Linguaggi Formali e Traduttori

Linguaggi regolari: automi finiti, espressioni regolari

Linguaggi liberi: grammatiche libere dal contesto, automi push-down

Proprietà dei linguaggi regolari e dei linguaggi liberi

Automa finito

- Un FA è una quintupla

$$A = (Q, \Sigma, \delta, q_0, F)$$

  - Q è un insieme finito di *stati*
  - Σ è un alfabeto (*simboli in input*)
  - δ è una *funzione di transizione*
  - $q_0 \in Q$ è lo stato *iniziale*
  - $F \subseteq Q$ è l'insieme degli stati *finali*

DFA	**NFA**	**ε-NFA**
$\delta: Q \times \Sigma \rightarrow Q$	$\delta: Q \times \Sigma \rightarrow 2^Q$	$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$

Funzione di transizione estesa alle stringhe

DFA:

$$\hat{\delta}(q,\varepsilon) = q$$

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

NFA:

$$\hat{\delta}(q,\varepsilon) = \{q\}$$

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

ε-NFA:

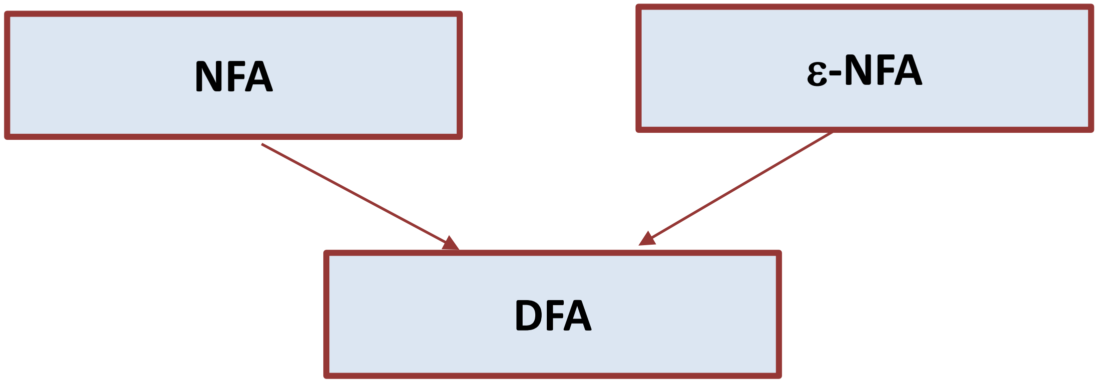
$$\hat{\delta}(q,\varepsilon) = \text{ECLOSE}(q)$$

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

Linguaggio <u>riconosciuto</u> (o <u>accettato</u>)

DFA	NFA	ε-NFA
$\{w \mid \hat{\delta}(q_0,w) \in F\}$	$\{w \mid \hat{\delta}(q_0,w) \cap F \neq \Phi\}$	$\{w \mid \hat{\delta}(q_0,w) \cap F \neq \Phi\}$

Automi finiti e Linguaggi regolari



- I linguaggi riconosciuti (accettati) da automi a stati finiti sono chiamati *linguaggi regolari.*

- Due automi sono *equivalenti* se accettano lo stesso linguaggio.

Espressioni regolari

Definizione

Base (espressioni elementari):

- Φ, ε sono espressioni regolari.
  $L(\Phi) = \{\ \}$ e $L(\varepsilon) = \{\varepsilon\}$.
- Se $a \in \Sigma$, allora *a* è un'espressione regolare.
  $L(a) = \{a\}$.

Induzione:

- Se R è un'espressione regolare, allora (R) è un'espressione regolare. $L((R)) = L(R)$.
- Se R e S sono espressioni regolari, allora R + S è un'espressione regolare. $L(R + S) = L(R) \cup L(S)$.
- Se R e S sono espressioni regolari, allora R.S (oppure RS) è un'espressione regolare. $L(R.S) = L(R).L(S)$.
- Se R è un'espressione regolare, allora R* è un'espressione regolare. $L(R^*) = (L(R))^*$.

Espressioni regolari

Precedenza degli operatori

1. Chiusura (*)
2. Concatenazione (.)
3. Unione (+)

Le espressioni regolari soddisfano le leggi algebriche che valgono per gli insiemi.

