Guida pratica per gli esercizi - Approccio Bresolin

🎯 QUANDO SERVE DIMOSTRARE REGOLARITÀ

Un linguaggio è **regolare** se può essere riconosciuto da:

- **Automa a Stati Finiti (DFA/NFA)**
- **Espressione Regolare (RE)**

TEOREMA: L è regolare ⟺ ∃ DFA che riconosce L ⟺ ∃ NFA che riconosce L ⟺ ∃ RE che descrive L

📚 METODO 1: COSTRUZIONE DFA/NFA

🔧 STRATEGIA GENERALE

```
OBIETTIVO: Costruire automa A tale che L(A) = L

COMPONENTI:
• Stati (memoria finita del "passato")
• Transizioni (come reagire a simboli)
• Stati finali (quando accettare)
```

🎯 PRINCIPI DI PROGETTAZIONE

Principio 1: Stati = Informazione Essenziale

```
DOMANDA: Che informazione del passato serve per decidere il futuro?

ESEMPI:
• L = stringhe che finiscono con "ab"
  → Stati: "che suffisso ho letto finora?"

• L = stringhe con numero pari di a
  → Stati: "ho letto pari o dispari a?"
```

Principio 2: Transizioni = Aggiornamento Informazione

```
REGOLA: δ(stato_attuale, simbolo) = nuovo_stato
dove nuovo_stato riflette informazione aggiornata
```

Principio 3: Stati Finali = Condizione Soddisfatta

```
REGOLA: Stato è finale ⟺ l'informazione memorizzata
        indica che la stringa fin qui soddisfa L
```

🧩 PATTERN COMUNI DI COSTRUZIONE

🔹 Pattern 1: Suffissi/Prefissi

_ESEMPIO: L = {w ∈ {a,b} | w finisce con "ab"}_*

```
STATI:
• q₀: non ho letto nulla di rilevante (o ho rotto il pattern)
• q₁: ho appena letto 'a' (possibile inizio di "ab")
• q₂: ho letto "ab" (FINALE)

DFA:
δ(q₀, a) = q₁    // vedo 'a', potenziale inizio
δ(q₀, b) = q₀    // vedo 'b', non rilevante
δ(q₁, a) = q₁    // altro 'a', nuovo potenziale inizio
δ(q₁, b) = q₂    // completo "ab"!
δ(q₂, a) = q₁    // nuovo 'a', nuovo potenziale
δ(q₂, b) = q₀    // 'b' dopo "ab", ricomincio

STATI FINALI: {q₂}
```

🔹 Pattern 2: Conteggio Modulo k

_ESEMPIO: L = {w ∈ {a,b} | #a (w) ≡ 0 (mod 3)}_*

```
STATI:
• q₀: #a ≡ 0 (mod 3) (FINALE)
• q₁: #a ≡ 1 (mod 3)
• q₂: #a ≡ 2 (mod 3)

DFA:
```

```
δ(q₀, a) = q₁    // 0+1 ≡ 1 (mod 3)
δ(q₁, a) = q₂    // 1+1 ≡ 2 (mod 3)
δ(q₂, a) = q₀    // 2+1 ≡ 0 (mod 3)
δ(qᵢ, b) = qᵢ    // 'b' non cambia il conteggio

STATI FINALI: {q₀}
```

🔹 Pattern 3: Combinazioni Logiche

_ESEMPIO: L = {w ∈ {a,b} | w contiene "aa" O numero pari di b}_*

```
APPROCCIO: Usa prodotto cartesiano di automi
• A₁ riconosce stringhe con "aa"
• A₂ riconosce stringhe con #b pari
• L = L(A₁) ∪ L(A₂)

COSTRUZIONE:
• Stati = Stati_A₁ × Stati_A₂
• (q₁,q₂) finale ⟺ q₁ finale in A₁ OR q₂ finale in A₂
```

🔹 Pattern 4: Proprietà Multiple

_ESEMPIO: L = {w ∈ {a,b} | w inizia e finisce con lo stesso simbolo}_*

```
STRATEGIA: Traccia primo simbolo + ultimo simbolo visto

STATI:
• q₀: stringa vuota (FINALE)
• qₐ: inizia con 'a', ultimo visto 'a' (FINALE)
• qₐb: inizia con 'a', ultimo visto 'b'
• qb: inizia con 'b', ultimo visto 'b' (FINALE)
• qbₐ: inizia con 'b', ultimo visto 'a'

TRANSIZIONI: Aggiorna "ultimo visto"
```

