B

Fondamenti dell'Informatica

A.A. 2003/2004

Corso A

Prima prova di esonero: 26/4/2005 ore 8.30 – 11.00

1. Definire una macchina di Turing deterministica *M* a nastro singolo e i concetti di configurazione e di transizione. Sintetizzare una macchina di Turing trasduttore che trovi il minimo di due numeri rappresentati in notazione binaria. Si supponga che i numeri abbiano lo stesso numero di cifre.

   Esempio: $q_0$11100001#11001000 $\vdash^*_M$ 11100001#11001000bq_F 11001000

   con q_0 stato iniziale e q_F stato finale.

   Si rappresenti la funzione di transizione mediante una matrice di transizione. Si specifichi per ogni stato qual è la funzione da esso svolta. (*7 punti*)

   Svolgimento simile a quello della traccia A.

2. Dare la definizione di macchina di Turing multinastro (MTM), di configurazione istantanea e di transizione. Riportare un esempio di MTM riconoscitore con almeno 2 nastri e mostrare due computazioni una relativa all'accettazione di una stringa e una relativa al rifiuto di una stringa. (*4 punti*). In generale, potresti simulare il comportamento di una MTM con una deterministica a nastro singolo? Come? E quanto costerebbe la simulazione? (*2 punti*).

3. Definire un programma RAM per il calcolo di

   $gauss(n) = \sum_{k=0}^{n} k$ con $n \geq 0$.

   Valutare la complessità del programma rispetto a un modello di costo logaritmico. [Suggerimento: al solo fine di valutare la complessità si tenga conto del seguente risultato matematico $gauss(n) = \frac{n(n+1)}{2}$] (*7 punti*)

   Svolgimento

   Si utilizzano 4 registri:

1	*2*	*3*
n	i=*n,n-1, …, 2, 1*	∑i

	READ 1	l(1)+l(n)	O(log *n*)	1	O(log *n*)
	LOAD #0	l(0)	O(1)	1	O(1)
	STORE 3	l(3)+l(0)	O(1)	1	O(1)
	LOAD 1	l(2)+l(n)	O(log *n*)	1	O(log *n*)
5	STORE 2	l(2)+l(i)	O(log *n*)	n+1	O(*n*log *n*)
	JZERO 13	l(n)	O(log *n*)	n+1	O(*n*log *n*)
	LOAD 2	l(2)+l(i)	O(log *n*)	n	O(*n*log *n*)
	ADD 3	l(3)+l(i)+l(∑i)	O(log *n*)	n	O(*n*log *n*)
	STORE 3	l(3)+l(∑i)	O(log *n*)	n	O(*n*log *n*)
	LOAD 2	l(2)+l(i)	O(log *n*)	n	O(*n*log *n*)
	SUB #1	l(1)+l(i)	O(log *n*)	n	O(*n*log *n*)
	JUMP 5	l(5)	O(1)	n	O(*n*)
13	WRITE 3	l(3)+ l(∑n^i)	O(log *n*)	1	O(log *n*)
	HALT	1	O(1)	1	O(1)

Nello svolgimento si è tenuto conto di quanto suggerito, cioè che

$$gauss(n) = \frac{n(n+1)}{2}$$

da cui discende che:

$$\log \sum_{k=1}^{n} k = \log \frac{n(n+1)}{2} = \log(n^2 - n) - \log 2 = O(\log n^2) = O(2 \log n) = O(\log n)$$

Quindi la complessità del calcolo della sommatoria è: O(*n*log *n*).

4. Dimostrare che la seguente sommatoria

   $\lambda x \lambda y.1+x+x^2+x^3+\ldots+x^y$ con $x>0$, $y \geq 0$

   è ricorsiva primitiva (*4 punti*).

Svolgimento

Detta *Spot(x,y)* la suddetta sommatoria, osserviamo che :

Spot(x,0) = 1

Spot(x,y+1) = Spot(x,y) + x^{y+1}

pertanto :

Spot(x,0)=S(0^0)

Spot(x,y+1)=h(x,y,z)=somma(U^3_3(x,y,z), potenza(U^3_1(x,y,z), S(U^3_2(x,y,z))))

dove *potenza* e *somma* sono le funzioni ricorsive primitive definite a lezione. Quindi *Spot(x,y)* è ricorsiva primitiva perché definita mediante lo schema di ricorsione primitiva a partire da funzioni ricorsive primitive (potenza, somma e quelle di base).

5. Dimostrare che il linguaggio SLF consente la definizione di tutte le funzioni ricorsive (*4 punti*).
6. Dimostrare a grandi linee che data una qualunque funzione ricorsiva è possibile definire un programma RAM π che la calcola. (*9 punti*) [1]

Svolgimento

Mediante un linguaggio di programmazione ad alto livello (e.g., C) è possibile scrivere una qualunque funzione ricorsiva *f*.

Infatti, se *f* è una funzione di base, abbiamo sicuramente delle funzioni nel linguaggio C che le calcolano (si pensi all'operatore ++ per il cancolo del successore).

Se *f* è definita mediante lo schema di composizione, anch'essa è esprimibile mediante un programma C perché in C possiamo richiamare una funzione all'interno di altre.

Se *f* è definita mediante lo schema di ricorsione primitiva:

$f(x_1, \ldots, x_n,y) = g(x_1, \ldots, x_n)$

$f(x_1, \ldots, x_n,y+1) = h(x_1, \ldots, x_n,y, f(x_1, \ldots, x_n))$

allora *f* è traducibile nel seguente modo:

```
int f(int x1, ..., int xn, int y) {
        if (y=0) return g(x1, ..., xn)
        else return h(x1, ..., xn,y, f(x1, ..., xn))
}
```

[1] La totalizzazione di un punteggio superiore a 30 punti equivale al *30 con lode*.

Se *f* è definite mediante lo schema di minimalizzazione:

$f(x_1, \ldots, x_n) = \mu t\ (g(x_1, \ldots, x_n, t) = 0)$

allora *f* è traducibile nel seguente modo:

```
int f(int x1, ..., int xn) {
        int i=0 ;
        while (!g(x1, ..., xn,t))  t++;
        return t
}
```

Avendo dimostrato che una qualunque funzione ricorsiva *f* può essere programmata in C è sufficiente osservare che il compilatore traduce il codice C in un codice macchina molto simile a quello di un programma RAM.