# Grammatiche Context-Free e Forma Normale di Chomsky

## Tutorato 5: CFG, Forma Normale di Chomsky

Espressioni Regolari, Linguaggi Context-free e loro proprietà

**Gabriel Rovesti**

Corso di Laurea in Informatica - Università degli Studi di Padova

Anno Accademico 2024-2025

## Contents

# 1 Espressioni Regolari e Equivalenze con Automi

## 1.1 Definizione e Sintassi delle Espressioni Regolari

Le espressioni regolari sono un formalismo dichiarativo per descrivere linguaggi regolari, equivalenti in potere espressivo agli automi a stati finiti (DFA e NFA).

**Concetto chiave**

Le espressioni regolari sono costruite utilizzando tre operatori base (unione, concatenazione e chiusura di Kleene) a partire da costanti elementari. Sono un modo conciso per specificare linguaggi regolari.

Le espressioni regolari sono costruite utilizzando:

- **Costanti di base**:
  - $\varepsilon$ per la stringa vuota
  - $\emptyset$ per il linguaggio vuoto
  - $a, b, \ldots$ per i simboli $a, b, \ldots \in \Sigma$
- **Operatori**:
  - $+$ per l'unione
  - $\cdot$ per la concatenazione
  - $*$ per la chiusura di Kleene
- **Parentesi** per il raggruppamento: ( )

**Suggerimento**

Le regole di precedenza per le espressioni regolari sono:

1. La chiusura di Kleene ($*$) ha la precedenza più alta
2. La concatenazione ($\cdot$) ha precedenza intermedia
3. L'unione ($+$) ha la precedenza più bassa

Usa le parentesi quando hai dubbi sulla precedenza degli operatori.

## 1.2 Semantica

Se $E$ è un'espressione regolare, allora $\mathcal{L}(E)$ è il linguaggio rappresentato da $E$. La definizione è induttiva:

**Caso Base**:

- $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(a) = \{a\}$ per $a \in \Sigma$

**Caso Induttivo**:

- $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$
- $\mathcal{L}(E \cdot F) = \mathcal{L}(E) \cdot \mathcal{L}(F)$
- $\mathcal{L}(E^*) = \mathcal{L}(E)^*$
- $\mathcal{L}((E)) = \mathcal{L}(E)$

**Errore comune**

Un errore comune è confondere l'espressione $01^* + 10^*$ (che rappresenta stringhe che iniziano con 0 seguite da un numero arbitrario di 1, o stringhe che iniziano con 1 seguite da un numero arbitrario di 0) con $(01)^* + (10)^*$ (che rappresenta stringhe formate da ripetizioni di 01 o ripetizioni di 10).

## 1.3 Conversione da Espressioni Regolari a $\varepsilon$-NFA

**Procedimento di risoluzione**

Per convertire un'espressione regolare $R$ in un $\varepsilon$-NFA $A$ tale che $\mathcal{L}(A) = \mathcal{L}(R)$:

1. **Casi base**:
   - Per $\varepsilon$: un singolo stato iniziale e finale
   - Per $\emptyset$: un automa che non accetta alcuna stringa
   - Per un simbolo $a \in \Sigma$: due stati collegati da una transizione etichettata $a$
2. **Casi induttivi**:
   - Per $R_1 + R_2$: costruisci un nuovo stato iniziale con $\varepsilon$-transizioni verso gli stati iniziali degli automi per $R_1$ e $R_2$
   - Per $R_1 \cdot R_2$: collega gli stati finali dell'automa per $R_1$ agli stati iniziali dell'automa per $R_2$ con $\varepsilon$-transizioni
   - Per $R^*$: aggiungi un nuovo stato iniziale/finale e $\varepsilon$-transizioni appropriate per implementare il ciclo

## 1.4 Da NFA a Espressioni Regolari

La conversione da NFA a espressioni regolari è più complessa e si basa sul metodo di eliminazione degli stati.

