Automi e Linguaggi Formali - Esame del 15 Settembre 2023

Problema 1 (9 punti)

Considera il linguaggio L = {0^m 1^n | 3m ≤ 2n}. 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^p 1^(⌈3p/2⌉) ∈ L.

Verifichiamo che w ∈ L: dobbiamo avere 3m ≤ 2n, cioè 3p ≤ 2⌈3p/2⌉.
Poiché ⌈3p/2⌉ ≥ 3p/2, abbiamo 2⌈3p/2⌉ ≥ 3p, quindi w ∈ L.

Inoltre, |w| = p + ⌈3p/2⌉ ≥ p, quindi il pumping lemma si applica.

Analisi della decomposizione: Poiché |xy| ≤ p e w inizia con p 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^(p-a-b) 1^(⌈3p/2⌉)

Derivazione della contraddizione: Consideriamo xy^0z = xz = 0^(p-b) 1^(⌈3p/2⌉).

Per essere in L, questa stringa deve soddisfare:
3(p-b) ≤ 2⌈3p/2⌉

Ma sappiamo che 2⌈3p/2⌉ ≥ 3p (come verificato sopra), quindi:
3(p-b) ≤ 3p
3p - 3b ≤ 3p

-3b ≤ 0
b ≥ 0

Questo è sempre vero dato che b > 0. Dobbiamo considerare il caso i = 2.

Consideriamo xy^2z = 0^(p+b) 1^(⌈3p/2⌉).

Per essere in L: 3(p+b) ≤ 2⌈3p/2⌉ ≤ 3p + 1 (usando il bound superiore del ceiling).

Quindi: 3p + 3b ≤ 3p + 1, che implica 3b ≤ 1, cioè b ≤ 1/3.

Ma b è un intero positivo, quindi b ≥ 1, che contraddice b ≤ 1/3.

Contraddizione! Quindi L non è regolare. □

Problema 2 (9 punti)

Dimostra che se L è context-free, allora delete#(L) = {xy | x#y ∈ L} 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 delete#(L).

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

1. **Nuovo simbolo iniziale:** S' è un nuovo simbolo non in V
2. **Nuove produzioni iniziali:** S' → S
3. **Regole di trasformazione:** R' è ottenuta da R come segue:

Per ogni produzione A → α in R:

- Se α non contiene #, aggiungi A → α a R'
- Se α = β#γ (una sola occorrenza di #), aggiungi A → βγ a R'
- Se α contiene multiple occorrenze di #, sostituisci ogni # con ε

Formalizzazione della trasformazione: Definiamo la funzione remove_hash: (Σ ∪ {#})* → Σ* tale che:

- remove_hash(ε) = ε
- remove_hash(a) = a per a ∈ Σ

- remove_hash(#) = ε
- remove_hash(αβ) = remove_hash(α) · remove_hash(β)

Allora R' = {A → remove_hash(α) | A → α ∈ R}

Correttezza:

Lemma: Per ogni A ∈ V e w ∈ Σ*, abbiamo A ⇒_{G'} w se e solo se esiste u ∈ (Σ ∪ {#}) tale che A ⇒*_G u e remove_hash(u) = w.

Dimostrazione del Lemma: Per induzione sulla lunghezza della derivazione.

Teorema principale: L(G') = delete#(L)

⊆**:** Se w ∈ L(G'), allora S' ⇒_{G'} w. Per costruzione, S' → S, quindi S ⇒_{G'} w. Dal lemma, esiste u tale che S ⇒*_G u e remove_hash(u) = w. Quindi u ∈ L e w = remove_hash(u), che significa w ∈ delete#(L).

⊇**:** Se w ∈ delete#(L), allora w = xy dove x#y ∈ L. Quindi S ⇒_G x#y. Dal lemma, S ⇒{G'} remove_hash(x#y) = xy = w. Per costruzione, S' → S, quindi S' ⇒*{G'} w, cioè w ∈ L(G').

Pertanto delete#(L) è context-free. □

Problema 3 (9 punti)

Dimostra che ogni linguaggio Turing-riconoscibile sull'alfabeto {0,1,2} può essere riconosciuto da una TM con alfabeto ternario.

Dimostrazione

Definizione: Una TM con alfabeto ternario ha Σ = {0,1,2} e Γ = {0,1,2,⊔}.

Teorema: Ogni linguaggio L ⊆ {0,1,2}* che è Turing-riconoscibile può essere riconosciuto da una TM con alfabeto ternario.

Dimostrazione: Sia L un linguaggio Turing-riconoscibile su {0,1,2}. Allora esiste una TM standard M = (Q, {0,1,2}, Γ', δ, q_0, q_acc, q_rej) che riconosce L, dove Γ' può contenere simboli arbitrari.

Costruiamo una TM con alfabeto ternario M' = (Q', {0,1,2}, {0,1,2,⊔}, δ', q_0', q_acc', q_rej') equivalente a M.

Strategia: Simulare M usando solo i simboli {0,1,2,⊔}.

