KER(LFA)

Dimostrazioni:

1) L LINGUAGGIO RICORSIVO → L^C LINGUAGGIO RICORSIVO
2) L RICORSIVO = L RICORSIVAMENTE NUMERABILE
3) L RICORSIVAMENTE NUMERABILE ⇔ L AMMETTE G CHE LO GENERA

[PARTE 1] INTRODUZIONE

- **Σ^*:** insieme di tutte le parole costruibili sull'alfabeto Σ, includendo ε;

- **Σ^+:** insieme di tutte le parole costruibili sull'alfabeto Σ, escludendo ε;

- Prodotto di giustapposizione = indicato con un puntino e non è altro che la concatenazione di due parole dello stesso alfabeto. Non è operazione commutativa ovviamente.

- Monoide libero generato da Σ = l'insieme Σ^* con l'operazione "·" e "ε" (l'elemento neutro di Σ^* rispetto all'operazione "·") costituisce un monoide, più precisamente la **terna (Σ^*, ·, ε)** è monoide libero generato da Σ

- Sottoparole di una parola:

  1) Fattore: sequenza di simboli "f" contenuti in "w" tali che $w = h \cdot f \cdot j$.

  2) Prefisso: sequenza di simboli "p" contenuti in "w" tali che $w = p \cdot j$.

  3) Suffisso: sequenza di simboli "s" contenuti in "w" tali che $w = h \cdot s$.

- Ci sono **4 tipi di linguaggi:**

  1) **∅:** linguaggio vuoto, cardinalità è zero, privo di qualsiasi elemento;
  2) **{ε}:** linguaggio di ε, cardinalità è uno, composto solo dalla parola vuota;
  3) **L FINITO:** cardinalità finita, composto da un numero finito d'elementi;
  4) **L INFINITO:** cardinalità infinita,composto da un numero d'elementi inenumerabile

- Le principali operazioni sui linguaggi sono:

  1) **UNIONE:** $A \cup B = \{w \in \Sigma^* \mid w \in A$ OR $w \in B\}$. A e B sono sottoinsiemi di $A \cup B$
  2) **INTERSEZIONE:** $A \cap B = \{w \in \Sigma^* \mid w \in A$ AND $w \in B\}$ $A \cap B$ è sottoinsieme di A e B
  3) **COMPLEMENTO:** $A^c = \{w \in \Sigma^* \mid w \notin A\}$ inoltre $A \cap A^c = \emptyset$
  4) **PRODOTTO:** $AB = \{w \mid w = x \cdot y$ con $x \in A$ e $y \in B\}$, non è operazione commutativa
  5) **POTENZA:** $A^k = A \cdot A \cdot A \cdot \ldots \cdot A$ (per k volte)
  6) **CHIUSURA DI KLEENE:** si indica in 2 modi diversi:
     a) $L^* = \{w \mid w = x_1 \cdot x_2 \cdot \ldots \cdot x_n$ dove $x_i \in L$ con $1 \le i < n, n \ge 1\} \cup \{\varepsilon\}$
     linguaggio formato dalle parole ottenute moltiplicando, in tutte le permutazioni possibili, parole di L, unitamente alla parola ε

     b) $L^+ = \{w \mid w = x_1 \cdot x_2 \cdot \ldots \cdot x_n$ dove $x_i \in L$ con $1 \le i < n, n \ge 1\}$
     L^n è il linguaggio formato dalle parole ottenute moltiplicando, in tutte le permutazioni possibili, parole di L prese "n" alla volta.

     $L^* = L^+ \cup L^0$
     $L^+ \neq L^* \setminus \{\varepsilon\}$

[PARTE 2] CODICI, PROCEDURE,ALGORITMI E PROGRAMMI

- Codice: Un linguaggio L si definisce codice, se ogni parola appartenente ad L^+ può essere decomposta in maniera univoca, come prodotto di parole di L → proprietà di decifrabilità. Deve quindi esserci uno e un solo modo di ottenere w come prodotto di parole di L

- Codici prefissi: linguaggio L in cui ogni parola non è prefisso di nessun'altra parola

- Codifica: Dato un alfabeto A e un codice C $\subseteq \Sigma^*$, una codifica di A in C è una corrispondenza biunivoca f: A → C. L'elemento a ∈ A è codificato da f(a) e si scriverà a = f(a).

