
🎯 TEMPLATE UNIVERSALI

Schema Dimostrazione Standard

```
Teorema: [Enunciato]
Dimostrazione: [Costruttiva/Per Assurdo/Induttiva]
[Corpo dimostrazione]
Abbiamo dimostrato che [conclusione]. ∎
```

Schema Costruzione Automa

```
Costruiamo A = (Q, Σ, δ, q₀, F) dove:
• Q = {stati con significato}
• Σ = [alfabeto dato]
• δ: [tabella o descrizione formale]
• q₀ = [stato iniziale]
• F = {stati finali}
```

Schema Costruzione Grammatica

```
Costruiamo G = (V, Σ, R, S) dove:
• V = {variabili con significato}
• Σ = [alfabeto terminali]
• R = {regole produzione}
• S = [variabile iniziale]
```

PARTE I: LINGUAGGI REGOLARI

🔧 TIPO 1: COSTRUZIONE AUTOMI

1.1 DFA per Proprietà Semplici

Pattern: "Stringhe che [proprietà]"

Algoritmo:

1. **Identifica invariante:** Cosa devo "ricordare"?
2. **Design stati:** Ogni stato = una "situazione" diversa
3. **Transizioni:** Come cambia la situazione con ogni simbolo?
4. **Stati finali:** Quali situazioni soddisfano la proprietà?

Esempio Template:

```
L = {w ∈ {0,1}* | w contiene almeno due 1 consecutivi}

Q = {q₀, q₁, q₂} dove:
• q₀: non ho ancora visto 1, o ho visto 1 non consecutivi
• q₁: ho appena visto un 1
• q₂: ho visto 11 (stato finale)

δ:  q₀ →⁰ q₀, q₀ →¹ q₁
    q₁ →⁰ q₀, q₁ →¹ q₂
    q₂ →⁰,¹ q₂

F = {q₂}
```

1.2 NFA per Proprietà Complesse

Quando usare: "Esistenza" di sottostringhe, unioni complesse

Template:

```
L = {w | w contiene 010 come sottostringa}

Strategia NFA: "Indovina quando inizia 010"
• q₀: stato iniziale, loop su tutto
• q₁: ho appena letto 0 (inizio possibile di 010)
• q₂: ho letto 01
• q₃: ho letto 010 (finale)

δ: q₀ →⁰'¹ q₀, q₀ →⁰ q₁
   q₁ →¹ q₂
   q₂ →⁰ q₃
   q₃ →⁰'¹ q₃
```

🔄 TIPO 2: CONVERSIONI REGOLARI

2.1 NFA → DFA (Costruzione per Sottoinsiemi)

Algoritmo Meccanico:

1. **Q' = 2^Q** (tutti sottoinsiemi di Q)
2. **q_0' = ε-closure({q_0})**
3. **δ'(S, a) = ε-closure(⋃_{q∈S} δ(q,a))**
4. **F' = {S ∈ Q' | S ∩ F ≠ ∅}**

2.2 Regex → NFA (Thompson)

Regole Composizionali:

- **a**: `○ →ᵃ ○`
- **AB**: `NFA(A) →ᵋ NFA(B)`
- **A|B**: `○ →ᵋ NFA(A) →ᵋ ○` e `○ →ᵋ NFA(B) →ᵋ ○`
- **A***: `○ →ᵋ NFA(A) →ᵋ ○` con loop

🚫 TIPO 3: PUMPING LEMMA REGOLARI

3.1 Template Standard

```
Teorema: L non è regolare
Dimostrazione: Supponiamo per assurdo che L sia regolare.

• Sia p la lunghezza data dal Pumping Lemma
• Consideriamo la parola w = [scegli w ∈ L con |w| ≥ p]
• Sia w = xyz una suddivisione tale che y ≠ ε e |xy| ≤ p
• [Analizza dove può cadere y]
• [Scegli i appropriato e dimostra xy^i z ∉ L]

Abbiamo trovato un assurdo quindi L non può essere regolare. ∎
```

3.2 Strategie di Scelta w

Pattern Comuni:

- **Dipendenza lineare:** w = a^p b^p → y contiene solo a
- **Dipendenza quadratica:** w = a^(p^2) → |xy^i z| = p^2 + (i-1)|y| ≠ quadrato perfetto
- **Comparazione parti:** w = a^p # b^p → y non può "vedere" entrambe le parti

