1 Alfabeto, Stringhe, Linguaggi

1.1 Alfabeto

Un Alfabeto è un **insieme finito** di elementi detti **simboli** o **caratteri**.
La **cardinalità** è il numero di simboli dell'alfabet.

1.2 Stringhe

La **stringa vuota** è indicata con ϵ.

1.2.1 Operazioni sulle stringhe

Concatenazione Il simbolo è il punto (.) tra stringhe:
"nano.tecnologie" diventa "nanotecnologie".

Riflessione Consiste nello scrivere una stringa al contrario, ovvero invertire l'ordine dei suoi simboli (caratteri).
x^R denota la riflessione della stringa x.
La riflessione della concatenazione di due stringhe è la concatenazione inversa delle loro riflessioni:
$(xy)^R = y^R x^R$

Potenza m-esima La potenza della stringa x è al concatenazione di se stessa m volte.
La **potenza** ha la **precedenza** sul concatenamento:
$abbc^3 = abbccc$

1.3 Linguaggi

1.3.1 Operazioni sui linguaggi

Unione

$$A \cup B = \{x \mid x \in A \; or \; x \in B\}$$

Concatenazione Il concatenamento di due linguaggi L_1 ed L_2 (notazione L_1L_2) è l'insieme ottenuto concatenando in **tutti i modi possibili**

le stringhe di L_1 con le stringhe di L_2.

$$L_1L_2 = \{x \mid x = yz \;\&\; y \in L_1 \;\&\; z \in L_2\}$$

$\{ab, abc\}\{ab, aa, cb\} = \{abab, abaa, abcb, abcab, abcaa, abccb\}$

Star

$$A* = \{x_1x_2x_3 \ldots x_k \mid k \geq 0 \textit{ and ogni } x_i \in A\}$$

2 Automi finiti ed Espressioni regolari

2.1 Automi finiti

2.1.1 DFA - Automa Finito Deterministico

È una quintupla:

$$A = \{Q, \Sigma, \delta, q_0, F\}$$

Q è un insieme finito di **stati**.

Σ è un alfabeto finito (**simboli in input**).

δ è una funzione di transizione $Q \times \Sigma \implies Q$

q_0 $\in$ Q è lo **stato iniziale**.

F è l'insieme degli **stati finali**.
La funzione di transizione δ si può estendere alle stringhe:

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

Definizione:

$$\hat{\delta}(q, \epsilon) = q$$
$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$$

Il linguaggio riconosciuto (o accettato) da A è:

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

I linguaggi riconosciuiti (o accettati) da AUTOMI A STATI FINITI sono chiamati *linguaggi regolari*.

Esempio:

$$\begin{aligned}
\hat{\delta}(q_0, aabbc) &= \delta(\delta(\hat{\delta})q_0, aabb), c) \\
&= \delta(\delta(\delta(\hat{\delta})q_0, aab), b), c) \\
&= \delta(\delta(\delta(\delta(\hat{\delta})q_0, aa), b), b), c) \\
&= \delta(\delta(\delta(\delta(\delta(\hat{\delta})q_0, a), a), b), b), c) \\
&= \delta(\delta(\delta(\delta(\delta(\delta(\hat{\delta})q_0, \epsilon), a), a), b), b), c) \\
&= \delta(\delta(\delta(\delta(\delta(q_0, a), a), b), b), c) \\
&= \delta(\delta(\delta(\delta(q_0, a), b), b), c) \\
&= \delta(\delta(\delta(q_0, b), b), c) \\
&= \delta(\delta(q_1, b), c) \\
&= \delta(q_1, c) = q_3
\end{aligned}$$

2.1.2 NFA - Automa Finito Non-Deterministico

È una quintupla:

$$A = \{Q, \Sigma, \delta, q_0, F\}$$

Q è un insieme finito di **stati**.

Σ è un alfabeto finito (**simboli in input**).

δ è una funzione di transizione $Q \times \Sigma \implies 2^Q$

$q_0 \in Q$ è lo **stato iniziale**.

