Linguaggi e Computabilità

DaveRhapsody

2 Ottobre 2019

Indice

Capitolo 1

L'esame

Avremo due compitini, uno a novembre ed uno a Gennaio, in un anno sono disponibili 5 appelli, se uno è del terzo anno, può fare i compitini, basta che ci sia spazio nelle aule, la precedenza va a coloro che sono del secondo anno.

Al secondo appello (Quello di Febbraio) puoi recuperare il voto negativo di uno dei due compitini. Non presentarsi è esattamente come provarci e non passare, quindi rischiate, conviene.

L'orale va sostenuto nello stesso appello dello scritto, cioè io faccio lo scritto, lo passo, l'orale lo devo fare in quella sessione. Per chi fa i compitini ed ha consegnato anche gli esercizi di lab. può fare un orale prima del 5 Febbraio OPPURE si può fare assieme a coloro che hanno fatto l'esame il 5.

Gli esercizi valgono dal momento che li invii fino a fine anno, quindi ha senso farli subito tutti

---

Capitolo 2

Linguaggi formali

Nascono per essere in grado di creare i linguaggi di programmazione, o meglio servono per gestire i protocolli di comunicazione e la possibilità di comunicare una determinata operazione al calcolatore.

2.1 Backus-naur form (Backus Normal Form)

Definizione Da Wikipedia: è una metasintassi, ovvero un formalismo attraverso cui è possibile descrivere la sintassi di linguaggi formali (il prefisso meta ha proprio a che vedere con la natura circolare di questa definizione). Si tratta di uno strumento molto usato per descrivere in modo preciso e non ambiguo la sintassi dei linguaggi di programmazione, dei protocolli di rete e così via, benché non manchino in letteratura esempi di sue applicazioni a contesti anche non informatici e addirittura non tecnologici. La BNF viene usata nella maggior parte dei testi sulla teoria dei linguaggi di programmazione e in molti testi introduttivi su specifici linguaggi.

2.2 Model checking

Usato per protocolli di comunicazione, per esempio per protocolli di pagamento, in realtà di qualsiasi tipo, chiaramente per la sicurezza questo è l'ideale, perchè si descrive lo stato di sistema, e si specifica se ogni stato è sicuro (Sicuro sia dal punto di vista dei risultati corretti che sicuri)

E' usato anche per il software, cioè in maniera automatica deduce in base alle condizioni di ingresso, se son corrette. Ce la fa? Si per programmi piccini, ma alla fine, ma ingenerale, non esiste una tecnica che preso un software ti dimostra che esso sia corretto in ogni caso. Non esiste nessuna procedura generale, se esistesse ci sarebbero contraddizioni logiche.

Cos'è una contraddizione logica? E' un paradosso, ma a livello un po' più infame, pensate alla frase "Questa frase è vera", se ci scavate a fondo, dopo un po' diventa una contraddizione.

2.3 Automi a stati finiti

Sono insiemi di stati ai quali arrivan dall'esterno dei dati, ed a seconda dello stato in cui si trovano, e del dato che arriva, allora potrebbero verificarsi le famose "Transizioni" che consistono nel cambiare stato.

La memoria del Latch SR, ad esempio, funziona come un automa, nel senso, varia a seconda dello stato interno, e del valore di ingresso.

Linguaggio Per E' uno dei primi linguaggi di scripting, anche se ce n'era qualcun altro prima, e contiene istruzioni per gestire espressioni regolari che possono essere applicate su testi lunghi per fare ricerche.
In pratica prendevano delle sequenze di DNA (tera di dati), e venivano analizzati (con espressioni regolari) da questo linguaggio.

Capitolo 3

Alfabeto

E' un insieme finito e non vuoto di simboli, ad esempio: {A, B, C, D, ... , Z} , {1, 2, 3, 4, ..., 9}.
Per gli alfabeti useremo lettere greche tipo: Σ, Λ, Γ, vediamo alcune definizioni ora:

Stringa La stringa è una sequenza di simboli, se è vuota si definisce vuota, può esistere. Data una stringa w, si indica la sua lunghezza con |w|. Per esempio: |acdas234| = 8, mentre se ho $|\epsilon|$ = 0, poichè si indica che una stringa è vuota dicendo che essa abbia solo una lettera greca dentro

Concatenazione tra stringhe La concatenazione fa in modo che date due stringhe w, x l'ultimo carattere di x sarà il successivo dell'ultimo di w. pertanto, w, x → w ∘ x = wx
Per esempio se ho una stringa vuota, e la concateno ad una stringa, otterrò la stringa (3+0 fa 0, no? :)) $\rightarrow \epsilon\circ$ w = w

Chiaramente si vanno a sommare le lunghezze delle due stringhe in ogni caso.

NON Commutatività di una stringa Concatenare due stringhe non è sempre possibile, a meno che siano perfettamente identiche

Potenze di un alfabeto Prendiamo un alfabeto Σ e per un k intero >= 0 $\Sigma^k = \Sigma x, \Sigma x, \Sigma x, \Sigma x$, ottengo una permutazione di k volte Σ, tutte appartenenti a Σ^k

Come sarà la sua cardinalità? $|\Sigma|$ = q $\rightarrow |\Sigma^k| - q^k$.

Per k = 1 avrei Σ^1 w = qualsiasi elemento di Σ(un solo elemento)

Se ho Σ = 0, 1
Σ^2 = Tutte le permutazioni che posso fare con 0, 1 **i lunghezza 2** (I valori di Σ)

Per definizione $\Sigma^0 = \epsilon$,

Attenzione Quello che è contenuto in Σ è un insieme di STRINGHE non caratteri o simboli.

Chiusura di Kleene $\Sigma^* =$ U per k ≥ 0 $\Sigma^* =$ U per k $\in 0$ ad ∞ di Σ^k

$\Sigma^+ = \Sigma^* - \Sigma^0$, invece Σ^* è considerabile come $\Sigma^+ \cup \Sigma^0$

ATTENZIONE La L che userò nei prossimi passaggi ($\rightarrow L$) è una MACCHINA AUTONOMA che verifica la stringa in questione

Linguaggio L su Σ E' un sotto insieme di Σ^*, o meglio L $\subseteq \Sigma^*$ Ad esempio:

$$\Sigma = a, b, c \rightarrow L_1 = aa, cbc \subseteq \Sigma^* L_2 = w \in a, b, c^* t.c.\ W\ contiene\ stesso\ numero\ di\ a\ e\ c$$

In pratica

$$L_2 = ac, ca, acb, abc, cab, cba, ...$$

abc $\in L_2$?? Yes
ccbb $\in L_2$?? Nope

Detto meglio (Problema di Membership)

$$W \in \Sigma^* \rightarrow L \begin{cases} SI,\ Se\ w \in L \\ NO,\ Se\ W \in \Sigma^* senza\ L(Complemento\ di\ L) \end{cases}$$

Attenzione, il linguaggio è un insieme, contiene quindi ELEMENTI, e di conseguenza può contenere anche l'insieme vuoto!

Osservazione: w può essere appartente a Σ^* MA non all'insieme vuoto. Occhio a non confondersi

In generale un linguaggio formale va studiato secondo due punti di vista almeno.

