Appunti del corso Linguaggi e Computabilità

Marco Natali

Il corso di Linguaggi e Computabilità riguarda l'informatica teorica e si occupa di definire la calcolabilità di un problema, di definire le grammatiche e i linguaggi formali con l'ausilio di anche di automi e macchine di Turing.

Incominciamo con la definizione dei componenti basilari attraverso cui svilupperemo poi i concetti del corso

Def. Si definisce come *alfabeto*, indicato con Σ, una sequenza di simboli, attraverso cui possiamo stabilire un alfabeto.

Esempio. $\Sigma = \{0, 1\}$ e $\Sigma = \{a, b, c\}$

Def. Si definisce come stringa, indicata solitamente con w, una sequenza di simboli appartenenti ad un alfabeto Σ

Esempio: su un alfabeto $\Sigma = \{0, 1\}$ definiamo le seguenti stringhe: $w = 10110 \;\; z = 10111111$

Considerando un qualsiasi alfabeto Σ, esisterà sempre la stringa ϵ, rappresentante la stringa vuota per cui attraverso questa considerazione si definisce una stringa in maniera induttiva come:

Def. Una stringa viene definita induttiva come:

caso base : ϵ è una stringa vuota

caso passo : se w è una stringa, allora anche $a \circ w$ è una stringa

Dopo aver definito le stringhe, definiamo le seguenti operazioni definite su le stringhe:

- lunghezza $|w| : \Sigma^* \to \mathbb{N}$: rappresenta il numero di caratteri presenti in una stringa, con Σ^* indicante una qualsiasi stringa e la definizione di lunghezza avviene induttivamente come segue:

  base : la lunghezza di $|\epsilon| = 0$

  passo : se $|w| = n$ con $n \in \mathbb{N}$ e sia $a \in \Sigma$ allora $|a \circ w| = 1 + |w| = n + 1$.

  Esempio. $w = abcdec \quad |w| = 6$

- insieme di stringhe: definiamo come Σ^k l'insieme di stringhe su Σ con k caratteri come segue:

$$\begin{aligned} \Sigma^0 &= \{\epsilon\} && (1)\\ \Sigma^1 &= \Sigma && (2)\\ \Sigma^2 &= insiemedistringhedidue caratteri && (3)\\ \ldots\ldots & && (4)\\ \Sigma^k &= insiemedistringhedikcaratteri && (5)\\ & && (6) \end{aligned}$$

  Le due più importanti insiemi di stringhe, usate per rappresentare l'insieme di stringhe di qualsiasi lunghezza, sono:

$$\begin{aligned} \Sigma^* &= \cup_{i=0}\Sigma^i && (7)\\ \Sigma^+ &= \Sigma^* - \{\epsilon\} && (8)\\ & && (9) \end{aligned}$$

- concatenazione $\circ : \sum^* \times \sum^* \to \sum^*$: rappresenta l'aggiunta dei caratteri della seconda stringa al termine della prima stringa ma vediamo ora una definizione più formale:

  Def. Date due stringhe $x = a_1a_2 \dots a_n$ e $y = b_1b_2 \dots b_n$ si definisce $x \circ y = a_1a_2 \dots a_nb_1b_2 \dots b_n$ con $|x \circ y| = |x| + |y| = n + m$.

  La concatenazione possiede le seguenti proprietà:

  - è associativa: $\forall x, y, z \in \sum^* x \circ (y \circ z) = (x \circ y) \circ z$.
  - non è commutativa infatti presi due stringhe x, y diverse risulta $x \circ y \neq y \circ x$.
  - possiede l'elemento neutro ϵ per cui $x \circ \epsilon = x = \epsilon \circ x$.

  Esempio.

$$\begin{aligned} w = 1011100 \quad z &= 1011110 \\ w \circ z &= 10111001011110 \\ z \circ w &= 10111101011100 \end{aligned}$$

Attraverso le seguenti operazioni si può stabilire che $(\Sigma^*, \circ, \epsilon)$ è un monoide libero su Σ.

Def. Definiamo come *Linguaggio* un'insieme di stringhe scelte su Σ^* scelte per far parte del linguaggio ossia $L \subseteq \Sigma^*$. Le componenti di un linguaggio sono:

alfabeto : insieme di simboli su cui definiamo poi il lessico e la sintassi

lessico : definisce il vocabolario del linguaggio e viene definito tramite una grammatica di tipo 3

sintassi : definisce come le varie frasi del linguaggio devono essere disposte nel linguaggio e ciò viene definito da una grammatica di tipo 2.

semantica : il significato attribuito alle frasi del linguaggio però nei linguaggi formali deriva dalla sintassi anche se in questo corso la semantica non verrà affrontata.