F è l'insieme degli **stati finali**.

Per semplicità possiamo estendere la definizione della funzione di transizione agli insiemi di stati:

$$\delta(\{r_1, r_2, \ldots, r_k\}, a) = \bigcup \{\delta(r_i, a) \mid i = 1, 2, \ldots, k\}$$

La funzione di transizione δ si può estendere alle stringhe:

$$\hat{\delta} : Q \times \Sigma^* \implies 2^Q$$

Definizione:

$$\hat{\delta}(q, \epsilon) = \{q\}$$
$$\hat{\delta}(q, xa) = \bigcup_{r \in \hat{\delta}(q,x)} \delta(r, a) = \delta(\hat{\delta}(q, x), a)$$

2.2 Conversione da NFA a DFA

2.2.1 Esempio 1

NFA:

	0	1
A	B	∅
B	B	B

DFA:

	0	1
A	B	C
B	B	B
C	C	C

C è lo stato morto, siccome il DFA non accetta ∅.
C viene inserito nella colonna degli stati perché per ogni stato possibile il DFA richiede un input.

2.2.2 Esempio 2

NFA:

	0	1
A	{A}	{A,B}
B	∅	∅

DFA:

	0	1
A	{A}	{A,B}
AB	A∪∅ = {A}	{AB}

AB è uno stato singolo.
AB invece di mettere B perché B non può essere raggiunto da nessuno degli stati precedenti.
Nella riga di AB, colonna degli stati, faccio l'unione di A e B per ogni input.

2.2.3 Esempio 3

NFA:

	a	b
$\Longrightarrow$ A	{A,B}	{C}
B	{A}	{B}
C	-	{A,B}

DFA:

	0	1
A	AB	C
AB	AB	CB
BC	A	AB
C	D	AB
D	D	D

BC e C sono stati finali perché sono quelli che contengono la C, che è lo stato finale dell'NFA.

2.3 Conversione da ϵ-NFA a NFA

Per convertire da ϵ-NFA a NFA bisogna usare la **ϵ-chiusura**.

ϵ-chiusura La **ϵ-chiusura** indica tutti gli stati raggiungibili da un determinato stato **solo** con l'inserimento di ϵ (carattere vuoto).

Stati Finali Diventano stati finali tutti quelli che, tramite la lettura di ϵ, raggiungono lo stato finale dell'ϵ-NFA.

Il **numero degli stati** rimane **uguale**. **Incrementa** il numero degli stati **finali**.

2.4 Espressioni Regolari

Una espressione regolare è un modo *dichiarativo* per descrivere un linguaggio regolare (ne fornisce una descrizione algebrica).

Esempio: 01 + 10
Un **Linguaggio regolare** su un alfabeto Σ è un linguaggio che può essere espresso mediante:

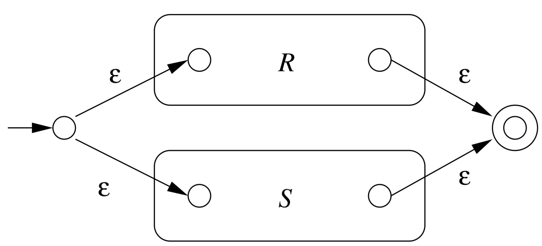
- Chiusura ($*$)
- Concatenazione (.)
- Unione (+)

Esempi:

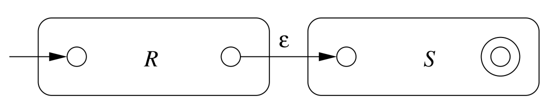
- $ab + bca$ **denota l'insieme** $\{ab, bca\}$
- ab^* **denota l'insieme** $\{abn|n \geq 0\}$
- $(ab)^+ + bca$ **denota l'insieme** $\{bca\} \cup (ab)n|n > 0$
- $(aa)^*$ **denota l'insieme** $\{a2n|n \geq 0\}$
- $(aa)^*a$ **denota l'insieme** $\{a2n + 1|n \geq 0\}$
- $(a + b + (cc)^*)^*$ **denota l'insieme** $\ldots$ (esercizio!)
- $1(0 + 1)^* + 0$ **denota i numeri binari**

