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

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

Concatenazione Il concatenamento di due linguaggi L_1 ed L_2 (notazione $L_1 L_2$) 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:

$$A = \{Q, \Sigma, \delta, q0, F\}$$

Q è un insieme finito di **stati**.

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

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

$q0 \in Q$ è lo **stato iniziale**.

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

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

Definizione:

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

2.1.2 NFA - Automa Finito Non-Deterministico

È una quintupla:

$$A = \{Q, \Sigma, \delta, q0, F\}$$

Q è un insieme finito di **stati**.

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

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

$q0 \in Q$ 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:

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

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	$\emptyset$
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 $\emptyset$.
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	$\emptyset$	$\emptyset$

DFA:

	0	1
A	{A}	{A,B}
AB	A$\cup\emptyset$ = {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
→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.

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

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

$S \to \underline{aA}b$
$\to a\underline{aA}bb$
$\to aaa\underline{A}bbb$
$\to aaabbb$

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
- Σ
- Γ

- Δ è la funzione di transizione
- q_0
- Z_0
- F

3.4.1 $FIRST(X)$ di una variabile in $LL(1)$

1. se X è un simbolo **terminale** (lettera minuscola) allora $FIRST(X) = X$.
2. se $X \to \epsilon$ è una **produzione** allora **aggiungi** $FIRST(X) = \{\epsilon\}$.
3. se X è un **non-terminale** e $X \to Y_1, Y_2 \dots Y_K$ è una **produzione** allora **aggiungi** $FIRST(Y_1)$ al $FIRST(X)$ se $Y_1 \to \epsilon$ (deriva ϵ) allora **aggiungi** $FIRST(Y_2)$ al $FIRST(X)$.

3.4.2 $FOLLOW(X)$ di una variabile in $LL(1)$

1. se X è il **simbolo di start** allora $FOLLOW(X) = \$$.
2. se $A \to \alpha B \beta$ è una **produzione** allora $FOLLOW(B) = FIRST(\beta)$ eccetto ϵ, se B contiene ϵ.
3. se $A \to \alpha B \beta$ o $A \to \alpha B \beta$ e $FIRST(\beta)$ contiene ϵ allora **aggiungi** $FOLLOW(A)$ al $FOLLOW(B)$.

3.4.3 Insieme Guida

- $GUI(X) \to FIRST(X)$ oppure
- $GUI(X) \to FIRST(X) \cup FOLLOW(X)$ se ϵ appartiene al $FIRST(X)$

3.5 Parser $LL(1)$

3.6 Proprietà principali delle Grammatiche $LL(1)$

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

3.6.1 Ricorsioni Sinistre

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

$$S \rightarrow Sa|b$$

3.6.2 Fattorizzazione Sinistra

Si ha quando una variabile ha multiple produzioni con lo stesso prefisso

$$A \rightarrow \mathbf{ab}c|\mathbf{ab}|\mathbf{ab}b$$

e si risolve così

$$A \rightarrow \mathbf{ab}A'$$

$$A' \rightarrow c|\epsilon|b$$

4 DOMANDE

4.1 Quando una grammatica è LL(1)?

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

4.2 Cosa è un linguaggio regolare? Come si verifica?

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

4.4 Quando una grammatica è ambigua?

4.5 Quando un automa è minimo?

4.6 Quando una grammatica è LL(1)?