Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

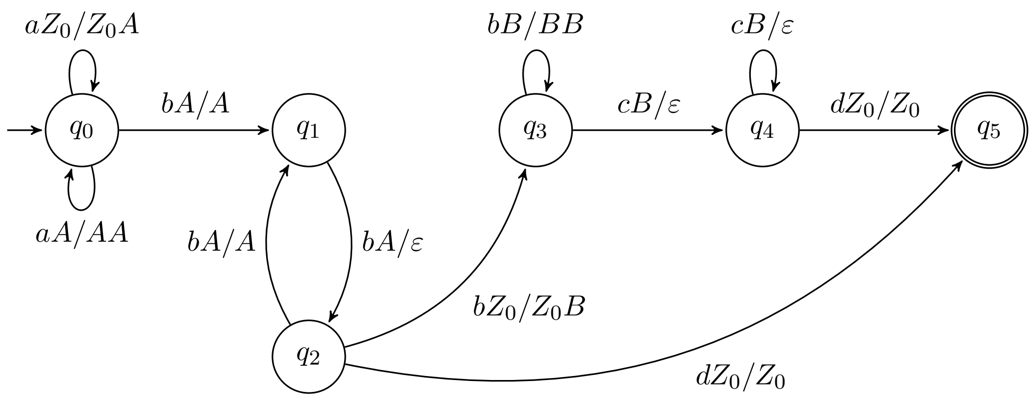
31 agosto 2022

Informatica teorica

Esercizio 1 (7 punti)
$L = \{a^n b^m c^o d;\ n, m, o \in \mathbb{N},\ n \geq 1, o \geq 0, m = 2n + o\}$. Utilizzare un formalismo a potenza minima (tra tutti quelli visti a lezione) che caratterizzi il linguaggio L.

Soluzione

Il linguaggio è libero dal contesto, riconoscibile da un automa a pila deterministico. Riscrivere la definizione come $L = \{a^n b^{2n} b^o c^o d;\ n, o \in \mathbb{N}, n \geq 1\}$ rende evidente la natura del linguaggio. Per riconoscerlo è sufficiente impilare un simbolo per ogni a, spilarne uno ogni due b, fino a quando la pila è vuota. Per le b successive, impilare un simbolo per ogni b e spilare un simbolo per ogni c effettua il conteggio del valore o, al termine del quale è sufficiente riconoscere la presenza della singola d.



Esercizio 2 (9 punti). Chi ha diritto alla riduzione del 30% della prova svolga unicamente il punto 1.

1. È possibile determinare se una generica macchina di Turing universale si arresta per ogni ingresso?
2. È possibile trovare, data la terza (secondo l'enumerazione di Gödel) macchina di Turing universale M, almeno un ingresso per cui essa non si arresta?
3. È possibile, data la macchina di Turing universale M di cui sopra, dire se essa si arresta per un generico ingresso n?

Soluzione

1. Sì: è possibile infatti stabilire che una qualunque MTU non si arresta per qualche input. Una MTU è in grado di emulare il comportamento di qualunque altra MT, il cui comportamento è codificato in parte del suo nastro di ingresso. Esiste quindi almeno un ingresso n che corrisponde a una MT che non si arresta per alcun input. Di conseguenza, non è vero che la MTU si arresta per ogni input fornito: essa infatti non si arresta almeno sull'input n.

2. Sì, è sufficiente considerare l'ingresso corrispondente alla codifica del comportamento di una MT che va sempre in loop.

3. Intuitivamente: no, l'ingresso n rappresenta il comportamento di una generica MT di cui servirebbe determinare se si arresta o meno a fronte di un generico ingresso. Più formalmente, è possibile ridurre la soluzione dell'halting problem alla soluzione del problema in questione. Si supponga che esista un algoritmo $\mathcal{A}(n)$, che riceve in ingresso una stringa n corrispondente all'input della MTU M e determina se M si arresta quando eseguita con input n. Data infatti una generica MT M_y ed un generico input x di cui si vuole determinare la terminazione, è sufficiente codificare la coppia $\langle M_y, x\rangle$ nel formato accettato da M, ottenendo la stringa $\bar{n}$. A questo punto, invocando $\mathcal{A}(\bar{n})$ sarebbe possibile risolvere l'halting problem per la generica istanza $M_y(x)$, il che è assurdo, data l'indecidibilità dello stesso.

Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

31 agosto 2022

Algoritmi e strutture dati

Esercizio 3 (8 punti)

Si abbiano due numeri interi x e y, con $x < y$. Si consideri il problema di partizionare una lista semplice, cioè con il solo puntatore al nodo successivo, in tre liste contenenti valori rispettivamente minori di x, compresi tra x e y, e maggiori di y. Si scriva lo pseudo-codice della procedura richiesta e se ne valutino le complessità spaziale e temporale.

Soluzione

```
tri-part(x, y, list)
    left := center := right := NIL
    while list != NIL
        if list.key < x then
            left := node(list.key, left)
        if list.key > y then
            right := node(list.key, right)
        else
            center := node(list.key, center)
        list := list.next
    return (left, center, right)
```

Le complessità richieste sono entrambe $\Theta(n)$, con n lunghezza della lista, essendoci un'unica scansione della stessa con copia dei valori contenuti.

Esercizio 4 (8 punti). Chi ha diritto alla riduzione del 30% della prova svolga unicamente il punto 1.