2.4.1 Da espressione regolare ad automa

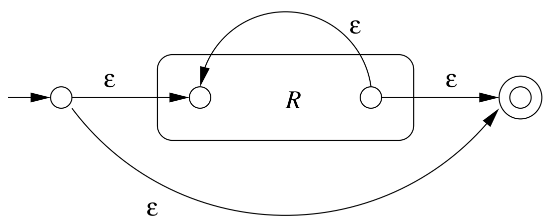
Unione



Concatenazione



Chiusura



2.4.2 Pumping Lemma

Sia L un linguaggio regolare.
Allora $\exists n$, che dipende solo dal linguaggio, tale che $\forall w \in L,\ |w| \geq n, w$ si può scrivere come la concatenazione di tre sottostringhe xyz tali che:

$$y \neq \epsilon$$
$$|xy| \leq n$$
$$\forall k \geq 0,\ xy^kz \in L$$

3 Grammatiche

3.1 Derivazioni di una Grammatica

La Grammatica è una quadrupla (V, T, P, S):

- V: insieme delle **variabili** o **non terminali**
- T: insieme dei **terminali**
- P: insieme delle **produzioni** o **regole di riscrittura**
- S: il simbolo **iniziale** o **assioma**. Da questo si parte, e con le regole, si generano le stringhe che formeranno il linguaggio.

Seguendo queste regole si generano delle stringhe che formano un linguaggio.

Esempio Consideriamo le seguente grammatica G_1

$$G_1 = (\{S, A\}, \{a, b\}, S, \{S \implies aAb, aA \implies aaAb, A \implies \epsilon\})$$

$$
\begin{aligned}
S &\implies \underline{aA}b \\
&\implies a\underline{aA}bb \\
&\implies aaa\underline{A}bbb \\
&\implies aaabbb
\end{aligned}
$$

3.2 Grammatiche Context-Free

La Grammatica Context-Free è una quadrupla (V, Σ, P, S):

- V: insieme delle **variabili** o **non terminali**
- Σ: insieme dei **terminali**
- P: insieme delle **produzioni** o **regole di riscrittura**
- S: il simbolo **iniziale** o **assioma**. Da questo si parte, e con le regole, si generano le stringhe che formeranno il linguaggio.

La **quadrupla** contiene queste regole che, seguite, generano delle stringhe. L'insieme di queste **stringhe** formano il **linguaggio** L generato dalla grammatica G:

$$L(G)$$

3.3 Alberi Sintattici

Un albero è un **albero sintattico** se:

- Ogni nodo interno è etichettato con una **variabile** V.
- Ogni foglia è etichettata con un simbolo in $V \cup T \cup \{\epsilon\}$.

Il **prodotto** di un albero sintattico è la stringa di foglie da **sinistra a destra**. Le derivazioni possono essere:

- Leftmost - si espandono per prima le **variabili** più a sinistra.
- Rightmost - si espandono per prima le **variabili** più a destra.

Una Grammatica si dice **ambigua** se esiste **almeno una** stringa che abbia **due o più alberi** sintattici.

3.4 Automi a Pila - Pushdown Automata (PDA)

Un **PDA** è un modo per implementare una Context-Free Grammar in un modo simile in cui implementiamo un **Linguaggio Regolare** usando gli **Automi a Stati Finiti**.
Essi hanno **più memoria** grazie allo **stack**.
Un PDA ha 3 componenti:

- Input - stringa di input
- Finite Control Unit - in base all'input fa la **PUSH** o **POP** dallo **stack**
- Stack (con memoria infinita)