Avendo un linguaggio L $\subseteq \Sigma^* posso \begin{cases} generarlo\ (grammatica) \\ riconoscerlo\ (macchina\ autonoma) \end{cases}$

Grammatica Insieme di regole che specificano come va fatta una stringa Una grammatica G è una quadrupla $\rightarrow$ G = (V, T, P, S) in cui

- V: variabili
- T: Simboli terminali
- P: insieme delle produzioni
- $S \in V$: simbolo di start

I tipi di grammatiche Esiste una gerarchia (Noam) Chomsky, che negli anni 50 si poneva domande su cosa accade nel cervello umano quando si elabora un linguaggio.

La sua ipotesi (smentita) c'è una sorta di grammatica codificata/cablata per elaborare il linguaggio, e (malgrado smentita) è saltata comunque fuori questa gerarchia

1. Grammatiche tipo 0:
   - Non hanno restrizioni sulle produzioni
   - Sono riconosciuti dalle macchine di Turing (Alan Turing)
   - linguaggi che generano sono i ricorsivamente numerati, li vedremo a computabilità (Sia deterministiche che non)
2. Grammatiche Tipo 1: La testa ha lunghezza $\geq$ corpo, ne vedremo solo due esempi
   - Linguaggi dipendenti dal contesto, riconosciuti da macchine particolari come la macchina di Turing, che lavorano spazio lineare Cioè Se n è la lunghezza della prima forma sentenziale da cui parto, tutte le altre forme sentenziali non potranno essere più lunghe, e quindi non può crescere il numero di simboli, tenderà sempre a diminuire.

   Le regole di produzione di tipo 1: $\alpha_1\, A\, \alpha_2\; \rightarrow\; \alpha_1\, \beta\, \alpha_2\, con\, \alpha_{1,2}, \beta\; \in\; (V \cup T)^*\,,\; \beta\; \neq\; \epsilon, A \in V$

   Problema di decisione E' un problema la cui risposta possibile è sì o no (Cioè alla fine true o false). Risolvere un problema di decisione non pè altro che risolvere un problema di membership.
3. Grammatiche Tipo 2: Le regole di produzione qui sono del tipo A$\in$B, con A$\in$V e $\beta\; \in\; (V \cup T)^*$ Sono linguaggi context free, e vengono riconosciuti da macchine (o automi) a pila monoterministica
4. Grammatiche Tipo 3: Sono le grammatiche regolari e quindi producono e generano semplicemente linguaggi regolari, e le produzioni delle grammatiche regolari si possono tutte trasformare in modo tale che A$\in$ aB $\wedge$ A $\in$ a con A, B$\in$V e a$\in$ T, riconosciuti da automi a stati finiti, deterministici o monodeterministici

Il complemento di un linguaggio può essere sia infinito che finito (Nel senso posso escludere elementi oppure posso considerare solo quelli!)

3.1 Linguaggio context-free (CFL) legati a grammatiche Context Free (CFG)

In questo caso si utilizza una forma ricorsiva per definire questi linguaggi,

Ricordiamoci del fatto che io posso mettere due linguaggi in serie, posso includerne uno in un altro MA non posso accavallarli. Nel senso, o tutto di entrambi, o niente. Ma torniamo ai **Context free**

La stringa palindroma Sono stringhe la cui lettura è identica in qualsiasi verso si leggano. Supponiamo $\Sigma = 0, 1$ es. $L_{pal} \subseteq \Sigma^* \rightarrow$ "0110", "11011", ϵ, perchè la stringa vuota è considerata palindroma.
Più in modo formale si può dore che w è **palindroma** quando $w = w^R$ Definizione induttiva:

$$\begin{cases} base: \ \epsilon, 0, 1 \in L_{pal} \\ passo\ induttivo: \ se\ w \in \ L_{pal}\ allora\ OwO,\ 1w1 \ \in \ L_{pal} \end{cases}$$

$S \in \epsilon$
$S \ \in \ 0$
$S \ \in \ 1$
$S \ \in \ 0S0$
$S \ \in \ 1S1$
Con S che è una variabile (categoria sintattica), e {0, 1} che sono i simboli terminali con cui si scrivono le stringhe del linguaggio.

Queste si chiamano regole di produzione in cui la testa è occupata in questo caso dalla freccia, mentre i vari 0 1 0S0 e 1S1 sono il corpo. S PUO' diventare il corpo

Detto questo possiamo dire che

$$G_{pal} = (V, T, P, S),\ in\ cui$$

- V = {S}
- T = {0, 1}
- P = $S \in \epsilon, S \in 0, S \in 1, S \in 0S0, S \in 1S1$

Più in generale

$$G_{pal} = (\{S\}, \{0, 1\}, P, S)\ dove\ P \ = \ \ \{S \in \epsilon, ...\}$$

Derivazione $S \implies 1S1 \implies 10S01$, dove 1S1 è una forma sentenziale e la S cambia in funzione delle regole che ho deciso sopra (per generare la stringa ovviamente.)

In modo compatto:
$S \in \epsilon|0|1|0S0|1S1$

C'è una precisazione da fare, se per esempio avessi la regola che le mie stringhe debbano iniziare per 0, quando andrò a fare 0S0, allora quell'S volendo può essere sostituita con una **nuova** variabile che chiamiamo **X**.

X non è altro che una variabile che rappresenta l'insieme di tutte le palindrome. Perchè cambiare variabile? Perchè se io voglio ad esempio le palindrome che iniziano per 0, devo avere, dato che non posso forzare l'ordine con cui vengono applicate le mie regolo, devo avere un "permesso" speciale che consenta di generare 0 all'inizio alla fine. Cioè, dentro ci può essere un pandemonio, ma fuori ci deve essere la regola che stabilisce l'esistenza di 0.

3.2 Parentesi bilanciate

T ={(,)}, in cui () $\in L_{pal}$, (()) $\in L_{pal}$, ()() $\in L_{pal}$, $\epsilon \in L_{pal}$

Se W $\in L_{pal}$, allora (W) $\in L_{pal}$ esattamente come WW$\in L_{pal}$

Ex 5.19 p 182 ϵ () ()() (w) $_{pal}$

$$\begin{cases} \underline{base} : \epsilon \\ \underline{Passo} : Se\ w \in L_{pal}\ allora\ ww \in L_{pal}\ AND\ (w) \in L_{pal} \end{cases}$$

Dato G=(V, T, P, B)
B $\rightarrow$ BB | (B) | ϵ,

V = {B} T = {(,)} e
P = {B $\rightarrow$ BB, B $\rightarrow$ (B), B$\rightarrow \epsilon$}

A questo punto, se avessi ()(()) otterrei:
B $\implies_1$ BB $\implies_2$ B(B) $\implies_2$ (B)(B) $\implies_3$ ()(B) $\implies_2$ ()((B)) $\implies_3$ ()(())
Quindi questa stringa è possibile generarla.