Gli automi finiti sono i riconoscitori dei linguaggi denotati dalle espressioni regolari:

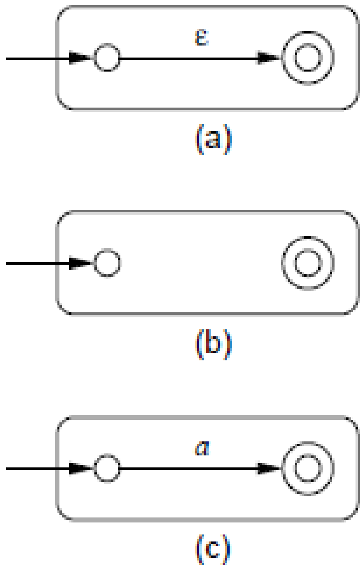
1. Per ogni automa finito A si può costruire un'espressione regolare *R* tale che L(*R*) = L(A)
2. Per ogni espressione regolare *R* esiste un automa finito A tale che L(A) = L(*R*)

Espressioni regolari ed automi finiti

Da espressione regolare ad automa

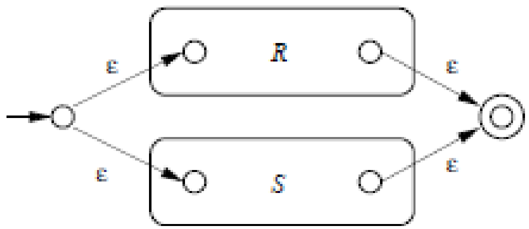
Costruzione per induzione strutturale.

Base: Automa per ε, Φ e *a*.

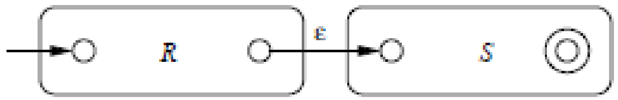


Da espressione regolare ad automa

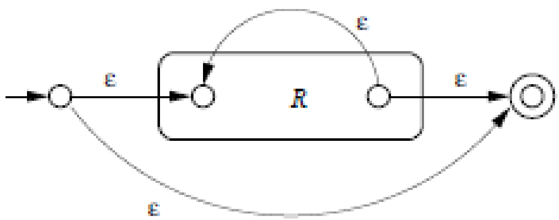
Induzione: Automa per *R* + *S*, *RS* e *R**



(a)



(b)



(c)

Un linguaggio regolare su un alfabeto Σ è un linguaggio che può essere espresso mediante concatenazione, unione e chiusura di Kleene a partire dai simboli di Σ.

Automa minimo e stati indistinguibili

Possiamo costruire una sequenza di partizioni degli stati:

$$\Pi_0 = (Q\text{-}F, F), \ \Pi_1, \Pi_2, \ldots, \Pi_i, \ldots$$

tali che i gruppi della partizione Π_i contengano gli stati non distinguibili da stringhe di lunghezza $\leq$ i.

Quando fermarsi?

È stato dimostrato che se $\Pi_i = \Pi_{i+1}$, la partizione degli stati non cambia più, cioè $\Pi_i = \Pi_{i+1} = \Pi_{i+2} = \ldots = \Pi_{i+k} = \ldots$
Allora Π_i (o Π_{i+1}) è la partizione finale Π_{final}, che contiene i gruppi degli stati indistinguibili.

Quanti passi possono servire?

Si può dimostrare che, se l'automa ha n stati, al massimo

$$\Pi_{final} = \Pi_{n-2}$$

Algoritmo di minimizzazione

1. L'algoritmo parte dalla partizione $\Pi = \Pi_0$ che distingue gli stati finali da quelli non finali e costruisce progressivamente partizioni i cui gruppi sono insiemi di stati non ancora identificati come distinguibili.
2. Due stati appartenenti ad insiemi diversi di una partizione sono invece già stati identificati come distinguibili.
3. Quando la partizione non può più essere modificata spezzando un gruppo in gruppi più piccoli, l'algoritmo costruisce il DFA minimo prendendo come stati i gruppi della partizione stessa (Π_{final}).
4. Il procedimento fondamentale consiste nel considerare un generico gruppo $\{s_1, s_2, \ldots, s_n\}$ e verificare per ogni coppia di stati s_i e s_j, se almeno un simbolo dell'alfabeto di input *a* li "distingue" nel senso che le transizioni da s_i e s_j con il simbolo *a* portano in stati che sono in gruppi diversi della partizione precedente.