PDA è una tupla di 7 elementi:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

dove

- Q è un insieme finito di stati
- Σ è un alfabeto finito di input
- Γ è un alfabeto finito di pila
- δ è la funzione di transizione
- q_0 è lo stato iniziale
- $Z_0 \in \Gamma$ è il simbolo iniziale per la pila
- $F \subseteq Q$ è l'insieme di stati di accettazione

3.4.1 Diagrammi di transizione

Un PDA può essere rappresentato da un diagramma di transizione in cui le etichette sono della forma

$$a, X/\gamma$$

in cui

- a è il simbolo di **input**
- X è la **cima della pila**
- γ è la stringa che sostituisce X sulla pila

3.4.2 Descrizioni istantanee

Un PDA passa da una configurazione ad un'altra configurazione consumando un simbolo di input usando le **descrizioni istantanee** (ID) del PDA. Una ID e' una tripla (q, w, γ) dove

- q è lo stato in cui si trova l'automa
- w è l'input rimanente (es. 0010). Lettura da sinistra a destra.
- γ è il contenuto della pila (es. ZZZ_0)

3.4.3 Da CFG a PDA

Sia $G = (V, T, Q, S)$ una CFG. Definiamo P_G come

$$(\{q\}, T, V \cup T, \delta, q, S)$$

dove

$$\delta(q, \epsilon A) = \{(q, \beta) \ : \ A \to \beta \in Q\}$$

per $A \in V$, e

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

per $a \in T$.
Example

$$S \to aSbS \mid bSaS \mid \epsilon$$

è definito da

$$\begin{aligned}
&\delta(q, \epsilon, S) = \{(q, aSbS), (q, bSaS), (q, \epsilon)\}\\
&\delta(q, a, a) = \{(q, \epsilon)\}\\
&\delta(q, b, b) = \{(q, \epsilon)\}
\end{aligned}$$

per esempio:

$$(q, ab, S) \vdash (q, ab, aSbS) \vdash (q, b, SbS) \vdash (q, b, bS) \vdash (q, \epsilon, S) \vdash (q, \epsilon, \epsilon)$$

3.5 $FIRST$

Data una grammatica e una **stringa** α.
$FIRST(\alpha)$ è l'insieme dei **terminali** con cui *iniziano* le stringhe derivabili dalla stringa data (α).
Formalmente:

$$FIRST(\alpha) = \{a \mid \alpha \rightarrow^* a\beta\} \cup \{\epsilon \mid se\ \alpha \rightarrow^* \epsilon\}$$

1. se X è un simbolo **terminale** (lettera minuscola) allora $FIRST(X) = X$.

2. se $X \implies \epsilon$ è una **produzione** allora **aggiungi** $FIRST(X) = \{\epsilon\}$.

3. se X è un **non-terminale** e $X \implies Y_1, Y_2 \dots Y_K$ è una **produzione** allora **aggiungi** $FIRST(Y_1)$ al $FIRST(X)$
   se $Y_1 \implies \epsilon$ (deriva ϵ) allora **aggiungi** $FIRST(Y_2)$ al $FIRST(X)$.

3.6 $FOLLOW$

Data una grammatica e una **variabile** A.
$FOLLOW(A)$(insieme dei **seguiti**) è l'insieme dei terminali con cui *iniziano* le stringhe che seguono A nelle derivazioni.
Formalmente:

$$FOLLOW(A) = \{a \mid S \rightarrow^* \alpha A a \beta\} \cup \{\$ \mid se\ S \rightarrow^* \alpha A\}$$

1. se X è il **simbolo di start** allora $FOLLOW(X) = \$$.

2. se $A \implies \alpha B \beta$ è una **produzione** allora $FOLLOW(B) = FIRST(\beta)$ eccetto ϵ, se B contiene ϵ.

3. se $A \implies \alpha B \beta$ o $A \implies \alpha B \beta$ e $FIRST(\beta)$ contiene ϵ allora **aggiungi** $FOLLOW(A)$ al $FOLLOW(B)$.

