Orale Linguaggi e Computabilità

UniShare

Davide Cozzi
@dlcgold

Indice

Capitolo 1

Linguaggi

- un **linguaggio** è un insieme di stringhe che può essere generato mediante un dato meccanismo con delle date caratteristiche; un linguaggio può essere riconosciuto, ovvero dando in input una stringa un meccanismo può dirmi se appartiene o meno ad un linguaggio. I meccanismi che generano linguaggi si chiamano *grammatiche*, quelli che li riconoscono *automi*. I linguaggi formali fanno parte dell'informatica teorica *(TCS)*

- si definisce **alfabeto** come un insieme finito e non vuoto di simbolo (come per esempio il nostro alfabeto o le cifre da 0 a 9). Solitamente si indica con Σ o Γ

- si definisce **stringa** come una sequenza finita di simboli (come per esempio una parola o una sequenza numerica). La stringa vuota è una sequenza di 0 simboli, e si indica con ε o λ

- si definisce **lunghezza di una stringa** il numero di simboli che la compone (ovviamente contando ogni molteplicità). Se si ha $w \in \Sigma^*$ è una stringa w con elementi da Σ^* (insieme di tutte le stringhe di tutte le lunghezze possibili fatte da Σ), allora $|w|$ è la lunghezza di w, inoltre $|\varepsilon| = 0$.

- si definisce **potenza di un alfabeto** Σ^k come l'insieme di tutte le sequenze (espressi come stringhe e non simboli) di lunghezza $k \in \mathbb{N}$, $k > 0$ ottenibili da quell'alfabeto (se Σ^2 si avranno tutte le sequenza di 2 elementi etc...). Se ho $k = 1$ si ha $\Sigma^1 \neq \Sigma$ in quanto ora ho stringhe e non simboli. Se ho $k = 0$ ho $\Sigma^0 = \varepsilon$. Dato k ho $|\Sigma|$ che è la cardinalità dell'insieme Σ (e non la sua lunghezza come nel caso delle stringhe); sia $w \in \Sigma^k = a_1, a_2, ..., a_k$, $a_i \in \Sigma$ e $|\Sigma| = q$ ora:

$$|\Sigma^k| = q^k$$

- si definisce Σ^* come **chiusura di Kleene** che è l'unione infinita di Σ^k ovvero
$$\Sigma* = \Sigma^0 \cup \Sigma^1 \cup ... \cup \Sigma^k$$
- si ha che Σ^+ è l'unione per $k \geq 1$ di Σ^k ovvero:
$$\Sigma+ = \Sigma^1 \cup \Sigma^2 \cup ... \cup \Sigma^k = \Sigma^* - \Sigma^0$$
- quindi un **linguaggio** L è un insieme di stringhe e:
$$L \subseteq \Sigma^*$$
si hanno sottoinsiemi particolari, come l'insieme vuoto, che resta però un linguaggio, il **linguaggio vuoto** e $\emptyset \in \Sigma^k$, $|\emptyset| = 0$ che è diverso dal linguaggio che contiene la stringa vuota $|\varepsilon| = 1$ (che conta come una stringa). Inoltre $\Sigma^* \subseteq \Sigma^*$ che ha lunghezza infinita. Posso concatenare due stringhe con un punto: $a \cdot b \cdot c = abc$ e $a \cdot \varepsilon = a$. Ovviamente la stringa concatenata è lunga come la somma delle lunghezze delle stringhe che la compongono. Vediamo qualche esempio di linguaggio:
  - Σ^* è un linguaggio per ogni alfabeto Σ
  - $\emptyset$, il linguaggio vuoto, e $\{\varepsilon\}$ sono un linguaggio rispetto a qualunque alfabeto

Si ha che una grammatica G è una quadrupla $G = (V, T, P, S)$ con:

- V simboli variabili
- T simboli terminali, ovvero i simboli con cui si scrivono le stringhe alla fine
- P regole di produzione
- S variabile di partenza *start*

Si hanno 4 grammatiche formali, *gerarchia di Chomsky*:

- **tipo 0:** non si hanno restrizioni sulle regole di produzione, $\alpha \to \beta$. Sono linguaggi ricorsivamente numerabili e sono rappresentati dalle *macchine di Turing*, deterministiche o non deterministiche (la macchina di Turing è un automa)

- **tipo 1:** il lato destro della produzione ha lunghezza almeno uguale a quello sinistro. Sono grammatiche dipendenti dal contesto (*contestuali*) e come automa hanno *la macchina di Turing che lavora in spazio lineare*:
$$\alpha_1 A \alpha_2 \to \alpha_1 B \alpha_2$$
con α_1 e α_2 detti *contesto* e $\alpha_1,\ \alpha_2,\ \beta \in (V \cup T)^*$
- **tipo 2:** sono quelle libere dal contesto, context free. Come regola ha $A \to \beta$ con $A \in V$ e $\beta \in (V \cup T)^*$ e come automa ha gli *automi a pila non deterministici*
- **tipo 3:** sono le grammatiche *regolari*. Come regole ha $A \to \alpha B$ (o $A \to B\alpha$) e $A \to \alpha$ con $A, B \in V$ e $\alpha \in T$. Come automi ha gli *automi a stato finito deterministici o non deterministici*

Definizione 1. *Prendo una grammatica $G = (V, T, P, S)$, grammatica CFG. Se $\alpha A \beta$ è una stringa tale che $\alpha, \beta \in (V \cup T)^*$, appartiene sia a variabili che terminali. Sia $A \in V$ e sia $a \to \gamma$ una produzione di G. Allora scriviamo:*
$$\alpha A \beta \to \alpha \gamma \beta$$
con $\gamma \in (V \cup T)^*$.
Le sostituzioni si fanno indipendentemente da α e β. Questa è quindi la definizione di ***derivazione****.*

Definizione 2. *Definisco il simbolo $\to_*$, ovvero il simbolo di derivazioni in 0 o più passi. Può essere definito in modo ricorsivo. Per induzione sul numero di passi.*

- *la base dice che $\forall \alpha \in (V \cup T)^*$, $\alpha \to_* \alpha$*
- *il passo è: se $\alpha \to_{G_*} \beta$ e $\beta \to_{G_*} \gamma$ allora $\alpha \to_* \gamma$*

Si può anche dire che $\alpha \to_{G_*} \beta$ sse esiste una sequenza di stringhe $\gamma_1, ..., \gamma_n$ con $n \geq 1$ tale che $\alpha = \gamma_1$, $\beta = \gamma_n$ e $\forall i,\ 1 < i < n-1$ si ha che $\gamma_1 \to \gamma_{i+1}$ la derivazione in 0 o più passi è la chiusura transitiva della derivazione

Definizione 3. *avendo ora definito questi simboli possiamo definire una forma sentenziale. Infatti è una stringa α tale che:*
$$\forall \alpha \in (V \cup T)^* \text{ tale che } S \to_{G_*} \alpha$$

Definizione 4. *data $G = (V, T, P, S)$ si ha che $L(G) = \{w \in T^* | \, S \to_{G_*} w\}$ ovvero composto da stringhe terminali che sono derivabili o 0 o più passi.*