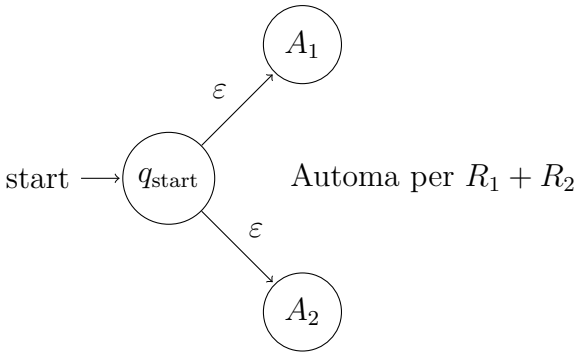


Figure 1: Schema di costruzione dell'automa per l'unione

**Procedimento di risoluzione**

Per convertire un NFA in un'espressione regolare equivalente:

1. **Trasformazione iniziale**: Converti l'NFA in un GNFA (Automa a Stati Finiti Non-deterministico Generalizzato) in forma speciale:
   - Aggiungi un nuovo stato iniziale $q_{\text{start}}$ che ha transizioni verso tutti gli altri stati, ma nessuna transizione entrante
   - Aggiungi un nuovo stato finale $q_{\text{accept}}$ che ha transizioni da tutti gli altri stati, ma nessuna transizione uscente
   - Assicurati che ci sia una transizione (possibilmente etichettata con $\emptyset$) per ogni coppia di stati
2. **Eliminazione iterativa degli stati**: Elimina uno ad uno tutti gli stati diversi da $q_{\text{start}}$ e $q_{\text{accept}}$:
   - Per ogni stato $q_{\text{rip}}$ da eliminare, aggiorna le etichette delle transizioni dirette tra gli altri stati
   - Se abbiamo transizioni $q_i \xrightarrow{R_1} q_{\text{rip}}$, $q_{\text{rip}} \xrightarrow{R_2} q_{\text{rip}}$ (ciclo), e $q_{\text{rip}} \xrightarrow{R_3} q_j$
   - Più una transizione esistente $q_i \xrightarrow{R_4} q_j$
   - Allora la nuova etichetta diventa: $R_1(R_2)^*R_3 + R_4$
3. **Risultato finale**: Quando rimangono solo $q_{\text{start}}$ e $q_{\text{accept}}$, l'etichetta della transizione da $q_{\text{start}}$ a $q_{\text{accept}}$ è l'espressione regolare equivalente all'NFA originale.

# 2 Linguaggi Context-free

## 2.1 Introduzione ai Linguaggi Context-free

Esistono linguaggi che non possono essere rappresentati da espressioni regolari o riconosciuti da automi a stati finiti. Un esempio classico è il linguaggio $\{0^n1^n | n \geq 0\}$. Per esprimere tali linguaggi, abbiamo bisogno di formalismi più potenti.

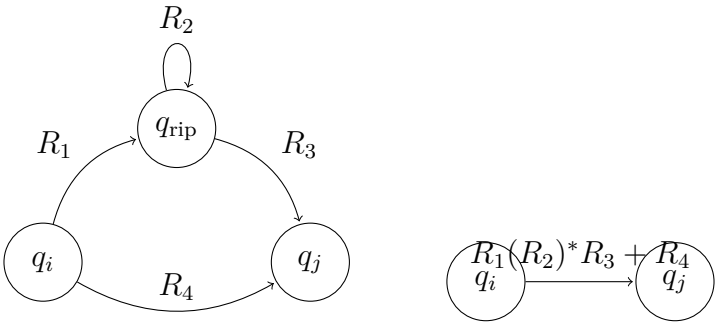


Figure 2: Schema di eliminazione di uno stato

**Concetto chiave**

I Linguaggi Context-Free (CFL) costituiscono una classe più ampia rispetto ai linguaggi regolari. Sono stati utilizzati nello studio dei linguaggi naturali dal 1950 e nello studio dei compilatori dal 1960.