3.3 Produzioni Context - Free

DISCLAIMER: D'ora in poi vedrete qualcosa di questo genere: $\implies^*_{lm/rm}$, dovete considerare tutte le volte in cui si presenteranno come se fossero =>$^*_{lm/rm}$, appena ho tempo poi li cambio tutti. cosa indicano rm lm e * E' scritto tranquilli

---

A $\rightarrow \gamma$ dove A $\in$ V e $\gamma \in (V \vee T)^*$

Agendo come prima:
B $\implies_1$ BB $\implies_2$ (B)B $\implies_3$ ()B $\implies_2$ ()(B) $\implies_2$ ()((B))$\implies_3$ ()(())

3.4 Derivazione left/right most

Per evidenziare che sia una letf o right most invece del numeretto, alla freccia si aggiunge un lm o rm (Left o Right most). Sempre con l'esempio di prima ->

B $\implies_{rm}$ BB $\implies_{rm}$ B(B) $\implies_{rm}$ B((B)) $\implies_{rm}$ B(()) $\implies_{rm}$ (B)(()) $\implies_{rm}$ ()(())

Ex 5.3, p 162 Regole di produzione: E $\in$ I | E + E | E * E | (E)

I $\in$ a | b | Ia | Ib | I0 | I1

G = (V, T, P, E) V = {E, I}
T = {+, *, (,), a, b, 0, 1}

La E diventa identificatore quindi $E \implies I \implies a$ (stessa roba per b) quindi $E \implies I \implies Ia \implies I0a \implies Ib0a \implies I1b0a \implies b1b0a$

Proviamo a generare a * (a+b00) (metodo Left-Most)
E $\implies_{lm}$ E * E $\implies_{lm}$ I * E $\implies_{lm}$ a * E $\implies_{lm}$ A * (E) $\implies_{lm}$ a * (E + E) $\implies_{lm}$ a * (I + E) $\implies_{lm}$ a * (a + E) $\implies_{lm}$ a * (a + I) $\implies_{lm}$ a * (a + I0) $\implies_{lm}$ a * (a + I00) $\implies_{lm}$ a *(a + b00)

Per parcondicio, ora faremo anche la generazione con il Right Most
E $\implies_{rm}$ E * E $\implies_{rm}$ E * (E) $\implies_{rm}$ E * (E + E) $\implies_{rm}$ E (E + I) $\implies_{rm}$ E * (E + I0) $\implies_{rm}$ E * (E + I00) $\implies_{rm}$ E * (E + b00) $\implies_{rm}$ E * (I + b00) $\implies_{rm}$ E * (a + b00) $\implies_{rm}$ I * (a + b00) $\implies_{rm}$ a * (a + b00)

Per indicare che "in qualche modo" è possibile ottenere una determinata stringa si scrive

$$E \implies^{*}_{rm/lm}$$

.

Data $\alpha A\beta,\ con\ \alpha\beta \in (VUT)^{*},\ con\ A \in V$
$A \in \gamma$ (Regole di produzione)
$\alpha A\beta \implies \alpha\gamma\beta$

Se A è var. più a sx $\implies_{lm}$ altrimenti diventa $\implies_{rm}$ Nel caso sia più a destra

3.5 Definizione di $\implies^{*}$

Per induzione:

$$\begin{cases} Base:\ \forall\alpha \in (VUT)^{*}, \alpha \implies {}^{*}\alpha Passo:\ Se\ \alpha \implies {}^{*}\beta e\beta \implies \gamma \\ allora\ \alpha \implies {}^{*}\gamma\ dove\ \alpha,\beta,\gamma \in (VUT)* \end{cases}$$

Pertanto $\alpha \implies {}^*\beta$ sse $\exists\gamma_1, \gamma_2, ..., \gamma_n \in (VUT)^*$ con n$\geq$ t.c $\alpha = \gamma_1, \beta = \gamma_n e \forall i = 1, 2, ..., n-1$ si ha che $\gamma_i \implies \gamma_{i+1}$

3.6 Definizione forma sentenziale

Sia G = (V, T, P, S) una CFG, e $\alpha \in (VUT)^*$ t.c. S $\implies {}^*\alpha$

Ogni volta che io genero nella forma sentenziale uno zero, in realtà se ne genera un altro, quindi se ho $S \implies 0S0 \implies 00S00$, imponendo un vincolo sugli zeri prima e dopo la S, quindi per esempio se avessi:

I $\rightarrow$ 0 | 1 | ϵ | I0 | I1 Reg.
$I \implies I0 \implies I00 \implies I000 \implies I1000$

3.7 Inferenza Ricorsiva

L'obbiettivo è dimostrare che dato un obbiettivo si può ricavare in 0 o più passi una determinata stringa.
(E' la stringa dell'esercizio precedente:) Si agisce ponendo una tabella composta in questo modo

/	Stringa ricavata	Variabile	N proof	Stringhe impiegate
(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	(6)
(7)	a+b00	E	2	(5),(6)
(8)	(a+b00)	E	4	(7)
(9)	a*(a+b00)	E	3	(5),(8)

3.8 Due teoremi importanti

3.8.1 Th. 1

Sia G = (V, T, P, S) una CFG, e sia $\alpha \in (VUT)^* allora S \implies {}^*\alpha\ sse\ \alpha\ e'\ ottenibile\ tramite\ procedura\ di\ inferenza\ ricorsiva$

3.8.2 Th. 2

Sia G = (V, T, P, S) una CFG, e $\alpha \in (VUT)^* allora S \implies {}^*\alpha$ se e solo se $S \implies^*_{rm} \alpha$ OPPURE $S \implies^*_{lm} \alpha$

3.9 Le Relazioni $\implies$

Sia G = (V, T, P, S) una CFG e sia $\alpha A\beta$ t.c. $\alpha, \beta \in (v \cup T)^\star$ e A $\in$ V.
Sia $A \rightarrow \gamma \in P$ allora $\alpha A\beta \implies \alpha\gamma\beta$

3.10 Le Relazioni $\implies^*$

$\alpha \implies {}^*\beta$, con $\alpha, \beta \in (V \cup T)^*$ se e solo se $\exists\gamma_1, \gamma_2, ..., \gamma_n \in (V \cup T)^*$ t.c. $\alpha = \gamma_1 \implies \gamma_2 \implies ... \implies \gamma_n = \beta$ Con $n \geq 1$

Capitolo 4

Esercizi sulle CFG

Esercizio 1: Formare una CFG per il linguaggio

$$L = \{0^n 1^n | n \geq 1\}$$

L = (01,0011,000111,00001111, $\rightarrow$)
Proviamo a scrivere la grammatica:
G = (V, T, P, S), dove T = {0, 1}
$S \rightarrow 0S1|01$, ricordiamoci che non possiamo metterci ϵ perchè si è specificatop che dobbiamo avere $n \geq 1$ (ricordandoci che n è il numero di 0 e 1), quindi

Se $L = \{0^n 1^n | n \geq 1\}$
S $\rightarrow 0S1|\epsilon$
L = ϵ, 01, 0011, 000111, ...

