Appello 23-08

1. Esercizio 1

Parola 🡪

La parola ma hai un num. diverso di 0 rispetto al numero di 1 (es. ) e il linguaggio non è regolare.

(Alternativamente – in modo completo)

Supponiamo per assurdo che L sia regolare.

Sia p la costante di pumping garantita dal lemma.

Consideriamo la stringa s = 1^(5p)0^(3p) ∈ L, poiché 5(5p) ≤ 3(3p).

|s| = 8p ≥ p, quindi possiamo applicare il pumping lemma.

Sia s = xyz, con |xy| ≤ p, |y| ≥ 1 e xy^i z ∈ L per ogni i≥0.

Poiché |xy| ≤ p, y può contenere solo 1.

Sia y = 1^k, con k≥1. Consideriamo i=0.

Allora xy^0 z = x1^0 z = xz = 1^(5p-k)0^(3p)

Ma (5p-k) non è divisibile per 5 se k≥1, quindi xz ∉ L.

Ciò è assurdo per il pumping lemma, quindi L non può essere regolare.

1. Esercizio 2

G’ (Forma Normale di Chomsky)

Esempio classico grammatica CF:

S -> aA

A -> aBa

B -> bCb

C ->

Caratteristiche:

G’ con:

* Stesso stato iniziale
* Produzione di regole
  + A -> a
  + Ogni regola che produce

una lettera “a” permette la traslitterazione T

* + A -> T(a)
* Stessa regola finale

Se è CF allora traslitterazione allora :

* (🡪) Ogni regola rispetta la funzione di transizione e genera per ogni ogni regola del tipo
* (🡨) Ogni produzione è in forma derivante da una funzione di transizione che mappa ogni simbolo secondo la forma normale di Chomsky 🡪 .

Sia L un linguaggio context-free su Σ.

Esiste quindi una grammatica context-free G tale che L = L(G).

Costruiamo una nuova grammatica G' su Γ come segue:

* G' ha gli stessi non-terminali di G
* Per ogni produzione A → α in G, G' ha la produzione A → T(α),
* dove T è applicata a ogni simbolo di α
* Gli stessi simboli iniziali

G' genera esattamente le stringhe T(w) per ogni w ∈ L(G).

Infatti, se S =>\* w in G, allora S =>\* T(w) in G',

e viceversa se S =>\* w' in G', w' = T(w) per qualche w tale che S =>\* w in G.

Quindi L(G') = T(L(G)) = T(L).

Poiché G' è context-free, anche T(L) lo è.

1. Esercizio (3)

k-PDA: k Pile

La pila fa due operazioni:

* Pop (toglie il simbolo)
* Push (mette il simbolo)

0-PDA = NFA

* Qua non abbiamo pile
* Semplicemente, è l’NFA normale
  + Stato iniziale / Insieme di stati finali

1-PDA = PDA

* Pop/Push

Domanda tipica: Mostrami che 2-PDA è più potente di 1-PDA

(2 Pile > 1 Pila)

2 Pile 🡪 TM Cosa Fa?

* Legge
* Scrive
* Va a sx
* Va a dx

Due possibilità per risolvere:

1)

* 1 Pila per leggere
* 1 Pila per scrivere
* Entrambe vanno a dx e a sx
* Risolto = fanno le stesse operazioni della TM

2)

* 1 Pila per gestire stesso nastro a sx
* 1 Pila per gestire stesso nastro a dx
* Entrambe leggono e scrivono
* Risolto = fanno le stesse operazioni della TM

Per mostrare che uno è più potente dell’altro:

1. Dimostriamo che uno legge un linguaggio che l’latro non può
2. Ogni nastro fa una certa cosa:

<https://cseweb.ucsd.edu/classes/fa99/cse105/hw3sol.pdf>

4) Esercizio 4

1)

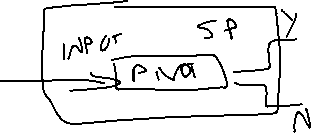
Dato ⟨n,W⟩ e un certificato C, possiamo verificare in tempo polinomiale che:

1. For i=1 ad N verificando che esistano tutti gli elettori
2. Esiste l’elemento pivot 🡪 W[n]
   1. Perché sicuramente esistevano “i” da 0 ad “n”
3. Disuguaglianze verificate

2)

Set Partition = Due sottoinsiemi S1 ed S2 tali che:

* La somma degli elementi in S1 è uguale alla somma degli elementi in S2
* Entrambi danno lo stesso risultato
* = non importa quanta roba c’è dentro, ottengo sempre lo stesso valore



Dato il problema:

Se aggiungo ad ed fallisce la condizione di .

(Per esempio):

Dato un'istanza ⟨S⟩ di SET-PARTITION, con S = {x\_1, ..., x\_m}, costruiamo:

- n = m+1

- W[i] = x\_i per i = 1, ..., m

- W[n] = (1/2) Σ\_{x∈S} x

Allora:

- Se ⟨S⟩ ∈ SET-PARTITION, esiste S\_1 ⊆ S tale che

Σ\_{x∈S\_1} x = (1/2) Σ\_{x∈S} x = W[n].

Quindi C = {i | x\_i ∈ S\_1} dimostra che m+1 è un pivot.

- Se ⟨n,W⟩ ∈ PIVOT, esiste C ⊆ {1, ..., m} che soddisfa le disuguaglianze.

Ponendo S\_1 = {x\_i | i ∈ C}, le disuguaglianze implicano che

Σ\_{x∈S\_1} x = (1/2) Σ\_{x∈S} x, quindi ⟨S⟩ ∈ SET-PARTITION

Data un’istanza del problema, usando un certo numero di elettori:

* Presi due sottoinsiemi (due pezzi a caso) di elettori, entrambi daranno esattamente quello che mi dice il problema:

Aggiungendo l’elemento pivot , tutto somma allo stesso numero , rispettando le condizioni del problema (i due sottoinsiemi hanno stessa somma)

(Adesso abbiamo fatto: usando SET PARTITIONING, risolvi PIVOT)

* Ora il contrario (avendo PIVOT, siamo in SP)

Avendo PIVOT, abbiamo tutti gli elettori.

Questo vuol dire che, aggiungendo W[n], allora entrambi i sottoinsiemi valutano ad .

grazie a

Problema risolto! (NP-Hard)

Appello 05/02/2024

1. Esercizio 3

(enumeratore) Ordinamento standard

Decidibile 🡪 Termina usando un oggetto finito

(🡪)

Usando un enumeratore, seguiamo l’ordinamento.

Il nostro E avrà a disposizione un alfabeto e stampa tutti i simboli nell’ordine previsto (esempio del ciclo for 🡪 , stampiamo tutti i caratteri seguendo l’ordine previsto dalla funzione di transizione. Se l’enumeratore termina, allora significa che è stato correttamente seguito l’ordine previsto dall’alfabeto.

(🡨)

Avendo l’ordinamento standard, esiste un enumeratore. Questo succede perché la funzione di transizione dell’enumeratore prende in input tutte le stringhe ordinate e, se non fossero ordinate, non riesce a stamparle. Correttamente lui stampa solo le stringhe come gli arrivano nel modo ordinato.

Risolto!

1. Esercizio 4

è una TM che, dati in input due numeri binari ed , ottiene

funzione di riduzione, prendendo in input la TM e input:

su input

* Simula sull’input
* Verifica l’esistenza di ed
* Se la TM riesce ad ottenere il prodotto binario, lo scrive sul nastro ed esegue
  + simula la propria esecuzione, avendo input
  + Se riesce a scrivere sul nastro, si ferma accettando, altrimenti rifiuta
* Restituisci

, allora la macchina si ferma accettando dato che ha svolto il prodotto tra i numeri binari previsti

, allora la macchina rifiuta dato che i simboli ed non hanno permesso di ottenere in output il prodotto in formato binario dei due numeri previsti