Teorema 1. *data la grammatica $G = \{V, T, P, S)$ CFG e $\alpha \in (V \cup T)^*$. Si ha che vale $S \to_* \alpha$ sse $S \to_{lm_*} \alpha$ sse $S \to_{rm_*} \alpha$. Con $\to_{lm_*}$ simbolo di left most derivation e $\to_{rm_*}$ simbolo di right most derivation*

1.0.1 Alberi Sintatici

Definizione 5. *Data una grammatica CFG, $G = \{V, T, P, S\}$ un* ***albero sintattico*** *per G soddisfa le seguenti condizioni:*

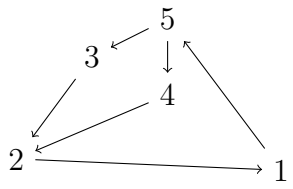
- *ogni nodo interno è etichettato con una variabile in V*
- *ogni foglia è anch'essa etichettata con una variabile o col simbolo di terminale T o con la stringa vuota ε (in questo caso la foglia è l'unico figlio del padre)*
- *se un nodo interno è etichettato con A i suoi figli saranno etichettati con X1, ..., Xk e $A \to X1, ..., Xk$ sarà una produzione di G in P. Se un X_i è ε sarà l'unica figlio e $A \to \varepsilon$ sarà comunque una produzione di G*

La concatenazione in ordine delle foglie viene detto ***prodotto dell'albero***

Data una CFG si ha che i seguenti cinque enunciati si equivalgono:

1. la procedura di inferenza ricorsiva stailisce che una stringa w di simboli terminali appartiene al linguaggio $L(A)$ con A variabile
2. $A \to^* w$
3. $A \to^*_{lm} w$
4. $A \to^*_{rm} w$
5. esiste un albero sintattico con radice A e prodotto w

queste 5 proposizioni si implicano l'uni l'altra:



da 1 a 5. si procede per induzione:

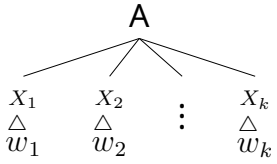
- **caso base:** ho un livello solo (una sola riga), $\exists A \to w$:

- **caso passo:** suppongo vero per un numero di righe $\leq n$, lo dimsotro per $n+1$ righe:

$$A \to X_1, X_2, ..., X_k$$

$$w = w_1, w_2, ..., w_k$$

ovvero, in meno di $n+1$ livelli:



□

da 5 a 3. procedo per induzione:

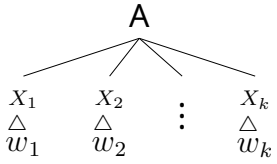
- **caso base (n=1):** $\exists A \to w$ quindi $A \to_{lm} w$, come prima si ha un solo livello:

- **caso passo:** suppongo che la proprierà valga per ogni albero di profondità minore uguale a n, dimostro che valga per gli alberi profondi $n+1$:

$$A \to X_1, X_2, ..., X_k$$

$$w = w_1, w_2, ..., w_k$$

ovvero, in meno di $n+1$ livelli:



$$A \to_{lm} X_1, X_2, ..., X_k$$

$x_1 \to^*_{lm} w_1$ per ipotesi induttiva si ha un albero al più di n livelli

quindi:

$$A \to_{lm} X_1, ..., X_k \to^*_{lm} w_1, X_2, ..., X_k \to^*_{lm} ... \to^*_{lm} w_1, ..., w_k = w$$

□

Definizione 6. *Una grammatica è definita ambigua se esiste una stringa w di terminali che ha più di un albero sintattico.*
Possono esserci più derivazioni di una stringa ma l'importante è che non ci siano alberi sintattici diversi. Capire se una CFG è ambigua è un problema indecidibile

Teorema 2. *Per ogni CFG, con $G = (V, T, P, S)$, per ogni stringa w di terminali si ha che w ha due alberi sintattici distinti sse ha due derivazioni sinistre da S distinte.*
Se la grammatica non è ambigua allora esiste un'unica derivazione sinistra da S

Definizione 7. *Un linguaggio L è inerentemente ambiguo se tutte le grammatiche CFG per tale linguaggio sono a loro volta ambigue*

1.0.2 Grammatiche Regolari

Sono le grammatiche che generano i linguaggi regolari (quelli del terzo tipo) che sono casi particolari dei CFL.
Si ha la solita grammatica $G = (V, T, P, S)$ con però vincoli su P:

- ε si può ottenere solo con $S \to \varepsilon$
- le produzioni sono tutte lineari a destra ($A \to aA$ o $A \to a$) o a sinistra ($A \to Ba$ o $A \to a$)

Espressioni Regolari

le regex sono usate per la ricerca di un pattern in un testo o negli analizzatori lessicali. Una regex denota il linguaggio e non la grammatica. Si hanno le seguenti operazioni tra due linguaggi L e M:

- **unione:** dati L, $M \in \Sigma^*$, l'unione $L \cup M$ è l'insieme delle stringhe che si trovano in entrambi i linguaggi o solo in uno dei due

  si ha che:
  $$L \cup M = M \cup L$$

- **concatenazione:** dati L, $M \in \Sigma^*$, la concatenazione $L \cdot M$ (o LM) è lisieme di tutte le stringhe ottenibili concatenandone una di L a una di M

  si ha che:
  $$L \cdot M \neq M \cdot L$$

- si definiscono:
  - $L \cdot L = L^2$, $L \cdot L \cdot L = L^3$ etc...
  - $L^1 = L$
  - $L^0 = \{\varepsilon\}$
- **chiusura di Kleene:** dato $L \subseteq \Sigma^*$ si ha che la chiusura di Kleen di L è:
$$L^* = \bigcup_{i \geq 0} L^i$$
ricordando che $l^0 = \varepsilon$ vediamo dei casi particolari:
  - $L = \{0^n | \, n \geq 0\}$ implica $|L| = \infty$ e quindi, essendo $L^i = L$, $i \geq 1$ e quindi $|L^i| = \infty$, $|L^*| = \infty$. Si ha quindi:
$$L^* = L^0 \cup L^1 \cup ... \cup L^i = L$$
  - $L = \emptyset$ implica $L^0 = \{\varepsilon\}$, $L^2 = L \cdot L = \emptyset$ e così via per ogni concatenazione di L. Si ha quindi:
$$L^* = L^0 = \{\varepsilon\}$$
  - $L = \{\varepsilon\}$ implica $L^0 = \{\varepsilon\} = L = L^1 = L^2 = ...$, si ha quindi:
$$L^* = \{\varepsilon\} = L$$

  L'insieme vuoto e l'insieme contenente la stringa vuota hanno le uniche chiusure di kleene finite

Si riporta la definizione ricorsiva di un'espressione regolare:

- **casi base:** si hanno tre casi base:
  1. ε e $\emptyset$ sono espressioni regolari
  2. se $a \in \Sigma$, con a che è un'esprssione regolare, $L(a) = \{a\}$
  3. le variabili che rappresentano linguaggi regolari sono espressioni regolari, $L(L) = L$
- **casi passo:** si hanno i 4 casi passo:
  1. **unione:** se E e F sono espressioni regolari allora anche $E + F = E \cup F$ è un'espressione regolare e si ha:
$$L(E + F) = L(E) \cup L(F)$$

