01-07 - Prima Parte

Esercizio 1:

Data una traslitterazione T : Σ → Γ *e un linguaggio regolare L ⊆ Σ*, dimostrare che T(L) = {w ∈ Γ* | w = T(a0)T(a1)...T(an) per qualche a0a1...an ∈ L} è regolare.

Dimostrazione:

1. Poiché L è regolare, esiste un DFA A = (Q, Σ, δ, q0, F) che lo riconosce.
2. Costruiamo un NFA N = (Q', Γ, δ', q0, F) che riconosce T(L):
   - Q' = Q ∪ {qij | q ∈ Q, i ∈ Σ, 1 ≤ j ≤ |T(i)|}
   - Lo stato iniziale e gli stati finali sono gli stessi di A
3. Per ogni transizione δ(q, a) = p in A, dove T(a) = b1b2...bk, aggiungiamo in N:
   δ'(q, b1) = {qa1}
   δ'(qaj, bj+1) = {qa(j+1)} per 1 ≤ j < k
   δ'(qak, ε) = {p}
4. Aggiungiamo ε-transizioni: δ'(q, ε) = {q} per ogni q ∈ Q
5. Correttezza:
   - Se w ∈ T(L), esiste a0a1...an ∈ L tale che w = T(a0)T(a1)...T(an)
   - A accetta a0a1...an, quindi N accetta w simulando il percorso di A
6. Completezza:
   - Se N accetta w, il percorso di accettazione corrisponde a una stringa in L
7. Conclusione: N è un NFA che riconosce T(L), quindi T(L) è regolare.

Esercizio 2:

Dimostrare che il linguaggio L2 = {x#y | x, y ∈ {0, 1}* e x ≠ y} non è regolare.

Dimostrazione per contraddizione usando il Pumping Lemma:

1. Supponiamo per assurdo che L2 sia regolare. Sia p la costante del pumping lemma.
2. Consideriamo la stringa s = 0p#1p ∈ L2
3. Per il pumping lemma, s può essere scritta come s = xyz, dove:
   - |xy| ≤ p
   - |y| > 0
   - xyiz ∈ L2 per ogni i ≥ 0
4. Dato che |xy| ≤ p, y deve essere composto solo da 0.
5. Consideriamo xy2z:

    - La parte prima del # avrà più di p zeri
    - La parte dopo il # avrà esattamente p uno
6. Ma questo significa che $xy2z \notin L2$, contraddicendo il pumping lemma
7. Conclusione: L2 non può essere regolare

Esercizio 3:

Date due stringhe w e t, diciamo che t è una permutazione di w se t ha gli stessi simboli di w con ugual numero di occorrenze, ma eventualmente in un ordine diverso.

Dimostrare che se $B \subseteq \{0, 1\}$ *è un linguaggio regolare, allora il linguaggio SCRAMBLE(B) = {t ∈ {0, 1}* | t è una permutazione di qualche w ∈ B} è un linguaggio context-free.

Dimostrazione:

1. Poiché B è regolare, esiste un DFA M che lo riconosce.
2. Costruiamo una CFG G = (V, Σ, R, S) per SCRAMBLE(B):
    - V = {S} ∪ {Aq | q è uno stato di M}
    - Σ = {0, 1}
    - S è il simbolo iniziale
3. Le regole di produzione R sono:
    - S → A0C | A1C | ε, dove A0 è lo stato iniziale di M
    - Aq → AqC | ε, per ogni stato q di M
    - C → 0C0 | 1C1 | ε
4. Per ogni transizione δ(q, a) = p in M, aggiungiamo la regola:
   Aq → aAp
5. Per ogni stato finale q di M, aggiungiamo la regola:
   Aq → ε
6. Correttezza:
    - La grammatica genera stringhe che sono permutazioni di stringhe in B
    - Le Aq generano stringhe accettate da M partendo dallo stato q
    - C genera coppie bilanciate di 0 e 1, permettendo le permutazioni
7. Completezza:
    - Ogni permutazione di una stringa in B può essere generata da G
8. Conclusione: G è una CFG che genera SCRAMBLE(B), quindi SCRAMBLE(B) è context-free.

01-07 - Seconda Parte

1. Macchina di Turing con "copia e incolla" (CPTM)

(a) Definizione formale della funzione di transizione di una CPTM:
$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S1, S2, C, P\}$
Dove:
Q = insieme degli stati
Γ = alfabeto del nastro
L, R = spostamenti a sinistra e destra
S1 = seleziona inizio della porzione da copiare
S2 = seleziona fine della porzione da copiare
C = copia la porzione selezionata
P = incolla la porzione copiata

(b) Dimostrazione che CPTM riconosce la classe dei linguaggi Turing-riconoscibili:

Possiamo simulare una TM standard con una CPTM:

1. Ignoriamo le operazioni S1, S2, C, P.
2. Usiamo solo L, R per gli spostamenti.
3. Usiamo la scrittura normale per modificare il nastro.

