepsilon + \langle\rangle + \lfloor\rfloor)(] +))^*$.

Quindi il riconoscimento di un linguaggio CF si riconduce a riconoscere un linguaggio di parentesi. Quindi si potrebbe pensare di costruire una macchina Il componente h^{-1} può essere realizzato come un componente a stati finiti. Quindi l'unica cosa non a stati finiti è il componente D_k.

2.6 Lingauaggi unari

Un linguaggio unario è un linguaggio tale che $|\Sigma| = 1$. Mostreremo che nel caso unario i linguaggi CF e i regolari sono la stessa classe.

Supponiao di avere un automa deterministico per un alfabeto unario Visto che c'è un solo simbolo, questi sono formati da una serie di stati e opzionali loop indietro. Il linguaggio riconosciuto da sopra è

$$L = \varepsilon, a^6, a^1 1, a^1 6, \cdots = a^0 + a^6(a^5)^*$$

Modificandolo in questo modo

$$L = a^0 + a^5(a^6)^* + a^6(a^5)^*$$

Questi rappresentano più o meno successioni numeriche.

Prendiamo ora l'automa questo riconosce

$$L = a^8 + a^{11}(a^5)^* + a^{12}(a^5)^*$$

Lemma 9. *Supponiamo di avere una serie di naturali*

$$l_1, l_2, l_3, \ldots$$

e un $q > 0$. Supponiamo di avere linguaggi del tipo

$$K_i = a^{l_i}(a^q)^*$$

allora l'unione

$$\bigcup_{i \geq 0} K_i \in Reg$$

Noi abbiamo visto che l'unione di linguaggi regolari è ancora regolare, ma qui il risultato importante è che l'unione può essere di un numero potenzialmente infinito di regolari.il risultato importante è che l'unione può essere di un numero potenzialmente infinito di regolari. Ad esempio questo non vale per l'unione infinita di

$$\bigcup_{i \geq 0} a^i b^i$$

Tutti questi possono essere ricondotti all'automa di sopra purché abbiano tutti lo stesso periodo q.

Lemma 10. *Sia N un intero e prendiamo dei linguaggi*

$$K_i = a^{l_i}(a^{q_i})^* \subseteq a^*$$

quindi senza un periodo fissato. Se però che $\forall i \mid 1 \leq q_i \leq N$, allora l'unione

$$\bigcup_i K_i \in Reg$$

Infatti possiamo generare per il lemma di prima i linguaggi regolari per ogni q, e visto che N è finito facciamo l'unione finita di tutti i linguaggi per ogni q nel range.

Teorema 2. *$L \subseteq a^* \in CFL$ implica L regolare.*

Dimostrazione. Utilizzeremo il pumping lemma per i linguaggi CF. Sia N la costante del pumping lemma per L. Prendiamo $a^m \in L$ con $m \geq N$, allora

$$a^m = uvwxy$$

sappiamo che

- $vx \neq \varepsilon$, e chiamiamo $q_m = |vx|$
- $|vwx| \leq N$
- $\forall i \geq 0 \mid uv^i wx^i y \in L$, la lunghezza di questa stringa è (escludendo il caso $i = 0$)
  $$m + (i-1)q^m$$
  quindi $a^{m+(i-1)q_m} \in L$ cioè $a^m(a^{q_m})^* \subseteq L$

quindi $q \leq q_m \leq N$. Prendiamo

$$L' = \{a^m \in L \mid m < N\}$$
$$L'' = \{a^m \in L \mid m \leq N\}$$

e vale che

$$L = L' \cup L'' \subseteq L' \cup \bigcup_{m \geq N} a^m (a^{q_m})^* \subseteq L$$

quindi

$$L = L' \cup \bigcup_{m \geq N} a^m (a^{q_m})^*$$

e L' è regolare perché finito e per il lemma di prima anche il secondo è ancora regolare, visto che $1 \leq q_m \leq N$ per il pumping lemma. E visto che l'unione di due regolari è regolare, allora L è regolare. ■

Dati si può vedere che generalmente il periodo è il MCD tra i due periodi dei linguaggi.