- I linguaggi vengono rappresentati in modo diverso, dipende se sono finiti o infiniti:

  1) **Rappresentazione linguaggi FINITI**
     - rappresentato estensivamente, mediante quindi l'elencazione di tutti gli elementi (parole) che compongono il linguaggio

  2) **Rappresentazione linguaggi INFINITI**
     - Rappresentato attraverso una proprietà P(w) tale che L = {w | P(w) = True}
     - Possono essere descritti a loro volta tramite:
       a) **Sistemi RICONOSCITIVI**
          - Sistema in grado di calcolare la funzione caratteristica tramite un algoritmo chiamato riconoscitore
          - Funzione caratteristica: funzione che permette di individuare, se una parola appartiene o meno al linguaggio L
       - → True se appartiene
       - → ~~False~~ se non appartiene
       b) **Sistemi GENERATIVI**
          - permettono di costruire parole del linguaggio → grammatiche

- Procedura = sequenza finita di istruzioni che possono essere eseguite automaticamente e che **possono portare ad un risultato** se il programma è progettato per fornirlo.

- Algoritmo = procedura che **termina sempre**, qualsiasi sia l'input fornitogli in ingresso, un algoritmo non prevede quindi situazioni di loop infiniti o di break;

- Programma = strumento mediante il quale vengono definite le procedure e gli algoritmi. Hanno 4 connotazioni:
  1) **SINTATTICO:** un programma è una parola w ∈ {0, 1}* ossia una sequenza di bit. Viene quindi inteso sintatticamente come un insieme di parole su Σ* ={0,1}

  2) **SEMANTICO:** un programma "w" è una procedura che genera una sequenza di passi di calcolo che possono terminare fornendo un risultato, indichiamo con F_w la semantica. Se "w" sintatticamente è una parola binaria che corrisponde al codice ASCII di un programma e "x" (input al programma) è sintatticamente una parola binaria → $F_w(x)$ = risultato dell'esecuzione del programma

  3) **NON TERMINANO:** programmi che vanno in loop → **$F_w(x)\uparrow$**

  4) **TERMINANO:** programmi che terminano o in 0 o in 1 → **$F_w(x)\downarrow$**

- Interprete: programma che prende in input un altro programma w ed un dato x. Se w è programma allora interprete porterà in output $F_w(x)$ in caso contrario porterà in output un risultato indefinito → $\perp$

[PARTE 3] CLASSIFICAZIONE LINGUAGGI

- Linguaggio RICORSIVO:

  - L è ricorsivo se esiste un algoritmo implementato dal programma tale che dando in input un dato $x \in \{0, 1\}^*$:
    - $F_w(x) = 1$ sse $x \in L$
    - $F_w(x) = 0$ sse $x \notin L$

  - Se L è ricorsivo allora il programma w termina sempre in quanto implementa un algoritmo

  - Linguaggio che può essere specificato con una quantità finita di informazioni; un linguaggio ricorsivo è quindi un linguaggio che ammette un sistema riconoscitivo.

  - dato $x \in \Sigma^*$, è possibile decidere se $x \in L$ eseguendo il programma w con input = x

- Linguaggio RICORSIVAMENTE NUMERABILE:

  - L ricorsivamente numerabile se esiste una procedura implementata dal programma tale che dando in input $x \in \{0 ,1\}^*$:
    - $F_w(x) = 1$ sse $x \in L$
    - $F_w(x)\uparrow$sse $x \notin L$

  - Se L è ricorsivamente numerabile allora il programma w non termina sempre in quanto implementa una procedura

  - Linguaggio che può essere specificato con una quantità finita di informazioni; risponde 1 quando $x \in L$ e non termina l'esecuzione quando $x \notin L$

  - se $x \notin L$ non è possibile recuperare l'informazione in un numero finito di passi di calcolo → L non ammette Sistema riconoscitivo

  - on risulta sempre possibile decidere se $x \in L$ eseguendo il programma w con input = x

    ➔ questo perché se $x \notin L$ il programma entra in loop e non vi è risultato da recuperare. In generale il risultato potrebbe metterci molto ad uscire in output e potremmo per sbaglio pensare che il programma sia entrato in loop e quindi ritenere che x non appartiene al linguaggio quando in realtà vi appartiene

- Linguaggio è sinonimo di Problema. A ciascun linguaggio esiste un problema indicato con la dicitura P_L. Un linguaggio quindi ha questa dicitura $L = \{w \mid P(w) = 1\}$
  - L ricorsivo ↔ P_L decidibile
  - L ricorsivamente numerabile ↔ P_L semidecidibile

- P_L in linea generale è cosi definito:
  - Se $w \in L$ allora, dato che $L = \{w \mid P(w) = 1\}$ w soddisfa anche P_L
  - Se $w \notin L$ allora, dato che $L = \{w \mid P(w) = 1\}$ w non soddisfa neanche P_L

- **L LINGUAGGIO RICORSIVO → L^C LINGUAGGIO RICORSIVO**

  - Se L è linguaggio ricorsivo allora esiste un algoritmo implementato da A tale che:

    - $F_A(x) = 1$ sse $x \in L$
    - $F_A(x) = 0$ sse $x \notin L$

  - Ricordiamo che **$L^c = \{w \in \Sigma^* \mid w \notin L\}$.** Si necessita quindi di un altro programma B tale che:

    - $F_B(x) = 0$ sse $x \in L$
    - $F_B(x) = 1$ sse $x \notin L$

  - Vediamo ora B come programma che riceve in input l'output di A e che quindi:

    - **$F_B(x) = 0$ sse $F_A(x) = 1$**
    - **$F_B(x) = 1$ sse $F_A(x) = 0$**

  - Come visto il programma B implementa un algoritmo = Linguaggio è Ricorsivo

L RICORSIVO = L RICORSIVAMENTE NUMERABILE

- Per per la dimostrazione bisogna dimostrare
  1) L ricorsivo ⊆ L ricorsivamente numerabile
  2) L ricorsivamente numerabile ⊆ L ricorsivo

1) *ricorsivo ⊆ ricorsivamente numerabile*

- L è un linguaggio ricorsivo, allora ∃ w tale che, input = x ∈ {0, 1}*:
  - **Fw(x) = 1 sse x ∈ L**
  - **Fw(x) = 0 sse x ∉ L**
- Si costruisce ora una procedura per la quale quando Fw(x) = 0 genera un loop quindi una computazione che non termina. Per farlo basta "peggiorare un algoritmo in modo da fargli fare un loop"

2) *Ricorsivamente numerabile ⊆ ricorsivo*

- Definisco D come un linguaggio ric num ma non ricorsivo:

  → D = { x ∈ {0, 1}* | Fu (x, x)↓ }
  → D^C = { x ∈ {0, 1}* | Fu (x, x)↑ }

- Adesso devo provare che:

a) **D E' RIC NUM**

- Implementiamo una procedura che prende in input x ∈ {0, 1}* e richiama l'interprete Fu passandogli x$x. Se x appartiene al linguaggio, return 1 se non appartiene, rimane in loop e non returna nulla.

```
RICNUM(x) {
      y = Fu(x$x);
      return 1;
}
```

- return 1 sse appartiene al linguaggio e e fa loop se non appartiene → è RIC NUM

b) **D NON E' RICORSIVO**

- Supponiamo D come linguaggio ricorsivo e implementiamo questa procedura

```
ASSURDOA(x){
      if(appartiene(x,D))
            return 1 - Fu(x$x);
      else
            return 0;
}
```

- Se codifichiamo ASSURDOA come programma "e" e lo passiamo a se stesso avremo in esecuzione l'interprete sul programma "e" con input "e"
  L'interprete u prende il programma 'e' e lo esegue passandogli quale input il programma 'e'

- Supponiamo che “e” appartenga a D, allora ci sarà un assurdo:

Fu(e,e) = 1 - Fu(e,e)

- Supponiamo che “e” non appartiene a D allora ci sarà comunque assurdo:

Fu (e,e) = 0

→ se il programma “e” non appartiene a D, allora vuol dire che non termina con input = “e” e di conseguenza si arriva ad un assurdo, perché il programma ASSURDOA(e) = 0

- Per ipotesi D linguaggio ricorsivo (quindi ∃ ‘e’). “e” non esiste quindi →ipotesi FALSA → **D NON è ricorsivo**

c) D^C NON E’ RIC NUM

- si utilizza la tecnica dell’assurdo
- quindi D è ric num e implementa una procedura “z” (che è RICNUM)
- anche D^C è ric num secondo la supposizione ed implementa “y” (che è RICNUM)
- si tenga a mente che

$x \in D^C \rightarrow x \notin D$

$x \in D \rightarrow x \notin D^C$

- e quindi $z \downarrow$ (termina) $\leftrightarrow$ $x \in D$

$y \downarrow$ (termina) $\leftrightarrow$ $x \notin D$

- Si utilizza ora una terza procedura “k” che passa ad entrambe (z e y) un input “x”:
  - Se $x \in D$ return 1
  - Se $x \notin D$ return 0

- In questo modo tramite un algoritmo posso riconoscere il linguaggio D.
  - algoritmo → D ricorsivo → ho dimostrato prima che D non è ricorsivo → **D^C NON E’ RIC NUM**

[PARTE 4] LE GRAMMATICHE

- sono un sistema generativo
- ci sono alcuni termini importanti da tenere a mente:

1) **METASIMBOLO:** posto all'interno dei tag <…> costituisce una parte di testo da sostituire con simboli terminali