📚 METODO 2: ESPRESSIONI REGOLARI

🔧 COSTRUZIONE SISTEMATICA

Operatori Base:

```
• ∅ = linguaggio vuoto
• ε = stringa vuota
• a = simbolo singolo
• R₁ + R₂ = unione (OR)
• R₁ · R₂ = concatenazione
• R* = stella di Kleene (0 o più ripetizioni)
```

Pattern Comuni:

```
• Σ* = tutte le stringhe
• a* = zero o più 'a'
• a⁺ = una o più 'a' = aa*
• (ab)* = zero o più ripetizioni di "ab"
• a*b* = zero o più 'a' seguite da zero o più 'b'
```

🎯 ESEMPI DI COSTRUZIONE RE

ESEMPIO 1: L = {w | w contiene almeno una 'a'}

```
STRATEGIA: Σ* - {stringhe senza 'a'}
RISPOSTA: Σ*aΣ* = (a+b)*a(a+b)*
```

ESEMPIO 2: L = {w | w ha lunghezza pari}

```
STRATEGIA: Ogni carattere in posizione pari + ogni in posizione dispari
RISPOSTA: ((a+b)(a+b))* = ((a+b)²)*
```

ESEMPIO 3: L = {w | w inizia con 'a' e finisce con 'b'}

```
STRATEGIA: 'a' + qualsiasi cosa + 'b'
RISPOSTA: a(a+b)*b + ab (caso speciale lunghezza 2)
SEMPLIFICATO: a(a+b)*b
```

🧩 ESEMPI COMPLETI TIPO D'ESAME

🔹 ESEMPIO 1: Stringhe senza "aa"

```
LINGUAGGIO: L = {w ∈ {a,b}* | w non contiene "aa"}

SOLUZIONE DFA:
STATI:
• q₀: OK, ultimo simbolo non era 'a' (FINALE)
• q₁: OK, ultimo simbolo era 'a' (FINALE)
• q₂: ERRORE, ho visto "aa" (NON FINALE)

TRANSIZIONI:
δ(q₀, a) = q₁    // primo 'a', ancora OK
δ(q₀, b) = q₀    // 'b', rimango OK
δ(q₁, a) = q₂    // secondo 'a' consecutivo → ERRORE
δ(q₁, b) = q₀    // 'b' dopo 'a', rompo sequenza
δ(q₂, a) = q₂    // rimango in errore
δ(q₂, b) = q₂    // rimango in errore

SOLUZIONE RE:
STRATEGIA: b*(ab+)*a?
SPIEGAZIONE:
• Inizia con b*
• Poi gruppi "ab+" (almeno una a seguita da almeno una b)
• Opzionalmente una 'a' finale
```

◆ ESEMPIO 2: Numero di a multiplo di 3

```
LINGUAGGIO: L = {w ∈ {a,b}* | #a(w) ≡ 0 (mod 3)}

SOLUZIONE DFA:
STATI: q₀ (resto 0), q₁ (resto 1), q₂ (resto 2)
[Come nel pattern 2 sopra]

SOLUZIONE RE:
STRATEGIA: Costruire da DFA con eliminazione stati
RISULTATO: b*(ab*ab*ab*)*
SPIEGAZIONE:
• b* iniziali
• Poi gruppi di esattamente 3 'a' separate da b*
```

◆ ESEMPIO 3: Stessa parità di a e b

```
LINGUAGGIO: L = {w ∈ {a,b}* | #a(w) e #b(w) hanno stessa parità}
```

```
SOLUZIONE DFA:
STATI:
• q₀₀: #a pari, #b pari (FINALE)
• q₀₁: #a pari, #b dispari
• q₁₀: #a dispari, #b pari
• q₁₁: #a dispari, #b dispari (FINALE)

TRANSIZIONI:
δ(qᵢⱼ, a) = q₍₁₋ᵢ₎ⱼ    // flip parità di a
δ(qᵢⱼ, b) = qᵢ₍₁₋ⱼ₎    // flip parità di b
```

🎯 STRATEGIE DI RISOLUZIONE

🔧 QUANDO USARE DFA vs NFA vs RE

Usa DFA quando:

- Logica deterministica chiara
- Stati rappresentano info "necessaria e sufficiente"
- Esercizio chiede esplicitamente DFA

