Guida pratica per gli esercizi - Approccio Bresolin

🎯 QUANDO SERVE DIMOSTRARE CONTEXT-FREE

Un linguaggio è **context-free** se può essere riconosciuto da:

- **Grammatica Context-Free (CFG)**
- **Automa a Pila (PDA)**

TEOREMA: L è context-free ⟺ ∃ CFG che genera L ⟺ ∃ PDA che riconosce L

📚 METODO 1: COSTRUZIONE CFG

🔧 STRATEGIA GENERALE

```
OBIETTIVO: Costruire CFG G tale che L(G) = L

PASSI:
1. Identifica la struttura ricorsiva del linguaggio
2. Progetta produzioni che catturano questa struttura
3. Verifica che L(G) = L (⊆ e ⊇)
```

🎯 PATTERN COMUNI PER CFG

Pattern 1: Conteggio Bilanciato

```
LINGUAGGIO: L = {a^n b^n | n ≥ 0}

CFG:
S → aSb | ε

SPIEGAZIONE:
• S genera stringhe con ugual numero di a e b
• Ricorsione centrale mantiene bilanciamento
```

Pattern 2: Annidamento

```
LINGUAGGIO: L = {a^n b^m c^n | n,m ≥ 0}

CFG:
S → aSc | T
T → bT | ε

SPIEGAZIONE:
• S genera a^n...c^n bilanciati
• T genera b^m nel mezzo
```

Pattern 3: Palindromi

```
LINGUAGGIO: L = {w ∈ {a,b}* | w = w^R}

CFG:
S → aSa | bSb | a | b | ε

SPIEGAZIONE:
• Costruisce simmetricamente da fuori verso dentro
• Casi base per lunghezze 0,1
```

Pattern 4: Unione di Linguaggi

```
LINGUAGGIO: L = L₁ ∪ L₂

CFG:
S → S₁ | S₂
[produzioni per L₁ con simbolo iniziale S₁]
[produzioni per L₂ con simbolo iniziale S₂]
```

Pattern 5: Concatenazione

```
LINGUAGGIO: L = L₁ · L₂

CFG:
S → S₁S₂
[produzioni per L₁ con simbolo iniziale S₁]
[produzioni per L₂ con simbolo iniziale S₂]
```

📚 METODO 2: COSTRUZIONE PDA

🔧 STRATEGIA GENERALE

```
OBIETTIVO: Costruire PDA P tale che L(P) = L

COMPONENTI PDA:
• Stati di controllo (finiti)
• Stack (memoria infinita LIFO)
• Transizioni: (stato, input, top_stack) → (nuovo_stato, push_stack)
```

🎯 TECNICHE COMUNI PER PDA

Tecnica 1: Push & Match

```
USO: Linguaggi con struttura a e b bilanciate

STRATEGIA:
1. Push simboli del primo tipo sullo stack
2. Pop e match simboli del secondo tipo
3. Accetta se stack vuoto alla fine

ESEMPIO L = {a^n b^n}:
δ(q₀, a, Z₀) = (q₀, aZ₀)  // push prima a
δ(q₀, a, a) = (q₀, aa)    // push altre a
δ(q₀, b, a) = (q₁, ε)     // inizia matching
δ(q₁, b, a) = (q₁, ε)     // continua matching
δ(q₁, ε, Z₀) = (q₂, Z₀)   // accetta se stack vuoto
```

Tecnica 2: Centro-Fuori (Palindromi)

```
USO: Palindromi e strutture simmetriche

STRATEGIA:
1. Push simboli nella prima metà
2. Guess del centro (transizione ε)
3. Pop e match simboli nella seconda metà

ESEMPIO L = palindromi:
δ(q₀, a, Z₀) = (q₀, aZ₀)  // push simboli
```

```
δ(q₀, b, X) = (q₀, bX) per ogni X
δ(q₀, ε, X) = (q₁, X)     // guess centro
δ(q₁, a, a) = (q₁, ε)     // match
δ(q₁, b, b) = (q₁, ε)     // match
```

Tecnica 3: Contatore su Stack

```
USO: Linguaggi che richiedono conteggio

STRATEGIA:
1. Usa stack come contatore
2. Push per incrementare
3. Pop per decrementare

ESEMPIO L = {a^i b^j c^k | i = j+k}:
// Push per ogni a
// Pop per ogni b
// Pop per ogni c
// Accetta se bilanciato
```

🧩 ESEMPI COMPLETI TIPO D'ESAME

🔹 ESEMPIO 1: {a^n b^m c^n | n,m ≥ 0}

SOLUZIONE CFG:

```
GRAMMATICA:
S → aSc | T
T → bT | ε

CORRETTEZZA:
• S ⇒* a^n T c^n per qualche n ≥ 0
• T ⇒* b^m per qualche m ≥ 0
• Quindi S ⇒* a^n b^m c^n
```

SOLUZIONE PDA:

```
STATI: {q₀, q₁, q₂, q₃}
TRANSIZIONI:
δ(q₀, a, Z₀) = (q₀, aZ₀)    // push a
```

```
δ(q₀, a, a) = (q₀, aa)      // push più a
δ(q₀, b, X) = (q₁, X)       // inizia b (per ogni X)
δ(q₁, b, X) = (q₁, X)       // continua b
δ(q₁, c, a) = (q₂, ε)       // inizia c, pop a
δ(q₂, c, a) = (q₂, ε)       // continua c, pop a
δ(q₂, ε, Z₀) = (q₃, Z₀)     // accetta
```