2) **SIMBOLO TERMINALE:** costituisce la parte fissa di testo da non sostituire

3) **PAROLA:** sequenza di simboli terminali, se w è una parola $w \in \Sigma^*$

4) **FORMA SENTENZIALE:** sequenza di simboli terminali e metasimboli. Se f è forma sentenziale allora $f \in (M \cup \Sigma)^*$

5) **ASSIOMA:** è il metasimbolo di partenza

6) **GRAMMATICA:** sistema definito dalla seguente quadrupla di elementi **$G = <\Sigma, M, P, S>$**

7) **PRODUZIONI:** regola di scrittura della forma $\alpha \rightarrow \beta$ dove

    - **$\alpha \in (\Sigma \cup M)^+$** → tutte le parole, simboli terminali o metasimboli esclusa ε
    - **$\beta \in (\Sigma \cup M)^*$** → tutte le parole, simboli terminali o metasimboli inclusa ε

- l'applicazione di una regola di produzione si chiama **derivazione.** Esistono 2 "tipi" di derivazioni:

    1) **Derivazioni in un passo $(x\ \alpha\ y) \Rightarrow (x\ \beta\ y)$**

        - date due parole $w, z \in (\Sigma \cup M)^*$, una derivazione in un passo è l'applicazione di una regola di produzione che trasforma w in z

    2) **Derivazioni in zero o più passi $(x\ \alpha\ y) \Rightarrow^* (x\ \beta\ y)$**

        - date due parole w e z, si dice <u>$w \Rightarrow^* z$</u> se z può essere ottenuta da w applicando un numero finito di regole di produzione

        - se w = z allora si dice in zero passi

8) **LINGUAGGIO GENERATO DA G: $L(G) = \{w \mid w \in \Sigma^* \text{ AND } S \Rightarrow^* w\}$**

9) **GRAMMATICHE EQUIVALENTI:** G1 e G2 sono equivalenti se L(G1) = L(G2). Equivalenti <u>non</u> significa comunque <u>avere stessa struttura e regole di derivazione</u>!

L RICORSIVAMENTE NUMERABILE ⇔ L AMMETTE G CHE LO GENERA

- Si dimostra che
  - (A) L ammette una grammatica G ⇒ L è enumerabile
  - (B) L è enumerabile ⇒ ammette una grammatica G

1) L ammette una grammatica G ⇒ L è enumerabile

- serve una procedura che
  - Per w ∈ L(G) return 1,
  - Per w ∉ L(G) non termini
- la procedura Elenca qua sotto elenca tutte le parole generate dalla grammatica G

```
Elenca (){
        F 0 = {S}
        i = 1
        while (i > 0) do
                costruisci Fi
                Ti = Fi ∩ Σ*
                Output(Ti)
                i = i+1
}
```

- si costruisce adesso la procedura definita al primo punto modificando Elenca:

```
Procedura w (x ∈ Σ*){
        F0 = {S}
        I=1
        While (i>0) do
                Costruisci Fi
                Ti=Fi ∩ Σ*
                If (x ∈ Ti ) return 1
                I=i+1
}
```

[PARTE 5] LA CLASSIFICAZIONE DI CHOMSKY

- si definiscono due versioni della classificazione

PRIMA VERSIONE

- **G di Tipo 0:** le regole di produzione sono completamente arbitrarie
- **G di Tipo 1:** ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che $|\beta| \geq |\alpha|$ (= non ammesse cancellazioni)
- **G di Tipo 2:** ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che **α sia un metasimbolo**
- **G di Tipo 3:** ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che si presenti in una delle seguenti due forme: $\alpha \rightarrow x$ oppure $\alpha \rightarrow y\beta$, con α e β metasimboli e x e y simboli terminali

- Un linguaggio si definisce di tipo k quando ammette una grammatica G di tipo k che lo genera
- Indicheremo con Rk la classe dei linguaggi di tipo k, esistono quindi 4 classi di linguaggi:
  - **classe R3:** linguaggi regolari
  - **classe R2:** linguaggi liberi da contesto
  - **classe R1:** linguaggi dipendenti da contesto
  - **classe R0:** linguaggi ricorsivamente numerabili

