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

7 luglio 2023

Informatica teorica

Esercizio 1 (8 punti)
Si consideri il linguaggio $L = \{uww^Rv \mid u, v, w \in \{a, b\}^+\}$.

1. Utilizzare un formalismo a potenza minima (tra tutti quelli visti a lezione) che caratterizzi il linguaggio L.
2. Cambierebbe il formalismo a potenza minima se il linguaggio fosse $L' = \{ww^Rv \mid v, w \in \{a, b\}^+\}$?

Soluzione

1. Tutte le stringhe generate quando $|w| > 1$ fanno già parte del caso $|w| = 1$, che pertanto è l'unico che occorra considerare. Il linguaggio è quindi $L = \{uwwv \mid u, v \in \{a, b\}^+ \text{ e } w \in \{a, b\}\}$, che è chiaramente regolare, in quanto esprimibile mediante l'espressione regolare $(a|b)^+(aa|bb)(a|b)^+$. Il formalismo a potenza minima per esprimere L è quello della logica monadica del prim'ordine (MFO). Una formula MFO che lo esprima è la seguente:
$$\exists x(x > 0 \wedge a(x) \wedge \exists y(succ(x, y) \wedge a(y) \wedge \neg last(y))) \vee$$
$$\exists x(x > 0 \wedge b(x) \wedge \exists y(succ(x, y) \wedge b(y) \wedge \neg last(y))).$$
La stessa formula, riscritta con il consueto zucchero sintattico, si semplifica come segue:
$$\exists x(x > 0 \wedge (a(x) \wedge a(x + 1) \vee b(x) \wedge b(x + 1)) \wedge \neg last(x + 1)).$$
2. Nel caso di L' non sarebbe più vero che le stringhe generate quando $|w| > 1$ sono comprese nel caso $|w| = 1$ ed è anzi richiesta la capacità di verificare la simmetria tra w e w^R, con la difficoltà aggiuntiva di non conoscere il punto in cui si passa da w a w^R. Occorrerebbe quindi un automa a pila nondeterministico.

Esercizio 2 (8 punti)

Si considerino le seguenti funzioni e si dica se sono calcolabili, motivando esaurientemente la risposta:

1. $h(x) = 1$ se $f_x(x) > x$, $h(x) = 0$ altrimenti;
2. $i(x, y, z) = f_y(x) + f_z(x)$.

Soluzione

1. No. Definiamo la funzione $g(x) = x + 1$ se $h(x) = 0$, $g(x) = \perp$ altrimenti. Se h fosse calcolabile, lo sarebbe anche g. Sia i l'indice di una MT che calcola g. Abbiamo due possibilità:
   (a) $g(i) = i + 1$, ma questo vuol dire $f_i(i) = g(i) \leq i$ oppure $f_i(i) = g(i) = \perp$;
   (b) $g(i) = \perp$, ma questo vuol dire $f_i(i) = g(i) > i$. Assurdo.
2. Calcolabile: basta costruire M_y e M_z con $\mathcal{E}$ e poi simularle in serie, passando x a entrambe. Se terminano entrambe, si deve restituire la somma dei risultati.

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

7 luglio 2023

Algoritmi e strutture dati

Esercizio 3 (8 punti)

Si consideri il seguente problema di traduzione: data una stringa $a^n b^m; n, m \in \mathbb{N}$, si vuole emettere come traduzione una stringa con le seguenti caratteristiche:

- ha lo stesso numero di a e b di quella in ingresso
- le a e le b compaiono in alternanza, a partire dall'inizio, fino a quando è possibile dato il numero di a e b.

A titolo di esempio, la traduzione di *aaabbb* è *ababab*, quella di *aaab* è *abaa*, quella di *abbb* è *abbb* stessa. Si descriva sinteticamente, ma con precisione, la procedura di traduzione a complessità temporale minima ottenibile utilizzando:

1. una macchina di Turing a k nastri,
2. una macchina di Turing a nastro singolo.

Soluzione

1. È sufficiente una MT a $k = 2$ nastri per ottenere la complessità temporale minima, ovvero $\mathcal{O}(n)$, con n lunghezza della stringa in ingresso. La MT opera nel seguente modo: scandisce la stringa in ingresso, utilizzando i due nastri come contatori, in unario, del numero di a e b. Fatto ciò, ripete la sequenza: stampa a, decrementa il contatore di a, stampa b, decrementa il contatore di b fino a quando entrambi i contatori non sono nulli. Stampa in seguito la quantità di lettere rimanente nell'eventuale contatore non nullo rimasto.
2. La MT a nastro singolo scorre la stringa sino all'inizio della porzione di b, ed inserisce un simbolo c prima dell'inizio di essa, eventualmente facendo scorrere in avanti le b di una posizione. A questo punto fino a quando: i) le b dopo la c non sono terminate oppure ii) è presente una b appena prima della c ripete il seguente procedimento: scorre la stringa fino alla fine, cancella una b, torna indietro fino ad incontrare la prima b prima della c, si sposta sulla a successiva, inserisce una b dopo di essa facendo scorrere in avanti tutto il resto della stringa. Alla fine di questo procedimento, la c viene cancellata.

   Il procedimento ha complessità temporale $\mathcal{O}(n^2)$, nel caso pessimo in cui l'input sia $a^n b^n$.

Esercizio 4 (8 punti)

Si consideri come struttura dati una lista semplice L, realizzata con nodi contenenti i soliti campi *L.value* e *L.next*. Una *lista a livelli multipli* è una lista dove il campo *value* di un nodo può o contenere un valore oppure puntare ad un altro nodo; questo tipo di lista viene usata tipicamente in linguaggi a tipizzazione dinamica, come Python o JavaScript.

Per capire come avviene la memorizzazione, si usi *Node(x,y)* per rappresentare $x = L.value$ e $y = L.next$. La lista [1,2,3] è dunque memorizzata come *Node(1,Node(2,Node(3,NIL)))*, mentre una lista a due livelli come [1,2,[3,4],5] viene memorizzata in questo modo: *Node(1,Node(2,Node(Node(3,Node(4,NIL)),Node(5,NIL))))*.

Si consideri il problema di appiattire una lista a livelli multipli: per es. dalla lista a tre livelli [1,2,[3,4,[5]],6,[7,8],9] si vuole ottenere [1,2,3,4,5,6,7,8,9]. Si scriva lo pseudo-codice di una procedura che risolva il problema, valutandone la complessità temporale. Si assuma di avere a disposizione una procedura IsNode(x) che restuisce *vero* se x è un nodo, *falso* altrimenti.

Soluzione

La seguente procedura risolve il problema, usando una lista r e un puntatore all'ultimo elemento di r, *tail*.

```
flatten(L)
  r := NIL
  tail := r
  flat(x)
    if x = NIL then return
    if IsNode(x) then
      flat(x.value)
      flat(x.next)
    else
      n := Node(x, NIL)
      if r = NIL then r := n; tail := n
      else tail.next := n; tail := n
  end
  flat(L)
  return r
end
```

Per analizzare la complessità, notiamo per prima cosa come la struttura dati di ingresso sia sostanzialmente analoga ad un albero binario: infatti dato $n = Node(x, y)$ ci sono tre possibilità: o x e y puntano ad altri nodi, quindi n è un nodo interno con 2 figli; oppure x è un valore e y punta ad un altro nodo, quindi n ha un solo figlio; oppure x è un valore e $y = \text{NIL}$, in questo caso n è una foglia. L'algoritmo in pratica effettua una visita dell'albero, aggiungendo i valori trovati in coda alla lista r, con complessità $T(n) = \Theta(n)$.