Esercizio 2: Formare una CFG per il linguaggio

$$L = \{a^n | n \geq 1\}$$

L = (aS,aaS,aaaS,aaaaS,aaaaa, ..., $\rightarrow$)
Proviamo a scrivere la grammatica:
$S \rightarrow aS|a$, ecco un esempio di applicazione di questo esercizio

Nel caso di questo esercizio è indifferente se la 'a' vien messa prima o dopo la S

Esercizio 3: Formare una CFG per il linguaggio

$$L = \{(ab)^n | n \geq 1\}$$

L = (ab, abab, ababab, abababab, ..., $\rightarrow$) (Rap Futuristico abababababab)
Scriviamo la grammatica $S \rightarrow abS|ab$ OPPURE, $S \rightarrow Sab|ab$

$S \implies Sab \vee abS \implies$ ... Non è sbagliato scriverli entrambi, ma sarebbe auspicabile costruire due insiemi S in cui hai uno con la prima regola ed uno in cui usi la seconda regola. Quindi si ha che:

$S \implies Sab \implies Sabab...$ $S \implies abS \implies ababS...$ Introduciamo le regole $A \rightarrow b$, $A \rightarrow bS$, $S \rightarrow aA$

Esercizio 4: Formare una CFG per il linguaggio

$$L = \{(a)^n cb^n | n \geq 1\}$$

L = $\{S \implies aSb \implies aaSbb \implies aaaSbbb...\}$
Scriviamo la grammatica $S \rightarrow aSb|acb$

Riassumendo: V = {S}
T = {a,b,c}
P = {S → aSb, S → acb}

Esercizio 5: Formare una CFG per il linguaggio

$$L = \{w \in \ \{a, b, c, d\}^{\star} | w = a^n, b^n, c^{\kappa}, d^{\kappa}\}| \ con \ n, \kappa \geq 0\}$$

L = $\{\epsilon, ab, aabb, aaabbb...$
$cd, ccdd, cccddd...$
$abcd, aabbcd, aaabbbcd, ...$
$abcd, abccdd, abcccddd, ...\}$
Si ha che X = $\{a^n, b^n\}$ e $Y = \{c^{\kappa}, d^{\kappa}\}$
Scriviamo la grammatica $S \rightarrow XY$
$X \rightarrow aXb|\epsilon$
$Y \rightarrow cYd|\epsilon$
Quindi alla fine abbiamo un linguaggio composto da due linguaggi, uno che è S, e poi rispettivamente X ed Y tali che: L = L_1L_2
$L_1 = \{a^nb^n | n \geq 0\}$ $L_2 = \{b^{\kappa}, b^{\kappa} | \kappa \geq 0\}$

Per concatenare due linguaggi devo concatenare una qualsiasi stringa presa da un linguaggio, e una qualsiasi presa dal secondo. Detto meglio:
Dati due linguaggi $L_1 ed L_2 \subseteq \Sigma^{\star}$,
$L_1 \circ L_2 = L_1L_2 = \{W | w = w_1w_2, con w_1 \in L_1 e w_2 \in L_2\}$, facciamo un esempio:

$\Sigma^{\star} = \{0, 1, a, b\} L_1 = \{\epsilon, 0, 00, 011\} L_2 = \{ab, b\} L_1L_2 = \{ab, b, 0ab, 0b, 00ab, 00b, 011ab, 011b\}$

Esercizio 6: Formare una CFG per il linguaggio

$$L = \{w \in \ \{a, b, c, d\}^{\star} | w = a^n, c^{\kappa}, d^{\kappa}, b^n | \ con \ n, \kappa \geq 0\}$$

Si ha che X = $\{b^{\kappa}, b^{\kappa}\}$ e notiamo che l'esercizio è praticamente come il precedente ma invece di concatenare andiamo ad inglobare uno dentro l'altro i linguaggi
Scriviamo la grammatica $S \rightarrow aSd|X$
$X \rightarrow bXc|\epsilon$
$Y \rightarrow cYd|\epsilon$

Vien fuori che:
$S \implies X \implies \epsilon$
$S \implies aSd \implies aXd \implies abXcd \implies abcd$

$S \implies X \implies bXc \implies bc$
$S \implies aSd \implies aXd \implies ad$

Esercizio 7: Formare una CFG per il linguaggio

$$L = \{w \in \ \{a,b,c\}^\star | w = a^n, c^\kappa, n^n | \ con\ n, \kappa \geq 0\}$$

Consideriamo come negli esempi precedenti che c^κsia una X (Ricordate X bestione? Quello del Dennunzio, ecco), per cui:
$S \rightarrow aSb|aXb$ }
$X \rightarrow cX|c$ } (Regolare)

Esercizio 8: Formare una CFG per il linguaggio

$$L = \{a^{n+m}xc^m yd^m n, m \geq 0\}$$

In quest'ultimo caso è leggermente più complesso perchè dovremo dividere il nostro a^{m+n} in due sotto casi.

- $a^n a^m$ MA non va bene perchè se lo traduco vien fuori

  $$a^n a^m x c^n y d^m$$

  E non va bene per via del fatto che c'è l'incrocio di m ed n

- $a^m a^n$ che è una soluzione accettabile perchè otteniamo:

  $$a^m a^n x c^n y d^m$$

  Se notate le n stanno dentro e le m stanno fuori, sono diciamo racchiuse, pertanto è corretto +

Esercizio 9: Formare una CFG per il linguaggio

$$L = \{a^{n+m}xc^n yd^m, \ con\ n, m \geq 0\}$$

Come possiamo notare avremo $a^n a^m x c^n y d^m$ che non va bene, non si può fare. Ma possiamo anche considerarla come $a^m a^n x c^n y d^m$

$S \rightarrow aSd|By$
$B \rightarrow aBc|x$
$S \implies By \implies xy\ S \implies aSd \implies aByd \implies aaBcyd \implies aaxcyd$

Precisazione: Poichè sono stato assente per prendere gli appunti della lezione successiva a questi esercizi ho lasciato nella cartella "Esercizi preAlberi" tutto ciò che è stato fatto prima del prossimo argomento. **Ringrazio di cuore Gaia per avere preso gli appunti di questa parte :)**

Capitolo 5

Alberi Sintattici

Un albero è una rappresentazione grafica che aiuta a comprendere in che modo una certa forma sentenziale con simboli terminali o variabili è stata ottenuta con la derivazione

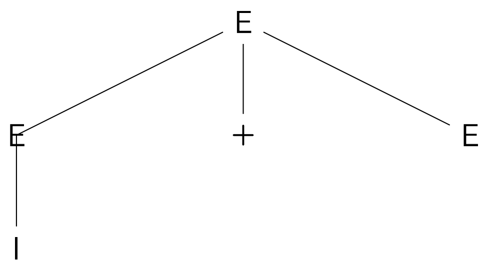
Definizione: Dato G = (V, T, P, S), l'albero sintattico è t.c.

- Ogni nodo interno è etichettato da una variabile
- Ogni foglia è etichettata da una variabile oppure un simbolo terminale, oppure anche ϵ

