Automi e Linguaggi Formali - Esame del 24 Giugno 2022

Problema 1 (12 punti)

Se L è un linguaggio regolare sull'alfabeto {0,1}, dimostra che anche ROR(L) è regolare, dove:

> ROR(L) = {aw | wa ∈ L, w ∈ {0,1}*, a ∈ {0,1}}

Dimostrazione costruttiva

Teorema: Se L è regolare, allora ROR(L) è regolare.

Dimostrazione: Costruiamo un NFA per ROR(L) dato un DFA per L.

Dato: $M = (Q, \{0,1\}, \delta, q_0, F)$ è un DFA che riconosce L.

Costruzione dell'NFA M' per ROR(L):

$M' = (Q', \{0,1\}, \delta', q_0', F')$ dove:

- **Stati:** $Q' = Q \cup \{(q,a) \mid q \in Q, a \in \{0,1\}\} \cup \{q_0'\}$
- **Stato iniziale:** q_0' (nuovo stato)
- **Stati finali:** $F' = \{(q,a) \mid \delta^*(q_0, a) = q, q \in F\}$
- **Transizioni:**

1. **Lettura del primo simbolo:**
   - $\delta'(q_0', a) = \{(q_0, a)\}$ per $a \in \{0,1\}$
2. **Elaborazione del resto della stringa:**
   - $\delta'((q,a), b) = \{(\delta(q,b), a)\}$ per ogni $q \in Q$, $a,b \in \{0,1\}$
3. **Transizioni su epsilon alla fine:**
   - Aggiungiamo transizioni ε da ogni stato (q,a) verso δ(q,a)
   - Se $\delta(q,a) \in F$, allora $(q,a) \in F'$

Spiegazione dell'algoritmo:

1. All'inizio, M' legge non-deterministicamente il primo simbolo a e lo "memorizza" negli stati
2. Simula M sul resto della stringa w, mantenendo a in memoria

3. Alla fine, verifica che il simbolo memorizzato a, se "aggiunto" alla fine, porterebbe a uno stato finale

Correttezza:

⇒**:** Se aw ∈ ROR(L), allora wa ∈ L. Quindi $\delta^*(q_0, wa) \in F$, cioè $\delta(\delta^*(q_0, w), a) \in F$. M' può:

- Leggere a e andare in (q_0, a)
- Simulare l'elaborazione di w: $(q_0, a) \to_w (\delta^*(q_0, w), a)$
- Verificare che $\delta(\delta^*(q_0, w), a) \in F$

⇐**:** Se M' accetta aw, allora durante la computazione ha memorizzato il primo simbolo a e ha verificato che $\delta(\delta^*(q_0, w), a) \in F$, il che significa wa ∈ L.

Approccio alternativo con espressioni regolari

Se L ha espressione regolare R, allora ROR(L) ha espressione regolare:

$$ROR(R) = 0 \cdot SHIFT(R, 0 \to \varepsilon, 1 \to \varepsilon) \cdot 0 + 1 \cdot SHIFT(R, 0 \to \varepsilon, 1 \to \varepsilon) \cdot 1$$

dove SHIFT sposta il primo simbolo alla fine.

Poiché le espressioni regolari sono chiuse sotto queste operazioni, ROR(L) è regolare. □

Problema 2 (12 punti)

*Dimostra che $L_2 = \{uv \mid u \in \Sigma, v \in \Sigma 1 \Sigma^* \text{ e } |u| \geq |v|\}$ non è regolare.***

Dimostrazione per contraddizione usando il Pumping Lemma

Definizione precisa di L_2:

$$L_2 = \{uv \mid u \in \{0,1\}^*, v \in \{0,1\}^*1\{0,1\}^*, |u| \geq |v|\}$$

Quindi v deve contenere almeno un '1' e la lunghezza di u deve essere almeno quella di v.

Assunzione: Supponiamo per contraddizione che L_2 sia regolare.