SECONDA VERSIONE

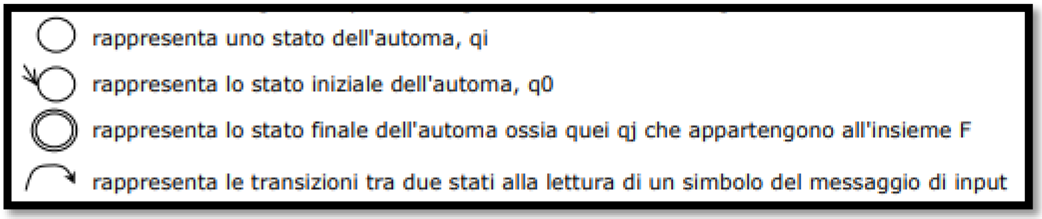
- **G di Tipo 0:** le regole di produzione sono completamente arbitrarie
- **G di Tipo 1:** ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che $|\beta| \geq |\alpha|$. Inoltre:
  - si aggiunge in M il metasimbolo S'
  - si aggiunge in P la regola $S' \rightarrow \varepsilon$ e $S' \rightarrow S$
- **G di tipo 2:** ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che **α sia un metasimbolo**
  - ~~$\beta \in (\Sigma \cup M)^+$~~ ma ora $\beta \in (\Sigma \cup M)^*$ (consente di aggiungere $A \rightarrow \varepsilon$)
- **G di tipo 3:** ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che si presenti in una delle seguenti due forme:
  - $\alpha \rightarrow x$ ⇨ con $\alpha \in M$ e $x \in \Sigma^*$ (consente creazione di regole nella forma $A \rightarrow \varepsilon$.)
  - $\alpha \rightarrow y\beta$ ⇨ con $\alpha \in M$, $y \in \Sigma^*$ e $\beta \in M$

- Le grammatiche di tipo 3 sono identificate da regole di produzione che assumono una delle seguenti forme:
  1) $A \rightarrow x$ oppure $A \rightarrow yB$
  2) $A \rightarrow \sigma$ oppure $A \rightarrow \sigma B$ oppure $A \rightarrow \varepsilon$
  3) $A \rightarrow \sigma B$ oppure $A \rightarrow \varepsilon$
  4) $A \rightarrow \sigma B$ oppure $A \rightarrow \sigma$

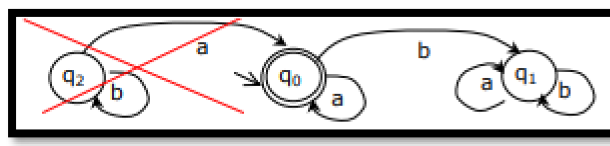
  → dove $A, B \in M$, $x, y \in \Sigma^*$, $\sigma \in \Sigma$

[PARTE 6] AUTOMI A STATI

- Un automa a stati "A" è un sistema A descritto dalla quintupla di elementi: **A = <Σ, Q, δ, Q₀, F>** dove
  - **Σ** = alfabeto dei simboli in input
  - **Q** = insieme degli stati finiti
  - **δ** = funzione di transizione con dominio **Q x Σ → Q**
  - **Q_0** = stato iniziale del sistema
  - **F** = insieme degli stati finali



- due automi si dicono equivalenti se riconoscono lo stesso linguaggio
- Gli automi a stati contengono spesso uno stato "**Trappola**":
  - Se dallo stato qi non si specifica una tipologia di transazione, questa, se accade, porta automaticamente allo stato trappola
  - è uno stato che **non consente di ritornare all'albero**, non è quindi possibile uscire in alcun modo dallo stato T
- gli stati non osservabili in un automa sono irrilevanti per quanto riguarda il riconoscimento e possono essere tranquillamente soppressi dall'automa, rendendo l'automa osservabile
- in questo caso per esempio q2 non è osservabile quindi si può rimuovere tranquillamente.



- un automa **completamente specificato:** esistono tanti archi uscenti quanti sono i simboli appartenenti a Σ
- un automa **non completamente specificato**: ∃ almeno uno stato qk il quale ha un numero di archi uscenti da detto stato sono inferiore al numero di simboli di Σ.

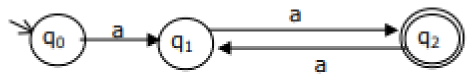
- stati **distinguibili ≉**: esiste almeno una parola w in Σ* tale che, lo stato raggiunto nell'automa, partendo da q_1, leggendo detta parola w sia diverso dallo stato raggiunto nell'automa, partendo da q_2

- stati **indistinguibili ≈:** lo stato raggiunto nell'automa, partendo da q1, leggendo una qualsiasi parola w deve essere obbligatoriamente uguale allo stato raggiunto nell'automa, partendo da q2, leggendo la medesima parola w, questo deve accadere per tutte le parole appartenenti a Σ*

- **Automa massimo:** l'automa osservabile con il maggior numero di stati (infiniti), che definisce il linguaggio L dove parole diverse corrispondono a stati diversi. ⇨ **ciascuna parola deve raggiungere un diverso stato** in modo da aumentare il numero di stati presenti dell'automa stesso e **ciascuno stato deve essere raggiunto da una e una sola parola**
- **Automa minimo:** automa con il minor numero di stati che definisce il linguaggio L

- **SE L È LINGUAGGIO REGOLARE ↔ ∃ AUTOMA A STATI FINITI CHE LO DEFINISCE**

  Per esempio:



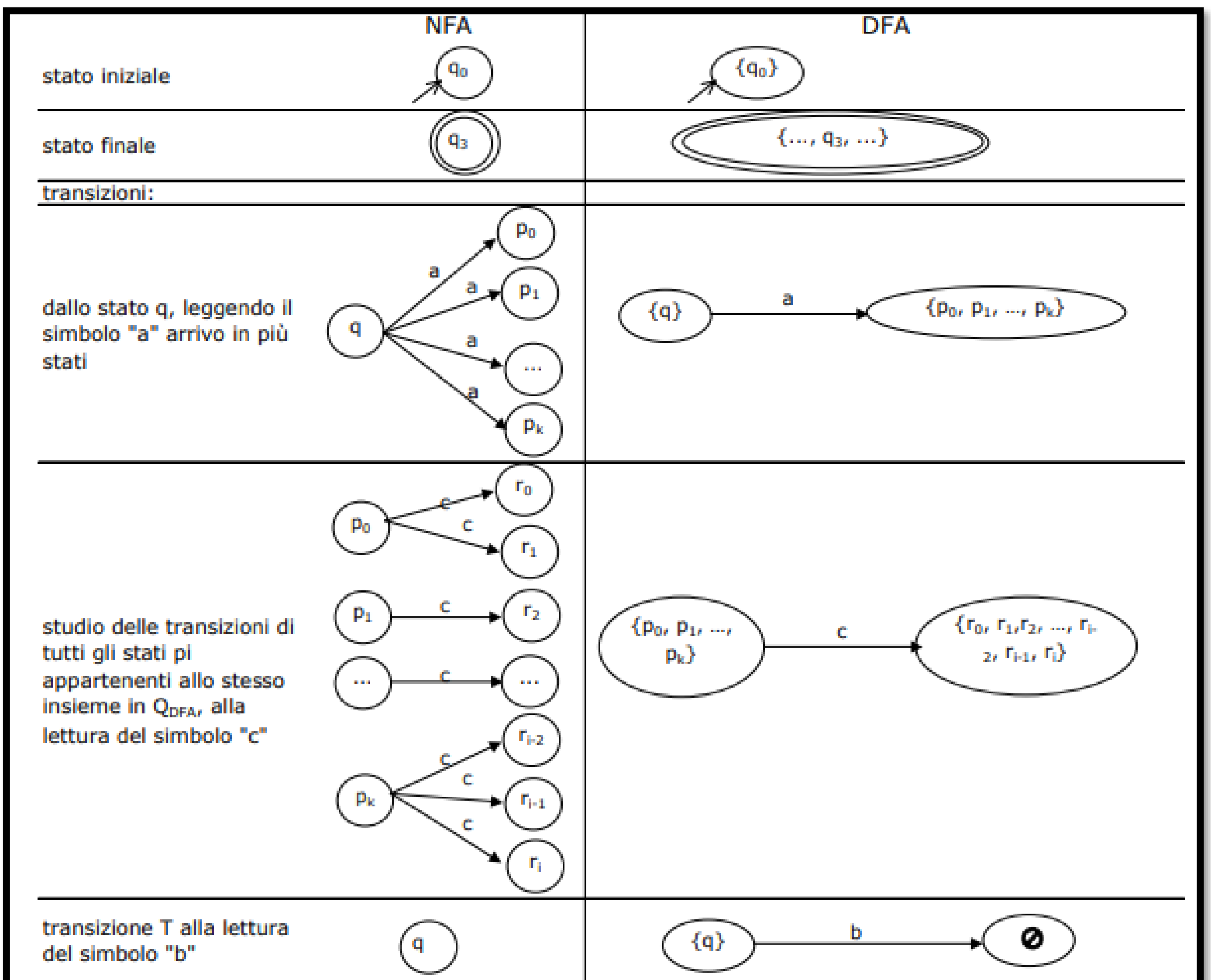
  G = <**Σ** = {a}, **M** = {q0, q1, q2}, **q0**, **P** = {q0 → aq1, q1 → aq2, q2 → ε| aq1}>

- **Automa non deterministico (NFA):** sistema A = <Σ, Q, q_0, R, F> dove **Q è un insieme finito di stati, Σ è un alfabeto** e **R è l'insieme delle relazioni di transizione**, non si parla quindi più di funzione bensì di relazione di transizione non deterministica:
  - **R (q, σ, p) = 0** "non si può raggiungere p da q tramite la lettura di σ "
  - **R (q, σ, p) = 1** "si può raggiungere p da q tramite la lettura di σ "