Grammatiche libere dal contesto

Una grammatica *G libera dal contesto* è una quadrupla <V, T, P, S> tale che:

- V è un insieme finito di simboli (*non terminali* o *variabili*)
- T è un insieme finito di simboli (*terminali*)
- P è un insieme di *regole di riscrittura* o *produzioni sintattiche*
- $S \in V$ è l'*assioma* o *start symbol* della grammatica

le regole sono coppie: $<A, \alpha>$, che scriviamo come $A \rightarrow \alpha$ ($A \in V$, $\alpha \in (V \cup T)^*$)

1. In una grammatica G la stringa β *produce* o *si riscrive come* la stringa γ: $\beta \Rightarrow \gamma$ se $\beta, \gamma \in (V \cup T)^*$ e $\beta = \delta A \mu$ e $\gamma = \delta \alpha \mu$ e $A \rightarrow \alpha \in P$
2. $\beta \Rightarrow^n \gamma$ $\beta \Rightarrow^* \gamma$ $\beta \Rightarrow^+ \gamma$
3. *Derivazione a sinistra* (*leftmost*) e *Derivazione a destra* (*rightmost*)
4. Il *linguaggio generato da una grammatica* (L(G)) è l'insieme delle *stringhe di terminali* prodotte dall'assioma:

$$L(G) = \{x \mid x \in T^* \ \& \ S \Rightarrow^+ x\}$$

Alberi sintattici

Alberi sintattici: rappresentazioni delle derivazioni in G.

1) Ogni nodo interno è etichettato da una variabile;
2) Ogni foglia è etichettata da una variabile, da un terminale o da ε;
3) Se una foglia è etichettata ε, allora è l'unico figlio del nodo padre;
4) Se un nodo interno è etichettato A e i suoi figli sono etichettati, da sinistra verso destra, $X_1, X_2, \ldots, X_k$, allora $A \rightarrow X_1, X_2, \ldots, X_k$ è una produzione della grammatica.

Per ogni albero sintattico si ha una e una sola derivazione destra e una e una sola derivazione sinistra.

Data una grammatica $G = \langle V, T, P, S \rangle$ i seguenti enunciati sono equivalenti:

1) La stringa w è nel linguaggio della variabile A ($w \in L_A$);
2) $A \Rightarrow^* w$;
3) $A \underset{rm}{\Rightarrow}^* w$;
4) $A \underset{lm}{\Rightarrow}^* w$;
5) Esiste un albero sintattico con radice A e *prodotto* w.

Grammatiche ambigue

- Una frase è *ambigua* se ha almeno due alberi sintattici distinti.

Poiché ad ogni albero sintattico è associata un'unica derivazione a destra (e un'unica derivazione a sinistra), si può dire che una frase è ambigua se ha più di una derivazione a destra (o a sinistra).

- Una grammatica è *ambigua* se almeno una frase del linguaggio generato è ambigua.

Più di un albero → più di un significato

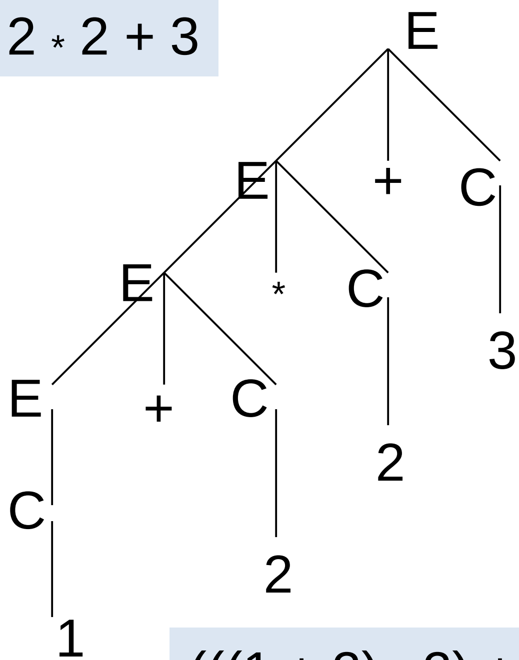
Un linguaggio è *inerentemente ambiguo* se tutte le grammatiche che lo generano sono ambigue.