Def. Si definisce come *Grammatica* un'insieme di regole che delineano le stringhe ammissibili del linguaggio e possono essere di due tipologie:

grammatica generativa : insieme di regole che permettono di generare tutte le stringhe di un lunguaggio partendo dalle stringhe base

grammatica analitica : analizza le stringhe passate in input e stabilisce l'appartenenza o meno al linguaggio

Nel 1956 il linguista Choumsky introdusse e definì la gerarchia delle grammatiche, che ha avuto una notevole importanza nell'informatica teorica anche se il suo intento era quello di catalogare le varie tipologie di linguaggi naturali:

grammatiche di tipo 3 : sono i linguaggi regolari, generati da grammatiche regolari e riconosciuti da automi a stati finiti. Una spiegazione dettagliata avverrà nel seguito dei paragrafi/capitoli.

grammatiche di tipo 2 : grammatiche libere dal contesto, riconosciute da automi a pila; una spiegazione più dettagliata avverrà nei paragrafi successivi.

grammatiche di tipo 1 : grammatica dipendente dal contesto in cui per definizione i lati destri delle produzioni delle grammatiche non posso essere più lunghi dei rispetti lati sinistri, ossia $\alpha \to \beta |\alpha| \geq |\beta|$. Per rappresentarli e stabilire se una stringa appartiene al linguaggio si usano gli *automi lineari*, che non sono oggetto del corso.

grammatiche di tipo 0 : grammatiche in cui non vi è alcun vincolo per le produzioni della grammatica e può essere rappresentato tramite una macchina di turing nondeterministica però questa tipologia di grammatica non verrà affrontata.

Capitolo 1

Grammatiche di tipo 2

Iniziamo ora a considerare le grammatiche di tipo 2 incominciando da un esempio e poi definendola in maniera formale.

Esempio. Dato il linguaggio delle stringhe palindrome $L_{pal} = \{w \in \Sigma^* : w = w^R\}$ dove w^R rappresenta la stringa reversa per cui ad esempio $''OTTO'' \in L_{pal}$ mentre $''PAPA''$ non appartiene al linguaggio.

Definiamo in maniera più formale e meno ambigua i componenti di L_{pal}:

caso base : $\epsilon, 0, 1 \in L_{pal}$

caso induttivo : se $w \in L_{pal}$ allora $0w0$ e $1w1$ appartengono a L_{pal} e nient'altro appartiene al linguaggio.

Le regole di derivazione per L_{pal} sono le seguenti, e derivano dalla definizione dei componenti:

- $P \to \epsilon$
- $P \to 0$
- $P \to 1$
- $P \to 0P0$
- $P \to 1P1$

Dopo aver dato le regole di derivazione dobbiamo capire se le seguente regole definiscono tutte e sole le stringhe del linguaggio per cui ci chiediamo per esempio se $010 \in L_{pal}$ e $10001 \in L_{pal}$?

$$P \Rightarrow 0P0 \Rightarrow 010 \quad \text{la stringa appartiene correttamente al linguaggio} \tag{1.1}$$

$$P \Rightarrow 1P1 \Rightarrow 10P01 \Rightarrow 10001 \quad \text{la stringa appartiene correttamente al linguaggio} \tag{1.2}$$

$$\tag{1.3}$$

Attraverso l'applicazione di una serie di regole di derivazione otteniamo una stringa $w \in \Sigma^*$ se e solo $w \in L$ della grammatica definita.

Def. Si definisce una grammatica context free come $G = (V, T, P_g, S)$, i cui componenti sono:

- V rappresenta l'insieme delle variabili usate per rappresentare un linguaggio
- T rappresenta l'insieme dei simboli terminali, ossia l'insieme dei simboli attraverso cui sono definite le stringhe del linguaggio per questo di solito coincide con l'alfabeto del linguaggio.
- S rappresenta la variabile di inizio della grammatica, ossia la variabile attraverso cui si definisce la grammatica per cui le altre variabili sono classi ausiliari di stringhe che aiutano a definire le stringhe del linguaggio.
- P_g indica l'insieme di regole, che rappresentano la definizione ricorsiva del linguaggio, della seguente forma:

$$P_g = \{X \to \beta | \beta \in (V \cup T)^* \text{ e } X \in V\}$$

  la variabile X rappresenta la testa della produzione mentre β indica il corpo

Nell'esempio del linguaggio palidromo la grammatica che lo genera è

$$G = (\{P\}, \{0, 1\}, \{P \to \epsilon, P \to 0, P \to 1, P \to 0P0, P \to 1P1\}, S\}$$

Dopo aver definito in maniera formale di grammatica introduciamo il concetto di derivazione, per stabilire se una stringa appartiene o meno al linguaggio.