3.7 Insieme Guida

L'**Insieme Guida** di una produzione (es. $A \rightarrow \alpha$) indica l'**insieme dei TERMINALI** con cui iniziano le stringhe generabili partendo dalla produzione stessa (\$ se ci si trova alla fine della parola).
Sostanzialmente serve a capire quali sono le **iniziali** (terminali, ad es. $a, b, c, \dots$) delle stringhe che si possono generare da una specifica produzione.

- $GUI(A \to \alpha) \implies FIRST(\alpha)$ oppure
- $GUI(A \to \alpha) \implies FIRST(\alpha) \cup FOLLOW(A)$ se ϵ appartiene al $FIRST(\alpha)$

	FIRST	FOLLOW
$S \implies ABCDE$	$\{a, b, c\}$	$\{\$\}$
$A \implies a \mid \epsilon$	$\{a, \epsilon\}$	$\{b, c\}$
$B \implies b \mid \epsilon$	$\{b, \epsilon\}$	$\{c\}$
$C \implies c$	$\{c\}$	$\{d, e, \$\}$
$D \implies d \mid \epsilon$	$\{d, \epsilon\}$	$\{e, \$\}$
$E \implies e \mid \epsilon$	$\{e, \epsilon\}$	$\{\$\}$

3.8 Parser $LL(1)$

3.9 Proprietà principali delle Grammatiche $LL(1)$

- Non ambigua (esiste solo una derivazione left-most)
- Assenza di Ricorsioni Sinistre

3.9.1 Ricorsioni Sinistre

Si ha una **Ricorsione Sinistra** quando una variabile *chiama* se stessa

$$S \implies S\alpha$$

questo non è gestibile da un Parser Top-Down. Una produzione del seguente tipo

$$A \implies A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_m$$

(le stringhe β_i non iniziano con A)
si può sostituire con una **fattorizzazione destra** in questo modo

$$\begin{aligned} A &\implies \beta_1 A' \mid \dots \mid \beta_m A' \\ A' &\implies \epsilon \mid \alpha_1 A' \mid \dots \mid \alpha_n A' \end{aligned}$$

3.9.2 Fattorizzazione Sinistra

Si ha una **Fattorizzazione Sinistra** quando una variabile ha multiple produzioni con lo stesso prefisso

$$A \implies \mathbf{ab}c \mid \mathbf{ab} \mid \mathbf{ab}b$$

e si risolve così

$$\begin{aligned} A &\implies \mathbf{ab}A' \\ A' &\implies c \mid \epsilon \mid b \end{aligned}$$

3.10 Analizzatore a Discesa Ricorsiva

Ad ogni variabile A con produzioni

$$\begin{aligned} A &\to \alpha_1 \\ A &\to \alpha_2 \\ &\dots \\ A &\to \alpha_k \end{aligned}$$

si associa una procedura:

```
function A()
      if (cc ∈ GUI(A → α1))
            body (α1)
      else if (cc ∈ GUI(A → αa))
            body (αa)
      ...
      else if (cc ∈ GUI(A → αk))
            body (αk)
      else ERRORE (...)
```

4 Traduzione Diretta della Sintassi

4.1 Attributi Ereditati o Sintetizzati

Sintetizzati Un attributo sintetizzato per una variabile A in un nodo n dell'albero di parsificazione è definito da una regola semantica associata alla produzione in n, e il suo valore è calcolato solo in termini dei valori degli attributi nei nodi figli di n ed in n stesso. (A è il simbolo a sinistra nella produzione).

$$E \to E_1 + T \ \{E.val = E_1.val + T.val\}$$

Ereditati Un attributo ereditato per una variabile A in un nodo n dell'albero di parsificazione è definito da una regola semantica associata alla produzione nel nodo padre di n e il suo valore è calcolato solo in termini dei valori degli attributi del padre di n, di n stesso e dei suoi fratelli. (A è un simbolo nel corpo della produzione, cioè al membro destro).

$$A \to BCD\{C.in = A.in,\ C.type = B.type\}$$

4.2 Definizioni S-attribuite ed L-attribuite

S-attribuite Una **SDD** è S-attribuita se tutti gli attributi sono sintetizzati.

