Automi e Linguaggi Formali - 1/7/2024

Primo appello – Prima e Seconda parte - Soluzioni

PRIMA PARTE

Esercizio 1 (12 punti) - Traslitterazione e linguaggi regolari

Teorema: Se L ⊆ Σ* è regolare e T è una traslitterazione, allora T(L) è regolare

Dimostrazione per costruzione di automa:

Dato che L è regolare, esiste un DFA A = (Q, Σ, δ, q_0, F) che riconosce L.

Una traslitterazione T: Σ → Γ* mappa ogni simbolo in una stringa. Possiamo estendere T a stringhe: T($a_1a_2...a_n$) = T(a_1)T(a_2)...T(a_n).

Costruiamo un NFA A' = (Q', Γ, δ', q_0, F) che riconosce T(L):

Costruzione:

- Q' = Q (stessi stati)
- Per ogni transizione δ(q, a) = r in A e per ogni simbolo b nella stringa T(a):
  - Aggiungiamo transizioni ε che "scorrono" attraverso T(a)

Descrizione più precisa:

Per ogni a ∈ Σ, sia T(a) = $b_1b_2...b_k$. Per la transizione δ(q, a) = r, introduciamo stati ausiliari e transizioni:

q →^(b_1) q_1 →^(b_2) q_2 →^...→ q_{k-1} →^(b_k) r

Descrizione implementativa alternativa:

Poiché i linguaggi regolari sono chiusi sotto omomorfismi e T è un omomorfismo (esteso alle stringhe), T(L) è regolare.

Dimostrazione formale con omomorfismo:

- T induce un omomorfismo h: Σ* → Γ* dove h(a) = T(a)

- h(L) = {h(w) | w ∈ L} = T(L)
- I linguaggi regolari sono chiusi sotto omomorfismi
- Quindi T(L) è regolare ■

Esercizio 2 (12 punti) - Linguaggio L_2 non regolare

Teorema: L_2 = {x#y | x, y ∈ {0,1}* e x ≠ y} 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 #0^k 1 ∈ L_2 (poiché 0^k ≠ 0^k 1).

Poiché |s| = 2k + 2 > 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| ≤ k, la sottostringa xy è contenuta nei primi k caratteri, quindi xy ⊆ 0^k.

Quindi y = 0^j per qualche 1 ≤ j ≤ k.

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

Per w ∈ L_2, dobbiamo avere che le parti prima e dopo # sono diverse.

- Parte sinistra: 0^(k+j)
- Parte destra: 0^k 1

Queste sono diverse, quindi sembra andare bene.

Caso i = 0: w = xz = 0^(k-j) #0^k 1

- Parte sinistra: 0^(k-j)
- Parte destra: 0^k 1

Anche queste sono diverse per j > 0.

Problema: Scegliamo una stringa diversa.

Correzione della dimostrazione:

Consideriamo s = 0^k 1#0^k 1 ∈ L_2 (poiché 0^k 1 ≠ 0^k 1 è falso!)

Nuova scelta: s = 0^k #0^(k-1) 1 ∈ L_2

Ora |xy| ≤ k implica y = 0^j con 1 ≤ j ≤ k.

Caso i = 0: w = xz = 0^(k-j) #0^(k-1) 1

Se j = 1: w = 0^(k-1) #0^(k-1) 1
Le parti sono 0^(k-1) e 0^(k-1) 1, che sono diverse. ✓

Però per j > 1: w = 0^(k-j) #0^(k-1) 1 rimangono diverse.

Scelta corretta: s = 0^k 1^k #0^k 1^k ∉ L_2 (sono uguali!)

Prendiamo s = 0^k 1^k #0^k 1^k 0 ∈ L_2

Ora xy ⊆ 0^k, quindi y = 0^j.

Caso i = 0: w = 0^(k-j) 1^k #0^k 1^k 0 **Caso i = 2:** w = 0^(k+j) 1^k #0^k 1^k 0

In entrambi i casi le parti sono diverse, contraddicendo il pompaggio...

Dimostrazione corretta:

Usiamo il complemento. L_2' = {x#y | x, y ∈ {0,1}* e x = y} non è regolare (linguaggio standard non-regolare).

Se L_2 fosse regolare, allora L_2' = {0,1}*#{0,1}* ∩ ¬L_2 sarebbe regolare (intersezione di regolari e complemento di regolare).

Ma L_2' non è regolare, quindi L_2 non può essere regolare. ■

Esercizio 3 (12 punti) - Linguaggio SCRAMBLE context-free

Teorema: Se B ⊆ {0,1}* è regolare, allora SCRAMBLE(B) è context-free

Dimostrazione per costruzione di grammatica context-free:

Idea: Una stringa t è permutazione di w ∈ B se t ha lo stesso numero di 0 e 1 di w.

Se w ∈ B ha n_0 zeri e n_1 uni, allora ogni permutazione ha esattamente n_0 zeri e n_1 uni.

Poiché B è regolare, l'insieme delle coppie (n_0, n_1) per stringhe in B è eventualmente periodico e quindi rappresentabile con grammatica context-free.

