Metodologia formale secondo Bresolin

🎯 OBIETTIVO GENERALE

Dimostrare che un linguaggio L è **decidibile** costruendo esplicitamente una Macchina di Turing **decisore** che lo risolve.

📚 DEFINIZIONI FONDAMENTALI

Linguaggio Decidibile

Un linguaggio $L \subseteq \Sigma^*$ è **decidibile** se esiste una TM M tale che:

- M **si ferma sempre** su ogni input $w \in \Sigma^*$
- M **accetta** w se e solo se $w \in L$
- M **rifiuta** w se e solo se $w \notin L$

Macchina Decisore

Una TM M è un **decisore** per L se:

- $\forall w \in \Sigma^*$: M(w) termina in tempo finito
- L(M) = L e M non va mai in loop

🔧 METODOLOGIA STANDARD

STEP 1: Formulazione del Problema

• **Definire formalmente** il linguaggio target • **Template**: L = {⟨oggetti⟩ | proprietà da verificare} • **Esempi**:

- ADFA = {⟨A,w⟩ | A è un DFA che accetta w}
- EQDFA = {⟨A,B⟩ | A e B sono DFA equivalenti}
- ACFG = {⟨G,w⟩ | G è una CFG che genera w}

STEP 2: Costruzione del Decisore

• **Template standard**:

```
M = "Su input ⟨parametri⟩:
1. [CONTROLLI VALIDITÀ]
   • Verifica che l'input abbia formato corretto
   • Se no, RIFIUTA

2. [ALGORITMO PRINCIPALE]
   • Applica procedura decidibile per la proprietà
   • Usa algoritmi noti per sottoproblemi

3. [DECISIONE FINALE]
   • Se proprietà verificata, ACCETTA
   • Altrimenti, RIFIUTA"
```

STEP 3: Prova di Correttezza

• **Terminazione**: Dimostra che l'algoritmo termina sempre • **Completezza**: Se w ∈ L, allora M accetta w • **Soundness**: Se M accetta w, allora w ∈ L

STEP 4: Analisi della Complessità (opzionale)

• Stimare il tempo di esecuzione dell'algoritmo • Identificare i passi più costosi

---

📋 PROBLEMI STANDARD E SOLUZIONI TIPO

🔸 PROBLEMI SU AUTOMI A STATI FINITI

Problema: ADFA

```
ADFA = {⟨A,w⟩ | A è un DFA che accetta w}

SOLUZIONE:
M = "Su input ⟨A,w⟩:
1. Controlla che A sia un DFA valido e w sia una stringa
2. Simula A su input w:
   • Mantieni lo stato corrente q = q₀
   • Per ogni simbolo aᵢ in w:
     - Aggiorna q = δ(q, aᵢ)
3. Se q ∈ F, ACCETTA; altrimenti RIFIUTA"
```

```
CORRETTEZZA:
• Terminazione: Simulazione richiede |w| passi
• Completezza: Se w ∈ L(A), simulazione raggiunge stato finale
• Soundness: Se simulazione raggiunge stato finale, allora w ∈ L(A)
```

Problema: EQDFA

```
EQDFA = {⟨A,B⟩ | A e B sono DFA equivalenti}

SOLUZIONE:
M = "Su input ⟨A,B⟩:
1. Costruisci DFA C che riconosce (L(A) \ L(B)) ∪ (L(B) \ L(A))
   • C = (A ∩ B̄) ∪ (B ∩ Ā)
2. Testa se L(C) = ∅ usando algoritmo per EDFA
3. Se L(C) = ∅, ACCETTA; altrimenti RIFIUTA"

GIUSTIFICAZIONE:
• A ≡ B ⟺ L(A) = L(B) ⟺ (L(A) \ L(B)) ∪ (L(B) \ L(A)) = ∅
```

🔸 PROBLEMI SU GRAMMATICHE CONTEXT-FREE

Problema: ACFG

```
ACFG = {⟨G,w⟩ | G è una CFG che genera w}

SOLUZIONE:
M = "Su input ⟨G,w⟩:
1. Se w = ε, controlla se S ⇒* ε usando algoritmo nullabile
2. Altrimenti, converti G in forma normale di Chomsky
3. Usa algoritmo CYK per testare se w ∈ L(G):
   • Costruisci tabella CYK di dimensione |w| × |w|
   • Riempi tabella bottom-up
4. Se CYK[1,|w|] contiene il simbolo iniziale, ACCETTA
5. Altrimenti, RIFIUTA"

COMPLESSITÀ: O(|w|³ · |G|)
```

Problema: ECFG

```
ECFG = {⟨G⟩ | G è una CFG con L(G) = ∅}

SOLUZIONE:
```

```
M = "Su input ⟨G⟩:
1. Marca tutti i simboli terminali come 'generativi'
2. Ripeti fino a convergenza:
   • Per ogni produzione A → α:
     – Se tutti i simboli in α sono marcati 'generativi'
     – Allora marca A come 'generativo'
3. Se il simbolo iniziale S è marcato 'generativo', RIFIUTA
4. Altrimenti, ACCETTA"

TERMINAZIONE: Al massimo |V| iterazioni
```