Nei linguaggi unari si può mostrare lo strano risultato che per passare da un nodeterministico a un deterministico il costo è $e^{\sqrt{n \ln n}}$

Teorema 3 (Teorema di Parikh)**.** *Dato un alfabeto Σ di qualsiasi dimensione e una stringa w. Sia ad esempio $\Sigma = \{a, b\}$, associamo a questa stringa il vettore*

$$\phi(w) = (\#_a(w), \#_b(w))$$

chiamato immagine di Parikh. E definiamo l'immagine di Parikh del linguaggio come

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

Se $L \in CFL$, allora $\exists L' \in Reg$ tale che $\phi(L) = \phi(L')$.

Quindi se non ci interessa l'ordine dei simboli, il linguaggi regolari e i linguaggi CF sono uguali. Un corollario è che vale l'uguaglianza del CF e dei regolari nel caso di alfabeti unari.

Sia

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

e

$$\phi(L) = \{(n, n) \mid n \geq 0\}$$

allora possiamo prendere il linguaggio regolare $R = (ab)^*$ e vale che

$$\phi(L) = \phi(R)$$

2.7 Automi a pila two-way

Degli automi a pila abbiamo studiato un modello one-way. A differenza degli FSA già nel modello one-way abbiamo differenza tra il modello detrministico e non-deterministico.

Vediamo ora il modello two-way, e prendiamo il linguaggio

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

nel caso one-way way l'informazione del numero di a viene distrutta durante il confronto con le b. Mentre nel caso two-way possiamo tornare indietro a recuperarla per confrontare le a anche con le c. Anche il caso

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

basta copiare w sulla pila fino al cancelletto, poi si salta fino in fondo e si confronta con la seconda stringa.

Pendiamo un altro linguaggio

$$L = \{a^{2^n} \mid n \geq 0\}$$

Dobbiamo "dividere per due" iterativametne. Questo è fatto scorrendo la stringa e ogni due simboli aggiungendone uno sulla stringa. Ad esempio In questo modo se arriviamo all'end marker in uno stato "dispari", è equivalente ad avere resto non zero e quindi non si accetta (tranne nel caso 2^0). Quindi è possibile riconoscere questo linguaggio, che non è CF.

Ritornando all'esempio di prima

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

con il sistema di prima riusciamo a trovare la metà con la pila vuota. Una volta a metò si carica la seconda parte della stringa sulla pila. Poi si ritrova la metà e si confronta con la prima parte della stringa al cotnratio. Ed è per giunta deterministico.

Chiamiamo i PDA (deterministici e non) two-way 2PDA, mentre i deterministici li chiamiamo 2DPDA. Abbiamo visto che sicuramente sono più potentei dei PDA normali perché abbiamo visto 3 linguaggio non CF che riescono a riconoscere. Non si sa però se 2PDA e 2DPDA siano classi distinti, cioè se il modello nondeterministico sia più potente di quello determnistico.

Inoltre sicuramente non si sà se la classe dei 2PDA sia strattamente inclusa nei CS, se sia più grande, o se siano inconfrontabili.

Inoltre non si sà se i linguaggi riconosciuti dai 2DPDA siano un superset dei CFL.

2.8 Automi limitati linearmente

Supponiamo ora di avere un automa two-way che può anche modificare il nastro. Questo è chiamato automa limitato linearmente, o LBA.

Il linguaggio

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

è riconoscibile cancellando la prima a, poi la prima b e poi la prima c e poi tonando indietro all'inizio della stringa. Si procede in questo modo finché non finiscono le a e se il nastro è tutto cancellato allora la parola è accettata.

Il linguaggio

$$L = \{a^{2^n} \mid n \geq 0\}$$

è riconosciuto similmente a prima: ogni due a se ne cancella una, una volta arrivati alla fine si torna all'inizio e si ripete.

Per il linguaggio

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

si trova il centro

$$\triangleright \quad a \quad b \quad b \quad b \quad a \quad b \quad b \quad b \quad \triangleleft$$

una volta trovata la metà il simbolo che la si ricorda nello stato e lo si cancella. si va fino in fondo e si cerca se l'ultimo simbolo è uguale e se

lo è lo si cancella e si va avanti.

