Automi e Linguaggi Formali - Esame del 15 Settembre 2021

Problema 1 (8 punti)

Considera il linguaggio L = {0^m 1^n | m > 3n}. Dimostra che L non è regolare.

Dimostrazione per contraddizione usando il Pumping Lemma

Assunzione: Supponiamo per contraddizione che L sia regolare.

Applicazione del Pumping Lemma: Allora esiste una costante p > 0 (pumping length) tale che ogni stringa w ∈ L con |w| ≥ p può essere decomposta come w = xyz con:

1. |xy| ≤ p
2. |y| > 0
3. xy^i z ∈ L per ogni i ≥ 0

Scelta della stringa di test: Consideriamo w = 0^(3p+1) 1^p ∈ L.

Verifichiamo che w ∈ L: dobbiamo avere m > 3n, cioè 3p+1 > 3p ✓

Inoltre, |w| = 4p+1 ≥ p, quindi il pumping lemma si applica.

Analisi della decomposizione: Poiché |xy| ≤ p e w inizia con 3p+1 occorrenze di 0, la substring xy deve essere contenuta interamente nella parte degli '0'. Quindi:

- x = 0^a per qualche a ≥ 0
- y = 0^b per qualche b > 0 (da |y| > 0)
- z = 0^(3p+1-a-b) 1^p

Nota che a + b ≤ p, quindi 3p+1-a-b ≥ 3p+1-p = 2p+1 > 0.

Derivazione della contraddizione: Consideriamo xy^0z = xz = 0^(3p+1-b) 1^p.

Per essere in L, questa stringa deve soddisfare:
3p+1-b > 3p

Questo implica 1-b > 0, cioè b < 1.

Ma b è un intero positivo (da |y| > 0), quindi b ≥ 1, che contraddice b < 1.

Contraddizione! Quindi L non è regolare. □

Problema 2 (8 punti)

Dimostra che se L è context-free, allora substring(L) = {v | uvw ∈ L per qualche u,w} è context-free.

Dimostrazione costruttiva

Dato: L è un linguaggio context-free con CFG G = (V, Σ, R, S).

Obiettivo: Costruire una CFG G' per substring(L).

Intuizione: Una stringa v è in substring(L) se esiste una derivazione in G che produce una stringa contenente v come substring. Dobbiamo "estrarre" tutte le possibili substring dalle derivazioni di G.

Costruzione di G': G' = (V ∪ {S', X}, Σ, R', S') dove:

- S' è un nuovo simbolo iniziale
- X è un nuovo simbolo ausiliario
- R' contiene le seguenti produzioni:

1. **Avvio dell'estrazione:** S' → X
2. **Copia delle produzioni originali:** Per ogni A → α in R, aggiungi A → α a R'
3. **Regole di "taglio" a sinistra:** Per ogni A → α in R, aggiungi X → α a R'
4. **Regole di "taglio" a destra:** Per ogni simbolo A ∈ V, aggiungi A → ε a R'

Spiegazione delle regole:

- Le regole (2) preservano la capacità di generare stringhe di L
- Le regole (3) permettono di iniziare la generazione da qualsiasi variabile, simulando il "taglio" del prefisso
- Le regole (4) permettono di fermare la generazione prematuramente, simulando il "taglio" del suffisso

Formalizzazione alternativa più rigorosa:

Definiamo R' come segue:

1. S' → S (può generare qualsiasi stringa di L)

2. Per ogni A → α in R:
    - Aggiungi A → α a R'
    - Aggiungi S' → α (può iniziare da qualsiasi produzione)
3. Per ogni A ∈ V, aggiungi A → ε a R' (può terminare prematuramente)

Correttezza:

Lemma: Se v ∈ substring(L), allora v ∈ L(G').