Adeguatezza strutturale

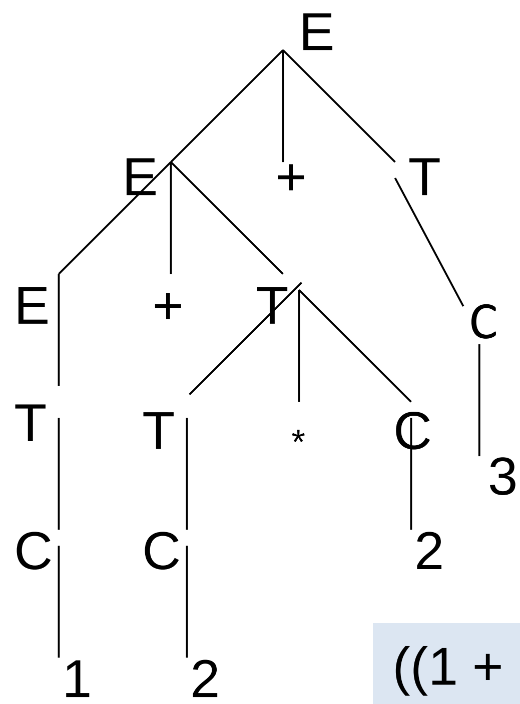
G_1 $\quad E \rightarrow E + C \mid E * C \mid C$
$\quad\quad C \rightarrow 0 \mid 1 \mid \ldots \mid 9$

G_2 $\quad E \rightarrow E + T \mid T$
$\quad\quad T \rightarrow T * C \mid C$
$\quad\quad C \rightarrow 0 \mid 1 \mid \ldots \mid 9$

$1 + 2 * 2 + 3$



$(((1 + 2) * 2) + 3)$



$((1 + (2 * 2)) + 3)$

Le due grammatiche sono non ambigue, ma solo la seconda è strutturalmente adeguata.

Automi a pila

Un *automa a pila* o *automa push-down* M è una 7-tupla

$$M = < Q, \Sigma, \Gamma, \delta, q_0, Z_0, F >$$

dove:

Q è l'*insieme degli stati dell'unità di controllo* (finito e non vuoto)

Σ è l'*alfabeto di ingresso*

Γ è l'*alfabeto della pila*

δ è la *funzione di transizione* $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$

$q_0 \in Q$ è lo *stato iniziale*

$Z_0 \in \Gamma$ è il *simbolo iniziale della pila*

$F \subseteq Q$ è l'*insieme degli stati finali*

$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_k, \gamma_k)\}$ $\qquad a \in \Sigma \cup \{\varepsilon\}$

Due tipi di transizioni: *mossa con lettura* e *mossa spontanea*.

Automi a pila: configurazione istantanea

Configurazione istantanea: (q, y, η)

q è lo stato del controllo
y è la parte della stringa ancora da esaminare
η è il contenuto della pila

Configurazione iniziale: (q_0, w, Z_0)

Transizioni da una configurazione ad un'altra:

$(q, ay, Z\eta) \vdash (p, y, \gamma\eta)$ se $(p, \gamma) \in \delta(q, a, Z)$

$(q, ay, Z\eta) \vdash (p, ay, \gamma\eta)$ se $(p, \gamma) \in \delta(q, \varepsilon, Z)$

Il *linguaggio riconosciuto* dall'automa $M = < Q, \Sigma, \Gamma, \delta, q_0, Z_0, \Phi >$ *per stack vuoto* è l'insieme delle stringhe w il cui esame, a partire dalla configurazione iniziale (q_0, w, Z_0), può portare l'automa M in una configurazione in cui la memoria è vuota:

$$N(M)= \{ w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$$

Relazione tra grammatiche libere e automi a pila

<u>Un linguaggio è generato da una CFG se e solo se è accettato da un PDA per pila vuota</u>.

Grammatica context-free $\Longleftrightarrow$ Automa a pila

($\Rightarrow$) **IDEA**: usare lo stack per simulare una derivazione leftmost in G

1. Definire lo start symbol come simbolo iniziale dello stack
2. Per ogni variabile A definire:

$$\delta(q, \varepsilon, A) = \{(q, \alpha) \mid A \to \alpha \text{ è una produzione di G}\}$$

N.B. $(q, \varepsilon) \in \delta(q, \varepsilon, A)$ se $A \to \varepsilon$ è una produzione

3. Per ogni simbolo terminale *a*:

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

Automi a pila deterministici

Automa a pila deterministico **M**:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

definisce per ogni terna: stato, simbolo in input, simbolo sullo stack, al massimo una transizione e non offre mai la scelta tra una mossa spontanea e una mossa con lettura, cioè:

- $| \delta(q, a, Z) | \leq 1$
- $| \delta(q, \varepsilon, Z) | \leq 1$
- se per una coppia (q, Z) è definita una mossa spontanea, non è definita nessuna mossa con lettura.