Si consideri la seguente funzione `g`, che riceve un array di interi `a` e restituisce un intero:

```
g(a)
    if a.length ≤ 1
        return 1
    k := g(a[1..n/2])
    j := g(a[n/2+1..n])
    for h := 1 to a.length
        i := 1
        while i ≤ a.length
            i := i * 2
            k := k + i * a[h]
            j := j - i * a[h]
```

```
    return (k + j) / 2
```

Detta n la lunghezza dell'array `a`,

1. si scriva la ricorrenza associata al codice della funzione;
2. si fornisca un limite asintotico superiore (possibilmente stretto) per la complessità temporale di `g(a)` in funzione di n.

Soluzione

La ricorrenza associata al codice della funzione è:

$$T(n) = \begin{cases} \Theta(1) \text{ se } n \leq 1 \\ 2T(n/2) + \Theta(n \log(n)) \text{ se } n > 1 \end{cases} \qquad \text{o anche } T(n) = \begin{cases} 1 \text{ se } n \leq 1 \\ 2T(n/2) + n \log_2(n) \text{ se } n > 1 \end{cases}$$

La funzione $f(n) = n \log_2(n)$ cresce più velocemente di $\Theta(n^{\log_2 2}) = \Theta(n)$, ma non polinomialmente più velocemente. Pertanto non si può applicare il teorema dell'esperto.

Osserviamo che $T(n)$ cresce più velocemente di $T'(n) = 2T'(n/2) + n$ ma meno di $T''(n) = 2T''(n/2) + n^{1+\epsilon}$, per ogni $\epsilon > 0$ (le relazioni $T(n) \geq T'(n)$ e $T(n) \leq T''(n)$ si mostrano per banale induzione). Pertanto $T(n) \in \Omega(n \log n)$ e $T(n) \in O(n^{1+\epsilon})$. Intuitivamente, ha quindi senso ipotizzare una complessità che sia più alta di $\Theta(n \log n)$, ma non polinomialmente più alta.

Formuliamo allora l'ipotesi che $T(n) \leq cn(\log_2 n)^2$ e procediamo per sostituzione.

$$\begin{aligned}
T(n/2) &\leq cn/2(\log_2(n/2))^2 && \text{Ipotesi induttiva} \\
T(n) &\leq cn(\log_2(n/2))^2 + n \log_2(n) && \text{Sostituzione} \\
cn(\log_2(n/2))^2 + n \log_2(n) &\leq cn(\log_2 n)^2 && \text{Da verificare asintoticamente} \\
c(\log_2(n/2))^2 + \log_2(n) &\leq c(\log_2 n)^2 && (n > 0) \\
c((\log_2 n)^2 - (\log_2(n/2))^2) &\geq \log_2(n) \\
c((\log_2 n)^2 - (\log_2 n - 1)^2) &\geq \log_2(n) \\
c(2 \log_2 n - 1) &\geq \log_2(n) \\
c &\geq \frac{1}{2} + \frac{\frac{1}{2}}{2 \log_2 n - 1}
\end{aligned}$$

Quest'ultima relazione è sempre soddisfatta per $n > 1$ e $c \geq 1$.

Nel caso base, $c1(\log_2(1))^2 = 0 \not\geq T(1) = 1$. Tramite la ricorrenza otteniamo $T(2) = 2T(1) + 2\log_2(2) = 4$ e $T(3) = 2T(1) + 3\log_2(3) = 2 + 3\log_2(3)$; inoltre $4 \leq c2(\log_2 2)^2$ e $2 + 3\log_2(3) \leq c3(\log_2 3)^2$ valgono entrambe per $c \geq 2$ e per $n > 3$ la ricorrenza non dipende più da $T(1)$. La relazione è soddisfatta anche nel caso base e quindi $T(n) = O(n(\log n)^2)$.

Si dimostra analogamente che $T(n) \geq dn(\log_2 n)^2$. Questa relazione è soddisfatta se

$$d \leq \frac{1}{2} + \frac{\frac{1}{2}}{2 \log_2 n - 1},$$

il che vale, ad esempio, per $d \leq 1/2$. Nel caso base, $T(1) = 1 \geq d1(\log_2(1))^2 = 0$. Quindi $T(n) = \Omega(n(\log n)^2)$.

Allora complessivamente $T(n) = \Theta(n(\log n)^2)$.

Un altro modo per risolvere la ricorrenza consiste nell'effettuare un cambio di variabile:

$$
\begin{aligned}
T(2^m) &= 2T(2^{m-1}) + 2^m m && \text{Poniamo } m = \log_2 n \text{ e quindi } n = 2^m \\
T(2^m)/2^m &= 2T(2^{m-1})/2^m + m && \text{Dividiamo per } 2^m \\
S(m) &= 2T(2^{m-1})/2^m + m && \text{Definiamo } S(m) = T(2^m)/2^m \\
S(m) &= S(m-1) + m && \text{Quindi } S(m-1) = 2T(2^{m-1})/2^m \\
S(m) &= S(m-2) + m - 1 + m && \text{Procedo per sostituzioni successive} \\
&\vdots \\
S(m) &= S(0) + 1 + 2 + \ldots + m - 1 + m && \text{Con } S(0) = T(2^0)/2^0 = 1 \\
S(m) &= \Theta(m^2) \\
T(2^m)/2^m &= \Theta(m^2) \\
T(n)/n &= \Theta((\log n)^2) \\
T(n) &= \Theta(n(\log n)^2)
\end{aligned}
$$