⚙️ TIPO 4: PROPRIETÀ DI CHIUSURA REGOLARI

Regolari chiusi sotto: ∪, ∩, ‾, ·, *, omorfeismo, quoziente

Template Operazioni Custom:

```
Per operazione f(L):
1. Analizza f: Come trasforma le stringhe?
2. Costruisci automa che "traccia" la trasformazione
3. Stati = informazione per decidere accettazione
```

PARTE II: LINGUAGGI CONTEXT-FREE

📝 TIPO 5: GRAMMATICHE CONTEXT-FREE

5.1 Costruzione CFG

Pattern Base:

```
L = {a^n b^n | n ≥ 0}

G: S → aSb | ε
```

Pattern Annidati:

```
L = {palindromi pari}

G: S → aSa | bSb | ε
```

Pattern Lineari:

```
L = {w | #a(w) = #b(w)}

G: S → aSbS | bSaS | ε
```

5.2 Derivazioni e Alberi

Derivazione Leftmost: Sostituisci sempre la variabile più a sinistra **Derivazione Rightmost:** Sostituisci sempre la variabile più a destra

Template Derivazione:

```
S ⇒ [passo 1] ⇒ [passo 2] ⇒ ... ⇒ stringa finale
```

5.3 Ambiguità

Per dimostrare ambiguità:

```
Trova stringa w ∈ L(G) con due alberi di parsing diversi
Mostra le due derivazioni leftmost diverse per w
```

Per eliminare ambiguità:

```
Ristruttura grammatica con precedenze/associatività
Aggiungi variabili intermedie per forzare parsing unico
```

🏗️ TIPO 6: FORME NORMALI

6.1 Forma Normale di Chomsky (CNF)

Regole ammesse: $A \to BC$, $A \to a$, $S \to \varepsilon$ (solo se $\varepsilon \in L$)

Algoritmo di Conversione:

```
1. Elimina ε-produzioni:
   - Trova variabili nullable
   - Per ogni regola, aggiungi versioni senza nullable

2. Elimina unit productions (A → B):
   - Per ogni A →* B → α, aggiungi A → α

3. Elimina simboli inutili:
   - Elimina non-generanti e non-accessibili

4. Converti in CNF:
   - Sostituisci terminali: A → aB diventa A → XB, X → a
   - Spezza regole lunghe: A → BCD diventa A → BY, Y → CD
```

6.2 Forma Normale di Greibach (GNF)

Regole ammesse: $A \to a\alpha$ (terminale seguito da variabili)

Quando serve: Costruzione PDA da CFG

🥞 TIPO 7: AUTOMI A PILA (PDA)

7.1 Definizione Formale

```
PDA = (Q, Σ, Γ, δ, q₀, Z₀, F) dove:
• Q: stati finiti
• Σ: alfabeto input
• Γ: alfabeto pila
• δ: Q × (Σ ∪ {ε}) × Γ → Insieme finito di (Q × Γ*)
• q₀: stato iniziale
• Z₀: simbolo iniziale pila
• F: stati finali
```

7.2 Costruzione PDA da CFG

Algoritmo Standard (CFG in GNF → PDA):

```
Dato G = (V, Σ, R, S) in GNF:

PDA P = ({q}, Σ, V ∪ Σ, δ, q, S, ∅)

δ definita da:
• Per A → aα in R: δ(q, a, A) ∋ (q, α)
• Per terminali: δ(q, a, a) ∋ (q, ε)
```

7.3 Costruzione CFG da PDA

Algoritmo Inverso:

```
Variabili: [q, A, r] per ogni q,r ∈ Q, A ∈ Γ
Significato: dalla configurazione (q, A) raggiungo r con pila vuota

Regole: Per δ(q, a, A) ∋ (p, B₁B₂...Bₖ):
[q, A, rₖ] → a[p, B₁, r₁][r₁, B₂, r₂]...[rₖ₋₁, Bₖ, rₖ]
per ogni scelta di r₁, r₂, ..., rₖ
```

