Automi e Linguaggi Formali - 15/7/2024

Secondo appello – Prima parte - Soluzioni

Esercizio 1 (12 punti) - Operazione SWAP e linguaggi regolari

Teorema: Se $L \subseteq \Sigma^*$ è regolare, allora SWAP(L) è regolare

Dimostrazione per costruzione di automa:

Dato che L è regolare, esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L.

Costruiamo un NFA $A' = (Q', \Sigma, \delta', q_0', F')$ che riconosce SWAP(L):

Costruzione:

- $Q' = Q \times \{even, odd, first\} \cup \{q_0'\}$
- q_0' è un nuovo stato iniziale
- $F' = F \times \{even, odd, first\}$

Funzione di transizione δ':

```
δ'(q₀', a) = {(δ(q₀, a), first)} per ogni a ∈ Σ
δ'((q, first), a) = {(δ(q, a), even)} per ogni q ∈ Q, a ∈ Σ
δ'((q, even), a) = {(δ(q, b), odd) | b ∈ Σ} per ogni q ∈ Q, a ∈ Σ
δ'((q, odd), b) = {(δ(δ(q, a), b), even) | δ'((p, even), a) = {(δ(p, c), odd)} e p tale che δ(p, c) = q}
```

Idea più semplice - Costruzione diretta:

Costruiamo un NFA che "indovina" come raggruppare i simboli:

A' = "Su input w:

1. Se $|w| = 0$: accetta se $\varepsilon \in L$
2. Se $|w| = 1$: accetta se $w \in L$
3. Altrimenti, per ogni possibile raggruppamento di w in coppie $a_0a_1, a_2a_3, \ldots$:
   - Forma la stringa w' scambiando ogni coppia: $a_1a_0a_3a_2\ldots$
   - Verifica se $w' \in L$ usando A

- Se sì, accetta"

Descrizione formale alternativa:

Poiché i linguaggi regolari sono chiusi sotto trasformazioni definite da transducer a stati finiti, e SWAP può essere implementata da un transducer finito, SWAP(L) è regolare.

Transducer per SWAP:

- Stati: $\{q_0, q_1, q_2\}$
- q_0: stato iniziale
- Su input a in q_0: va in q_1, memorizza a, non produce output
- Su input b in q_1: va in q_0, produce ba
- Su EOF in q_1: produce a

Componendo questo transducer con l'automa per L otteniamo un automa per SWAP(L). ■

Esercizio 2 (12 punti) - Linguaggio delle permutazioni

Teorema: L_2 = {uv | u, v ∈ {0,1}* e u è permutazione di v} non è regolare

Dimostrazione per Pumping Lemma:

Supponiamo per contraddizione che L_2 sia regolare. Sia k la costante di pompaggio.

Consideriamo la stringa s = 0^k 1^k 0^k 1^k ∈ L_2, dove:

- u = 0^k 1^k
- v = 0^k 1^k
- u è chiaramente una permutazione di v (stessi simboli, stesso numero)

Poiché |s| = 4k > k, per il Pumping Lemma esistono x, y, z tali che:

1. s = xyz
2. |xy| ≤ k
3. |y| > 0
4. xy^i z ∈ L_2 per ogni i ≥ 0

Analisi dei casi:

Dato che $|xy| \leq k$, la sottostringa xy è contenuta nei primi k simboli di s, quindi $xy \subseteq 0^k$.

Quindi $y = 0^j$ per qualche $1 \leq j \leq k$.

Caso i = 0: Consideriamo w = xz = s senza la parte y. $w = 0^{(k-j)} 1^k 0^k 1^k$

Per essere in L_2, w deve poter essere scritta come uv dove u è permutazione di v.

Le possibili divisioni sono:

- Se $u = 0^a 1^b$ con $a + b \leq 2k - j$, allora $v = 0^{(2k-j-a)} 1^{(2k-b)}$
- Per essere permutazioni: $a = 2k-j-a$ e $b = 2k-b$
- Da cui: $2a = 2k-j$ e $2b = 2k$, quindi $b = k$ e $a = k - j/2$

Ma $j \geq 1$, quindi a non è intero quando j è dispari, contraddicendo l'esistenza della divisione.

Caso i = 2: $w = xy^2z = 0^{(k+j)} 1^k 0^k 1^k$

Analogamente, per ogni divisione in uv dove u è permutazione di v, dobbiamo avere:

- Numero totale di 0 in $w = 2k + j$
- Numero totale di 1 in $w = 2k$
- Per essere permutazioni: u e v devono avere lo stesso numero di 0 e 1

Ma $(2k + j) + 2k$ è dispari quando j è dispari, quindi non può essere diviso ugualmente.

In entrambi i casi otteniamo una contraddizione. Quindi L_2 non è regolare. ■

Esercizio 3 (12 punti) - Linguaggio PALINDROMIZE

Teorema: Se B è regolare, allora PALINDROMIZE(B) = $\{ww^R \mid w \in B\}$ è context-free

Dimostrazione per costruzione di grammatica context-free:

Dato che B è regolare, esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce B.

Costruiamo una grammatica context-free $G = (V, \Sigma, R, S)$ per PALINDROMIZE(B):

Variabili:

- $V = \{S\} \cup \{[q, r] \mid q, r \in Q\}$

- S è il simbolo iniziale
- [q, r] genera tutte le stringhe w tali che $\delta(q, w) = r$

Regole di produzione:

1. **Regola iniziale:** $S \rightarrow [q_0, f]$ per ogni $f \in F$
2. **Regole per palindromi:**
    - $[q, r] \rightarrow a[\delta(q,a), s]a$ per ogni $q, r, s \in Q$ e $a \in \Sigma$, dove esiste un cammino da s a r leggendo l'input al contrario
3. **Regola per stringa vuota:** $[q, q] \rightarrow \varepsilon$ per ogni $q \in Q$

Descrizione più sistematica:

Costruiamo una PDA che:

1. Legge la prima metà della stringa, verificando che appartenga a B
2. Confronta la seconda metà per verificare che sia il rovesciato della prima

Descrizione implementativa della PDA P:

```
P = "Su input w:
1. Fase 1 - Lettura e push:
   - Simula A su simboli di input
   - Push ogni simbolo letto sullo stack
   - Indovina non-deterministicamente quando siamo a metà

2. Fase 2 - Confronto:
   - Per ogni simbolo successivo:
     - Pop dallo stack
     - Verifica che sia uguale al simbolo letto
   - Alla fine dell'input:
     - Stack deve essere vuoto
     - Simulazione di A deve essere in stato finale

3. Accetta se entrambe le condizioni sono soddisfatte"
```

Correttezza:

- P accetta ww^R ⟺ $w \in B$ e la seconda metà è esattamente il rovesciato della prima

- Il non-determinismo permette di indovinare il punto medio
- Lo stack garantisce il confronto palindromico

Quindi PALINDROMIZE(B) è context-free. ■

Costruzione alternativa con grammatica:

Se B ha grammatica regolare con produzioni della forma A → aB | ε, possiamo costruire:

```
S → ASₐ per ogni produzione A → a in G_B
Sₐ → aSₐa per ogni a ∈ Σ
Sₐ → ε
```

Questo genera tutte le stringhe ww^R dove w ∈ B.