2. **concatenazione:** se E e F sono espressioni regolari allora anche $EF = E \cdot F$ è un'espressione regolare e si ha:

$$L(EF) = L(E) \cdot L(F)$$

3. **chiusura:** se E è un'espressione regolare allora E^* è un'espressione regolare e si ha:

$$L(E^*) = (L(E))^*$$

4. **parentesi:** se Eè un'espressione regolare allora (E) è un'espressione regolare e si ha:

$$L((E)) = L(E)$$

Si ha una precedenza degli operatori, in ordine di precedenza (si valuta da sinistra a destra):

1. chiusura di Kleene *
2. concatenazione $\cdot$, che è associativo $((E \cdot F) \cdot G = E \cdot (F \cdot G))$ ma non è commutativo $(E \cdot F \neq F \cdot E)$
3. unione + che è associativa $((E+F)+G = E+(F+G))$ ed è commutativo $(E + F = F + E)$
4. infine le parentesi

si hanno anche delle proprietà algebriche:

- due espressioni regolari sono equivalenti se denotano le stesso linguaggio
- due espressioni regolari con variaboli sono equivalenti se lo sono $\forall$ assegnamento alle variabili
- l'unione è commutativa e associativa, la concatenazione è solo associativa
- si definiscono:
  - **identità:** ovvero un valore unito all'identità è pari a se stesso (elemento neutro della somma $0+x = x+0 = x$). $\emptyset$ è identità per l'unione $(\emptyset + L = L + \emptyset = L)$, $\{\varepsilon\}$ è identità per la concatenazione $(\varepsilon L = L\varepsilon = L)$
  - **annichilitore:** ovvero un valore concatenato all'annichilatore da l'annichilitore (l'elemento nullo del prodotto $0x = x0 = 0$). $\emptyset$ è l'annichilitore per la concatenazione $(\emptyset L = L\emptyset = \emptyset)$

- **distributività:** dell'unione rispetto alla concatenazione (che non è commutativa):
  - **distributività sinistra:** $L(M + N) = LM + LN$
  - **distributività destra:** $(M + N)L = ML + NL$
- **idempotenza:** $L + L = L$
- $(L^*)^* = L^*$
- $\emptyset^* = \varepsilon$ infatti $L(\emptyset) = \{\varepsilon\} \cup L(\emptyset) \cup L(\emptyset) \cdot L(\emptyset) \cup ... = \{\varepsilon\} \cup \emptyset \cup \emptyset ... = \varepsilon$
- $\varepsilon^* = \varepsilon$ infatti $L(\varepsilon^*) = \{\varepsilon\} \cup L(\varepsilon) \cup L(\varepsilon) = \{\varepsilon\} \cup \{\varepsilon\} \cup ... = \{\varepsilon\} = L(\varepsilon)$
- $L^+ = L \cdot L^* = L^* \cdot L$ (quindi con almeno un elemento che non sia la stringa vuota)
- $L^* = l^+ + \varepsilon$

1.1 Automi

un automa a stati finiti ha un insieme di stati e un controllo che si muove da stato a stato in risposta a input esterni. Si ha una distinzione:

- **automi deterministici:** dove l'automa non può essere in più di uno stato per volta
- **automi non deterministici:** dove l'automa può trovarsi in più stati contemporaneamente

1.1.1 Automi deterministici

Un automa a stati finiti deterministico (*DFA*), un automa che dopo aver letto una qualunque sequenza di input si trova in un singolo stato. Il termine *deterministico* concerne il fatto che per ogni input esiste un solo stato verso il quale l'automa passa dal suo stato corrente. Un automa a stati finiti deterministico consiste nelle seguenti parti:

- un insieme finito di stati, spesso indicato con Q
- un insieme finito di simboli di input , spesso indicato con Σ

- una funzione di transizione, che prende come argomento uno stato e un simbolo di input e restituisce uno stato. La funzione di transizione sarà indicata comunemente con δ. Nella rappresentazione grafica informale di automi δ è rappresentata dagli archi tra gli stati e dalle etichette sugli archi. Se q è uno stato e a è un simbolo di input, $\delta(q, a)$ è lo stato p tale che esiste un arco etichettato con a da q a p[2]
- uno stato iniziale, uno degli stati in Q
- un insieme di stati finali, o accettanti , F. L'insieme F è un sottoinsieme di Q

Nel complesso un DFA è rappresentato in maniera concisa con l'enumerazione dei suoi elementi, quindi con la quintupla:

$$A = (Q, \Sigma, \delta, q_0, F)$$

con A nome del DFA, Q insiem degli stati, Σ rappresentante i simboli di input, δ la sua funzione di transizione, q_0 il suo stato iniziale e F l'insieme degli stati accettanti.
Vediamo come decidere se accettare o meno una stringa (sequenza di caratteri) in input mediante un DFA.
Ho una sequenza in input $a_1...a_n$. Parto dallo stato iniziale q_0, consultando la funzione di transizione δ, per esempio $\delta(q_0, a_1) = q_1$ e trovo lo stato in cui il DFA entra dopo aver letto a_1. Poi passo a $\delta(q_1, a_2) = q_2$ e così via, $\delta(q_{i-1}, a_i) = q_i$ fino a ottenere q_n. Se q_n è elemento di F allora $a_1...a_n$ viene accettato, altrimenti viene rifiutato.

1.1.2 Automi non deterministici

Un automa a stati finiti non deterministici (*NFA*) può trovarsi in diversi stati contemporaneamente. Come i DFA accettano linguaggi regolari e spesso sono più semplici da trattare rispetto ai DFA.
Un NFA è definito come un NFA ma si ha un diverso tipo di transizione δ, che ha sempre come argomenti uno stato e un simbolo di input ma restituisce zero o più stati.
Definisco quindi un NFA come una quintupla:

$$A = (Q, \Sigma, \delta, q_0, F)$$

con, a differenza dei DFA:

$$\delta : Q \times F \to 2^Q$$

Possiamo ora definire$\hat{\delta}$, delta cappuccio che prende in ingresso uno stato e l'intera stringa w. Definisco ricorsivamente:

- **caso base:** se $|w| = 0$ ovvero se $W = \varepsilon$ si ha:

$$\hat{\delta}(q, \varepsilon) = \{q\}$$

- **caso passo:** se $|w| > 0$, allora $W = xa$, $a \in \Sigma$ e $x \in \Sigma^*$. Posto $\hat{\delta}(q, x) = \{p_1, ..., p_k\}$ si ha:

$$\hat{\delta}(q, w) = \cup\delta(p_i, a)$$

Per il linguaggio L accettato dall'automa si ha:

$$L(A) = \{w \in \Sigma^* | \, \hat{\delta}(q_0, q) \cap F \neq \emptyset\}$$

Troviamo ora un algoritmo che trasformi un NFA in un DFA.
Definiamo questo algoritmo che avrà:

- come input un NFA $N = (Q_n, \Sigma, \delta_N, q_0, F_N)$
- come output un DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ tale che $L(D) = L(N)$

con:

- $Q_D = 2^{Q_N}$ (quindi se $Q_N = n$ si ha $|Q_D| = 2^n$) ma gli stati non accessibili possono essere eliminati
- $F_D = \{S \subseteq Q_n | \, S \cap F_N \neq \emptyset\}$
- $\forall S \subseteq Q_N$ e $\forall a \in \Sigma$:

$$\delta_D(S, a) = \cup\delta_n(p, a)$$

  per esempio:

$$\delta_D(\{q_0, q_1, q_2\}, 0) = \delta_N(q_0, 0) \cup \delta_N(q_1, 0) \cup \delta_N(q_2, 0)$$