L-attribuite Una **SDD** è L-attribuita se gli attribuiti sono:

- **sintetizzati** oppure
- **ereditati** e soddisfano il seguente vincolo:
  per ogni produzione $A \to X_1, X_2, \ldots, X_n$ ogni attributo ereditato di X_j dipende solo da:
  - attributi ereditati o sintetizzati dei simboli $X_1, X_2, \ldots, X_{j-1}$ a sinistra di X_j nella produzione
  - attributi ereditati di A
  - attributi ereditati o sintetizzati di X_j purché non vi siano cicli nel grafo delle dipendenze formati dagli attributi di questa occorrenza di X_j.
    Cioè nel grafo delle dipendenze gli archi tra gli attributi associati a una produzione possono andare da sinistra a destra, ma non da destra a sinistra.

Esempio:

$$P_1: \ S \to MN \ \{S.val = M.val + N.val\}$$
$$P_2: \ M \to PQ \ \{M.val = P.val * Q.val \ and \ P.val = Q.val\}$$

P_1 è L-attribuita, invece P_2 non lo è.
In P_1, S è un attributo **sintetizzato** e nelle definizioni L-attribuite è permesso.
Invece P_2 non rispetta la definizione L-attribuita siccome P dipende da Q che è alla sua **destra**.

5 DOMANDE

5.1 Quando una grammatica è LL(1)?

Se per ogni coppia di produzioni a partire da uno stesso simbolo non terminale A :
$A \implies \alpha$ e $A \implies \beta$, si ha
$GUI(A \implies \alpha) \cap GUI(A \implies \beta) = \Phi$.
Ovvero, gli **insiemi guida** $GUI()$ di un simbolo con due produzioni distinte non coincidono (non hanno elementi in comune).

5.2 Cosa è un linguaggio regolare? Come si verifica?

5.3 Differenza tra linguaggi Context-Free e non? Come lo stabilisco?

Regular grammar is either right or left linear, whereas context free grammar is basically any combination of terminals and non-terminals. Hence you can see that regular grammar is a subset of context-free grammar.

5.4 Quando una grammatica è ambigua?

Una grammatica è ambigua se almeno una frase del linguaggio generato è ambigua.
Una frase è ambigua se ha almeno due **alberi sintattici** distinti.

5.5 Completare la definizione della funzione di transizione σ_D dell'automa a stati finiti deterministico

5.6 Quando due stati p e q sono indistinguibili? E i-indistinguibili?

Sono i-indistinguibili se e solo se nessuna stringa di lunghezza $\leq i$ distingue p da q.
Sono indistinguibili se e solo se nessuna stringa (di qualunque lunghezza) distingue p da q.

5.7 Due stati i-indistinguibili quando sono i+1-distinguibili?

Questo si verifica quando con la stessa stringa di input i due stati raggiunti appartengono a due sottoinsiemi diversi dalla Partizione i (Π_i)

5.8 Quando un automa è minimo?

Un automa è minimo quando la partizione degli stati contiene i gruppi degli stati indistinguibili.

5.9 Quando una grammatica è LL(1)?

Una grammatica è $LL(1)$ se per ogni non terminale A e per ogni coppia di produzioni

$$A \to \alpha$$
$$A \to \beta$$

gli insiemi guida sono disgiunti:

$$GUI(A \to \alpha) \cap GUI(A \to \beta) = \Phi$$

Una grammatica **non può essere** $LL(1)$ se è **ricorsiva sinistra**.
Perché l'intersezione degli insiemi GUI delle produzioni sono uguali.