Def. Sia $G = (V, T, P_g, S)$ una grammatica context-free, sia $\alpha A \beta$ una stringa di terminali e variabili con $A \in V$ e sia infine $A \to \gamma$ una regola di derivazione allora $\alpha A \beta \Rightarrow_g \alpha \gamma \beta$.

Def. Si indica $\Rightarrow_g^*$, il simbolo di applicazione di zero,uno o più step di derivazione, definiti nel seguente modo:

caso base : per ogni stringa α di terminali e variabili, si ha $\alpha \Rightarrow_g^* \alpha$

caso passo : se $\alpha \Rightarrow_g^* \beta$ e $\beta \Rightarrow_g \gamma$ allora $\alpha \Rightarrow_g^* \gamma$

Le stringhe che otteniamo sono delle forme sentenziali, ossia stringhe appartenenti a $(V \cup T)^*$ e un particolare sottoinsieme, in cui le stringhe sono composte da letterali, definisce le stringhe del linguaggio.

Sempre considerando l'esempio del linguaggio delle stringhe palindrome la derivazione di 10011001 è la seguente:

$$P \Rightarrow 1P1 \Rightarrow 10P01 \Rightarrow 100P001 \Rightarrow 1001P1001 \Rightarrow 10011001$$

Al fine di ridurre il numero di scelte nella derivazione di una stringa introduciamo ora:

left derivation : sostituiamo la variabile più a sinistra nell'applicazione di una regola di derivazione e ciò viene rappresentato con il simbolo $\Rightarrow_{lm}$.

right derivation : sostituiamo la variabile più a destra nell'applicazione di una regola di derivazione ed essoviene rappresentato con il simbolo $\Rightarrow_{rm}$.

Def. Data una grammatica context-free $G = (V, T, P_g, S)$ si definisce un linguaggio context-free L come:

$$L_g = \{w \in T^* | S \Rightarrow_g^* w\}$$

Nei linguaggi context-free si può soltanto effettuare la concatenazione e l'annidamento dei sottolinguaggi come vediamo nei seguenti esempi:

Esempio. Mostriamo ora un esempio di linguaggio definito come la concatenazione di due linguaggi

$$L_g = \{w \in \{0,1\}^* | w = 0^m 1^{m+1} 01^n 001^n \ n \geq 0\}$$

Per sapere come costruire la grammatica del linguaggio dobbiamo visualizzare la struttura della stringa

$0^m 11^m 01^n 001^n$ lo spazio rappresenta i blocchi su cui viene formata la stringa

Si può vedere che i 3 blocchi sono 3 linguaggi che concatenati generano il linguaggio L_g per cui le regole di derivazione di L_g sono $P_g = \{S \rightarrow X0Y, X \rightarrow 1|0X1, Y \rightarrow 1001|1Y1\}$.

Esempio. Mostriamo ora un esempio di linguaggio definito come l'annidamento di linguaggi:

$$L = \{w = \{a, b, c, d\}^* | w = a^n b^m c^m d^n \ m > 0, n \geq 0\}$$

Le regole di produzione del seguente linguaggio sono $P_g = \{S \rightarrow aSd|Y, Y \rightarrow bYc|bc\}$ e come si nota il linguaggio viene definito come una sequenza di a e d intrammezzate da un blocco Y, formato da b e c, e ciò è l'annidamento tra diversi blocchi di una stringa che formano le stringhe del linguaggio.

Un'altra modalità per stabilire l'appartenza di una stringa di un linguaggio è *inferenza ricorsiva*, il quale a differenza delle altre applica le regole dal corpo alla testa, ossia concateniamo ogni terminale che appare nel corpo e inferiamo che la stringa trovata è nel linguaggio delle variabili, presenti in testa alle regole.

Viene poco utilizzato in quanto è più naturale e chiaro pensare secondo la derivazione per cui ne vediamo soltanto un esempio e lo usiamo in un importante teorema sull'equivalenza delle modalità di derivazione, che verrà presentato prossimamente.