Per il linguaggio delle parentesi a un simbolo. Si cerca la prima parentesi chiusa e si controlla se il primo simbolo precedente (non cancellato) è una aperta. Si procede iterativamente finché non si arriva all'end-marker.

$$\triangleright(()())(())\triangleleft$$
$$\triangleright(\not(\not)()())(())\triangleleft$$

Vediamo il linguaggio di quadrati di interi in notazione unaria

$$L = \{a^{n^2} \mid n \geq 0\}$$

possiamo usare il fatto che il quadrato di n è uguale alla somma dei primi n numeri dispari

$$n^2 = \sum_{i=1}^{n}(2i+1)$$

Ora procediamo le seguente modo

$$\triangleright aaaaaaaaaaaaaaaa\triangleleft$$
$$\triangleright Xaaaaaaaaaaaaaaa\triangleleft$$
$$\triangleright XYYYaaaaaaaaaaa\triangleleft$$
$$\triangleright XYYYXXXXXaaaaaaa\triangleleft$$
$$\triangleright XYYYXXXXXYYYYYYY\triangleleft$$

Quindi l'idea per un blocco è fare la copia del blocco precedente e aggiungere 2.

Gli automi LBA (non deterministici) corrispondono alla classe dei lin-

guaggi CS. Ricordiamo che le grammatice CS sono della forma

$$\alpha \to \beta, \qquad |\beta| \geq |\alpha|$$

L'idea dell'automa è di simulare la derivazione applicando le regole al contrario: se si trova una parte dell'input che corrisponde ad una parte destra di una regola, lo si sostituisce con la sua parte sinistra (nel caso riepmpiendo lo spazio extra con un simbolo nullo).

In generale si può mostrare che i linguaggi CS possono essere riconosciuti anche da macchine che hanno un nastro di input e un nastro di lavoro separato (di lunghezza limitata). Infatti i linguaggi CS sono riconosciuti da tutte queste macchine che hanno una memoria limitata linearmente rispetto l'input.

Vediamo ora le proprietà di chiusura di questi linguaggi:

- chiusi rispetto all'unione, il motivo è lo stesso della chiusura rispetto all'unione dei CF
- chiusi rispetto all'intersezione: si esegue il primo automa e se questo accetta si esegue il secondo. Per evitare che il primo automa sporchi l'input per il secondo, si "sdoppia" l'input e ogni cella contiene una coppia di simboli e il primo automa tocca solo uno dei due elementi.
- chiusi rispetto al complemento (dimostrato da Immermann nell'88). Se c'è una macchina nondeterministica che lavora in spazio $s(n)$ della lunghezza di input, si può ottenre una macchina limitata da $s(n)$ per il complemento

Non si sà se il modello nondeterministico sia più potente o no di quello deterministico

$$\text{DLBA} \stackrel{?}{=} LBA$$

Teorema 4 (Teorema di Sevitch)**.** *Se ho una classe di linguaggi riconosciuta da una macchina nondeterministica, si può sempre costruire una macchina determinisitca che riconosce in spazio quadratico.*

$$NSPACE(s(n)) \subseteq DSPACE(s^2(n))$$

Quindi da questo

$$\text{CS} = \text{NSPACE}(n) \subseteq \text{DSPACE}(n^2)$$

ma non si sà se si può fare meglio di così.

2.9 Problemi di decisione sui linguaggi CF

Teorema 5. *Sia $L \in CF$ e N la sua costante del pumping lemma per L, allora*

$$\begin{cases} L \neq \varnothing & sse\ \exists w \in L \mid |w| < N \\ L\ \grave{e}\ infinito & sse\ \exists w \in L \mid N \leq |w| < 2N \end{cases}$$

La dimsotrazione di questo è uguale allo stesso teorema per i regolari.

Per mostrare che $L \neq \varnothing$, supponiamo che L sia generato dalla grammatica

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

Possiamo costrire l'insieme delle variabili che sono in grado di generare terminali, e se questo contiene S allora il linguaggio non è vuoto. Iniziamo costruendo l'insieme

$$T_0 = \Sigma$$

e

$$T_{i+1} = T_i \cup \{A \in V \mid \exists A \to B \in P \mid \beta \in T_i^*\}$$

Vale che

$$T_0 \subseteq T_1 \subseteq \cdots \subseteq \Sigma \cup V$$

Mostriamo ora se un linguaggio L è finito o infinito. Supponiamo che L sia riconosciuto dalla grammatica G in FNC, priva di simboli raggiungibili e produttivi. Costruiamo il grafo delle produzioni: i vertici sono le variabili ed esiste un arco da A a B se esiste una variabile C tale che

$$A \to BC \in P \vee A \to CB \in P$$

Dato questo grafo, L è infinito sse il grafo contiene un ciclo. Infatti vuol dire che esiste una serie di produzioni per cui una variabile può riprodurre sé stessa e visto che tutte le variabili in FNC sono produttive, la stringa non può che crescere.