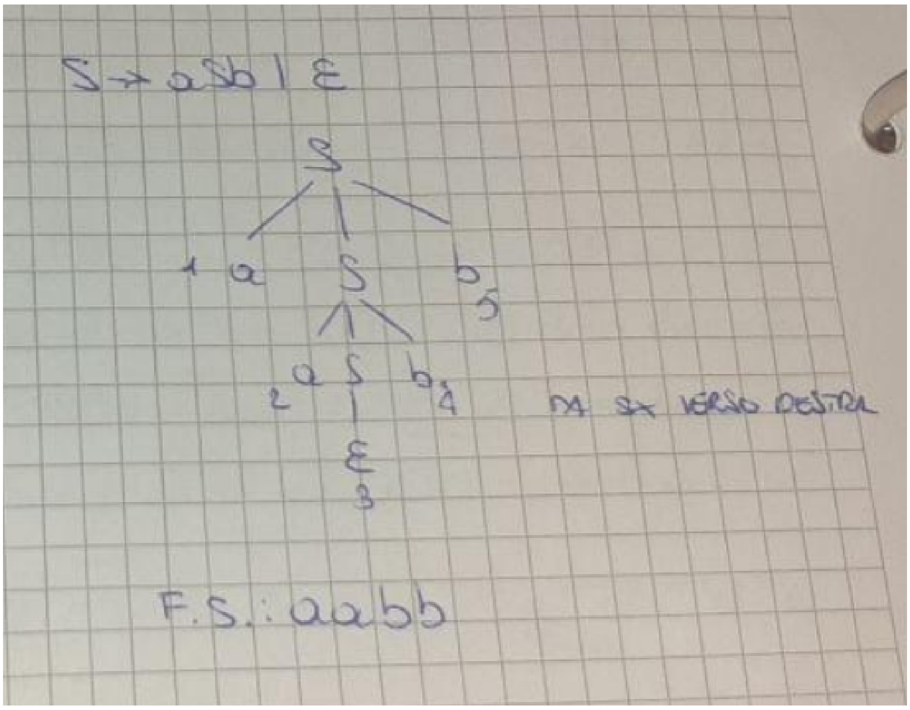
Però se è etichettata con ϵ significa che è l'unico figlio del padre. Inoltre, se un nodo interno è etichettato con A(variabile) e i figli sono etichettati da Sx verso Dx con $x_1, x_2, ..., x_\kappa$ allora $A \to x_1, x_2, ..., x_\kappa$ e P.

Esempio CFG: Dato $E \to I|E + E|E * E|(E)$ in cui la E sta per Espressione e la I indica un identificatore. E
$I \to a|b|Ia|Ib|I0|I1$

Forma Sentenziale I + E:



Perciò se avessimo: $S \to aSb|\epsilon$ verrebbe fuori:



La computazione di prolog non è altro che una visita di un albero in modo left most, a seconda di come si visita l'albero infatti cambia ma il prodotto finale rimane quello.

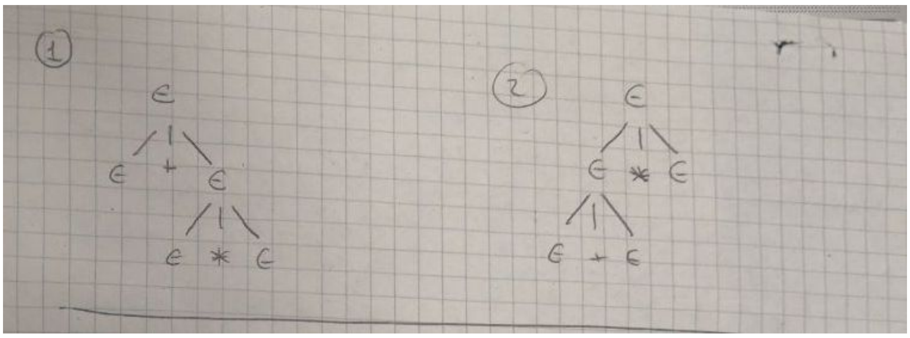
Dato un albero quindi non c'è un'unica derivazione.

Che rapporto c'è perciò tra alberi e derivazioni? Data una CFG (Grammatica context free) G i seguenti enunciati si equivalgono:

1. L'inferenza ricorsiva (quella della tabella in cui in base alla riga sapevam dire che valori poteva assumere partendo da quelli precedenti) stabilisce che W (stringa) è nel linguaggio della variabile A
2. Da A si può derivare in zero o più passi la stringa W
3. Se esiste una derivazione sinistra di W in 0 o più passi, allora esisterà per forza anche una derivazione da destra per W
4. Esisterà un albero sintattico con radice A e prodotto W

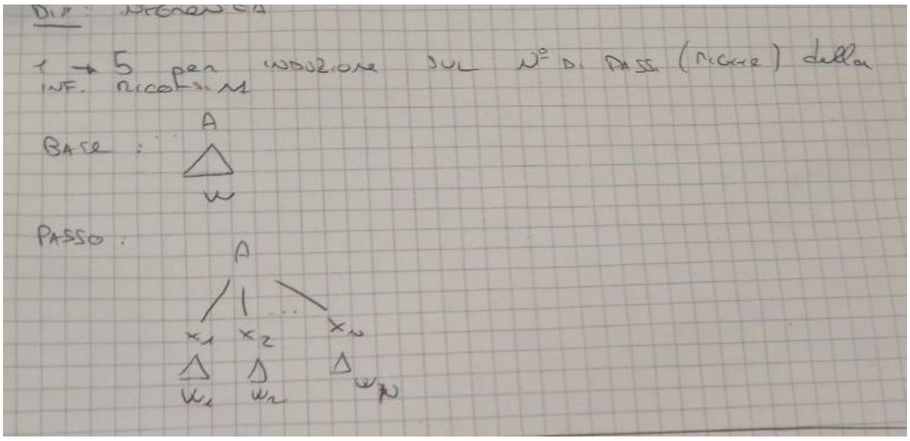
Per esempio:

fig 5.7 p 175



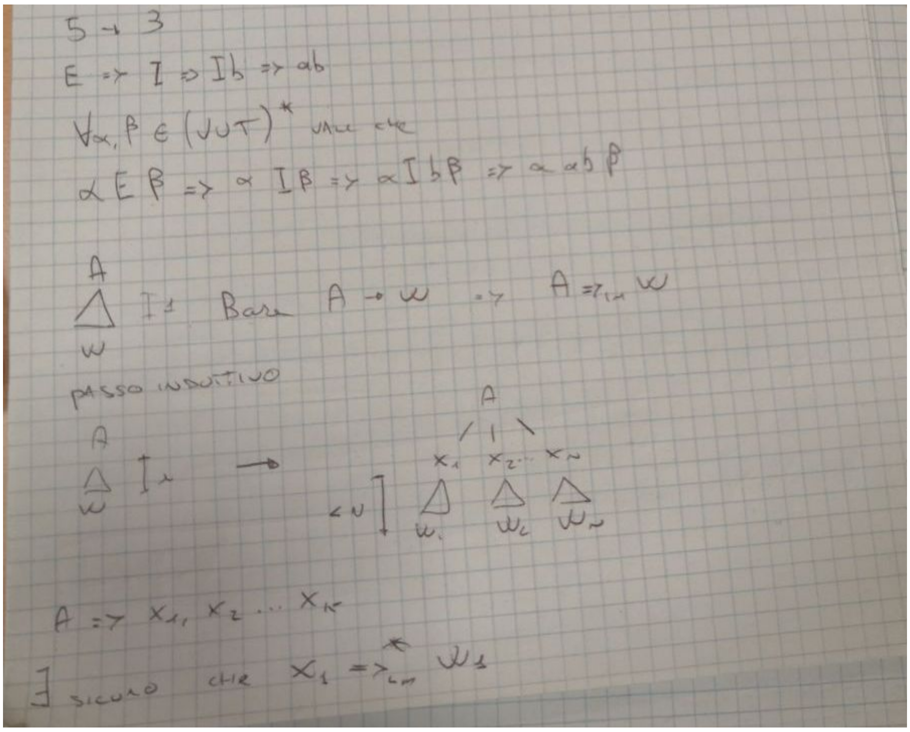
Nel caso dell'inferenza ricorsiva, si dimostra per induzione ($1 \to 5$) il numero di passi (righe)

dalla inferenza ricorsiva: $\begin{cases} base: \ A \to w \in P \\ passo\ induttivo: \ Inferenza \ \ ricorsiva \ \ di \ \ n+1 \ \ righe, \ \ ultima \\ \quad riga \ \ A \ \underbrace{\to w_1, w_2, ..., w_\kappa}_{w} \leq n \ righe \end{cases}$