Proprietà dei linguaggi

Regolari		Context-free
<u>*Proprietà Pumping*</u>.		<u>*Proprietà Pumping*</u>.
<u>*Proprietà di chiusura*</u>.		<u>*Proprietà di chiusura*</u>.
$R_1 \cup R_2 \in$ Reg	Unione	$L_1 \cup L_2 \in$ C-F
$R_1.R_2 \in$ Reg	Concatenazione	$L_1.L_2 \in$ C-F
$R^* \in$ Reg	Chiusura di Kleene	$L^* \in$ C-F
$R^R \in$ Reg	Inversione	$L^R \in$ C-F
$\Sigma^*- R \in$ Reg	Complemento	$\Sigma^*- L \notin$ C-F
$R_1 - R_2 \in$ Reg	Differenza	$L_1 - L_2 \notin$ C-F
$R_1 \cap R_2 \in$ Reg	Intersezione	$L_1 \cap L_2 \notin$ C-F
		$L \cap R \in$ C-F

Relazione tra linguaggi liberi e linguaggi regolari

Una grammatica context-free è *unilineare destra* (*sinistra*) se le sue regole hanno la forma:

$A \rightarrow wB$	$w \in T^*$ e $B \in V \cup \{\varepsilon\}$	*unilineare destra*
$A \rightarrow Bw$	$w \in T^*$ e $B \in V \cup \{\varepsilon\}$	*unilineare sinistra*

Nota: In una grammatica unilineare non si possono mescolare produzioni del primo e del secondo tipo.

Ogni grammatica unilineare destra è *equivalente* a una grammatica unilineare sinistra, e viceversa.

Le grammatiche unilineari destre (sinistre) generano i linguaggi regolari.

$\Downarrow$

Un linguaggio regolare è anche libero dal contesto.

Limiti delle grammatiche context-free

Le grammatiche libere dal contesto:

1. non sono in grado di generare stringhe con *tre o più gruppi di caratteri ripetuti un numero di volte in relazione tra loro.*

   Esempio: $L = \{0^n1^n2^n \mid n > 0\}$.

2. non sanno generare *coppie con lo stesso numero di simboli, se le coppie sono "intrecciate".*