7.4 Pattern di Costruzione PDA

Linguaggio Matching:

```
L = {a^n b^n | n ≥ 0}

δ(q₀, a, Z₀) = {(q₀, aZ₀)}  // push a
δ(q₀, a, a) = {(q₀, aa)}     // push a
δ(q₀, b, a) = {(q₁, ε)}     // pop a, cambia stato
```

```
δ(q₁, b, a) = {(q₁, ε)}     // pop a
δ(q₁, ε, Z₀) = {(q₂, Z₀)}   // accetta
```

Linguaggio Palindromi:

```
L = {ww^R | w ∈ {a,b}*}

Strategia: Push prima metà, pop e confronta seconda metà
Non deterministico: "indovina" il centro
```

🚫 TIPO 8: PUMPING LEMMA CONTEXT-FREE

8.1 Template Context-Free

```
Teorema: L non è context-free
Dimostrazione: Supponiamo per assurdo che L sia context-free.

• Sia p la lunghezza data dal Pumping Lemma per CFL
• Consideriamo w = [scegli w ∈ L con |w| ≥ p]
• Sia w = uvxyz decomposizione con:
  - |vxy| ≤ p
  - |vy| ≥ 1
• [Analizza dove cadono v,y rispetto a struttura di w]
• [Scegli i appropriato e dimostra uv^i xy^i z ∉ L]

Abbiamo trovato un assurdo quindi L non può essere context-free. ∎
```

8.2 Strategie Specifiche CFL

Tripla Dipendenza:

```
L = {a^n b^n c^n | n ≥ 0}

w = a^p b^p c^p
|vxy| ≤ p ⟹ vxy tocca al più 2 delle 3 sezioni
Pompando si rompe bilanciamento su terza sezione
```

Copia Esatta:

```
L = {ww | w ∈ {a,b}*}

w = (ab)^p (ab)^p
```

```
vxy non può "vedere" entrambe le copie
Pompando si distruggono le copie identiche
```

⚙️ TIPO 9: PROPRIETÀ DI CHIUSURA CFL

9.1 Chiusure Positive

CFL chiusi sotto: ∪, ·, *, omorfeismo

Template Unione:

```
Date CFG G₁, G₂ per L₁, L₂:
G = G₁ ∪ G₂ ∪ {S → S₁ | S₂}
```

Template Concatenazione:

```
G = G₁ ∪ G₂ ∪ {S → S₁S₂}
```

9.2 Chiusure Negative

CFL NON chiusi sotto: ∩, ‾

Controesepio Standard:

```
L₁ = {a^n b^n c^m | n,m ≥ 0} (CFL)
L₂ = {a^m b^n c^n | m,n ≥ 0} (CFL)
L₁ ∩ L₂ = {a^n b^n c^n | n ≥ 0} (NON CFL)
```

📊 TIPO 10: PROBLEMI DECISIONALI CFL

10.1 Decidibili

Appartenenza: $w \in L(G)$? → **CYK Algorithm**

```
Algoritmo CYK (G in CNF, w = a₁...aₙ):
1. Tabella T[i,j] = {A | A ⇒* aᵢ...aᵢ₊ⱼ₋₁}
2. Base: T[i,1] = {A | A → aᵢ}
3. Ricorsione: T[i,k] = {A | A → BC, B ∈ T[i,j], C ∈ T[i+j,k-j]}
4. Accetta iff S ∈ T[1,n]
```

Vuotezza: $L(G) = \emptyset$? → Test accessibilità variabile iniziale

10.2 Indecidibili

Equivalenza: $L(G_1) = L(G_2)$? **Complemento:** $L(G) = \Sigma^*$? **Intersezione:** $L(G_1) \cap L(G_2) = \emptyset$?

🏛️ TIPO 11: GERARCHIA DI CHOMSKY

11.1 Inclusioni Proprie

```
REGOLARI ⊊ CONTEXT-FREE ⊊ CONTEXT-SENSITIVE ⊊ RICORSIVAMENTE ENUMERABILI

Linguaggi Separatori:
• REG vs CFL: {a^n b^n | n ≥ 0}
• CFL vs CSL: {a^n b^n c^n | n ≥ 0}
• CSL vs RE: Halting Problem variants
```

11.2 Caratterizzazioni Alternative

```
REGOLARI = DFA = NFA = Regex = Grammatiche Lineari
CONTEXT-FREE = CFG = PDA
CONTEXT-SENSITIVE = LBA (Linear Bounded Automata)
RIC. ENUMERABILI = TM = Grammatiche Unrestricted
```