Per descrivere i linguaggi context-free, utilizziamo due formalismi principali:

- Grammatiche context-free (CFG)
- Automi a pila (pushdown automata)

## 2.2 Grammatiche Context-Free

**Concetto chiave**

Una grammatica context-free è definita formalmente come una quadrupla $G = (V, \Sigma, R, S)$, dove:

- $V$ è un insieme finito di variabili (o non-terminali)
- $\Sigma$ è un insieme finito di simboli terminali, disgiunto da $V$
- $R$ è un insieme di regole di produzione, ciascuna della forma $A \to \alpha$ dove $A \in V$ e $\alpha \in (V \cup \Sigma)^*$
- $S \in V$ è la variabile iniziale

La grammatica $G_1$ definita come:

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

genera il linguaggio $\{0^n\#1^n | n \geq 0\}$.

## 2.3 Derivazioni e Alberi Sintattici

Una grammatica genera stringhe attraverso derivazioni successive a partire dalla variabile iniziale.

**Procedimento di risoluzione**

Per generare una stringa utilizzando una grammatica context-free:

1. Si parte dalla variabile iniziale
2. Si sostituisce iterativamente una variabile con la parte destra di una regola che la contiene come parte sinistra
3. Si continua finché non rimangono solo simboli terminali

Per esempio, la derivazione $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000\#111$ nella grammatica $G_1$ genera la stringa 000#111.

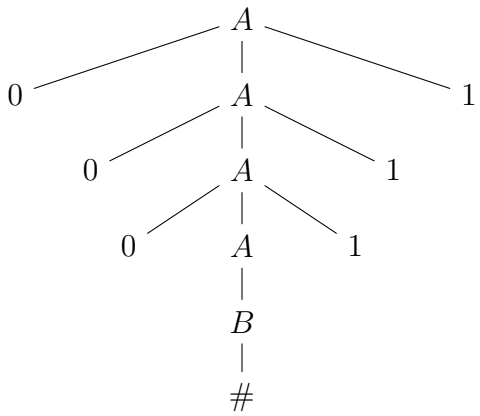


Figure 3: Albero sintattico per 000#111 nella grammatica $G_1$

Un **albero sintattico** (o parse tree) è una rappresentazione grafica di una derivazione dove:

- La radice è la variabile iniziale
- I nodi interni sono variabili
- Le foglie sono terminali o $\varepsilon$

# 3 Proprietà delle Grammatiche Context-free

## 3.1 Grammatiche Ambigue

**Concetto chiave**

Una grammatica è ambigua se esiste almeno una stringa nel linguaggio che può essere generata da più di un albero sintattico distinto.

Ad esempio, la grammatica:

$$\langle EXPR \rangle \to \langle EXPR \rangle + \langle EXPR \rangle | \langle EXPR \rangle \times \langle EXPR \rangle | (\langle EXPR \rangle) | a$$

è ambigua, perché la stringa $a + a \times a$ può essere interpretata in due modi diversi: come $(a + a) \times a$ o come $a + (a \times a)$.

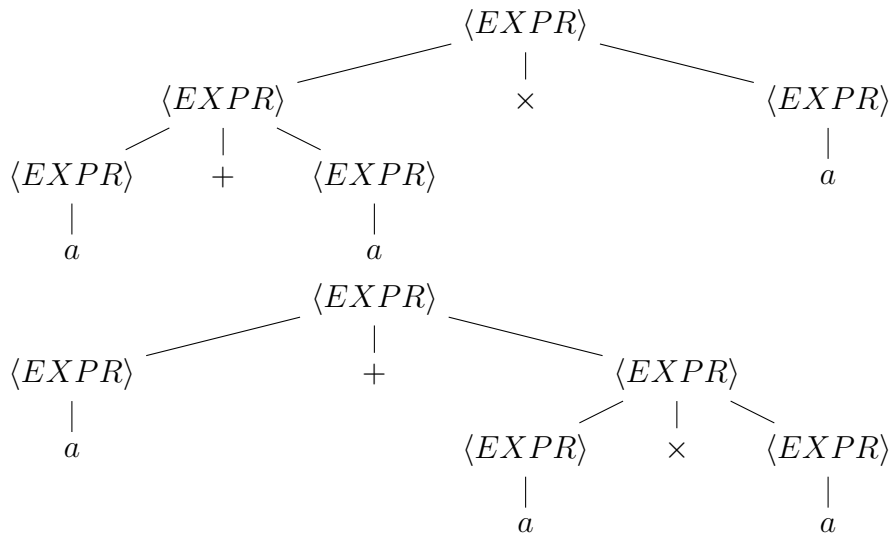


Figure 4: Due diversi alberi sintattici per la stessa espressione $a + a \times a$