Si definisce l'$\varepsilon - NFA$, l'automa astati finiti non deterministici con ε transizioni. Con la transizione ε posso saltare i nodi, ovvero avanza senza aggiungere caratteri. Si definisce la funzione $ECLOSE : Q \to 2^Q$, con $ECLOSE(q)$ insieme degli stati raggiungibili da q tramite $\varepsilon - mosse$. si ha inoltre che:

- $ECLOSE2^Q \to 2^Q \;\; P \subseteq Q$
- $ECLOSE(P) = \cup ECLOSE(p)$

- $ECLOSE(\emptyset) = \emptyset$

torniamo a dare bene qualche definizione per $\hat{\delta}$ in un DFA:

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

con: $q \in Q,\ \ w \in \Sigma^*\ \ e\ \ \hat{\delta}\ (p, w) = q$

- **caso base:** $w = \varepsilon \to |w| = 0 \to \hat{\delta}\ (q, \varepsilon) = q$
  caso passo: $|w| \neq 0 \to w = ax$ con $a \in \Sigma, x \in \Sigma^*$: $\hat{\delta}\ (q, w) = \hat{\delta}\ (q, ax) = \hat{\delta}\ (\delta(q, a), x)$

- **caso base:** $w = \varepsilon \to |w| = 0 \to \delta(q, w) = \delta(q, \varepsilon) = q$
  caso passo: $|w| \neq 0 \to w = xa$ con $a \in \Sigma, x \in \Sigma^*$: $\hat{\delta}\ (q, w) = \hat{\delta}\ (q, xa) = \hat{\delta}\ (\delta(q, x), a)$

in un NFA si ha:

- **caso base:** $\hat{\delta}\ (q, \varepsilon) = \{q\},\ \forall q \in Q$

- **caso passo:** posto $w = ax$ e $\hat{\delta}\ (q, a) = \{p_1, ..., p_n\}$ allora $\hat{\delta}\ (q, w) = \cup$ $\hat{\delta}\ (p_i, x) = \{r_1, ..., r_n\}$.
  $\hat{\delta}\ (q, a) = p$
  $\hat{\delta}\ (q, w) = \hat{\delta}\ (q, xa) = \hat{\delta}(p, x) = r$

oppure:

- **caso base:** $\hat{\delta}\ (q, \varepsilon) = \{q\}$, $\forall q \in Q$

- **caso passo:** posto $w = xa$ si ha $\hat{\delta}(q, q) = (q, xa) = \cup\delta(p, a)$ con $\hat{\delta}$ $(q, x) = \{p_1, ..., p_n\}$

Se ho un NFA $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ con $\delta_N : Q \times \Sigma^* \to 2^{Q_N}$ che accetta un lingiaggio L posso ottenere un DFA $D = (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$ equivalente con $Q_D = 2^{Q_N}$ e $q_{0_D} = \{q_{0_N}\}$ che accetta L.
$\forall S \subseteq Q_N$ e $\forall a \in \Sigma$ si ha:

$$F_D = \{S \subseteq Q_N \,|\, S \cap F_n \neq \emptyset\}$$

con $\delta_D(S, a) = \cup\delta_N(p, a)$ Si ha che:

$$|Q_D| = |2^{Q_N}| = 2^{|Q_N|}$$

Si ha che dato un ε-NFA $E = (Q_E, \Sigma, \delta_E, q_{0_E}, F_E)$ che riconosce L in un NFA $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ equivalenti con $Q_N = 2^{Q_E}$ stati in Q_E ε-close: *ECLOSE(S)=S* e $q_N = ECLOSE(q_E)$:

$$F_N\{S \in Q_F,\ S \subseteq Q_E | S \cap F_E \neq \emptyset\}$$

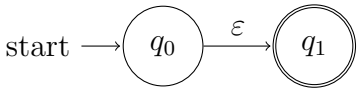
quindi:

$$\forall a \in \Sigma \text{ e } \forall S = \{p_1, ..., p_k\}, \forall S \in Q_F \text{ e } Q_E$$
$$\delta_F(S, a) \text{ si ottiene } \cup \delta_E(p_i, a) = \{r_1, ..., r_n\}$$
$$\delta_N(S, a) = ECLOSE(\{r_1, ..., r_n\})$$

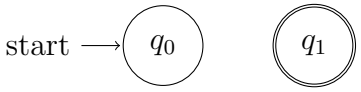
1.1.3 Da espressioni regolari a automi E-NFA

- **caso base**:

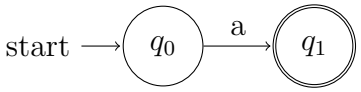
  1. se $R = \varepsilon$ ovvero $L(R) = \{\varepsilon\}$ allora:



  2. se $R = \emptyset$ ovvero $L(R) = \emptyset$ allora:

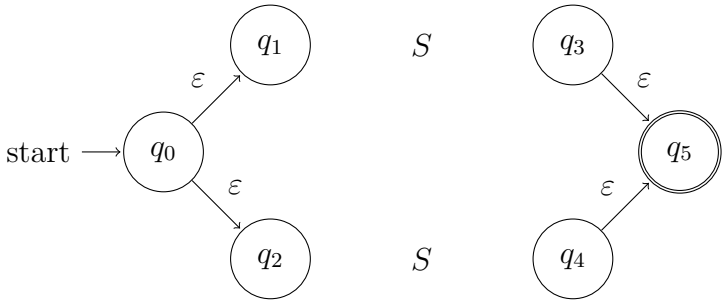


  3. se $R = a$ ovvero $L(R) = \{a\}$ allora:

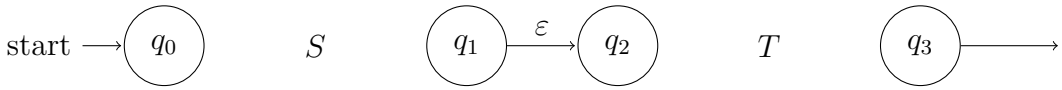


- **caso passo:**

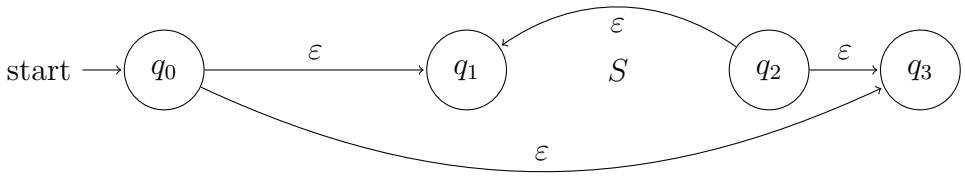
  1. se $R = S + T$ quindi $L(R) = L(S) + L(T)$ allora:



2. se $R = ST$ quindi $L(R) = L(S)L(T)$:



3. se $R = S^*$ quindi $L(R) = (L(S))^*$:



Vediamo ora l'algoritmo che trasforma DFA in una ER. Questo algoritmo permette di verificare quale linguaggio è accettato o meno dall'automa:
dato DFA $A = (Q, \Sigma, \delta, q_0, F)$ con Q e F stati non finali e Q, F, q_0 che sono i primi stati da eliminare. L'algoritmo procede per eliminazioni successive. Si costruisce quindi l'automa B che ha solo q_0 e $F = \{q_1, ..., q_k\}$. Scrivo quindi K espressioni regolari considerando solo q_0 e il k-esimo stato finale eliminando pian piano gli altri stati. Alla fine ottengo le varie espressioni regolari da unire:

$$E = E1 + E2 + \cdots + Ek$$

1.1.4 Chiusura di un Linguaggio regolare

Sia REG la classe dei linguaggi regolari (ovvero ogni linguaggio regolare su un alfabeto finito non vuoto). Si ha la seguente proprietà:

$$L, M \in REG \rightarrow L \cup M \in REG$$

si può dimostrare in due maniere:

- **con le espressioni regolari:**

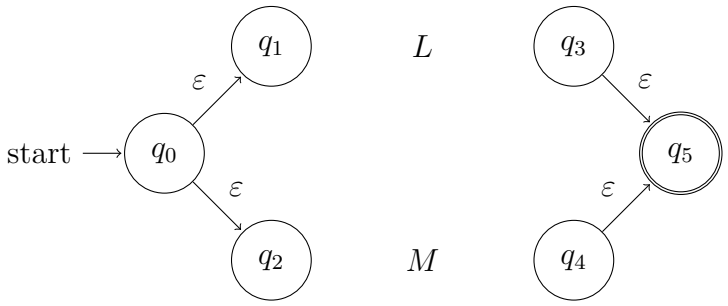
  $$L, M \in REG \rightarrow \exists R, S \text{ ER tali che } L(R) = L \text{ e } L(S) = M$$

  $$L \cup M = L(R + S) \text{ e quindi appartenente a } REG$$

- **con gli automi:**

  se $L, M \in REG \rightarrow \exists \varepsilon - NFA$ che con L e M va dallo stato iniziale a quello finale

  :



accetta $L \cup M$ che quindi è *REG*

Inoltre siano due i due alfabeti $\Sigma \subseteq \Gamma$, si ha che:

$$\text{se L è regolare su } \Sigma \rightarrow \text{ L è regolare su } \Gamma$$

inoltre:

$$\text{se L è REG su } \Sigma_1,\ \text{M è REG su } \Sigma_2,\ \text{allora } L \cup M \text{ è REG su } \Sigma_1 \cup \Sigma_2$$

Teorema 3. *Si ha che:*
Se L e M sono linguaggi regolari allora LM, ovvero la concatenazione è regolare.
Se L è un linguaggio regolare allora L^* è regolare

Teorema 4. *se $L \in REG$ su Σ allora $\overline{L} = \Sigma^* - L \in REG$*

Dimostrazione. se $L \in REG$ allora $\exists$ DFA $A = (Q, \Sigma, \delta, q_0, F)$ tale che $L(A) = L$. Costruisco $B = (Q, \Sigma, \delta, q_0, Q - F)$, $L(B) = \overline{L}$.

□

si osserva che:

$$L \in REG \text{ su } \Sigma \text{ e } \Sigma \subseteq \Gamma$$

inoltre:

$$\overline{L} = \Sigma^* - L \text{ oppure } \overline{L} = \Gamma^* - L$$

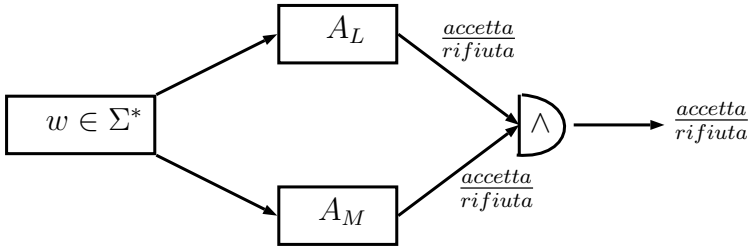
Se ho un espressione regolare per L e voglio ottenere un'espressione regolare per $\overline{L}$ devo ottenere un $\varepsilon - NFA$ che devo convertire in DFA. A quel punto complemento con F e ottengo un'espressione regolare per $\overline{L}$.
Passiamo ora all'intersezione:

Teorema 5. *se L e M sono regolari allora la loro intersezione è regolare*

Dimostrazione.

$$L, M \in REG \rightarrow \exists A_L, A_M(DFA) \text{ tali che } L(A_L) = L \text{ e } L(A_M) = M$$

e quindi:



□

Passiamo al problema della **chiusura dei linguaggi regolari rispetto all'intersezione**, ovvero che due automi che usano entrambi contemporaneamente una certa stringa in input, il ché non è possibile. Si risolve usando l'**automa prodotto**:

$$A = (Q_L \times Q_M, \Sigma, \delta, (q_{OL}, q_{OM}), F_L \times F_M)$$

con:

$$(\delta(p,q),a) = (\delta_L(p,a), \delta_M(q,a))$$
$$A_L = (Q_L, \Sigma, \delta_L, q_{OL}, F_L)$$
$$A_M = (Q_M, \Sigma, \delta_M, q_{OM}, F_M)$$

1.1.5 Minimizzazione

definiamo la **relazione di equivalenza tra stati**:

Definizione 8. *Sia A un DFA, $A = (Q, \Sigma, \delta, q_0, F)$. Siano:*

$$p, q \in Q \to p \approx q \text{ vale se } \forall w \in \Sigma^* \quad \hat{\delta}(p,w) \in F \longleftrightarrow \hat{\delta}(q,w) \in F$$

inoltre p e q sono distinguibili se $p \not\approx q$ ovvero:

$$\exists w \in \Sigma^* \text{ tale che:}$$

$$\hat{\delta}(p,w) \in F \text{ e } \hat{\delta}(q,w) \notin F \text{ o } \hat{\delta}(p,w) \notin F \text{ e } \hat{\delta}(q,w) \in F$$

quindi si ha una **relazione di equivalenza** e quindi si hanno le tre proprietà:

- **riflessività:** $\forall p \in Q, p \approx p$
- **simmetricità:** $\forall p, q \in Q, p \approx q \to q \approx p$
- **transitività:** $\forall p, q, r \in Q, p \approx q \vee q \approx r \to p \approx r$

e quindi **in ogni classe di equivalenza ci sono stati tra loro equivalenti** due stati, uno finale e uno non finale, sono sicuramente distinguibili, basti pensare alla stringa vuota.