Quindi questa è una quesione decidibile.

Vedremo invece che non possiamo decidere se

$$L \stackrel{?}{=} \Sigma^*$$

Si possono considerare modelli diversi rispetto ai LBA, ad esempio un automa con un nastro di input read-only e con n-nastri ausiliari.

Definizione 5 (Memoria di Turing ad un nastro)**.** Un macchina di Turing è un automa con una testina e un nastro illimitato che, sulle celle che non contengono l'input, contiene un carattere speciale detto *blank*.

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

- Q un insieme di stati
- Σ un alfabeto, e $\Sigma \subseteq \Gamma$
- Γ un alfabeto di lavoro, e il blank $\# \in \Gamma \setminus \Sigma$
- $q_0 \in Q$
- $F \subseteq Q$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ nel caso determinsitico, dove
  - -1 è un movimento a destra
  - 0 nessun movimento
  - $+1$ un movimento a destra

Quindi una macchina di Turing può utilizzare una memoria illimitata.

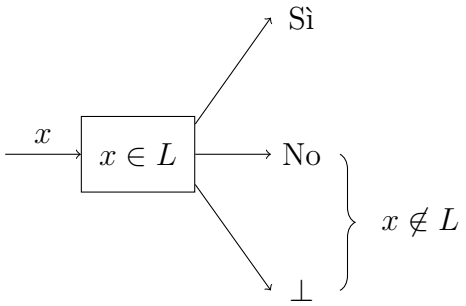
Ci sono diverse varianti dell'automa, ad esempio con un nastro infinito solo da un lato, o più nastri.

Assumeremo che gli stati finali siano stati *halting*, quindi che non ci siano transizioni da essi, e inoltre assumeremo che il blank non possa essere scritto.

È importante notare che la macchina potrebbe entrare in un loop.

Nel caso delle macchine di Turing, il modello nondeterministico e il modello deterministico sono gli stessi, se non ci sono vincoli di risorse. Infatti una macchina nondetermininistica in ogni momento può fare delle scelte, nella versione deterministica si utilizza una BFS.

Si può mostrare che la classe dei linguaggi riconosciuti da questa macchina sono i linguaggi di tipo 0. Infatti si può pensare di, tenendo l'input sul nastro, e applicando tutte le produzioni al contrario finché non si trova l'assioma, opzionalmente andando avanti in eterno. Possiamo avere quindi tre possibili risposte



per questo i linguaggi sono detti ricorsivi numberabili.

Possiamo vedere le macchine di Turing anche come macchine che calcolano funzioni, e le funzioni calcolate da una macchina di Turing sono le stesse calcolate da un linguaggio di programmazione generico.

Vediamo ora alcuni problemi indecibili per le macchine di Turing.

- problema dell'arresto (HALT), tale che abbiamo in input una macchina di Turing M e una stringa $x \in \Sigma^*$, e ci chiediamo se la macchina termini su input x
- vuotezza del linguaggio riconosciuto, tale che abbiamo in input una macchina di Turing M e ci chiediamo se $L(M) = \varnothing$

Definizione 6 (Configurazione)**.** Definiamo ora la configurazione di una macchina di Turing. Questa non contiene l'intero nastro infinito, ma solo la parte non-blank. Suponiamo che la parte a sinistra della testina sia chiamata x e che la parte della stringa da dove la testina si trova all'estremo destro si chiami y, una configurazione è

$$xqy$$

con $q \in Q$ e $x, y \in \Gamma^*$.

Definizione 7 (Configurazione iniziale)**.** La configurazione iniziale su input w è

$$q_0 w$$

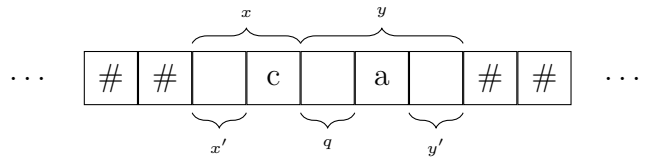
Definizione 8 (Configurazione finale)**.** Una configurazione è finale o acecttante se lo stato q è finale.