Usa NFA quando:

- Serve "guess" o scelte non-deterministiche
- Più semplice di DFA equivalente
- Linguaggio ha struttura "or" naturale

Usa RE quando:

- Pattern testuale evidente
- Composizione di linguaggi più semplici
- Esercizio chiede espressione regolare

🔧 PROCESSO DI PROGETTAZIONE

Step 1: Analizza il linguaggio

```
DOMANDE:
• Che proprietà deve avere una stringa accettata?
• Quale informazione del "passato" è rilevante?
```

```
• Ci sono pattern/substring da riconoscere/evitare?
• La decisione dipende da conteggi?
```

Step 2: Progetta stati

```
PRINCIPIO: Uno stato per ogni "situazione distintiva"
ESEMPI:
• "ho visto prefisso X"
• "sono a distanza Y dalla fine"
• "ho contato Z modulo k"
• "sono in errore (stringa già rifiutata)"
```

Step 3: Definisci transizioni

```
REGOLA: Per ogni (stato, simbolo) → nuovo_stato
VERIFICA: Il nuovo stato riflette correttamente la nuova situazione
```

Step 4: Identifica stati finali

```
CRITERIO: Stato finale ⟺ le stringhe che terminano qui sono in L
```

⚠️ ERRORI COMUNI DA EVITARE

❌ Stati insufficienti

```
ERRORE: Non distinguere situazioni che richiedono comportamenti diversi
ESEMPIO: Per "finisce con ab", non basta stato "ho visto a"
CORREZIONE: Servono stati per "niente rilevante", "visto a", "visto ab"
```

❌ Transizioni mancanti

```
ERRORE: Non definire δ(q,a) per qualche stato q e simbolo a
CORREZIONE: DFA deve avere transizione per ogni (stato, simbolo)
```

❌ Stati finali sbagliati

```
ERRORE: Stato finale quando non dovrebbe (o viceversa)
```

```
CORREZIONE: Verifica: "Se termino qui, la stringa è davvero in L?"
```

❌ RE mal costruite

```
ERRORE: (a+b)* per "contiene almeno una a"
PROBLEMA: Include anche stringhe senza a
CORREZIONE: (a+b)*a(a+b)*
```

📝 TEMPLATE VELOCE PER ESAMI

🔧 Template DFA

```
TEOREMA: L è regolare

DIMOSTRAZIONE: Costruiamo DFA A con L(A) = L

DFA: A = (Q, Σ, δ, q₀, F) dove:
Q = [stati con spiegazione del significato]
Σ = [alfabeto]
δ = [tabella transizioni o descrizione]
q₀ = [stato iniziale]
F = [stati finali con giustificazione]

CORRETTEZZA: [spiegazione strategia]

Quindi L è regolare □
```

🔧 Template RE

```
TEOREMA: L è regolare

DIMOSTRAZIONE: L è descritto dalla RE: [espressione]

SPIEGAZIONE: [come la RE cattura esattamente L]

Quindi L è regolare □
```

✅ CHECKLIST FINALE

Per ogni DFA verifica:

- [ ] Stato per ogni situazione distintiva rilevante
- [ ] Transizione definita per ogni (stato, simbolo)
- [ ] Stati finali corretti (termino qui ⟺ stringa in L)
- [ ] Stato iniziale appropriato
- [ ] Correttezza testata su esempi

Per ogni NFA verifica:

- [ ] Transizioni ε usate appropriatamente
- [ ] Non-determinismo semplifica la costruzione
- [ ] Accettazione corretta (almeno un cammino finale)

Per ogni RE verifica:

- [ ] Operatori usati correttamente (*, +, ·)
- [ ] Espressione cattura esattamente L
- [ ] Casi limite gestiti (ε, simboli singoli)
- [ ] Precedenza operatori rispettata

Linguaggi tipici d'esame che SONO regolari:

- [ ] Pattern prefissi/suffissi
- [ ] Conteggi modulo k costante
- [ ] Combinazioni booleane di proprietà regolari
- [ ] Linguaggi con memoria finita

Tecniche da ricordare:

- [ ] Stati = informazione essenziale del passato
- [ ] Prodotto cartesiano per combinazioni logiche
- [ ] Complemento: scambia finali/non-finali
- [ ] RE da pattern testuali evidenti
- [ ] Eliminazione stati per DFA → RE