Algoritmi e Principi dell'Informatica

Appello del 18 Settembre 2014

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1, 2, 3 in 1 ora e 15 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 4, 5 e 6 in 1 ora e 15 minuti.

NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (punti 5)

Si consideri il linguaggio L delle stringhe a^nb^n, $a^nb^na^nb^n$, ... dove a^nb^n può essere ripetuto 1 o più volte, con $0 < n < 4$.
Esempi di stringhe in L sono: aabb, abab. Esempi di stringhe non in L sono invece: abb, aabbab.
Si scrivano un automa e/o una grammatica a potenza minima per L e per $L1 = L \cap \{(a^mb^m)^+, m > 0\}$.
NB: è preferita una soluzione che presenti sia una grammatica che un automa ma è accettabile anche uno solo dei due formalismi.

Esercizio 2 (punti 8)

Punto a.

Specificare in logica del prim'ordine il predicato binario substring(x,y) che indica che il primo argomento x è una sottostringa del secondo argomento y, ossia una sequenza di caratteri *consecutivi* che compare in y. NB: come caso particolare la stringa nulla è sottostringa di qualsiasi stringa.

Ad esempio

substring(ac,abcd) è falso

substring(bc,abcd) è vero

Per la specifica, oltre al predicato di uguaglianza, si può usare *solo* la funzione binaria concat (concatenazione tra stringhe), che restituisce la concatenazione dei suoi argomenti.

Ad esempio: concat(ab,cd) restituisce abcd

Punto b.

Si specifichi poi il predicato substring2(x,y,i,j) che indica che x è uguale alla sottostringa di y che va dal carattere in posizione i di y al carattere in posizione j ≥ i di y (estremi inclusi). Si noti che per una stringa di lunghezza n, le posizioni dei suoi caratteri vanno da 1 a n; si noti anche che questa definizione implica che x non sia la stringa nulla.

Per la specifica, in aggiunta a quanto consentito al punto a, si possono usare le 4 operazioni aritmetiche, le costanti intere e la funzione unaria len che restituisce la lunghezza di una stringa.

Ad esempio: len(abcd) restituisce 4

Si può evitare, in quanto sottinteso, l'uso di predicati string(x) e integer(i) per specificare che una variabile rappresenta una stringa o un intero, rispettivamente; di conseguenza non è necessario specificare esplicitamente domini e codomini delle funzioni utilizzate.

Punto c.

Infine, osservando le restrizioni imposte ai punti precedenti, ossia utilizzando esclusivamente predicati e funzioni ivi definiti, si specifichi il predicato binario reverse(x,y) che indica che la stringa x, letta da sinistra a destra, è uguale alla stringa y letta da destra a sinistra.

Ad esempio:

reverse(abcd,dcba) è vero

reverse(abcd,cba) è falso

Se utile, è possibile usare anche i predicati definiti ai punti precedenti.

Esercizio 3 (punti 3)

E' decidibile il problema di stabilire se, date due qualunque stringhe ed eventualmente due interi, essi soddisfino i predicati definiti nell'Esercizio 2?

Esercizio 4 (punti 4)

Qual è la complessità migliore ottenibile da MT *a nastro singolo* che riconoscano, rispettivamente, i linguaggi L e L1 di cui all'esercizio 1? Giustificare brevemente la risposta.

Esercizio 5 (punti 4)

Si risolva la seguente equazione alle ricorrenze:

$T(n) = T(n-1) + n^2$

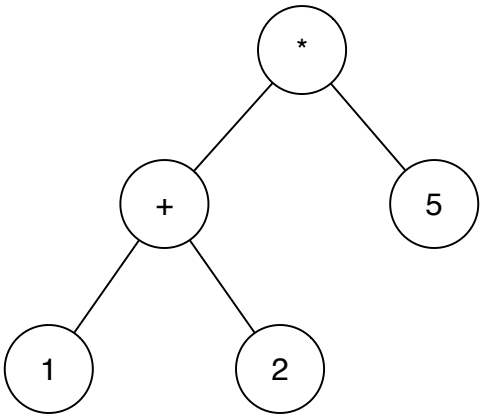
Esercizio 6 (punti 7)

Scrivere lo pseudocodice e calcolare la complessità temporale di una funzione che, dato un albero binario che rappresenta una espressione aritmetica ne calcoli il valore e lo restituisca in output.

Nota: la radice ed i nodi intermedi rappresentano gli operatori, le foglie dell'albero rappresentano gli operandi e la loro chiave ne contiene il valore. Le parentesi non sono rappresentate.