PARTE III: PARSING

🔍 TIPO 12: ALGORITMI DI PARSING

12.1 Top-Down Parsing

LL(1) Conditions:

```
Per ogni A → α | β:
1. FIRST(α) ∩ FIRST(β) = ∅
2. Se ε ∈ FIRST(α), allora FIRST(β) ∩ FOLLOW(A) = ∅
```

Costruzione Tabella LL(1):

```
Per regola A → α:
• Per ogni a ∈ FIRST(α): M[A,a] = A → α
• Se ε ∈ FIRST(α), per ogni b ∈ FOLLOW(A): M[A,b] = A → α
```

12.2 Bottom-Up Parsing

LR(0) Items: [A → α·β] **SLR(1):** Usa FOLLOW per risolvere conflitti **LR(1):** Usa lookahead per maggiore precisione

🧮 CALCOLI FIRST/FOLLOW

12.3 Algoritmi FIRST/FOLLOW

FIRST(α):

```
Se α = ε: FIRST(α) = {ε}
Se α = aβ: FIRST(α) = {a}
Se α = Aβ:
  - Se ε ∉ FIRST(A): FIRST(α) = FIRST(A)
  - Se ε ∈ FIRST(A): FIRST(α) = FIRST(A) ∪ FIRST(β)
```

FOLLOW(A):

```
1. $ ∈ FOLLOW(S) (simbolo fine)
2. Per B → αAβ: FIRST(β)\{ε} ⊆ FOLLOW(A)
3. Per B → αA o B → αAβ con ε ∈ FIRST(β): FOLLOW(B) ⊆ FOLLOW(A)
```

⚡ SHORTCUTS PER ESAME

Riconoscimento Pattern Veloce

Linguaggi Regolari:

```
"contiene substring": NFA con indovinamento
"proprietà locali": DFA con stati = "memoria recente"
"counting mod k": k stati
"prefissi/suffissi": costruzione lineare
```

Linguaggi Context-Free:

```
"matching parentesi": stack-based, CFL
"palindromi": CFL (tranne finite exceptions)
"a^n b^n": classico CFL
"equal counts": CFL ma attenzione dipendenze multiple
```

NON Context-Free:

```
"a^n b^n c^n": tripla dipendenza
"ww": copia esatta
"quadratic growth": non CFL
"a^(n²)": crescita quadratica
```

Error Prevention Checklist

Automi:

- [ ] Stati finali chiari e motivati
- [ ] Transizioni complete o deterministico specificato
- [ ] Semantica stati esplicita

Grammatiche:

- [ ] Tutte le produzioni necessarie
- [ ] Casi base (often ε-productions)
- [ ] Variabili con ruoli chiari

Pumping Lemma:

- [ ] Vincoli $|xy| \leq p$ (regolari) o $|vxy| \leq p$ (CFL)
- [ ] Scelta i motivata
- [ ] Verifica appartenenza originale $w \in L$

Costruzioni:

- [ ] Correttezza almeno accennata
- [ ] Casi edge (ε, stringhe vuote)
- [ ] Determinismo vs non-determinismo chiaro

🎯 TIME MANAGEMENT ESAME

```
• Costruzione DFA/NFA: 5-8 min
• Pumping Lemma: 8-12 min
• CFG da linguaggio: 6-10 min
• PDA construction: 10-15 min
• Conversioni (NFA→DFA): 8-12 min
• CYK/parsing: 8-15 min
• Proprietà chiusura: 5-8 min
```

🚨 RED FLAGS COMUNI

```
✗ Dimenticare ε-transizioni in NFA
✗ Confondere accettazione PDA (stati finali vs pila vuota)
✗ Pumping lemma: scegliere w troppo semplice
✗ CFG: dimenticare casi base
✗ Equivalenza vs appartenenza nei problemi decisionali
✗ Mixing up FIRST/FOLLOW calculations
✗ Confondere LL(1) vs LR(1) conditions
```

🎭 FILOSOFIA FINALE

Remember:

- **Costruisci sempre esplicitamente** - l'intuizione non basta
- **Ogni affermazione deve essere giustificata**
- **Le costruzioni sono più importanti delle definizioni**
- **In dubbio, formalizza tutto**

La precisione batte la velocità. La correttezza batte l'eleganza. La dimostrazione batte l'intuizione. 🎯