Esempio. Sia $G = (V, T, O, E)$, con $V = \{E, I\}$ e $T = \{a, b, 0, 1, (,), +, *\}$ quindi ho le seguenti regole, è di tipo 3:

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$

7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

voglio ottenere $a * (a + b00)$ sostituisco sempre a destra (right most derivation)

$$E \to E * E \to E * (E) \to E * (E + E) \to E * (E + I) \to E + (E + I0)$$

$$\to R + (I + b00) \to E * (a + b00) \to I * (a + b00) \to a * (a + b00)$$

usiamo ora *l'inferenza ricorsiva*:

passo	stringa ricorsiva	var	prod	passo stringa impiegata
1	a	I	5	\
2	b	I	6	\
3	b0	I	9	2
4	b00	I	9	3
5	a	E	1	1
6	b00	E	1	4
7	a+b00	E	2	5,6
8	(a+b00)	E	4	7
9	a*(a+b00)	E	3	5, 8

Introduciamo ora un'importante forma grafica per vedere le regole di derivazioni applicate per formare una stringa

Def. Data una grammatica context-free G, un albero sintattico per G è un albero composto come:

- ogni nodo interno è etichettato con una variabile $X \in V$, con la radice etichettata con S.
- ogni foglia è etichettata con una variabile, un simbolo terminale o ϵ; se una foglia viene etichettata con ϵ allora dev essere l'unico figlio del padre.
- se un nodo è etichettato con A e i suoi figli sono etichettati come $x_1, x_2, \ldots, x_k$, allora $A \to x_1 x_2 \ldots x_k$ è una regola di produzione della grammatica.

Eseguendo il prodotto delle stringhe foglia otteniamo una stringa w appartenente alla grammatica di cui abbiamo svolto l'albero sintattico

Esempio. Prendendo il linguaggio L definito come segue:$L = \{w \in \{a, b, c, d\}^* | w = a^n cb^m cd^{n+m} n \geq 0, m > 0\}$ stabiliamo una grammatica context-free e fare l'albero sintattico di $aacbbbcddddd \in L$: le regole di produzione di L sono $P_g = \{S \Rightarrow aSd|cY, Y \Rightarrow bcd|bYd\}$ per cui la grammatica è:

$$G = (\{S, Y\}, \{a, b, c, d\}, P_g, S)$$

L'albero sintattico di $aacbbbcddddd$ è il seguente:

1.1 Equivalenza tra le derivazioni

In questo paragrafo consideriamo un importante teorema sulle equivalenze tra le varie modalità di derivazione, definito come segue:

Thm: 1.1. Data una grammatica context-free $G = (V, T, P_g, S)$ abbiamo che le seguenti modalità di derivazione sono equivalenti:

1. la procedura di inferenza ricorsiva che determina che una stringa di terminali w appartiene ad un linguaggio di variabili A
2. $A \Rightarrow^* w$
3. $A \Rightarrow^*_{lm} w$
4. $A \Rightarrow^*_{rm} w$
5. esiste un albero sintattico con radice A e come prodotto di foglie la stringa w.

I primi due sono facilmente dimostrabili dato che le derivazione a sinistra e a destra sono delle derivazioni invece iniziamo a dimostrare:

Thm: 1.2. pippo

1.2 Grammatiche e Linguaggi Ambigui

In questo paragrafo analizziamo le grammatiche e i linguaggi ambigui, incominciando da un esempio per catturare gli aspetti essenziali, che poi verranno definiti formalmente.

Incominciamo per cui a considerare le espressioni algebriche definite su $T = \{+, *, a, b, (,)\}$ con le seguenti regole di produzione:

$$P_g = \{E \Rightarrow I|E + E|E * E|(E), I \Rightarrow a|b|aI|bI\}$$

La grammatica per definire gli identificatori I è una grammatica di tipo 3, cosa che analizzeremo nel prossimo capitolo, però questa grammatica è ambigua dato che preso ad esempio $a + b * a$ avremmo due derivazioni left-most, che sono le seguenti:

Diamo ora una definizione formale di grammatica ambigua:

Def. Si dice che una grammatica è ambigua se e solo se esiste una stringa $w \in L$ tale per cui w ammette due derivazioni lm e/o rm diverse oppure ammette due alberi sintattici definiti sulla stessa grammatica G

L'obiettivo è quello di avere grammatiche non ambigue perchè esse permettono di definire in maniera univoca, per cui automamilizzabile con una procedura, l'insieme delle stringhe del linguaggio e fortunatamente nella maggior parte dei casi, quando il linguaggio non è ambiguo, data una grammatica ambigua è possibile definire una grammatica non ambigua G' tale che $L(G) = L(G')$.