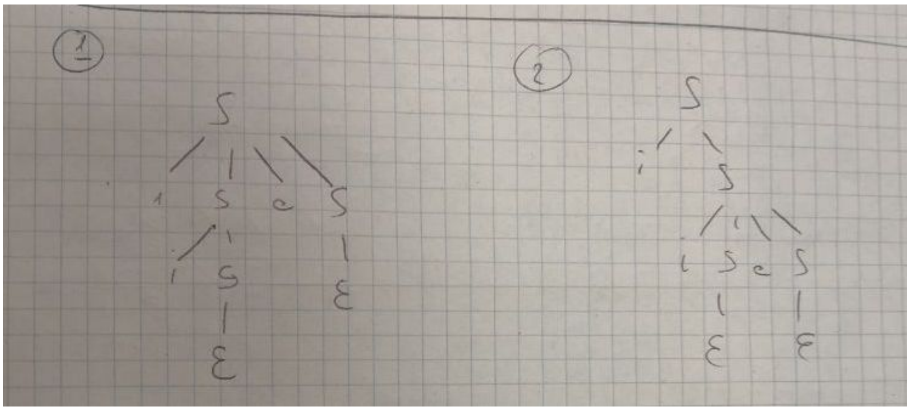
$E \implies I \implies Ib \implies ab \forall \alpha, \beta \in (V \cup T)^* \, vale\, che \alpha E \beta \implies \alpha I \beta \implies \alpha I b \beta \implies \alpha a\, b \beta$

Per induzione sull'altezza dell'albero:

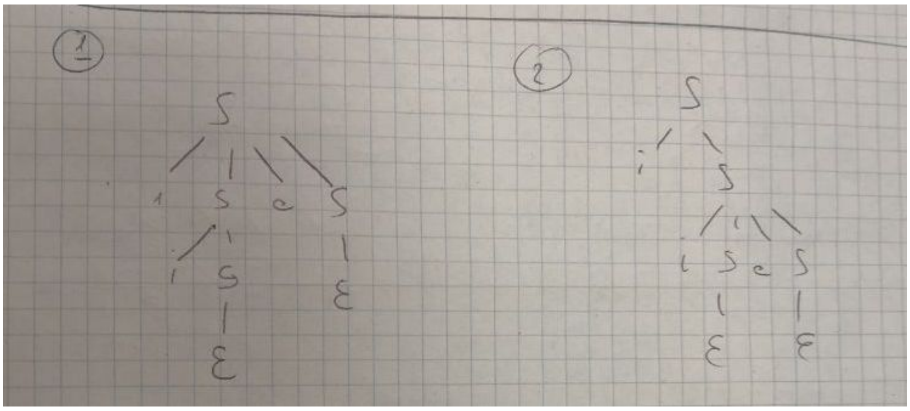


5.1 Ambiguità

E + E * E
$E \to E+E|E*E$ 1) $E \implies E+E \implies E+E*E$
2) $E \implies E*E \implies E+E*E$



Data la seguente CFG: $S \to \epsilon|SS|iS|iSeS$
Dobbiamo ottenere iie:
1) $S \implies iSeS \implies iiSeS \implies iieS \implies iie$
2) $S \implies iS \implies iiSeS \implies iieS \implies iie$



Se per una stringa ci sono più di un albero sintattico allora essa è ambigua, invece non c'è problema se lo stesso albero dia più derivazioni è easy. Ma perchè è un problema se è ambigua? Perchè non si sa come è cicciata fuori fondamentalmente.

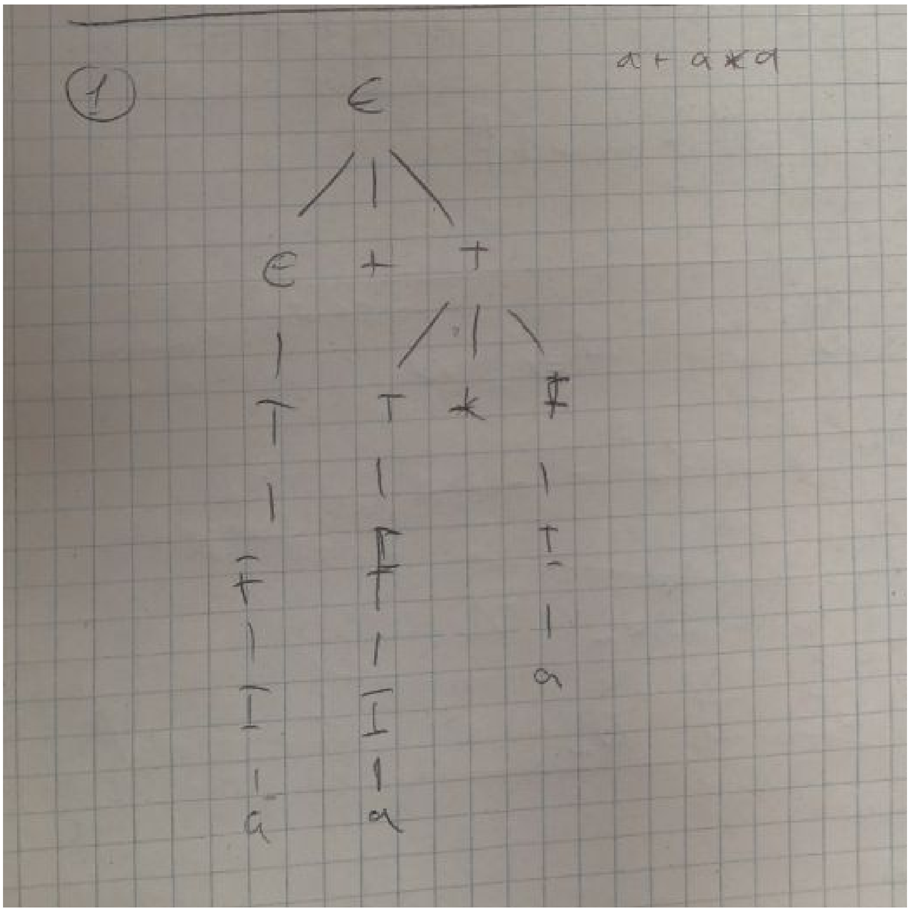
Non c'è un algoritmo che data una grammatica ti dica se è ambigua o non ambigua, c'è pure una dimostrazione ma non la vediamo, è uno di quei problemi per cui non si riesce a trovare una soluzione.

Non è detto che una grammatica ambigua sia trasformabile in una grammatica non ambigua PERO' in realtà in casi tipo Linguaggi di Programmazione (la materia dico) o negli

XML ci son delle regole che si sa che funzionino.

Ci son linguaggi che sono inerentemente ambigui, ma ora vediamo anche degli esempi.

Riprendiamo gli identificatori: I → a|b|Ia|Ib|I0|I1
$F \rightarrow I|(E)$
$T \rightarrow F|T * F$
$E \rightarrow T|E + T$



Teorema: $\forall CFG$
G = (V,T,P,S) e $\forall w \in T^{\star}$, w ha due alberi sintattici distinti se e solo se ha due derivazioni sx distinte.
(Solo se): Supponendo due alberi distinti
(Se): Supponendo due alberi di derivazioni sx distinte
C'è un oppure, nel senso che questo è applicabile anche alle derivazioni da dx.