Teorema 6. *due stati non distinti dall'algoritmo riempi-tabella sono equivalenti. Se ne calcola la complessità:*

$$|Q| = n \to \frac{n(n-1)}{2} \; caselle = O(n^2)$$

se ho una crocetta a iterazione ho il caso peggiore $n^2n^2 = O(n^4)$

quindi per vedere se due linguaggi regolari sono equivalenti si ha che:

$$L, M \in REG$$

$$\exists A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$$
$$\exists A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$$

costruisco $A = (Q_L \cup Q_M, \Sigma, \delta, q_L, F_L \cup F_M)$
δ è δ_L per Q_L e δ è δ_M per Q_M per vedere se $q_L \approx q_M$ uso il riempi tabella e se sono equivalenti si ha che $L = M$ passiamo ora alla **minimizzazione**, che prende in input un DFA A e restituisce un DFA A_{min} tale che $L(A) = L(A_{min})$ e che A_{min} ha il numero più piccolo possibile di stati per distinguere $L(A)$.
Procedo così:

1. si rimuovono gli stati non raggiungibili
2. applico la tabella per scoprire le tabelle di equivalenza, gli stati del nuovo automa sono le classi di equivalenza

si ha che lo stato iniziale è la classe di equivalenza dello stato finale e gli stati finali sono le classi di equivalenza degli stati finali.
La minimizzazione si dimostra per assurdo:
Sia M il DFA ottenuto dalla tabella. Suppongo esista un DFA N tale che $L(A) = L(N)\,|Q_N| < |Q_M|$. I due stati iniziali sono indistinguibili. Sia $p \in M$ $q \in N$ tali $p \approx q$ $\forall a \in \Sigma$ quindi $\delta p, a) = \delta(q, a)$, Ogni stato $p \in Q$,è quindi indistinguibile da almeno uno stato di N. Avendo però N meno stati si avranno almeno due stati di M indistinguibili dallo stesso di N ma non sono indistinguibili per la tabella. Si ha un assurdo.

1.1.6 Pumping Lemma per i linguaggi regolari

Questo lemma serve a dimostrare che un linguaggio L non è regolare. Si procede per assurdo.
Partiamo da un esempio:

$$L_{01} = \{0^n 1^n | \, n \geq 1\} = \{01, 0011, ...\}$$

supponiamo che questo linguaggio sia rappresentabile da un DFA A con k stati ($L(A) = L_{01}$).
0^k implica $k+1$ prefissi: $\varepsilon, 0, 00, 000, ..., 0^k$ e quindi $\exists i, j$ tali che 0^i e 0^j finiscono nello stesso stato. Allora l'automa è ingannabile e accetterebbe $0^i 0^j$ con $i \neq j$.
Formalmente si ha:

Teorema 7. *Sia $L \in REG$ allora $\exists n$ che dipende da L tale che $\forall w \in L$ con $|w| \geq n$, w può essere scomposta in tre stringhe $w = xyz$ in modo che:*

$$y \neq \varepsilon$$

$$|xy| \leq n$$

$$\forall k \geq 0 \;\; zy^k z \in L$$

in altre parole posso sempre trovare una stringa non vuota y non troppo distante dall'inizio di w, da replicare da ripetere o cancellare ($k = 0$) senza uscire dal linguaggio L

Dimostrazione. essendo $L \in REG$ esiste un DFA A tale che $L = L(A)$. Suppongo $|Q| = n$ e considero $w = a_1 a_2 ... a_n$ con $m \geq n$. Sia:

$$p_i = \hat{\delta}(q_0, a_1, ...a_i) \;\; \forall i = 0, 1, ..., n$$

quindi:

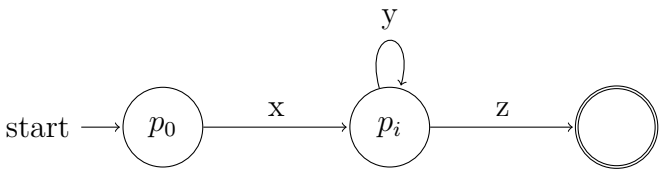
$$p_0 = q_0, p_1, ..., p_n \rightarrow n + a \; stati$$

allora $\exists i, j$, con $0 \leq i < j \leq n$ tali che $p_i = p_j$. Scompongo ora w in $w = xyz$ con:

$$x = a_1 a_2 ... a_n$$

$$y = a_{i+1} a_{i+2} ... a_j$$

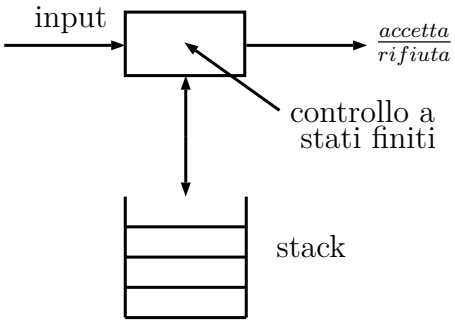
$$z = a_{j+1} a_{j+2} ... a_m$$



□

1.2 Automi a Pila

Si introduce un nuovo tipo di automa, il PDA (push down automata) che può essere pensato come un $\varepsilon - NFA$ col supporto di una pila (stack):



e viene definito un PDA P come:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

con;

- Q: insieme finito e non vuoto di stati
- Σ: alfabeto di simboli di input
- Γ: alfabeto di simboli di stack
- $q_0 \in Q$: stato iniziale
- $z_0 \in \Gamma \backslash \Sigma$: simbolo iniziale dello stack
- $F \in Q$: insieme degli stati accettanti o finali

si ha che:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\} \times \Gamma \to 2^{Q \times \Gamma^*}$$

quindi:

$$\delta(q_0, a, X) = \{(p_1, X_1), (p_2, X_2), ...\} \, insieme \;\; finito \; p_i \in Q \; X_i \in \Gamma^*$$

si hanno dei casi particolari:

- lo stato p potrebbe coincidere con Q e si avrebbe un cappio
- se $\Gamma = \varepsilon$ si ha il pop di X dallo stack
- se $\Gamma = X$ si lascia lo stack invariato

- se $\Gamma = Y \neq X$ si ha la sostituzione di X con Y in cima allo stack
- se Γ è una stringa di simboli si ha il la rimozione di X dallo stack e l'aggiunta a uno a uno dei simboli nello stack

analizziamo meglio i PDA. Si ha che la **descrizione istantanea (ID)** di un PDA è una tripla:

$$ID : (q, w, \gamma)$$

con $q \in Q$ stato attuale $w \in \Sigma^*$ input rimanente e $\gamma \in \Gamma^*$ contenuto attuale dello stack.
Definiamo ora il concetto di **mossa in un passo** dato $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ la mossa è una relazione $\overset{\vdash}{p}$:

$$(p, \alpha) \in \delta(q, a, X) \; allora \; \forall w \in \Sigma^* \; e \; \forall \beta \in \Gamma^* \rightarrow (q, aw, X\beta) \vdash (p, w, \alpha\beta$$

e

$$(p, \alpha) \in \delta(q, \varepsilon, X) \; allora \; \forall w \in \Sigma^* \; e \; \forall \beta \in \Gamma^* \rightarrow (q, w, X\beta) \vdash (p, w, \alpha\beta)$$

ora possiamo anche definire la relazione con 0 o più mosse in forma induttiva $\overset{*}{\underset{p}{\vdash}}$:

- **caso base:** $\forall ID \; I, I \overset{*}{\vdash} I$
- **caso passo:** $I \overset{*}{\vdash} J$ se $\exists ID \; K$ tale che $I \vdash K \; e \; K \overset{*}{\vdash} J$

chiamiamo **computazione** una sequenza di mosse, non necessariamente di successo. Si hanno alcune proprietà:

- se una se una sequenza di ID è lecita per un PDA P allora è lecita anche la sequenza di Id ottenuta concatenando $w \in \Sigma^*$ in ogni ID
- se una se una sequenza di ID è lecita per un PDA P e resta una coda di input non consumata allora posso rimuovere tale coda in ogni ID e ottenere un'altra sequenza lecita
- se una se una sequenza di ID è lecita per un PDA P allora è lecita la sequenza ottenuta aggiungendo $\gamma \in \Gamma^*$ in coda alla terza sequenza di ogni ID

del resto però:

$$(q, Xw, \alpha\gamma) \overset{*}{\underset{p}{\vdash}} (p, Yw, \beta\gamma) \not\rightarrow (q, X, \alpha) \overset{*}{\vdash} (p, y, \beta), \;\; x, w, y \in \Sigma^* \; \alpha, \beta, \gamma \in \Gamma^*$$

per queste proprietà valgono i seguenti teoremi:

Teorema 8. *per la seconda: Se* $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ *è un PDA e* $(q, Xw, \alpha) \overset{*}{\underset{p}{\vdash}} (p, Yw, \beta)$ *allora vale anche:*

$$(q, X, \alpha) \overset{*}{\underset{p}{\vdash}} (p, Y, \beta)$$

Teorema 9. *per la prima e la terza: Se* $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ *è un PDA e* $(q, X, \alpha) \overset{*}{\underset{p}{\vdash}} (p, Y, \beta)$ *allora:*

$$\forall \gamma \in \Gamma^* \ \textit{vale anche} \ (q, Xw, \alpha\gamma) \overset{*}{\underset{p}{\vdash}} (p, Yw, \beta\gamma)$$

Si definiscono due modalità di accettazione per i PDA:

1. **per stato finale:** sia $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ si ha che:

   $$L(P) = \{w \in \Sigma^* | \, (q_0, w, z_0) \overset{*}{\underset{p}{\vdash}} (q, \varepsilon, \alpha)\}$$

   con $q \in F$ e $\forall \alpha \in \Gamma^*$

2. **per stack vuoto:** sia $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ si ha che:

   $$N(P) = \{w \in \Sigma^* | \, (q_0, w, z_0) \overset{*}{\underset{p}{\vdash}} (q, \varepsilon, \varepsilon)\}$$

   con $q \in Q$ e in questo caso l'insieme degli stati finali F non ha alcuna influenza

In realtà si ha che la classe di linguaggi accettati dai PDA per stato finale è uguale a quella per stack vuoto, anche se passare da un tipo all'altro di PDA è complesso. SI ha il seguente teorema per la trasformazione:

Teorema 10. *se* $L = N(P_N)$ *per un PDA* $P_N = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ *allora* $\exists$ *PDA* P_F *tale che* $L = L(P_F)$

Dimostrazione. Sia $x_0 \in \Gamma$, che indica la fine dello stack di P_F. Si ha:

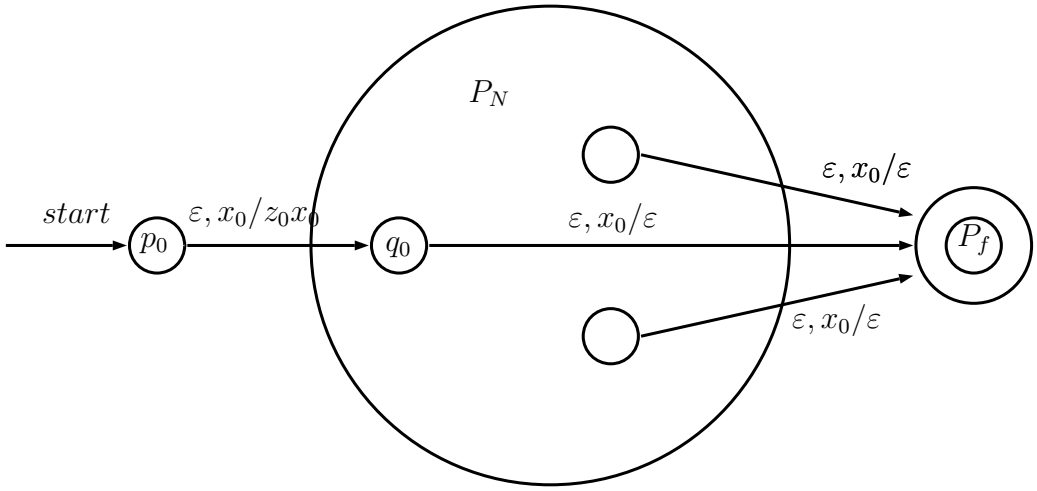
$$\delta(p_0, \varepsilon, x_0) = \{(q_0, z_0 x_0\}$$

e:

$\forall q \in Q$, $\forall a \in \Sigma \cup \{\varepsilon\}, , \forall y \in \Sigma :$ $\delta_F(q, a, y)$ contiene tutte le coppie di $\delta_N(q, a, y)$

$$\forall q \in Q, \delta_F(q, \varepsilon, x_0) = \{(P_F, \varepsilon)\}$$

quindi graficamente:



Bisogna dimostrare che effettivamente $w \in L(P_F) \longleftrightarrow \in N(P_N)$.
se $w \in N(P_N)$ $\exists$ una sequenza di ID $(q_0, w, z_0) \vdash^*_{P_N} (q, \varepsilon, \varepsilon)$ per un qualche $q \in Q$:

$$(q_0, w, z_0x_0) \vdash^*_{P_N} (q, \varepsilon, x_0)$$

inoltre:

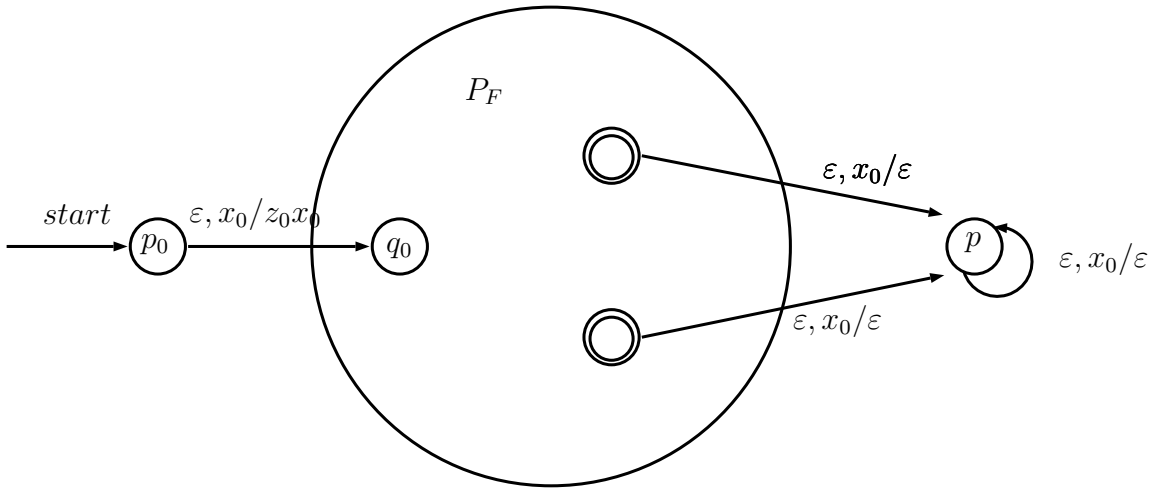
$$(q_0, w, z_0x_0) \vdash^*_{P_F} (q, \varepsilon, x_0)$$

e quindi:

$$(p_0, w, x_0) \vdash_{P_F} (q_0, w, z_0x_0) \vdash^*_{P_F} (q, \varepsilon, x_0) \vdash_{P_F} (P_f, \varepsilon, \varepsilon)$$

solo se togliendo il primo e l'ultimo passo di P_F ripercorro all'indietro quanto scritto sopra. □