Def. Un linguaggio $L \subseteq \sum^*$ è detto *ambiguo* se per ogni grammatica G, tale per cui $L = L_G$, risulta che G è ambigua.

La grammatica data delle espressioni algebriche ha due cause di ambiguità:

1. la precedenza degli operatori non viene rispettata in quanto andrebbe raggruppato prima il silmbolo di $*$ rispetto a $+$
2. una sequenza di uguali operatori può essere reggruppata sia da sinistra che da destra e si stabilisce per convenzione che si raggruppa da sinistra a destra.

Considerate queste cause di ambiguità, per eliminarla si introducono altre variabili definite come:

1. un fattore è un espressione che non può essere spezzata da nessun operatore, sia il $*$ che il $+$, per cui gli unici fattori nel nostro linguaggio delle espressioni sono gli identificatori e ogni espressioni dentro le parentesi.
2. un termine è un espressione che non può spezzata dall'operatore $+$.
3. un espressione può riferirsi a qualsiasi possibile espressione, incluse quelle che possono essere spezzate da un adiacente $*$ o $+$

La grammatica non ambigua del linguaggio delle espressioni è la seguente:

$$P_{exp} = \{E \to T|E+T, T \to F|T*F, F \to I|(E), I \to a|b|aI|bI|0I|1I\}$$

Mostriamo ora un esempio di linguaggio ambiguo L definito come segue:

$$L = \{w = a^n b^n c^m d^m | n, m \geq 1\} \cup \{z = a^n b^m c^m d^n | n, m \geq 1\}$$

Le regole di produzione che generano il seguente linguaggio sono le seguenti:

$$P_L = \{S \to XY|Z, X \to ab|aXb, Y \to cd|cYd, Z \to aZd|aRd, R \to bc|bRc\}$$

Questa grammatica è ovviamente ambigua dato che per esempio la stringa *aabbccdd* ha due derivazioni sinistre:

La dimostrazione che il linguaggio è ambiguo è molto complessa ma il concetto alla base è quello di provare che tutte le grammatiche del linguaggio generano un numero finito di stringhe con almeno due derivazioni sinistre/destre e nel nostro caso definiamo che il linguaggio è ambiguo perche per $n = m$ qualsiasi grammatica usiamo per definire il linguaggio avremmo i due sottolinguaggi che generano la stessa stringa in maniera diversa.

Capitolo 2

Linguaggi Regolari

Per definire le stringhe appartenenti ai linguaggi regolari, di tipo 3, vi può utilizzare le *grammatiche regolari*, in cui vengono definite delle regole per stabilire se e quando una stringa appartiene al linguaggio, e le *espressioni regolari*

Incominciamo a considerare le grammatiche regolari, sottoinsieme delle grammatiche di tipo 2 secondo la gerarchia di Choumsky, utilizzate per generare i linguaggi regolari.

Prevedono dei maggiori vincoli sulle produzioni infatti:

1. ϵ può apparire solo nel corpo del simbolo start ossia può essere solo $S \rightarrow \epsilon$.
2. le produzioni sono del seguente tipo, dove $a \in T$ e $A, B \in V$,:
   - $A \rightarrow aB$ oppure $A \rightarrow a$(lineari a destra)
   - $A \rightarrow Ba$ oppure $A \rightarrow a$(lineari a sinistra)
3. non si può avere produzioni sia lineari sinistre che lineari destre

Esempio: i due insiemi di produzioni per definire gli identificatori di un espressione algebriche sono le seguenti:

$$P_1 = \{I \rightarrow a|b|Ia|Ib|I0|I1\} \quad \text{lineare a destra} P_2 = \{I \rightarrow aJ|bJ, J \rightarrow aJ|bJ|0J|1J|0|1|a|b\} \quad \text{lineare a sinistra}$$

Esempio:data la grammatica $G(\{S\}, \{0,1\}, \{S \rightarrow \epsilon|0S|1S\}, S)$ capire che linguaggio rappresenta: $L = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \dots\}$

Esempio:Produrre le produzioni lineari sinistra e lineari destra del seguente linguaggio:

$$L = \{w \in \{a, b\} * |w = a^n b^m n, m \geq 0\}$$

Esempio: Produrre le produzioni lineari sinistra e lineari destra del seguente linguaggio:

$$L = \{w \in \{a, b, c, d, e\} * |w = ab^n cd^m en \geq 0, m > 0\}$$

Esempio: Fornire le produzioni lineari sinistra e lineari destra del seguente linguaggio:

$$L = \{w \in \{0, 1\} * |\text{w contiene almeno uno 0 oppure almeno un 1}\}$$

2.1 Espressioni Regolari

Le espressioni regolari permettono di definire, utilizzando una notazione algebrica, un linguaggio regolare e vengono utilizzate per estrarre parole da un testo ed altre notevole applicazioni, che verranno analizzate nel corso dei paragrafi.