Dimostrazione: Se v ∈ substring(L), allora esiste uvw ∈ L per qualche u,w. Poiché uvw ∈ L, abbiamo S ⇒*_G uvw. Questa derivazione può essere vista come una sequenza di passi che prima genera u, poi genera v, poi genera w. In G', possiamo:

- Usare S' → α dove α inizia la generazione della parte che produce v
- Applicare le produzioni per generare v
- Usare le regole A → ε per fermarci prima di generare il suffisso

Lemma: Se v ∈ L(G'), allora v ∈ substring(L).

Dimostrazione: Se v ∈ L(G'), allora S' ⇒*_{G'} v. Questa derivazione può essere trasformata in una derivazione che genera una stringa completa di L contenente v come substring, estendendo opportunamente con prefisso e suffisso usando le produzioni originali.

Pertanto substring(L) = L(G') è context-free. □

Problema 3 (8 punti)

Mostra che ogni linguaggio Turing-riconoscibile può essere riconosciuto da un automa deterministico a coda.

Dimostrazione di equivalenza

Definizione formale di automa a coda: Un automa deterministico a coda è una tupla M = (Q, Σ, Γ, δ, q_0, F) dove:

- Q è l'insieme finito di stati
- Σ è l'alfabeto di input
- Γ è l'alfabeto della coda
- δ: Q × (Σ ∪ {⊔}) × (Γ ∪ {ε}) → Q × (Γ ∪ {ε}) è la funzione di transizione

- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme di stati finali

La transizione $\delta(q, a, b) = (q', c)$ significa:

- Stato corrente: q, simbolo input: a, simbolo in testa alla coda: b
- Nuovo stato: q', simbolo da scrivere in coda (push): c
- Se $b \neq \varepsilon$, il simbolo b viene rimosso dalla coda (pull)

Teorema: Automi a coda deterministici ≡ Turing Machine

Dimostrazione:

Parte 1: TM ⊆ Automa a coda

Data: Una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q\_acc, q\_rej)$
Obiettivo: Costruire un automa a coda M' equivalente

Strategia: Simulare il nastro della TM usando la coda come memoria di lavoro.

Rappresentazione del nastro: Il contenuto del nastro viene rappresentato nella coda come: [simbolo_pos_-k]...[simbolo_pos_-1][MARKER][simbolo_pos_0][simbolo_pos_1]...[simbolo_pos_j]

dove MARKER è un simbolo speciale che indica la posizione corrente della testina.

Algoritmo di simulazione:

1. **Inizializzazione:**
   - Copia l'input nella coda preceduto da MARKER
   - Stato iniziale corrisponde a q_0
2. **Simulazione di una transizione $\delta(q,a) = (q',b,D)$: Per movimento a destra (D = R):**

   a. Estrai tutti i simboli fino a trovare MARKER (memorizza in stati)
   b. Estrai il simbolo successivo a (quello sotto la testina)
   c. Reinserisci i simboli estratti
   d. Push b (nuovo simbolo)
   e. Push MARKER (nuova posizione testina)
   f. Cambia stato a q'

   Per movimento a sinistra (D = L):

a. Estrai simboli fino a MARKER
b. L'ultimo simbolo estratto prima di MARKER è quello a sinistra
c. Riorganizza: push MARKER, poi b, poi gli altri simboli
d. Cambia stato a q'

3. **Accettazione/Rifiuto:**
   - L'automa entra in stato finale quando la TM raggiunge q_acc
   - L'automa si ferma senza accettare quando la TM raggiunge q_rej

Dettagli implementativi:

- Servono stati ausiliari per memorizzare i simboli durante la riorganizzazione
- Il numero di stati è finito perché dobbiamo memorizzare al più una quantità finita di simboli durante ogni transizione

Parte 2: Automa a coda ⊆ TM

Data: Un automa a coda $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
Obiettivo: Costruire una TM M' equivalente

Strategia: Simulare la coda usando il nastro della TM.