Codifica dei simboli: Per ogni simbolo $s \in \Gamma'$, definiamo una codifica $encode(s) \in \{0,1,2\}^+$ tale che:

- I simboli distinti hanno codifiche distinte
- Le codifiche hanno tutte la stessa lunghezza $k = \lceil \log_3 |\Gamma'| \rceil$

Esempio di codifica: Se $\Gamma' = \{0,1,2,\sqcup,a,b,c\}$, allora $k = 2$ e possiamo usare:

- $0 \to 00$, $1 \to 01$, $2 \to 02$, $\sqcup \to 10$
- $a \to 11$, $b \to 12$, $c \to 20$

Costruzione di M':

1. **Stati:** Q' include stati per:
   - Simulazione diretta di Q
   - Stati ausiliari per navigazione e codifica/decodifica
2. **Inizializzazione:** M' codifica l'input sostituendo ogni simbolo con la sua codifica
3. **Simulazione di una transizione $\delta(q,s) = (q',s',D)$:**
   - Localizza la posizione corrente della testina simulata
   - Decodifica il simbolo attuale s
   - Applica la transizione originale
   - Codifica il nuovo simbolo s'
   - Sposta la testina simulata nella direzione D
4. **Gestione della testina:**
   - La posizione della testina originale è tracciata usando stati
   - Movimento richiede navigazione attraverso k celle per simbolo

Algoritmo dettagliato per simulare una transizione:

```
1. Trova_inizio_simbolo_corrente()
2. simbolo := Decodifica_simbolo_corrente()
3. (nuovo_stato, nuovo_simbolo, direzione) := δ(stato_corrente, simbolo)
4. Sovrascrivi_con_codifica(nuovo_simbolo)
5. Se direzione = L:
     Sposta_testina_k_posizioni_sinistra()
   Altrimenti:
     Sposta_testina_k_posizioni_destra()
6. stato_corrente := nuovo_stato
```

Correttezza:

- Ogni configurazione di M corrisponde a una configurazione di M'
- Ogni transizione di M è fedelmente simulata da una sequenza di transizioni in M'
- M' accetta se e solo se M accetta

Conclusione: M' riconosce L usando solo l'alfabeto ternario {0,1,2,⊔}. □

Problema 4 (9 punti)

Parte (a): Formulazione come linguaggio SUM_TM

Definizione del problema: Una TM M "somma correttamente" se per ogni input della forma x#y dove x,y sono rappresentazioni binarie di numeri naturali, M termina con output che è la rappresentazione binaria di x+y.

Linguaggio SUM_TM:

```
SUM_TM = { ⟨M⟩ | M è una TM che somma correttamente }
```

Formalmente:

```
SUM_TM = { ⟨M⟩ | ∀x,y ∈ {0,1}*, M(x#y) = bin(val(x) + val(y)) }
```

dove:

- val(x) è il valore numerico della stringa binaria x
- bin(n) è la rappresentazione binaria del numero n

- M(w) denota l'output di M sull'input w (assumendo che M termini)

Parte (b): SUM_TM è indecidibile

Teorema: SUM_TM è indecidibile.

Dimostrazione per riduzione da HALT_TM:

Useremo HALT_TM = {⟨M,w⟩ | M si ferma su input w}, che è indecidibile.

Riduzione: HALT_TM ≤ SUM_TM

Dato: Un'istanza ⟨M,w⟩ di HALT_TM
Costruzione: Costruiamo una TM M' tale che ⟨M'⟩ ∈ SUM_TM ⟺ ⟨M,w⟩ ∈ HALT_TM

Costruzione di M':

```
M'(input):
1. Analizza se input ha la forma x#y con x,y binari
2. Se no, rifiuta
3. Se sì:
   a. Simula M su w per |x| + |y| passi
   b. Se M si ferma entro questi passi:
      - Calcola e output bin(val(x) + val(y))
   c. Se M non si ferma entro questi passi:
      - Output "errore" (non la somma corretta)
```

Correttezza della riduzione:

⇒**:** Se ⟨M,w⟩ ∈ HALT_TM, allora M si ferma su w in k passi per qualche k.

Per ogni input x#y con |x| + |y| ≥ k, M' eseguirà almeno k passi di simulazione, quindi M si fermerà e M' produrrà la somma corretta.

Per input x#y con |x| + |y| < k, M' potrebbe non simulare abbastanza a lungo, ma possiamo modificare la costruzione per simulare sempre almeno un numero fisso di passi sufficienti per gli input "piccoli".

Versione corretta di M':

```
M'(input):
1. Analizza se input ha la forma x#y con x,y binari
2. Se no, rifiuta
3. Se sì:
   a. Simula M su w per max(|x| + |y|, 1000) passi
   b. Se M si ferma: output bin(val(x) + val(y))
   c. Se M non si ferma: output "0" (scorretto per somme > 0)
```

Allora M' somma correttamente solo se M si ferma su w (in tempo ragionevole).

⇐**:** Se ⟨M'⟩ ∈ SUM_TM, allora M' somma correttamente tutti gli input. In particolare, considera x = "1", y = "1". Allora M' deve output "10" (binario per 2). Questo significa che M deve fermarsi nella simulazione, quindi M si ferma su w.

Computabilità: M' può essere costruita algoritmicamente da M e w.

Poiché HALT_TM è indecidibile e HALT_TM ≤ SUM_TM, concludiamo che SUM_TM è indecidibile.
□