  → un esempio della sua utilità è lo string matching

- **Automa deterministico (DFA):** è un particolare automa non deterministico A = <Σ, Q, q_0, R, F> dove è sempre possibile sostituire la funzione di transizione δ con una relazione di transizione R

- **per ogni automa a stati finiti non deterministico è possibile effettuare la costruzione di un automa a stati finiti deterministico a lui equivalente.**



[PARTE7] ESPRESSIONI REGOLARI

- servono per denotare un linguaggio
- Sono espressioni regolari base: **∅ , ε, σ ∈ Σ**
- se p e q sono espressioni regolari allora lo sono anche **p + q**, **p x q**, **p* (chiusura di Kleene)**
- ad ogni espressione regolare p associamo un linguaggio L
  - ∅ → ∅
  - ε → {ε}
  - σ → {σ}
  - p → L1
  - q → L2
  - p+q → L1 ∪ L2
  - pxq → L1 x L2
  - p* → L1*

- **TEOREMA DI KLEENE:** **L' è denotato da un'espressione regolare ↔ L riconosciuto da DFA** (pag 68)

[PARTE 8] LINGUAGGI LIBERI DAL CONTESTO

- un linguaggio è detto acontestuale se è generato da una grammatica di tipo 2
- una grammatica G si dice di tipo 2 se ogni regola è della forma $\alpha \rightarrow \beta$ dove $\alpha \in M$
- Data una grammatica di tipo 2, che genera il linguaggio L(G), un albero di derivazione della parola $w \in L(G)$ in G è un albero ordinato composto da:
    - **Radice =** etichettata con assioma della grammatica
    - **Nodi =** interni all'albero e sono etichettati con meta-simboli
    - **Foglie =** posti alle estremità inferiori di ciascun ramo
    - **Archi =** indicano l'applicazione di una regola di produzione

ESEMPIO

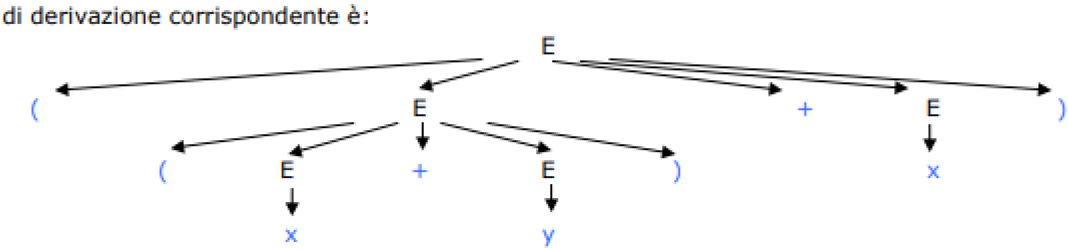
Data la grammatica per le espressioni aritmetiche sulle incognite x e y:

$G = <\Sigma = \{(,), +, x, y\}, M = \{E\}, P = \{E \rightarrow (E + E) \mid x \mid y\}, E>$

Generiamo la parola w = ((x + y) + x) mediante left most

$E \Rightarrow (E + E) \Rightarrow ((E + E) + E) \Rightarrow ((x + E) + E) \Rightarrow ((x + y) + E) \Rightarrow ((x + y) + x)$

L'albero di derivazione corrispondente è:



- Derivazione **left-most**: quando una derivazione viene effettuata derivando sempre a partire dal metasimbolo più a sinistra. Nell'esempio sopra si utilizza.
- Data una grammatica G, affermeremo che **due derivazioni sono equivalenti** se hanno associato lo stesso albero di derivazione.
    - Per ogni albero, infatti, esiste **una e una sola derivazione left most** e **per ogni derivazione left most esiste uno e un solo albero di derivazione**.

- Una grammatica G di tipo 2 si dice **ambigua** se **genera almeno una parola ambigua**
    - Se $\exists\, w \in L(G)$ che ammette due diversi alberi di derivazione $\Rightarrow$ w è ambigua $\Rightarrow$ G è ambigua
    - Se $\exists\, w \in L(G)$ che ammette due diverse derivazioni left most $\Rightarrow$ w è ambigua $\Rightarrow$ G è ambigua
    - Se $\exists\, w \in L(G)$ che ammette due diversi significati $\Rightarrow$ w è ambigua $\Rightarrow$ G è ambigua

- Una grammatica G di tipo 2 si dice **non ambigua** se genera tutte e sole parole non ambigue
    - Se $\forall\, w \in L(G)\ \exists!$ Albero di derivazione $\Rightarrow$ w non è ambigua $\Rightarrow$ G non è ambigua
    - Se $\forall\, w \in L(G)\ \exists!$ derivazione left most $\Rightarrow$ w non è ambigua $\Rightarrow$ G non è ambigua
    - Se $\forall\, w \in L(G)\ \exists!$ significato $\Rightarrow$ w non è ambigua $\Rightarrow$ G non è ambigua