Costruzione più precisa:

1. **Analisi di B:** Per ogni stringa w ∈ B, definiamo il suo "profilo" prof(w) = $(\#_0(w), \#_1(w))$
2. **Insieme dei profili:** P = {prof(w) | w ∈ B}
3. **Grammatica per SCRAMBLE(B):**
   - Per ogni $(n_0, n_1) \in P$, aggiungiamo regole per generare tutte le permutazioni con n_0 zeri e n_1 uni

Costruzione con PDA:

Costruiamo una PDA P che riconosce SCRAMBLE(B):

Descrizione implementativa:

```
P = "Su input t:
1. Non-deterministicamente indovina w ∈ B tale che t sia permutazione di w

2. Fase di conteggio:
   - Simula un DFA per B su w (memorizzando w)
   - Conta 0 e 1 in w: ottieni (n0, n1)

3. Fase di verifica:
   - Leggi t carattere per carattere
   - Mantieni contatori per 0 e 1 in t
   - Alla fine verifica che i contatori siano (n0, n1)

4. Accetta se B accetta w e i contatori coincidono"
```

Implementazione con Stack:

Più precisamente, usiamo una grammatica context-free:

Costruzione sistematica:

Per ogni stato q di un DFA per B e per ogni coppia (i, j):
- Variabile A[q,i,j] genera stringhe che portano il DFA in q con i zeri e j uni rimanenti da distribuire

Regole:
- $A[q,i,j] \to 0\, A[\delta(q,0), i-1, j]$ se $i > 0$
- $A[q,i,j] \to 1\, A[\delta(q,1), i, j-1]$ se $j > 0$
- $A[q,0,0] \to \varepsilon$ se q è finale

Simbolo iniziale: $S \to A[q_0,n_0,n_1]$ per ogni $(n_0,n_1) \in P$

Dove P è calcolabile poiché B è regolare.

Quindi SCRAMBLE(B) è context-free. ■

SECONDA PARTE

Esercizio 1 (12 punti) - Macchina di Turing con "copia e incolla" (CPTM)

(a) Definizione formale della funzione di transizione di una CPTM

Una **CPTM** è una macchina di Turing deterministica che può copiare porzioni di nastro.

Definizione formale:

Una CPTM è una 7-tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ dove la funzione di transizione è estesa:

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, SELECT\_START, SELECT\_END, COPY\}$

Semantica operazionale:

- Configurazione: (q, w, i, start, end) dove start, end sono posizioni selezionate (o ∅)
- **SELECT_START**: start ← i
- **SELECT_END**: end ← i
- **COPY**: se start, end sono definiti e start ≤ end, copia w[start:end] dalla posizione i, sovrascrivendo; altrimenti nessun effetto

(b) Dimostrazione di equivalenza con TM standard

Teorema: Le CPTM riconoscono esattamente i linguaggi Turing-riconoscibili.

Dimostrazione analoga a SRTM:

(⊆) Data CPTM M, costruiamo TM N che simula M mantenendo traccia delle posizioni start/end e implementando COPY con operazioni standard.

(⊇) Data TM M, costruiamo CPTM N che ignora le operazioni speciali e simula M direttamente.

Esercizio 2 (12 punti) - Linguaggio quasi-palindromo

(a) Formulazione come QPALTM

Definizione:

```
QPALTM = {⟨M⟩ | M è una TM e L(M) è quasi-palindromo}
```

dove B è quasi-palindromo se contiene al più una stringa non palindroma.

(b) Indecidibilità di QPALTM

Dimostrazione per riduzione da ATM:

Costruiamo il decisore S per ATM:

```
S = "Su input ⟨M, w⟩:
1. Costruisci M':
   M' = "Su input x:
   a) Se x = 0 o x = 1: accetta
   b) Se x = 01: simula M su w
      - Se M accetta: accetta
      - Altrimenti: rifiuta
   c) Se x = 10: accetta
   d) Altrimenti: rifiuta"

2. Esegui R su ⟨M'⟩
3. Se R accetta: accetta
4. Se R rifiuta: rifiuta"
```

Analisi:

- Se M accetta w: L(M') = {0, 1, 01, 10} (al più una stringa non-palindroma: 01)
- Se M non accetta w: L(M') = {0, 1, 10} (tutte palindrome)

Quindi S decide ATM, contraddicendo l'indecidibilità. ■

Esercizio 3 (12 punti) - Problema COMMITTEE

(a) COMMITTEE ∈ NP

Verificatore:

> V = "Su input $\langle\langle D_1,\ldots,D_m,I\rangle, C\rangle$:
> 1. Verifica $|C \cap D_i| = 1$ per ogni i
> 2. Verifica che non esistano $(d_1,d_2) \in I$ con $d_1,d_2 \in C$
> 3. Accetta se entrambe valgono"

(b) COMMITTEE è NP-hard

Riduzione da 3SAT:

Data formula $\varphi = C_1 \wedge \ldots \wedge C_m$, costruiamo:

Dipartimenti:

- Per ogni variabile x_i: dipartimento $D_i = \{x_i, \neg x_i\}$
- Per ogni clausola C_j: dipartimento $D'_j = \{c_{j,1}, c_{j,2}, \ldots, c_{j,k}\}$ (gadget di scelta)

Inimicizie:

- $(x_i, \neg x_i) \in I$ per ogni variabile
- Collegamenti tra gadget clausola e variabili per forzare soddisfacimento

La riduzione preserva soddisfacibilità ⟺ esistenza commissione. ■