Definizione 9 (Mossa)**.** Una mossa, scritta $C \vdash C'$, o nel caso di più mosse $C \overset{*}{\vdash} * C'$ è una applicazione di δ.

Definizione 10 (Linguaggio riconosciuto da una macchina di Turing)**.** Data la definizione di mossa definiamo il linguaggio riconosciuto da una macchina di Turing M come

$$L(M) = \{w \in \Sigma^* \mid q_0 w \overset{*}{\vdash} xqy, x, y \in \Gamma^*, q \in F\}$$

Supponiamo di trovarci nella configurazione



con $xqy = xqay'$. possiamo avere tre casi di mossa $\delta(q, a)$

- $(p, b, 0)$ fa passare alla configurazione $xqay' \vdash xpby'$
- $(p, b, +1)$ fa passare alla configurazione $xqay' \vdash xbpy'$
- $(p, b, -1)$ fa passare alla configurazione $xqay' \vdash x'pcby'$

Chiamiamo (questo cancelletto non è il blank, è un altro)

$$\Delta = Q \cup \Gamma \cup \{\#\}$$

e definiamo un linguaggio su questo alfabeto

$$L'_{succ} = \{\alpha \# \beta^R \mid \alpha \text{ e } \beta \text{ sono configurazioni di } M \text{ e } \alpha \vdash \beta\}$$

Per riconoscere questo basta un automa a pila, infatti ad esempio sia $\alpha = xqay'$ e $\beta = y'^R bpx^R$, carico x e con la funzione di transizione calcolo con cosa sostituire qa e carico y' e poi scarico dopo il cancelletto. Definiamo anche

$$L''_{succ} = \{\alpha^R \# \beta \mid \dots\}$$

come sopra. Questi automi a pila sono deterministici, visto che il successore è deterministico.

Definiamo ora il linguaggio delle computazioni valide per un automa M, o le computazioni accettanti

$$\text{valid}(M) = \{\alpha_1 \# \alpha_2^R \# \alpha_3 \# \dots \# \alpha_k^{(R)} \mid \alpha_i \in (Q \cup \Gamma)^*\}$$

(R) indica che se k è dispari allora non è rovesciato, altrimenti sì. E devono valere le seguenti condizioni:

- per $i \in 1, \ldots, k$ deve valere che $\alpha_i \in \Gamma^* Q \Gamma^*$, cioè α_i rappresenta una configurazione di M
- α_1 rappresenta una configurazione iniziale
- α_k rappresenta una configurazione accettante
- per $i \in 2, \ldots, k$, deve valere che $\alpha_{i-1} \vdash \alpha_i$, cioè α_i è raggiunto in una mossa da α_{i-1}

La prima, la seconda e la terza condizioni possono essere controllate da automi a stati finiti. Per la quarta condizione, abbiamo visto che un passaggio è il linguaggio L'_{succ} (o L''_{succ}), ma confrontare dopo aver confrontato α_1 e α_2^R, abbiamo distrutto la pila per α_3. Quello che possiamo fare con un automa a pila è confrontare α_1 e α_2^R, α_3 e α_4^R e così via. Poi si può fare un secondo automa che confronta α_2^R e α_3, α_4^R e α_5, e così via. Quindi questo linguaggio è l'intersezione di due linguaggi CF.

Teorema 6. *Esistono due linguaggi CF L_1 e L_2 tali che valid$(M) = L_1 \cap L_2$, ed esiste un algoritmo che data M costruisce PDA (deterministico) o CFG per L_1 e L_2.*

Se la macchina M riconosce l'insieme vuoto, quindi

$$L(M) = \varnothing$$

sse non ci sono computazioni valide, quindi

$$\text{valid}(M) = \varnothing$$

sse $L_1 \cap L_2 = \varnothing$. Quindi non è possibile decidiere se l'intersezione di due linguaggi CF è vuota, perché se si potesse allora potremmo anche decidere se un certo linguaggi riconosciuto da una macchina di Turing è vuota.