Per risolvere questa ambiguità, si possono introdurre livelli di priorità utilizzando diverse variabili. Ad esempio:

$$\begin{aligned} \langle EXPR \rangle &\to \langle EXPR \rangle + \langle TERM \rangle | \langle TERM \rangle \\ \langle TERM \rangle &\to \langle TERM \rangle \times \langle FACTOR \rangle | \langle FACTOR \rangle \\ \langle FACTOR \rangle &\to (\langle EXPR \rangle) | a \end{aligned}$$

## 3.2 Linguaggi Inerentemente Ambigui

**Concetto chiave**

Esistono linguaggi context-free che sono inerentemente ambigui, cioè ogni grammatica che li genera è necessariamente ambigua.

Un esempio classico è il linguaggio $\{a^i b^j c^k | i = j \text{ oppure } j = k\}$.

## 3.3 Forma Normale di Chomsky

**Concetto chiave**

Una grammatica context-free è in Forma Normale di Chomsky (CNF) se ogni regola è della forma:

- $A \to BC$ dove $B$ e $C$ sono variabili non iniziali, oppure
- $A \to a$ dove $a$ è un terminale

Inoltre, può esistere la regola $S \to \varepsilon$ per la variabile iniziale $S$.

Ogni linguaggio context-free può essere generato da una grammatica in Forma Normale di Chomsky.

L'algoritmo per trasformare una grammatica in Forma Normale di Chomsky prevede i seguenti passi:

1. Aggiungere una nuova variabile iniziale
2. Eliminare le $\varepsilon$-regole
3. Eliminare le regole unitarie
4. Trasformare le regole restanti nella forma corretta

**Procedimento di risoluzione**

Passi per convertire una grammatica in Forma Normale di Chomsky:

1. Aggiungere una nuova variabile iniziale $S_0$ e la regola $S_0 \to S$.
2. Eliminare le $\varepsilon$-regole:
   - Per ogni regola $A \to \varepsilon$, identificare tutte le regole che contengono $A$ nel lato destro.
   - Per ciascuna di queste regole, aggiungere nuove regole che considerano tutte le possibili combinazioni di presenza/assenza di $A$.
3. Eliminare le regole unitarie:
   - Per ogni regola $A \to B$, sostituirla con l'insieme di regole $A \to \alpha$ per ogni regola della forma $B \to \alpha$.
4. Trasformare le regole restanti:
   - Per regole della forma $A \to X_1 X_2 \dots X_n$ con $n > 2$, introdurre nuove variabili per spezzare la regola.
   - Per regole con terminali e variabili mescolati, introdurre nuove variabili per i terminali.

# 4 Considerazioni Finali

**Concetto chiave**

I linguaggi regolari e i linguaggi context-free formano due livelli fondamentali nella gerarchia di Chomsky. Mentre i linguaggi regolari sono più limitati ma più facili da implementare, i linguaggi context-free offrono maggiore espressività necessaria per modellare strutture più complesse come le espressioni aritmetiche e le costruzioni sintattiche dei linguaggi di programmazione.

La teoria dei linguaggi formali fornisce strumenti essenziali per:

- Analizzare la sintassi dei linguaggi di programmazione
- Implementare parser e compilatori
- Sviluppare strumenti per l'elaborazione del linguaggio naturale

- Comprendere i limiti teorici di ciò che può essere computato

**Suggerimento**

Per approfondire questi argomenti, si consiglia la lettura dei seguenti testi:

- Hopcroft, Motwani, Ullman: "Introduction to Automata Theory, Languages, and Computation"
- Sipser: "Introduction to the Theory of Computation"