5.10 Differenza tra SDD S-attribuiti e l-attribuiti

- $S-$attribuiti tutti gli attributi sono sintetizzati
- $L-$attribuiti gli attributi possono essere sia sintetizzati che ereditati, ereditati con le seguenti regole:
  - pagina 29/36 di 4/22-traduzione1.pdf

5.11 MEMO

I Linguaggi Regolari sono sottoinsiemi dei CFG (Context Free Grammar).
Linguaggi context free (generati da CFG) riconosciuti da

- PDA
- NPDA

Linguaggi regolari riconosciuiti da

$$\left.\begin{array}{l} \bullet\ \text{DFA} \\ \bullet\ \text{NFA} \\ \bullet\ \epsilon\text{-NFA} \end{array}\right\}\ \text{Riconoscono linguaggi regolari}$$

Minimizzazione Automi Per Minimizzare NFA occorre prima convertirli in DFA e poi minimizzare il DFA.

6 Domande PDF

6.1 Parte I

1. **Cos'è un alfabeto?**

   Un insieme di simboli con cui comporre stringhe

2. **Cos'è una stringa?**

   Un'insieme di simboli (tra quelli dell'alfabeto) che rispetta le regole di costruzione di una certe grammatica

3. **Quando due stringhe sono uguali?**

   Se i loro caratteri, letti ordinatamente da sx a dx, coincidono.

4. **Cos'è una sottostringa?**

   La stringa Y è una sottostringa della stringa X se esistono delle stringhe U e V tali che
   $X = UYV$

5. **Cos'è un prefisso?**

La stringa Y è un prefisso della stringa X se esiste una stringa V tale che $X = YV$

6. **Cos'è un suffisso?**

   La stringa Y è un suffisso della stringa X se esiste una stringa U tale che $X = UY$

7. **Cos'è la concatenazione? Quali proprietà ha?**

   La concatenazione di due stringhe è la stringa formata dai simboli della prima stringa seguiti da quelli della seconda.

- NON è commutativa
- E' associativa
- HA un ELEMENTO NEUTRO
- la lunghezza della stringa concatenazione è la somma delle lunghezze delle due stringhe.

8. **Cos'è la riflessione? Quali proprietà ha?**

   La riflessione è la stringa ottenuta scrivendo i caratteri in ordine inverso.
   Il riflesso del riflesso di una stringa è la stringa stessa
   La riflessione della concatenazione di due stringhe è la concatenazione inversa delle loro riflessioni.
   La riflessione della stringa vuota è la stringa vuota
   La riflessione ha la precedenza sul concatenamento.

9. **Cos'è la potenza?**

   La potenza n-esima della stringa X è il concatenamento di X con se stessa n volte.

10. **Cos'è un linguaggio?**

Un linguaggio è un insieme di stringhe su quell'alfabeto

11. **Cos'è l'insieme dei prefissi?**

12. **Quando un linguaggio è privo di prefissi?**

Se non contiene i prefissi delle sue stringhe.
$L \cap$ Prefissi $(L) = \Phi$

13. **Cosa sono la riflessione, la concatenazione e la potenza di un linguaggio?**

- La riflessione di un linguaggio L è l'insieme delle stringhe riflesse di L
- La concatenazione di due linguaggi $L1$ e $L2$ è l'insieme ottenuto concatenando in tutti i modi possibili le stringhe di $L1$ con le stringhe di $L2$
- La potenza n-esima di un linguaggio L è il concatenamento di L con se stesso n volte.

14. **Cos'è la chiusura di Kleene di un linguaggio L? Quali proprietà ha?**

E' l'unione di tutte le potenze di L.
Le proprietà:

- monotonia
- chiusura rispetto al concatenamento
- idempotenza
- commutatività della riflessione con la chiusura di Kleene e con la potenza.

15. **Che cos'è la chiusura non riflessiva?**

E' l'unione di tutte le potenze positive di L

16. **Che cos'è il linguaggio universale su un alfabeto Σ ?**

E' la sua chiusura di Kleene.

17. **Che cos'è il complemento di un linguaggio L su un alfabeto Σ rispetto ad un alfabeto Δ?**

E' la differenza tra Δ^* ed L

6.2 Parte II