Corollario 1. *Non è possibile decidere se l'intersezione di due CFL (e anche DCFL) è vuota.*

Vediamo ora invece

$$(\text{valid}(M))^C = \Delta^* \setminus \text{valid}(M)$$

Costruiamo un dispositivo nondeterministico che scommette quele delle quattro condizioni non sia rispettata.

- se la prima è infranta basta controllare che tra due cancelletti non ci sia uno stato, o ci siano più stati.
- ...
- ...
- esiste almeno un i tale che α_i non è raggiunta da una mossa da α_{i-1}. Questo è verificabile con una pila.

Le prime proprietà sono controllabili ancora con automi a stati finiti. Mentre la quarta è riconoscibile da una pila.

Teorema 7. *$(valid(M))^C \in CFL$ ed esiste un algoritmo che, data M costruisce una CFG o un PDA per $(valid(M))^C$.*

Vale che $L = \varnothing$ sse $\text{valid}(M) = \varnothing$ sse $(\text{valid}(M))^C = \Delta^*$. Se esistesse un algoritmo per determinare se un $L \in \text{CFL} = \Delta^*$, allora potremmo determinare che $L(M) = \varnothing$.

Corollario 2 (Problema dell'universalità)**.** *Dato $L \in CFL$ non è possibile determinare (è indecidibile) se $L = \Sigma^*$.*

Invece visto che i DCFL sono chiusi rispetto al complemento, basta costruire l'automa a pila per un linguaggio e vedere se il suo complemento riconosce il linguaggio vuoto.

Corollario 3 (Problema dell'equivalenza)**.** *Dati $L_1, L_2 \in CFL$ non è possibile decidere se $L_1 = L_2$.*

Questo perché il problema dell'universalità è un caso particolare dell'equivalenza ($L_1 = \Delta^*$).

Corollario 4 (Problema dell'equivalenza con regolari)**.** *Dati $L \in CFL$ e $L \in Reg$, non è possibile decidere se $L = R$.*

La motivazione è la stessa di sopra.

Invece dati due DPDA è possibile determinare se sono equivalenti.

Un automa a pila con due pile ha la stessa potenza computazionale di una macchina di Turing.

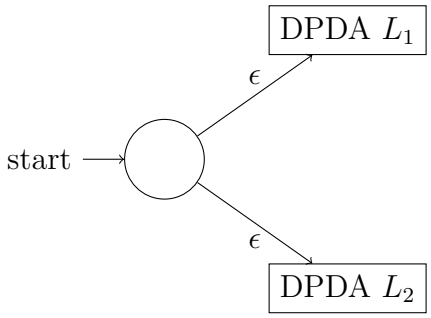
Teorema 8. *Esistono L_1, L_2 linguaggi CF deterministici, tali che*

$$valid(M) = L_1 \cap L_2$$

ed esiste un algoritmo che, data la macchina M, ci costruisce due automi a pila deterministici per L_1 e L_2 (o grammatiche non ambigue).

Corollario 5. *$L_1 \cap L_2 = \varnothing$ sse valid$(M) = \varnothing$ sse $L(M) = \varnothing$. Visto che non è possibile capire se un linguaggio di tipo 0 è vuoto, non è decidibile sapere se l'intersezione di due linguaggi CF (anche determinisitic) è vuota.*

Se invece analizziamo $L_1 \cup L_2$, questo contiene sia computazioni valide che non. Prendiamo $\beta \in \Delta^*$, e supponiamo che $\beta \in L_1 \cup L_2$. Definiamo allora il PDA $\mathcal{A}$



se la stringa non viene accettata, allora β sicuramente non appartiene a $L(M)$. Altrimenti se viene accettata abbiamo alcuni casi

- se $\beta \in L_1 \cap L_2$ allora abbiamo 2 computazioni accettanti
- se $\beta \in (L_1 \setminus L_2$ o $\beta \in (L_2 \setminus L_1)$, allora c'è solo una computazione accettante

Allora vale che $\mathcal{A}$ è ambiguo sse $L_1 \cap L_2 = \varnothing$. Quindi $\mathcal{A}$ non è ambiguo sse $L_1 \cap L_2 = \varnothing = \text{valid}(M)$ sse $L(M) = \varnothing$.