Per dimostrare che CPTM può essere simulata da una TM standard:

1. Usiamo un secondo nastro per memorizzare la porzione copiata.
2. Simuliamo S1, S2 marcando l'inizio e la fine sul nastro principale.
3. Per C, copiamo la porzione marcata sul secondo nastro.
4. Per P, copiamo dal secondo nastro al nastro principale.

Questo dimostra l'equivalenza computazionale tra CPTM e TM standard.

2. Linguaggi quasi-palindromi

(a) Formulazione del problema come linguaggio QPAL_TM:

QPAL_TM = {⟨M⟩ | M è una TM e L(M) contiene un linguaggio quasi-palindromo infinito}

(b) Dimostrazione dell'indecidibilità di QPAL_TM:

Riduzione da INFINITO_TM a QPAL_TM:

Dato ⟨M⟩, costruiamo una TM M' che:

1. Su input w:
    - Se w non è della forma x#x^R, rifiuta.

    - Se w = x#x^R, simula M su x.
    - Se M accetta x, M' accetta w.
2. ⟨M⟩ ∈ INFINITO_TM se e solo se ⟨M'⟩ ∈ QPAL_TM:
    - Se L(M) è infinito, L(M') contiene {x#x^R | x ∈ L(M)}, che è un linguaggio quasi-palindromo infinito.
    - Se L(M) è finito, L(M') è finito e quindi non contiene un linguaggio quasi-palindromo infinito.

Poiché INFINITO_TM è indecidibile, QPAL_TM è indecidibile.

3. Problema COMMITTEE

(a) Dimostrazione che COMMITTEE è un problema NP:

Certificato: Una selezione di docenti C.
Verifica in tempo polinomiale:

1. |C| = k
2. Per ogni dipartimento, esattamente un docente è in C.
3. Non ci sono coppie di docenti in C che si detestano.

(b) Dimostrazione che COMMITTEE è NP-hard:

Riduzione da 3SAT a COMMITTEE:

Per una formula 3SAT φ con n variabili e m clausole:

1. Crea 2n+m dipartimenti:
    - 2 per ogni variabile (x_i e not_x_i)
    - 1 per ogni clausola
2. k = n+m
3. Per ogni variabile x_i:
    - Un docente in x_i e not_x_i
    - Si detestano a vicenda
4. Per ogni clausola c_j:
    - Un docente per ogni letterale nella clausola
    - Collega questo docente al dipartimento del letterale
5. Docenti di letterali contrari si detestano

φ è soddisfacibile se e solo se esiste una buona commissione:

- Scegliere un docente per variabile corrisponde ad un'assegnazione di verità

- Scegliere un docente per clausola corrisponde a soddisfare la clausola

15-07 - Prima Parte

Certo, risolverò in dettaglio ciascuno dei tre esercizi.

1. (12 punti) Dimostrazione che SWAP(L) è regolare se L è regolare:

Dato che L è regolare, esiste un DFA A = (Q, Σ, δ, q0, F) che lo riconosce.
Costruiamo un NFA N = (Q', Σ, δ', q0', F') che riconosce SWAP(L):

Q' = Q ∪ {q0', qswap}
F' = F se ε ∈ L, altrimenti F' = F ∪ {q0'} se ε ∈ L

Definiamo δ' come segue:

1. δ'(q0', ε) = {q0} (per gestire il caso w = ε)
2. δ'(q0', a) = {q0'} per ogni a ∈ Σ (per gestire il caso |w| = 1)
3. Per ogni coppia di simboli a, b ∈ Σ e stato q ∈ Q:
   δ'(q, ba) = {p | p ∈ δ(δ(q, a), b)}
4. δ'(qswap, a) = δ(q0, a) per ogni a ∈ Σ

L'NFA N simula il DFA A, ma legge i caratteri a coppie scambiate. La transizione 3 permette di leggere ba invece di ab. Lo stato qswap gestisce il caso in cui la stringa ha lunghezza dispari, permettendo di leggere l'ultimo carattere normalmente.

Poiché N è un NFA che riconosce SWAP(L), e i linguaggi riconosciuti dagli NFA sono regolari, SWAP(L) è regolare.

2. (12 punti) Dimostrazione che L2 non è regolare:

Useremo il Pumping Lemma per dimostrare che L2 non è regolare.

Supponiamo per assurdo che L2 sia regolare. Sia p la costante del pumping lemma.
Consideriamo la stringa s = 0p1p ∈ L2 (è una permutazione di se stessa).

Per il pumping lemma, s può essere scritta come s = xyz, dove:

1. |xy| ≤ p
2. |y| > 0
3. xyiz ∈ L2 per ogni i ≥ 0

Data la struttura di s, y deve essere composta solo da 0.
Sia y = 0k con 0 < k ≤ p.

Consideriamo xy2z = 0p+k1p

Per essere in L2, questa stringa dovrebbe essere una permutazione di 0p1p, ma ha p+k zeri e p uni, quindi non è una permutazione valida.

Questo contraddice il punto 3 del pumping lemma, quindi L2 non può essere regolare.