Si ma come fa un albero ad avere due derivazioni sx diverse? Esempio:
1) $E \rightarrow E+E \implies I+E \implies a+E \implies a+E*E \implies A+I*E \implies a+a*E \implies a+a*I \implies a+a*a$
2) $E \implies E*E \implies E+E*E \implies I+E*e \implies a+E*E \implies A+I*E \implies a+a*E \implies a+a*I \implies a+a*a$

Vediamo ora un esempio di linguaggio inerentemente ambiguo:
$\nexists$ CFG non ambigua, cioè:
L = $\{a^n b^n c^m d^m | n, m \geq 1\} \cup \{a^n b^n c^m d^m | n, m \geq 1\}$
Definiamo le regole di inferenza:

$$\begin{cases} S \to AB|C \\ A \to aAb|ab \\ B \to cBd|cd \\ C \to aCd|aDd \\ D \to bDc|bc \end{cases}$$

Ora vediamo come derivare la seguente stringa: "aabbccdd": (n = m - 2) TUTTO LEFT-MOST
1) $S \implies AB \implies aAbB \implies aabbB \implies aabbcBd \implies aabbccdd$
2) $S \implies C \implies aCd \implies aaDdd \implies aabDcdd \implies aabbccdd$

Tutti i nodi vanno trasformati in questo, non si può trovare una grammatica fondamentalmente. Per non essere inerentemente ambiguo dovrebbe esserci una intersezione tra le due, e invece abbiamo ben due derivazioni sinistre diverse.

Per dimostrare davvero che sia inerentemente ambigue bisognerebbe fare una dimostrazione vera e propria ma diventa davvero complesso

5.2 Grammatiche regolari

Generano linguaggi di tipo 3 che si chiamano (Regolari)
G = (V, T, P, S): Analizziamo le Produzioni (P)

Le produzioni hanno i seguenti vincoli:

1. ϵ può comparire solo in $S \to \epsilon$ (S sta per start eh)
2. Le produzioni sono tutte lineari a dx oppure tutte lineari a sx
3. (a) lin a dx: $A \to aB$, oppure A→a con A,B∈V e a∈T
   (b) lin a sx: $A \to Ba$, oppure A→a con A,B∈V e a∈T

Vediamo subito un esempio:
$I \to a|b|Ia|Ib|I0|I1$ lin sx
Si vuole rappresentare "b01" da destra verso sinistra
$I \implies I1 \implies I01 \implies b01$
In pochi passaggi si è ottenuta subito la nostra stringa
Ora invece vediamo il lin dx
Ovvero si vuole rappresentare "b01"
$I \to a|b|aI|bI|0I|1I$
$I \implies bI \implies b01 \implies b01I...$?
Come vediamo non si risolve in questo caso, non esce la stringa, vediamo come fare.

Imponiamoci la regola che:

$$\begin{cases} I \rightarrow aJ|bJ|a|b \\ J \rightarrow a|b|aJ|bJ|0J|1J|0|1 \end{cases}$$

Vediamo subito un esempio per capirci meglio:
G = ({S},{0,1},P,S)
lin dx: $S \rightarrow \epsilon|0|1|0S|1S$
Da qui vediamo che dobbiamo escludere 0 ed 1 poichè $L(G) = \{0,1\}^\star$

Ora proviamo a produrre con lin sx 01101:
lin sx: $S \implies \epsilon|S0|S1$

Esercizio Si forniscano due grammatiche regolari lin dx ed sx per $L = \{a^n b^n | n, m \geq 0\}$

1. lin dx: G=({S,B}, {a,b},P,S)
   $S \rightarrow \epsilon|aS|bB$
   $B \rightarrow bB|b$, però così c'è un problema, nel senso che arriva alla B in cui può produrre soltando per l'appunto delle b, o comunque arriva che esce con b.

   Come si risolve questo problema? Aggiungendo la b singola alla prima espressione con S: $S \rightarrow \epsilon|aS|bB|b$

2. lin sx: G=({S,B}, {a,b},P,S)
   $S \rightarrow \epsilon|Sb|A0|a$
   Con $A \rightarrow Aa|a$

Esercizio: Grammatica lin dx e sx per:
L = $\{ab^n cd^m e | n \geq 0, m\, 0\}$

1. lin dx:
   $$\begin{cases} S \rightarrow aA \\ A \rightarrow bA|cB \\ B \rightarrow dB|dE \\ E \rightarrow e \end{cases}$$

2. lin sx:
   $$\begin{cases} S \rightarrow Xe \\ X \rightarrow Xd|Yd \\ Y \rightarrow Zc \\ Z \rightarrow Zb|a \end{cases}$$

3. lin dx: (modo equivalente) $A \rightarrow aB$
   $A \rightarrow w$
   $w \in T^+$

Per esempio:

$$A \to ciao \to \begin{cases} A \to cB \\ B \to iC \\ C \to aD \\ D \to o \end{cases}$$

4. lin sx:
$$\begin{cases} A \to Bo \\ B \to Ca \\ C \to Di \\ D \to c \end{cases}$$

Esercizio: Data G = ({S,T},{0,1},P,S)

$\begin{cases} S \to \epsilon|0S|1T \\ T \to 0T|1S \end{cases}$ L(G) =$\{w \in 0,1^{\star}|\ w\ contiene\ un\ n\ pari\ di\ 1\}$

Altro esercizio: Grammatica lin dx ed sx per:
L = -$\{w \in \{0,1\}^{\star} -$ | contiene almeno uno 0 oppure almeno un 1}
L = $\{0,1\}^{+}$

1. lin dx: G({S},{0,1},P,S)
   $\begin{cases} S \to 0|1|0S|1S \end{cases}$
2. lin sx: G({S},{0,1},P,S)
   $\begin{cases} S \to 0|1|S0|S1 \end{cases}$

Capitolo 6

Espressioni Regolari

Esistono linguaggi regolari (Di tipo 3) che possono generare grammatiche regolari(lin dx oppure lin sx) e rappresentare (dimostrare) espressioni regolari.
Tramite espressioni regolari puoi dimostrare che una stringa fa parte di un linguaggio.
Inoltre puoi riconoscerle (accettare) automi a stati finiti (DFA, NFA, ϵ-NFA)

6.1 Operazioni sui linguaggi

1. Unione: $L \cup M$ (Unione tra sistemi)
   Esempio:
   L = {001,10,111} M =$\{\epsilon, 001\}$ $L \cup M$ = {ϵ, 10, 001, 111}

2. Concatenazione ($L \cdot M$)
   Esempio:
   L = {001,10,111} M = $\{\epsilon, 001\}$ $L \cdot M$ = {ϵ001, ϵ10, ϵ111, 001001, 10001, 111001}

3. Chiusura di Kleene: $L^\star$
   L =$\bigcup\limits_{i=1} L^1 = \bigcup\limits_{i=1}^{\infty} L^1$
   Dove: $L^0 = \{\epsilon\}, L^1 = L$, fino ad arrivare a L^i = $\underbrace{LLLLL...L}_{i\ volte}$ i volte (per i $\geq$ 1)

   Esempio:
   L = {0,11}
   $L^\star = \{\epsilon\} \cup \{0, 11\} \cup \{00, 011, 110, 1111\} \cup \{000, 0011, 01111, 1100, 11011, 11110, 111111\} \cup$
   ...