- un linguaggio è **inerentemente ambiguo** se **ogni grammatica che lo genera è ambigua**

[PARTE 9] FORME NORMALI DI CHOMSKY E GREIBACH, AUTOMI A PILA E ALBERI

Forma normale di GREIBACH (FNG)

- Una grammatica G = <Σ, M, P, S> di **tipo 2** si dice in forma normale di **Greibach** se le regole di produzione sono in questa forma: **A → σW** con **A ∈ M σ ∈ Σ W ∈ M***

ESEMPIO DI TRASFORMAZIONE IN FNG

Data la grammatica per le espressioni aritmetiche parentesizzate:

G = <Σ = {(,), +, *, x, y}, M = {E}, P = {E → (E + E) | (E * E) | x | y}, E>

Analizziamo le regole di produzione di cui si compone la grammatica e trasformiamole in regole in FNG:

- E → x   # è della forma A → σW   con σ = x, W = ε   ok per FNG
- E → y   # è della forma A → σW   con σ = y, W = ε   ok per FNG
- E → (E * E)   # non è della forma A → σW, si necessita di trasformarla:
  - Introduciamo due nuovi metasimboli (G, H) e le seguenti regole di produzione:
    - G → *   # è della forma A → σW   con σ = *, W = ε   ok per FNG
    - H →)   # è della forma A → σW   con σ =), W = ε   ok per FNG
  - Ottengo quindi E → (EGEH # è della forma A → σW   con σ = (, W = EGEH   ok per FNG
- E → (E + E)   # non è della forma A → σW, si necessita di trasformarla:
  - Introduciamo un nuovo metasimbolo (P) e la seguente regola di produzione:
    - P → +   # è della forma A → σW   con σ = +, W = ε   ok per FNG
  - Ottengo quindi E → (EPEH # è della forma A → σW   con σ = (, W = EPEH   ok per FNG

G_{FNG} = <Σ_{FNG}=Σ, M_{FNG} = {E, G, H, P}, P_{FNC} = {E → x | y | (EGEH | (EPEH, G → *, H →), P → +}, E>

Forma normale di CHOMSKY (FNC)

- Una grammatica G = <Σ, M, P, S> di tipo 2 si dice in forma normale di Chomsky se le regole di produzione sono in una di queste forme:
  - **A → BC**   con A, B, C ∈ M
  - **A → σ**   con σ ∈ Σ

ESEMPIO DI TRASFORMAZIONE IN FNC

Data la grammatica che genera il linguaggio L = $a^n b^n$:

G = <Σ = {a, b}, M = {S}, P = {S → aSb | ab}, S>

Analizziamo le regole di produzione di cui si compone la grammatica e trasformiamole in regole in FNC:

- S → ab   # non è né della forma A → σ né della forma A → BC, si necessita di trasformarla:
  - Introduciamo due nuovi metasimboli (A, B) e le seguenti regole di produzione:
    - A → a   # è della forma A → σ   con σ = a   ok per FNC
    - B → b   # è della forma A → σ   con σ = b   ok per FNC
  - Ottengo quindi S → AB # è della forma A → BC   con B = A e C = B   ok per FNC
- S → aSb   # non è né della forma A → σ né della forma A → BC, si necessita di trasformarla:
  - Utilizziamo i metasimboli A e B con rispettive regole di produzione,
    ottengo quindi S → ASB # non è né della forma A → σ né A → BC, si necessita di ridurla:

$$S \longrightarrow \underbrace{A \quad S}_{C} \quad B$$

  - Introduciamo un nuovo metasimbolo (C) e la seguente regola di produzione:
    - C → AS   # è della forma A → BC con B = A e C = S   ok per FNC
  - Ottengo quindi S → CB # è della forma A → BC   con B = C e C = B   ok per FNC

G_{FNC} = <Σ_{FNC}=Σ, M_{FNC} = {S, A, B, C}, P_{FNC} = {S → AB | CB, A → a, B → b, C → AS}, S>

Abbiamo trasformato G in G_{FNC} dove quest'ultima risulta essere in forma normale di Chomsky.

- Considero G in fnc, allora ogni parola generata in G ha associato un **albero di derivazione binario**. è composto da:
  - **RAMO** = sequenza di metasimboli in cui ogni metasimbolo è figlio del precedente. Inizia con l'assioma e termina con un simbolo terminale
  - **ALTEZZA** = lunghezza del ramo più lungo
    - Se l'albero ha forma a catena ⇒ altezza = numero di foglie
    - Se l'albero ha forma bilanciata ⇒ altezza = $\log_2$ (numero di foglie)
-