3. (12 punti) Dimostrazione che PALINDROMIZE(B) è context-free se B è regolare:

Dato che B è regolare, esiste un DFA M = (Q, Σ, δ, q0, F) che lo riconosce.
Costruiamo una CFG G = (V, Σ, R, S) per PALINDROMIZE(B):

V = {S} ∪ {[q, p] | q, p ∈ Q}
S è il simbolo iniziale

Le regole di produzione R sono:

1. S → [q0, q] per ogni q ∈ F
2. [q, p] → a[δ(q, a), p]a per ogni q, p ∈ Q e a ∈ Σ
3. [q, q] → ε per ogni q ∈ Q

La grammatica funziona come segue:

- La regola 1 inizia la derivazione con uno stato iniziale e uno stato finale.
- La regola 2 genera simultaneamente la prima metà della stringa (seguendo le transizioni di M da q0) e la seconda metà (che deve portare a uno stato finale).
- La regola 3 termina la derivazione quando raggiungiamo lo stesso stato al centro della stringa.

Correttezza:

- Se wwR ∈ PALINDROMIZE(B), allora w ∈ B, e G può generare wwR usando le regole sopra.
- Se G genera una stringa, questa sarà della forma wwR dove w ∈ B.

Completezza:

- G può generare tutte le stringhe in PALINDROMIZE(B) e solo quelle.

15-07 - Seconda Parte

Certo, risolverò in dettaglio ciascuno dei tre esercizi presenti nell'immagine.

1. Macchine di Turing con un solo stato (JTM)

(a) Definizione formale della funzione di transizione di una JTM:

$\delta: \Gamma \to \Gamma \times \{L, R, H\}$

Dove:
Γ = alfabeto del nastro (include NAY, YEA, e simboli A, B, #)
{L, R, H} = azioni (spostamento a sinistra, a destra, halt)

La funzione δ mappa ogni simbolo del nastro a un nuovo simbolo da scrivere e un'azione da eseguire.

(b) Dimostrazione che le JTM riconoscono la classe dei linguaggi Turing-riconoscibili:

Possiamo simulare una JTM con una TM standard:

1. Usiamo il nastro della TM standard per simulare il nastro della JTM.
2. Implementiamo la logica della funzione δ della JTM negli stati della TM standard.
3. Usiamo gli stati della TM per tenere traccia della posizione corrente e simulare il comportamento della JTM.

Per simulare una TM standard con una JTM:

1. Codifichiamo gli stati della TM standard sul nastro della JTM.
2. Usiamo una codifica del nastro che permette di simulare gli spostamenti della testina.
3. Implementiamo la funzione di transizione della TM nella funzione δ della JTM.

Questa simulazione bidirezionale dimostra l'equivalenza computazionale tra JTM e TM standard, provando che le JTM possono riconoscere tutti i linguaggi Turing-riconoscibili.

2. Problema GROSS_TM

(a) Formulazione del problema come linguaggio GROSS_TM:

GROSS_TM = {⟨M⟩ | M è una TM e L(M) contiene almeno una parolaccia}

(b) Dimostrazione dell'indecidibilità di GROSS_TM:

Riduzione dal problema della fermata (HALT_TM) a GROSS_TM:

Data una TM M e un input w, costruiamo una TM M' che:

1. Simula M su w.
2. Se M si ferma su w, M' accetta una stringa predefinita che è una parolaccia.

Ora, ⟨M,w⟩ ∈ HALT_TM se e solo se ⟨M'⟩ ∈ GROSS_TM:

- Se M si ferma su w, allora L(M') contiene una parolaccia.
- Se M non si ferma su w, allora L(M') è vuoto.

Poiché HALT_TM è indecidibile, GROSS_TM è indecidibile.

3. Problema ALIBABA

(a) Dimostrazione che ALIBABA è un problema NP:

Certificato: Una selezione di oggetti S'.
Verifica in tempo polinomiale:

1. Calcolare il peso totale degli oggetti in S': $\Sigma(s \in S')\ p[s] \leq t$
2. Calcolare il valore totale degli oggetti in S': $\Sigma(s \in S')\ v[s] \geq L$

(b) Dimostrazione che ALIBABA è NP-hard:

Riduzione da SUBSET-SUM a ALIBABA:

Data un'istanza di SUBSET-SUM ⟨S,t⟩:

1. Crea un'istanza di ALIBABA con:
    - Gli stessi oggetti S
    - p[s] = s per ogni s ∈ S (il peso è il valore dell'elemento)
    - v[s] = s per ogni s ∈ S (il valore è lo stesso del peso)
    - t rimane lo stesso
    - L = t

Ora, esiste un sottoinsieme S' ⊆ S tale che $\Sigma(s \in S')\ s = t$ se e solo se esiste una soluzione per ALIBABA:

- Se esiste S' per SUBSET-SUM, allora S' è una soluzione valida per ALIBABA (soddisfa sia il vincolo di peso che di valore).
- Se esiste una soluzione per ALIBABA, questa soluzione risolve anche SUBSET-SUM.

Poiché SUBSET-SUM è NP-completo, questa riduzione dimostra che ALIBABA è NP-hard.

Essendo sia in NP che NP-hard, ALIBABA è NP-completo.