Applicazione del Pumping Lemma: Esiste p > 0 tale che ogni stringa $w \in L_2$ con $|w| \geq p$ può essere decomposta come w = xyz con:

1. |xy| ≤ p
2. |y| > 0
3. xy^i z ∈ L_2 per ogni i ≥ 0

Scelta della stringa di test: Consideriamo w = 0^(2p) 1 ∈ L_2.

Verifichiamo che w ∈ L_2:

- u = 0^(2p), v = 1
- |u| = 2p ≥ 1 = |v| ✓
- v = 1 ∈ {0,1}*1{0,1}* ✓

Inoltre, |w| = 2p + 1 ≥ p.

Analisi della decomposizione: Poiché |xy| ≤ p e w inizia con 2p zeri, xy deve essere contenuto interamente nei primi p zeri. Quindi:

- x = 0^a per qualche a ≥ 0
- y = 0^b per qualche b > 0
- z = 0^(2p-a-b) 1

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

Per essere in L_2, questa stringa deve essere decomponibile come uv dove:

- v contiene almeno un '1'
- |u| ≥ |v|

Le possibili decomposizioni sono:

1. u = 0^k, v = 0^(2p-b-k) 1 per qualche 0 ≤ k ≤ 2p-b

Per la condizione |u| ≥ |v|:
k ≥ 2p-b-k+1
2k ≥ 2p-b+1
k ≥ p - b/2 + 1/2

Ma k ≤ 2p-b, quindi:
p - b/2 + 1/2 ≤ 2p-b
-p + b/2 + 1/2 ≤ 0

b/2 ≤ p - 1/2
b ≤ 2p - 1

Questo è sempre vero. Proviamo con xy^2z.

Consideriamo xy^2z = 0^(2p+b) 1.

Le decomposizioni possibili sono u = 0^k, v = 0^(2p+b-k) 1. Per |u| ≥ |v|: k ≥ 2p+b-k+1 2k ≥ 2p+b+1
k ≥ p + b/2 + 1/2

Ma k ≤ 2p+b, quindi:
p + b/2 + 1/2 ≤ 2p+b
-p + b/2 + 1/2 ≤ 0

Questo è la stessa disuguaglianza di prima.

Strategia alternativa: Consideriamo w = 0^p 1^p.

Verifichiamo che w ∈ L_2:

- u = 0^p, v = 1^p
- |u| = |v| = p, quindi |u| ≥ |v| ✓
- v = 1^p ∈ {0,1}*1{0,1}* ✓

Nella decomposizione xyz con |xy| ≤ p:

- y consiste solo di zeri (y = 0^b, b > 0)
- xy^0z = 0^(p-b) 1^p

Per essere in L_2, dobbiamo avere una decomposizione uv con |u| ≥ |v|.
L'unica possibilità è u = 0^(p-b), v = 1^p.
Ma allora |u| = p-b < p = |v|, contraddizione!

Contraddizione! Quindi L_2 non è regolare. □

Problema 3 (12 punti)

Mostra che per ogni PDA P esiste un PDA P_2 con due soli simboli di stack tale che L(P_2) = L(P).

Dimostrazione costruttiva

Teorema: Ogni linguaggio context-free può essere riconosciuto da un PDA con alfabeto di stack binario.

Dimostrazione: Dato un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, costruiamo P_2 con alfabeto di stack $\{0, 1\}$.

Strategia: Codificare ogni simbolo $\gamma \in \Gamma$ con una stringa binaria $encode(\gamma) \in \{0,1\}^+$.

Codifica dell'alfabeto di stack

Assegnazione delle codifiche: Sia $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_k\}$. Definiamo:

- $k = |\Gamma|$
- $\ell = \lceil \log_2 k \rceil$ (lunghezza delle codifiche)
- $encode(\gamma_i)$ = rappresentazione binaria di $i-1$ con ℓ bit

Esempio: Se $\Gamma = \{Z_0, A, B, C\}$, allora:

- $encode(Z_0) = 00$
- $encode(A) = 01$
- $encode(B) = 10$
- $encode(C) = 11$

Costruzione di P_2

$P_2 = (Q', \Sigma, \{0,1\}, \delta_2, q_0', 0, F')$ dove:

Stati: $Q' = Q \cup \{\text{stati ausiliari per codifica/decodifica}\}$

Simulazione di una transizione $\delta(q, a, \gamma) = (q', \alpha)$:

1. **Decodifica del simbolo in cima:**
   - Leggi ℓ bit dalla pila per ricostruire γ
   - Verifica che corrisponda al simbolo aspettato
2. **Applicazione della transizione:**
   - Se $\alpha = \varepsilon$ (pop), non fare nulla
   - Se $\alpha = \gamma_1\gamma_2...\gamma_m$ (push), scrivi $encode(\gamma_m)...encode(\gamma_2)encode(\gamma_1)$
3. **Aggiornamento dello stato:** $q \to q'$

Algoritmo dettagliato

Per simulare $\delta(q, a, \gamma) = (q', \gamma_1\gamma_2...\gamma_m)$:

```
Fase 1: Decodifica (stati ausiliari q_decode_1, ..., q_decode_ℓ)
- Leggi ℓ bit dalla pila
- Ricostruisci γ e verifica che sia quello aspettato

Fase 2: Elaborazione simbolo input
- Consuma a dall'input (se a ≠ ε)

Fase 3: Codifica e push (stati ausiliari q_encode_1, ..., q_encode_m)
- Per i = m, m-1, ..., 1:
  - Scrivi encode(γ_i) sulla pila (bit per bit)

Fase 4: Transizione di stato
- Vai in stato q'
```

Gestione dello stack vuoto

Problema: Come riconoscere quando lo stack è vuoto?

Soluzione: Usiamo un marker speciale:

- encode(Z_0) include sempre un pattern riconoscibile (es. inizia con 00)
- Quando decodifichiamo e troviamo questo pattern, sappiamo di essere al fondo

Correttezza

Invariante: In ogni momento, il contenuto dello stack di P_2 è la concatenazione delle codifiche dei simboli nello stack di P.

Lemma: Ogni configurazione $(q, w, \gamma_1\gamma_2...\gamma_n)$ di P corrisponde alla configurazione $(q, w, \text{encode}(\gamma_1)\text{encode}(\gamma_2)...\text{encode}(\gamma_n))$ di P_2.

Dimostrazione per induzione:

- **Base:** Configurazione iniziale corrisponde
- **Passo:** Ogni transizione di P è fedelmente simulata da una sequenza di transizioni in P_2

Complessità della costruzione

- **Numero di stati:** $|Q'| = O(|Q| \cdot |\Gamma| \cdot \ell)$

- **Lunghezza transizioni:** Ogni transizione di P richiede $O(\ell \cdot |\alpha|)$ transizioni in P_2
- **Alfabeto stack:** {0, 1} (fissato)

Conclusione: P_2 riconosce L(P) usando solo due simboli di stack. □

Esempio pratico

PDA originale:

$\delta(q_0, a, Z_0) = (q_1, AZ_0)$
$\delta(q_1, b, A) = (q_1, AA)$
$\delta(q_1, c, A) = (q_2, \varepsilon)$

PDA con stack binario:

Codifica: $Z_0 \rightarrow 00$, $A \rightarrow 01$

$\delta_2(q_0, a, 0) = (temp_1, \varepsilon)$ // leggi primo bit di Z_0
$\delta_2(temp_1, \varepsilon, 0) = (temp_2, \varepsilon)$ // leggi secondo bit di Z_0
$\delta_2(temp_2, \varepsilon, \varepsilon) = (temp_3, 0)$ // push primo bit di A
$\delta_2(temp_3, \varepsilon, \varepsilon) = (temp_4, 1)$ // push secondo bit di A
$\delta_2(temp_4, \varepsilon, \varepsilon) = (temp_5, 0)$ // push primo bit di Z_0
$\delta_2(temp_5, \varepsilon, \varepsilon) = (q_1, 0)$ // push secondo bit di Z_0

Questo dimostra che la costruzione è sempre possibile e preserva il linguaggio riconosciuto.