Teorema 11. *Sia $P_F = (Q, \Sigma, \Gamma, \delta_f, q_0, Z_0, F)$.*
Si aggiunge una transizione ε a un nuovo stato p da ogni accettante di P_F. quando si ha p P_N svuota lo stack senza consumare input. Quindi se $P : F$ entra in uno stato accettante dopo aver consumato l'input w, P_N svuota lo stack dopo aver consumato w. Per evitare che si svuoti lo stack per una stringa non accettata uso x_0 per indicare il fondo dello stack. Il nuovo P_N parte da p_0 che ha il solo scopo di inserire il simbolo iniziale di P_F e passare al suo stato iniziale. Si ottiene quindi:



e si ha formalmente:

$$P_F = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{x_0\}, \delta_N, p_0, x_0)$$

dove δ_N è così definita:

1. *$\delta_N(p_0, \varepsilon, x_0) = \{(q_0, Z_0x_0)\}$ inserisce il simbolo iniziale di P_F nello stack e va allo stato iniziale di P_F*

2. *$\forall q \in Q$ ogni simbolo di input $a \in \Sigma$, compreso l'input vuoto, e $\forall y \in \Gamma$, $\delta_N(q, a, y)$ contiene tutte le coppie di $\delta_F(q, a, y)$. Quindi P_N simula P_F*

3. *per tutti gli stati accettanti $q \in F$ e i simboli di stack $y \in \Gamma$, compreso x_0, si ha che $\delta_N(q, \varepsilon, y)$ contiene (p, ε), quindi ogni volta che P_F accetta P_N inizia scaricare lo stack senza consumare ulteriori input*

4. *per tutti i simboli di stack $y \in \Gamma$, compreso x_0, si ha che $\delta_N(q, \varepsilon, y) = \{(p, \varepsilon)\}$, quindi giunti allo stato p, ovvero quando P_F ha accettato, P_N elimina ogni simbolo nel suo stack fino a svuotarlo*

inoltre formalmente voglio dimostrare che:

$$w \in L(P_F) \rightarrow w \in N(P_N)$$

e quindi ho le seguenti mosse:

$$(q_0, w, z_0) \vdash^*_{P_F} (q, \varepsilon, \alpha)\ q \in F,\ \alpha \in \Gamma^*$$

$$(p_o, w, x_0) \vdash (q_0, w, z_0x_0) \vdash^*_{P_N} (q, \varepsilon, \alpha, x_0) \vdash^*_{P_N} (p, \varepsilon, \varepsilon)$$

Teorema 12. *sia $G = (V, T, P, S)$ una CFG:*

$$\exists\ PDA\ Q = (\{q\}, T, V \cup T, \delta, q, S)\ tale\ che\ N(Q) = L(G)$$
$$\forall A \in V\ \delta(q, \varepsilon, A) = \{(q, \beta) |\ A \to B\ e'\ una\ produzione\ di\ G\}$$
$$\forall a \in T\ \delta(q, a, a) = \{(a, \varepsilon)\}$$

Questo dimostra che ogni CFL può essere accettato da un PDA accettante per stack vuoto. Per il teorema visto in precedenza, posso sempre costruire un altro PDA accettante per stati finale. I PDA accettano tutti e soli i linguaggi CF. Mostrare che accettano solo linguaggi di tipo 2 è complicato. Un tipo di PDA interessante, soprattutto per i parse, è il PDA deterministico, il **DPDA**.
Un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ è deterministico se:

1. $|\delta(q, a, x)| \leq 1\ \forall q \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall x \in \Gamma$
2. se $|\delta(q, a, x)| \neq 0$ per qualche $a \in \Sigma$ allora $|\delta(q, \varepsilon, x)| = 0$

Teorema 13. *$L \in REG \to \exists PDA\ P$ tale che $L = L(P)$*

Dimostrazione.

$$L \in REG \to \exists DFA\ A = (Q, \Sigma, \delta_A, q_0, F)\ tale\ che\ L = L(A)$$

costruisco il DPDA $P = (Q, \Sigma, \{z_0\}, \delta_p, q_0, z_0, F)$ con:

$$\delta_p(q, a, z_0) = \{p, z_0\}\ \forall p, q \in Q\ tali\ che\ \delta_A(q, a) = 0$$

vale:

$$(q_0, w, z_0) \overset{A}{\underset{P}{\vdash}} (p, \varepsilon, z_0) \longleftrightarrow \overset{\wedge}{\delta}_A (q_0, w) = p$$

□

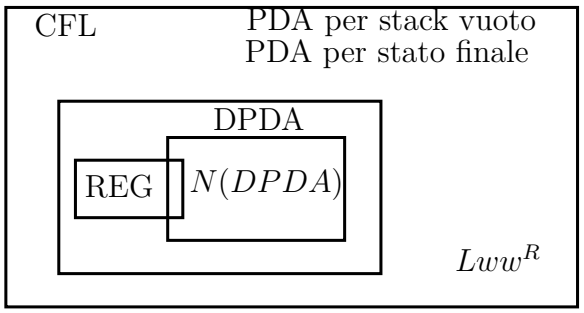
si ha inoltre il seguente teorema:

Teorema 14. *L è $N(P)$ per un DPDA P sse L è $L(P')$ per un DPDA P' e L ha le proprietà di prefisso* ***prefix-free***

definiamo così la proprietà di prefisso:

$$\nexists x, y \in L\ tali\ che\ x \neq y\ e\ x \text{ è prefisso di } y$$

per esempio $L = \{0\}^0 = \{\varepsilon, 0, 00, 000, ...\}$ non ha la proprietà di prefisso. Osserviamo che se la stringa vuota appartiene al linguaggio, tale stringa è prefissa di tutte le altre e quindi il linguaggio non può avere la proprietà di prefisso. Affermiamo che L è regolare, quindi è accettato da un DPDA per stati finali ma non da uno per stack vuoto. Completiamo il diagramma precedente sulle classi di linguaggi:



SI ha che L_{wcw^R} gode della proprietà di prefisso:

$$y = wcw^R \in L \; Se \; x \neq y, \text{ prefisso di } y, x \notin L$$

tornando alle grammatiche si hanno ora due teoremi:

Teorema 15. *se $L = N(P)$ per un DPDA P, allora L ha una CFG non ambigua*

Teorema 16. *se $L = L(P)$ per un DPDA P, allora L ha una CFG non ambigua*

dimostriamo il secondo:

Dimostrazione. $L = L(P)$ per un DPDA P, costruiamo $L^{'} = L$, quindi $L^{'}$ ha la proprietà di prefisso. Esiste quindi un DPDA $P^{'}$ tale che $L^{'}N(P)$, esiste quindi per il teorema sopra una CFG $G^{'}$ tale che $L(G^{'}) = L^{'}$ che non è ambigua.
Costruiamo G per L con le stesse produzioni di $G^{'}$ più $\$ \to \varepsilon$, applicata solo all'ultimo passo. □

Vogliamo scoprire se è vero il viceversa: per ogni L che ha una CFG non ambigua è vero che L è accettato da un DPDA? No, mostriamo infatti un controesempio:
$S \to 0S0|1S1|\varepsilon$ produce L_{ww^R} che non è accettato da alcun PDA