Ci sono dei casi particolari

1. $L = \varnothing$ (Non contiene stringhe)

2. $L = \{\epsilon\}$(Cotiene solo la stringa vuota)

3. $\varnothing = \epsilon$

4. $\varnothing^i = \varnothing \quad \forall i \geq 1$

5. $\varnothing^\star = \varnothing^0, \varnothing^1, ..., \varnothing^n$

6. $\{\epsilon\}^\star = \{\epsilon\}^1, \{\epsilon\}^2, ..., \{\epsilon\}^n$ = $\{\epsilon\}$

7. Qualunque insieme se non vuoto, e non avente solo la stringa vuota, è in grado di generare qualsiasi stringa, di lunghezza che può estendersi fino ad ∞

Tornando alle epsressioni regolari:
Si dice che esse denotino linguaggi regolari, dice come sono fatte le stringhe, in maniera più esplicita e compatta di una grammatica, e si definiscono in modo ricorsivo,(matematicamente per induzione, sempre lì si torna).

Ci saranno dei casi base e una serie di passi, vediamo i 3 casi base:

1. $\epsilon, \varnothing$ sono espressioni regolari, perciò ϵ ha già più di un significato, di solito si indica in grassetto, per comodità in questo contesto sappiamo che indica un'espressione regolare (ϵ)
   L(ϵ) = $\{\epsilon\}, L(\varnothing) = \varnothing$

2. Se a $\in \Sigma$, a è una espressione regolare
   L(a) = {a}

3. Variabili che rappresentano linguaggi: (L per esempio) sono espressioni regolari

Vediamo ora i seguenti passi induttivi:

1. Unione: Se E, F sono ER, allora E+F è una ER (E+F = L(E) $\cup$ L(F))

2. Concatenazione: Se E, F sono ER, allora EF è una ER (L(EF) = L(E)L(F))
   Ad esempio
   $\Sigma^1 = \{0, 1\}$, E = 0, F = 1 EF=01

3. Chiusura: Se E è una ER, allora $E^\star$ è una ER ($E^\star = (L(E))^\star$)

4. Parentesi: Se E è una ER, allora (E) è una ER (L((E))=L(E))
   Vedrete lisp, a proposito di parentesi..

6.2 Precedenza operatori

1. Chiusura di Kleene ($\star$)

2. Concatenazione A $\circ$ B

3. Associativa, commutativa (l'unione)

6.3 Esercizi

Es. 1 Data la seguente ER = $(0+1)^\star 0^\star (01)^\star$
Dire se 001 è reaizzabile con il suddetto insieme:
$(0+1)^\star$ = 1 0 1 0
$L((0+1)^\star) = (L(0+1))^\star = (L(0) \cup L(1))^\star = (\{0\} \cup \{1\}) = (\{0,1\})^*$, quindi è possibile realizzarlo

Es. 2 Data la seguente ER = $0^\star(01)^\star$
Dire se 001 è reaizzabile con il suddetto insieme:
001 Si' 1001 NO
0101 Si'

6.4 Identità ed annichilatori

6.4.1 L'identità

E' un valore tale per cui la somma con esso (identità + valore) darà il valore stesso (a+x = x+a $\forall x \in$ ESISTENZA)
$\varnothing$ è un'identità per +: $\varnothing + E = E + \varnothing = E$
ϵ è identità per$\circ$: $\epsilon E = E\epsilon = E$

6.4.2 Annichilatore

$\varnothing$è annichilatore per$\circ$: $\varnothing E = E\varnothing = \varnothing$

Esercizio tipo esame Data ER = $((01)^\star 10(0+1)^\star)^\star$ E' possibile realizzare 0101? Nope
01000? Nemmeno
01011 Nemmeno
10111 SI PUO' FARE WOOOOO (Perdonatemi, lo svolgimento lo aggiungerò successivamente quando ci studierò sopra, perchè al momento mi è troppo lungo scriverlo)

6.5 Distributività

Aritmeticamente se ho $x(y+z)$, posso riscriverlo come $xy + xz$. Consideriamo due tipi di distributività:

- Distributività SX della concatenazione rispetto all'unione:
  $L(M+N) = LM + LN$
- Distributività DX: $(M+N)L = ML + NL$

Vediamo subito un esempio: Data una ER '$0 + 01^\star$'

$$L(0+01^\star) = L(0) + L(01^\star) = \{0\} \cup L(0)L(1^\star) =$$

$$= \{0\} \cup \{0\}(L(1))^\star) = \{0\} \cup \{0\}\{1\}^\star =$$
$$\{0, 01, 011, 0111, 01111, ...\}$$

Con:

$$\begin{cases} \{1\}^\star = \epsilon, 1, 11, 111, 1111, ... \\ \{0\}\{1\}^\star = \epsilon, 01, 011, 0111, 01111, ... \end{cases}$$

A questo punto passiamo a considerare '$0 + 01^\star$':

$$0 + 01^\star = 0\epsilon + 01^\star = 0(\epsilon + 1^\star) = 01^\star$$
$$= (0(\epsilon + 1^*)) = L(0)L(\epsilon + 1^*) =$$
$$= \{0\}(\{\epsilon\} \cup L(1))^*)...$$

6.6 Idempotenza

Dal punto di vista dell'unione si ha che L + L = L, ma se visto aritmeticamente viene leggermente più chiaro da capire, infatti:

$$x + x \neq x$$

La somma non gode dell'indepotenza

$$x \cdot x \neq x$$

Ed allo stesso modo non ne gode nemmeno il prodotto

6.7 Proprietà chiusura

$$(L^\star)^\star = L^\star$$
$$(L^\star)^\star = \bigcup_{i \geq 0} (L^\star)^i$$

Da questo si deriva che $\varnothing^\star = \epsilon$ e $\epsilon^\star = \epsilon$, ora passiamo ad L^+

$$L^+ = LL^* = L^*L = L + LL + LLL + LLL = L\epsilon + LL * LLL =$$
$$L(\epsilon + L + LL + LLL * ....) = LL^*$$

Esercizio 1: Data ER = $(00^*1^*)^*$
Fin da subito osserviamo che avremo una serie di blocchi tutti inizianti per 0 di questo tipo: $0xxxx0xxxx0xxxx$, potrebbero capitare blocchi aventi una serie di 0 ed una serie di 1.

- Posso avere una stringa tutta di 1? No
- Una stringa avente uno 0 e tutti 1? Sì

Non l'ho specificato ma $\epsilon \in L$
Quello che si nota è che quindi ogni stringa che avrò inizia per 0, perciò potrò scrivere qualcosa del tipo $0(0 + 1)^* + \epsilon$

Esercizio 2: Quale linguaggoo si denota da questa ER: $a(a + b)^*b$?
L = $\{w \in \{a, b\}^*|$ w inizia con a e termina con b$\}$, spiegato peggio sarebbero tutte le strinche che iniziano con a e b, ed all'interno hanno qualsiasi combinazione

Esercizio 3: Quale linguaggoo si denota da questa ER: $(0^*1^*)000(0 + 1)^*$?
E' in pratica come dire, qualsiasi insieme di 0 ed 1 (zeri ed uni) che in mezzo da qualche parte avranno un punto in cui ci sarà una sequenza di 3 zeri