Per riuscire a definire in maniera formale le espressioni regolari dobbiamo definire le seguenti operazioni sui linguaggi regolari:

- Unione: dati $L, M \subseteq \Sigma*$ si definisce $L \cup M = \{w \in \Sigma * | w \in L \vee w \in M\}$ Esempio: $L = \{001, 10, 111\}$ e $M = \{\epsilon, 001\}$ risulta $L \cup M = \{\epsilon, 10, 001, 111\}$
- Concatenazione: dati due linguaggi $L, M \subseteq \Sigma*$ si ha $L \circ M = LM$, ossia linguaggio formato da tutte le strighe ottenute concatenando le stringhe in L con le stringhe in M. Esempio: $L = \{001, 10\}$ e $M = \{\epsilon, 111\}$ risulta $L \circ M = \{001, 10, 001111, 10111\}$
- Chiusura di Kleene: dato un linguaggio L si ha $L*$ definito induttivamente come:

$$\begin{aligned} L^0 = \{epsilon\} L^1 = L \\ L^2 = L \circ L \\ \dots \\ L^i = L^{i-1} \circ L \\ L^* = L^0 \cup L^1 \cup L^2 \cup \dots \\ L^+ = L^* - \{epsilon\} \end{aligned}$$

  Esempio: dato $L = \{0, 1\}$ abbiamo:

$$\begin{aligned} L^0 = \{epsilon\} \\ L^1 = \{0, 1\} \\ L^2 = \{00, 01, 10, 11\} \\ L^3 = \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111\} \end{aligned}$$

  Il linguaggio $L*$ è generalmente un linguaggio infinito in quanto è l'unione di un numero infinito di linguaggi finiti ma esistono due linguaggi la cui chiusura è finita, che analizziamo ora:

  - il linguaggio $L = \{0\}$ la sua chiusura di kleene è finita dato che si ha:

$$\begin{aligned} L^0 = \epsilon \\ L^1 = L \\ L^2 = L \circ L = L \\ L^3 = L^2 \circ L = L \\ L^i = L^{i-1} \circ L = L \\ L^* = L^0 \cup L^1 \cup L^2 \cup \dots = L \cup epsilon = L \end{aligned}$$

– il linguaggio $L = \emptyset$ la sua chiusera di kleene è finita in quanto:

$$\begin{aligned} L^0 &= \epsilon \\ L^1 &= \emptyset \\ L^2 &= \emptyset \circ \emptyset = \emptyset \\ L^i &= \emptyset^i \circ \emptyset = \emptyset \\ L^* &= \emptyset \cup \{\epsilon\} = \{epsilon\} \end{aligned}$$

Dopo aver definito le operazioni sui linguaggi regolari, definiamo ora:

Def. Si definisce *espressione regolare* induttivamente come segue, considerando anche il linguaggio che generano:

caso base : la base consiste in 3 parti:

1. ϵ e $\emptyset$ sono RegEx e generano $L(\epsilon) = \{epsilon\}$, $L(\emptyset) = \emptyset$
2. se a è un simbolo allora a è una Regex e questa espressione genera $L(a) = \{a\}$
3. una variabile, rappresentanti linguaggi regolari, sono RegEx, e generano $L(L) = L$

caso induttivo : la parte induttiva delle espressioni regolari sono composte da 4 tipologie:

1. se E e F sono delle RegEx allora $E + F$ è una RegEx e rappresentano $L(E + F) = L(E) \cup L(F)$
2. se E e F sono delle RegEx allora $E \circ F = EF$ è una RegEx per cui rappresentano $L(EF) = L(E)L(F)$
3. se E è una RegEx allora $E*$ è una RegEx, che denota la chiusura di $L(E)$ infatti $L(E^*) = (L(E))^*$
4. se E è una RegEx allora (E) è una RegEx in cui $L((E)) = L(E)$.