Si assuma che l'espressione contenga solo gli operatori aritmetici binari: +, -, *, / e numeri interi.

Ad esempio l'espressione ((1+2)*5) è rappresentata dall'albero in figura



Tracce delle soluzioni

Esercizio 1

L è un linguaggio regolare perché n può assumere solo i valori 1, 2, 3 e poi può seguire un'altra stringa con le stesse caratteristiche; non è quindi necessaria una capacità di conteggio illimitata. Una grammatica regolare che genera L è la seguente, facilmente convertibile in un automa a stati finiti:

```
S -> a A1 | a B1 | a C1
A1 -> b A2 | b
A2 -> a A1
B1 -> a B2
B2 -> b B3
B3 -> b B4 | b
B4 -> a B1
C1 -> a C2
C2 -> a C3
C3 -> b C4
C4 -> b C5
C5 -> b C6 | b
C6 -> a C1
```

L1 coincide con L perché $\{(a^m b^m)^+, m > 0\}$ evidentemente contiene L; quindi la stessa grammatica e lo stesso automa definiscono L e L1.

Esercizio 2

Punto a.

$\forall x \forall y$ substring(x,y) $\leftrightarrow \exists p \exists q \exists r$ (r = concat(p,x) $\wedge$ y = concat(r,q))

Punto b.

$\forall x \forall y \forall i \forall j$, substring2(x,y,i,j) $\leftrightarrow$

$\exists p \exists q \exists r$ (r = concat(p,x) $\wedge$ y = concat(r,q) $\wedge$ len(p) = i-1 $\wedge$ len(x) = j-i+1)

Punto c.

$\forall x \forall y$ reverse(x,y) $\leftrightarrow$

(len(x) = len(y) = 0) /* x e y sono entrambe la stringa nulla*/ $\vee$

($\forall c \forall i$ substring2(c,x,i,i) $\rightarrow$ substring2(c,y,len(y)-i+1,len(y)-i+1)) $\wedge$

($\forall c \forall i$ substring2(c,y,i,i) $\rightarrow$ substring2(c,x,len(x)-i+1,len(x)-i+1)))

Una formula alternativa ed equivalente è la seguente:

$\forall x \forall y$ reverse(x,y) $\leftrightarrow$

(len(x) = len(y) = 0) /* x e y sono entrambe la stringa nulla*/ $\vee$

len(y) = len(x) = 1 ∧ x = y

∨

(∃z, ∃w, ∃r, ∃q, (x = concat(z,w) ∧ (y = concat(r,q)) ∧

(len(z) = len(q) ≥ 1∧ len(w) = len(r) ≥ 1) ∧

(reverse(z,q) ∧ reverse(w,r)))

Esercizio 3

Evidentemente SI: è molto facile scrivere algoritmi che stabiliscano rispettivamente se due stringhe sono una sottostringa dell'altra, se una stringa è la concatenazione di altre due, o calcolino la concatenazione tra due stringhe, ecc.

Esercizio 4

Una MT a nastro singolo estende l'automa a stati finiti che costituisce il suo organo di controllo, quindi può eseguire esattamente le mosse che esegue un automa a stati finiti. Essendo L1 = L un linguaggio regolare, anche la complessità di un MT a nastro singolo che lo riconosca sarà O(n), per la precisione $T_M(n) = n+1$.

Esercizio 5

E' facile sviluppare T(n) in $n^2 + (n-1)^2 + (n-2)^2 + (n-3)^2 \ldots + c$

ossia $T(n) = \sum_{i=1}^{n} i^2$ che è $\Theta(n^3)$, infatti $\sum_{i=1}^{n} i^2 \leq n^3$ e $\sum_{i=1}^{n} i^2 \geq \left(\frac{n}{2}\right)^2 \left(\frac{n}{2}\right) = \frac{n^3}{8}$.

Esercizio 6

calcola-valore (tree t)

{

- **if** t è una foglia **return** key (t)
- **if** key(t) = "+" **return** (calcola-valore (left(t)) + calcola valore (right(t))) /*nodo intermedio: operatore +*/
- **if** key(t) = "- "**return** (calcola-valore (left(t)) - calcola valore (right(t))) /*nodo intermedio: operatore - */
- **if** key(t) = "*" **return** (calcola-valore (left(t)) * calcola valore (right(t))) /* nodo intermedio: operatore * */
- **if** e key(t) = "/ "**return** (calcola-valore (left(t)) / calcola valore (right(t))) /*nodo intermedio: operatore /*/

else return errore "espressione mal formata"

}

Complessità: $\Theta(n)$, con n numero dei nodi, visto che è una visita dell'albero.