Rappresentazione della coda:

- La coda è rappresentata su una porzione del nastro
- Una marca speciale indica la testa (destra) della coda
- Un'altra marca indica la fine (sinistra) della coda

Simulazione:

- **Pull:** La TM si sposta alla testa della coda, legge il simbolo, lo cancella, e aggiorna la marca
- **Push:** La TM si sposta alla fine della coda, scrive il nuovo simbolo, e aggiorna la marca
- Il controllo degli stati è diretto

Correttezza: Ogni operazione della coda è fedelmente simulata sul nastro.

Conclusione: Automi a coda deterministici ≡ TM ≡ Linguaggi Turing-riconoscibili. □

Problema 4 (8 punti)

Parte (a): Constrained-4-Color ∈ NP

Problema:

$$\text{Constrained-4-Color} = \{\langle G, f_1, \ldots, f_n\rangle \mid \exists \text{ colorazione } c_1, \ldots, c_n \text{ tale che } c_v \neq f_v \ \forall v \text{ e } c_u \neq c_v \text{ se } (u,v) \in E\}$$

Certificato: Una colorazione c: V → {rosso, blu, verde, giallo}

Verificatore polinomiale:

```
Verify(⟨G, f₁, ..., fₙ⟩, c):
1. Per ogni vertice v:
   a. Verifica che c(v) ≠ fᵥ
2. Per ogni arco (u,v) ∈ E:
   a. Verifica che c(u) ≠ c(v)
3. Return true se tutte le verifiche passano
```

Analisi di complessità:

- Dimensione del certificato: $O(n \log 4) = O(n)$ bit
- Tempo di verifica: $O(n + |E|)$ per controllare vincoli
- La verifica è polinomiale

Quindi Constrained-4-Color ∈ NP. □

Parte (b): Constrained-4-Color è NP-hard

Riduzione: 3-Color $\leq_p$ Constrained-4-Color

Problema 3-Color (NP-completo):

$$\text{3-Color} = \{\langle G\rangle \mid G \text{ può essere colorato con 3 colori}\}$$

Dato: Un grafo G = (V, E) con $V = \{v_1, v_2, \ldots, v_n\}$

Costruzione dell'istanza Constrained-4-Color:

- Grafo G' = G (stesso grafo)
- Colori proibiti: f_i = giallo per ogni $v_i \in V$

Spiegazione: Proibiamo il quarto colore (giallo) a tutti i vertici, effettivamente riducendo il problema a una 3-colorazione.

Correttezza della riduzione:

⇒**:** Se G ∈ 3-Color, allora G può essere colorato con {rosso, blu, verde}. Questa stessa colorazione è valida per Constrained-4-Color perché:

- Non usa il colore giallo (rispetta i vincoli f_i = giallo)
- Rispetta i vincoli di adiacenza (era una 3-colorazione valida)

Quindi $\langle G', f_1, ..., f_n \rangle \in$ Constrained-4-Color.

⇐**:** Se $\langle G', f_1, ..., f_n \rangle \in$ Constrained-4-Color, allora esiste una colorazione valida c. Poiché f_i = giallo per ogni vertice, la colorazione c usa solo {rosso, blu, verde}. Poiché c rispetta i vincoli di adiacenza, è una 3-colorazione valida di G.

Quindi G ∈ 3-Color.

Computabilità: La trasformazione è chiaramente polinomiale (identità sul grafo, assegnazione costante dei colori proibiti).

Alternativa più sofisticata (riduzione da 4-Color): Se preferissimo usare 4-Color come riferimento:

Dato: Un grafo G
Costruzione:

- G' = G ∪ {nuovo vertice v_new}
- Connetti v_new a tutti i vertici di G
- f_i = rosso per tutti i vertici originali
- f_new = blu per il nuovo vertice

Questo forza una struttura più complessa ma la riduzione da 3-Color è più diretta e elegante.

Poiché 3-Color è NP-completo e 3-Color $\leq_p$ Constrained-4-Color, concludiamo che Constrained-4-Color è NP-hard. □

Conclusione: Constrained-4-Color è NP-completo.