🔹 ESEMPIO 2: {ww^R | w ∈ {a,b}*}

SOLUZIONE CFG:

```
GRAMMATICA:
S → aSa | bSb | ε

CORRETTEZZA:
• Costruisce palindromi simmetricamente
• Ogni derivazione produce ww^R per qualche w
```

SOLUZIONE PDA:

```
STRATEGIA: Push prima metà, pop e match seconda metà
STATI: {q₀, q₁, q₂}

δ(q₀, a, Z₀) = (q₀, aZ₀)    // push simboli
δ(q₀, b, X) = (q₀, bX)      // push simboli
δ(q₀, ε, X) = (q₁, X)       // guess fine prima metà
δ(q₁, a, a) = (q₁, ε)       // match
δ(q₁, b, b) = (q₁, ε)       // match
δ(q₁, ε, Z₀) = (q₂, Z₀)     // accetta
```

🔹 ESEMPIO 3: {a^i b^j | i ≠ j}

SOLUZIONE CFG:

```
GRAMMATICA (unione di due casi):
S → S₁ | S₂                  // i > j OR i < j
S₁ → aS₁b | aS₁ | a          // i > j
S₂ → aS₂b | bS₂ | b          // i < j

SPIEGAZIONE:
• S₁ genera stringhe con più a che b
```

```
• S₂ genera stringhe con più b che a
• Unione copre tutti i casi i ≠ j
```

🎯 STRATEGIA DI RICONOSCIMENTO

🔧 QUANDO USARE CFG vs PDA

Usa CFG quando:

- Struttura ricorsiva è chiara
- Linguaggio ha pattern "nested" semplici
- Serve dimostrazione rapida
- Esercizio chiede esplicitamente CFG

Usa PDA quando:

- Serve memoria di tipo stack
- Linguaggio richiede "matching" di simboli
- Struttura più complessa con stati
- Esercizio chiede esplicitamente PDA

🔧 COME PROGETTARE CFG

Step 1: Identifica Struttura

```
DOMANDE CHIAVE:
• Ci sono simboli da bilanciare? (a^n b^n pattern)
• È una unione di linguaggi più semplici?
• C'è struttura ricorsiva evidente?
• Serve concatenazione di parti?
```

Step 2: Progetta Produzioni

```
PRINCIPI:
• Una produzione per ogni "caso base"
• Una produzione per ogni "passo ricorsivo"
• Simboli non-terminali per ogni "componente"
• Mantieni bilanciamenti/relazioni
```

Step 3: Verifica Correttezza

```
CONTROLLI:
• Ogni stringa in L può essere derivata?
• Ogni stringa derivabile è in L?
• Le produzioni catturano tutti i casi?
• Non ci sono derivazioni "spurie"?
```

⚠️ ERRORI COMUNI DA EVITARE

❌ CFG mal progettate

```
ERRORE: S → aS | bS | ε per L = {a^n b^n}
PROBLEMA: Genera tutte le stringhe, non solo quelle bilanciate
CORREZIONE: S → aSb | ε
```

❌ PDA con stack management scorretto

```
ERRORE: Non gestire il simbolo di bottom Z₀
PROBLEMA: Stack underflow o non-terminazione
CORREZIONE: Sempre controllare Z₀ per accettazione
```

❌ Non considerare string vuota

```
ERRORE: Dimenticare ε nei linguaggi
PROBLEMA: CFG/PDA non accetta stringa vuota quando dovrebbe
CORREZIONE: Aggiungere produzioni/transizioni per ε
```

📝 TEMPLATE VELOCE PER ESAMI

🔧 Template CFG

```
TEOREMA: L è context-free

DIMOSTRAZIONE: Costruiamo CFG G con L(G) = L

GRAMMATICA:
[Produzioni]
```

```
CORRETTEZZA:
• L(G) ⊆ L: [ogni derivazione produce stringa in L]
• L ⊆ L(G): [ogni stringa in L può essere derivata]

Quindi L è context-free □
```

🔧 Template PDA

```
TEOREMA: L è context-free

DIMOSTRAZIONE: Costruiamo PDA P con L(P) = L

PDA: P = (Q, Σ, Γ, δ, q₀, Z₀, F)
dove:
Q = [stati]
Σ = [alfabeto input]
Γ = [alfabeto stack]
δ = [transizioni]
[stati iniziale e finali]

CORRETTEZZA: [spiegazione strategia]

Quindi L è context-free □
```

✅ CHECKLIST FINALE

Per ogni CFG verifica:

- [ ] Produzioni catturano struttura ricorsiva
- [ ] Ogni stringa in L può essere derivata
- [ ] Ogni derivazione produce stringa in L
- [ ] Gestione corretta di stringa vuota
- [ ] Simboli non-terminali ben motivati

Per ogni PDA verifica:

- [ ] Stack usato correttamente per memoria
- [ ] Transizioni ε per "guess" quando necessario
- [ ] Gestione Z_0 per riconoscere fine input

- [ ] Stati finali raggiungibili
- [ ] Accettazione per stack vuoto o stato finale

Linguaggi tipici d'esame che SONO context-free:

- [ ] {a^n b^n | n ≥ 0}
- [ ] {a^n b^m c^n | n,m ≥ 0}
- [ ] {ww^R | w ∈ Σ*} (palindromi)
- [ ] {a^i b^j | i ≠ j}
- [ ] Linguaggi di parentesi bilanciate

Pattern da ricordare:

- [ ] Bilanciamento → S → aSb | ε
- [ ] Palindromi → S → aSa | bSb | a | b | ε
- [ ] Unione → S → S_1 | S_2
- [ ] Concatenazione → S → S_1S_2
- [ ] Stack come contatore nel PDA