🔸 PROBLEMI SU LINGUAGGI REGOLARI

Problema: INFINITEREG

```
INFINITEREG = {⟨A⟩ | A è un DFA con L(A) infinito}

SOLUZIONE:
M = "Su input ⟨A⟩:
1. Costruisci il grafo degli stati di A
2. Trova tutti gli stati raggiungibili da q₀
3. Trova tutti gli stati da cui si può raggiungere uno stato finale
4. Controlla se esiste un ciclo in un stato che soddisfa entrambe le
condizioni
5. Se sì, ACCETTA; altrimenti RIFIUTA"

PRINCIPIO: L(A) infinito ⟺ ∃ cammino q₀ →* q →+ q →* f con f ∈ F
```

🧩 TECNICHE DI COSTRUZIONE AVANZATE

🔹 Composizione di Algoritmi

• **Principio**: Combina algoritmi decidibili per sottoproblemi • **Esempio**: Per EQDFA, usa EDFA come subroutine • **Template**:

```
1. Riduci problema a sottoproblemi decidibili
2. Risolvi ogni sottoproblema con algoritmo noto
3. Combina risultati per decisione finale
```

🔹 Costruzioni per Prodotto

• **Uso**: Per operazioni tra linguaggi (∩, ∪,) • **Tecnica**: Costruisci automa prodotto • **Esempio**: A ∩ B̄ per differenza simmetrica

🔹 Conversioni tra Formalismi

• **CFG → PDA**: Per problemi su grammatiche • **Regex → NFA → DFA**: Per problemi su espressioni regolari • **Principio**: Usa il formalismo più conveniente per l'algoritmo

🔹 Programmazione Dinamica

• **Uso**: Algoritmi CYK, riempimento tabellare • **Principio**: Risolvi sottoproblemi più piccoli prima • **Applicazioni**: Parsing, membership testing

---

⚠️ ERRORI COMUNI DA EVITARE

❌ Problemi di Terminazione

• **Errore**: Non dimostrare che l'algoritmo termina sempre • **Soluzione**: Identifica misura di progresso che decresce

❌ Confusione Decidibile/Riconoscibile

• **Errore**: Costruire riconoscitore invece di decisore • **Soluzione**: Assicurarsi che la TM si fermi sempre

❌ Algoritmi Non Costruttivi

• **Errore**: Dare solo esistenza senza costruzione esplicita • **Soluzione**: Fornire algoritmo concreto passo-passo

❌ Formato Input Scorretto

• **Errore**: Non validare formato dell'input • **Soluzione**: Sempre includere controlli di validità

📝 TEMPLATE GENERICO PER OGNI ESERCIZIO

```
PROBLEMA: [Nome del linguaggio]
DEFINIZIONE: L = {(parametri) | proprietà}

TEOREMA: L è decidibile.
```

```
DIMOSTRAZIONE:
Costruiamo un decisore M per L:

M = "Su input ⟨parametri⟩:
1. [VALIDAZIONE INPUT]
   • Controlla formato e validità parametri
   • Se invalido, RIFIUTA

2. [ALGORITMO PRINCIPALE]
   • [Passo 1]: [descrizione]
   • [Passo 2]: [descrizione]
   • ...
   • [Passo n]: [descrizione]

3. [DECISIONE]
   • Se [condizione soddisfatta], ACCETTA
   • Altrimenti, RIFIUTA"

CORRETTEZZA:
• TERMINAZIONE: [argomento per terminazione garantita]
• COMPLETEZZA: [dimostrazione ⟨parametri⟩ ∈ L ⟹ M accetta]
• SOUNDNESS: [dimostrazione M accetta ⟹ ⟨parametri⟩ ∈ L]

COMPLESSITÀ: [analisi opzionale]

CONCLUSIONE: L è decidibile. □
```

🎯 CHECKLIST FINALE

Prima di consegnare, verifica:

- [ ] Linguaggio definito formalmente
- [ ] Decisore costruito esplicitamente
- [ ] Algoritmo termina sempre (no loop infiniti)
- [ ] Controlli di validità input inclusi
- [ ] Correttezza dimostrata (completezza + soundness)
- [ ] Terminazione argomentata esplicitamente
- [ ] Utilizzo corretto di algoritmi standard noti
- [ ] Conclusione esplicita di decidibilità

Algoritmi Standard da Conoscere:

- [ ] Simulazione DFA/NFA
- [ ] Algoritmo CYK per CFG
- [ ] Test di vuotezza per DFA/CFG
- [ ] Costruzioni per operazioni su linguaggi
- [ ] Algoritmi di raggiungibilità su grafi