Teorema 9. *Non esiste algoritmo che, dato un PDA $\mathcal{A}$, decide se $\mathcal{A}$ è ambiguo.*

Lo stesso vale per le grammatiche CF.
Dati due linguaggi L_1 e L_2, il loro quoziente è definito come

$$L_1/L_2 = \{x \in \Sigma^* \mid \exists y \in L_2 . xy \in L1\}$$

Supponiamo che $L_2 = \{z\}$, allora

$$L/z = \{x \in \Sigma^* \mid xz \in L\}$$

avevamo dimostrato che se L_1 e L_2 sono regolari, allora anche il loro quoziente è regolare.

Definiamo il problema della regolarità, come il problema che dato un linguaggio $L \in$ CFL, ci domandiamo se L è regolare. Prendiamo $G = \langle V, \Sigma, P, S\rangle$ una grammatica CF. Chiamiamo $L = L(G)$. Noi sappiamo che esistono dei linguaggi CF che non sono regolari, sia L_0 uno di questo, per cui

$$L_0 \in \text{CFL} \setminus \text{Reg}$$

generato da una gramamtica G_0. Costruiamo ora

$$L_1 = L_0\#\Sigma^* \cup \Sigma^*\#L$$

con $\# \notin \Sigma$. Ora

- se $L = \Sigma^*$, allora $L_1 = L\#\Sigma^* \cup \Sigma^*\#\Sigma^* = \Sigma^*\#\Sigma^*$, e questo linguaggio è regolare.
- se $L \neq \Sigma^*$, sia $w \notin L$, calcoliamo

  $$L_1/\#w = L_0$$

  siccome $w \notin L$, tutte le stringhe di $\Sigma^*\#L$ non sono nel quoziente.

Visto che ??? sono chiusi rispetto al quoziente, allora $L_1 \in \text{CFL}\setminus\text{Reg}$. Allora $L = \Sigma^*$ sse $L_1 \in$ Reg. Quindi se so decidere la regolarità di un CF, saprei decidere anche la sua universalità.

Definiamo ora il linguaggio delle computazioni valide su un input fissato

$$\text{valid}(M, w), w \in \Sigma^*$$

Questo non cambia rispetto al valid(M), e bisogna solo controllare che la computazione inizi per w. Anche il suo complemento rimane invariato e vale che

$$\begin{aligned}(\text{valid}(M,w))^C &= \Delta^* \setminus \text{valid}(M,w) \\ &= \begin{cases} \Delta^* & \text{se } w \notin L(M) \\ \Delta^* \setminus \{p\} & \text{se } w \in L(M) \text{ e } p \text{ codifica la computazione che accetta } w \end{cases} \in \text{Reg}\end{aligned}$$

Teorema 10. *Non esiste un algoritmo che dato PDA o CFG per linguaggio regolare, produce un automa a stati finiti equivalente.*

Dimostrazione. Sia M una macchina di Turing che riconosce un linguaggio ricorsivamente enuemrabile, ma non ricorsivo. Sia $w \in \Sigma^*$, costruisco la grammatica G_w per $(\text{valid}(G, w))^C$, un linguaggio regolare.

- se $w \in L(M)$, allora $L(G_w) = \Delta^* \setminus \{\beta\} \neq \Delta^*$
- se $w \notin L(M)$, allora $L(G_w) = \Delta^*$

vale quindi che $w \in L(M)$ sse $L(G_w) = \Delta^*$. Quindi non si può costruire l'automa a stati finiti. ■

Vale inoltre che data una grammatica CF con n simboli che genera un linguaggio regolare, non c'è limite al numero di stati che l'automa regolare corrispondente. Infatti se ci fosse un upper bound potremmo esplorare tutti gli automi costruibili finché non si trova quello corretto. Questo si chiama *tradeoff*. Abbiamo visto che passare da DFA nondeterminisitici a DFA deterministici a un tradeoff esponenziale. In questo secondo caso invece il tradeoff è illimitato.

Decidibilità

	Reg	DFCL	CF
$L \stackrel{?}{=} \varnothing$	D	D	D
L è finito?	D	D	D
$L \stackrel{?}{=} \Delta^*$	D	D	U
$L_1 \stackrel{?}{=} L_2$	D	D	U
$L \stackrel{?}{\in} \text{Reg}$	/	D	U
$L_1 \cap L_2 \stackrel{?}{=} \varnothing$	D	U	U