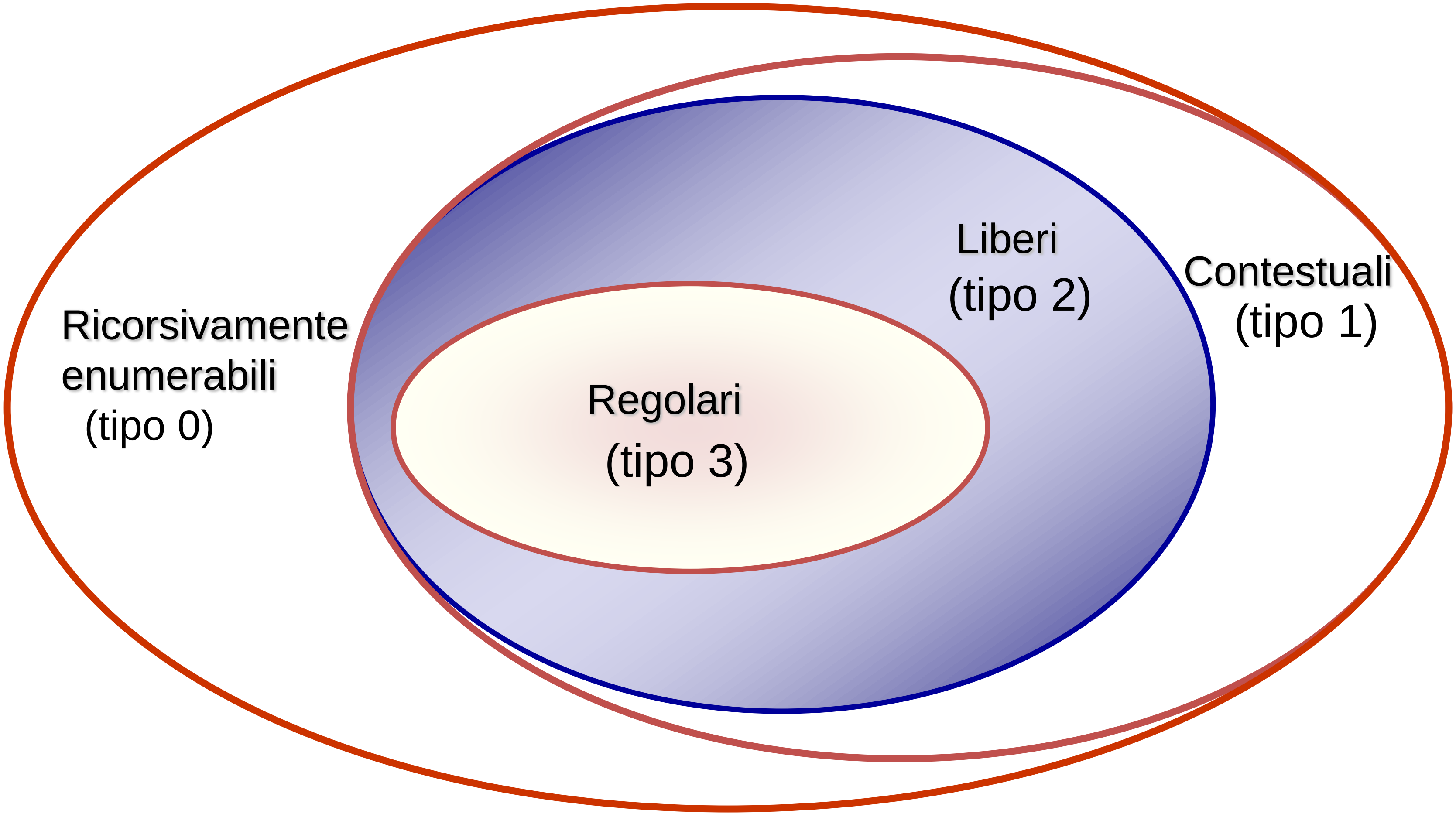
   Esempio: $L = \{0^i1^j2^i3^j \mid i, j > 0\}$.

3. non sanno *ripetere una stringa di lunghezza arbitraria*, se la stringa è su un alfabeto di cardinalità maggiore di 1.

   Esempio: $L = \{ww \mid w \in \{0,1\}^*\}$.

Classificazione di Chomsky

Chomsky ha introdotto una classificazione “storica” dei linguaggi, distinguendoli in tipo 3 (regolari), tipo 2 (liberi dal contesto), tipo 1 (contestuali) e tipo 0 (ricorsivamente enumerabili).



Il processo di compilazione

Analisi lessicale

L'*analizzatore lessicale*

- riceve in input la descrizione dei *lessemi*, sottostringhe che soddisfano un pattern, cioè un linguaggio, di solito fornita da espressioni regolari;
- legge la stringa di caratteri e riconosce i lessemi per mezzo di una automa (un algoritmo che lo implementa);
- per ogni lessema crea una coppia, chiamata *token*, che contiene le informazioni necessarie per le fasi successive.

Lo studio dei linguaggi regolari risponde alla domanda:

Come descrivere i lessemi e
come riconoscerli ?

Il processo di compilazione

Analisi sintattica

L'*analizzatore sintattico* o *parsificatore*

- riceve la descrizione del linguaggio fornita da una grammatica G;
- legge la stringa di token e se appartiene al linguaggio L(G) ne produce una derivazione o un albero sintattico, altrimenti si ferma segnalando l'errore (o prosegue ignorando le sottostringhe contaminate dall'errore).

Lo studio dei linguaggi context-free risponde alle domande:

Come descrivere il linguaggio di programmazione e come verificare che il programma in esame sia corretto almeno sintatticamente ?

Quali caratteristiche sono auspicabili per una grammatica e quali quelle da evitare ?

Generazione di codice intermedio

I linguaggi context-free sono utilizzati anche nella fase di generazione del codice intermedio con tecniche di traduzione guidate dalle grammatiche libere.