Informatica Teorica

Appello d'esame – 3 Febbraio 2010

Tempo a disposizione: 2h

Esercizio 1 (12 punti)

Si consideri la seguente formula in logica del I ordine

$\forall x\ (x \in L \Leftrightarrow \exists y\ (x = y b^n y \wedge y \in L' \wedge n > 0)$

Si descriva l'automa a potenza minima per riconoscere L nei seguenti casi

1) $L' = \{ x \in \{a,b\}^* \mid |x| = 3 \}$
2) $L' = \{a^m \mid m > 0\}$
3) $L' = \{b^m \mid m > 0\}$
4) $L' = \{ a^m b^m \mid m > 0\}$

Esercizio 2 (10 punti)

Considerato il seguente insieme di coppie di numeri naturali

$S = \{ \langle y, z \rangle \mid \exists x (f_y(x) \neq \perp \wedge f_z(x) \neq \perp \wedge f_y(x) = 2 * f_z(x)) \}$,

rispondere alle seguenti domande (motivando brevemente le risposte)

1. S è semidecidibile?
2. Il complemento di S è semidecidibile?

Esercizio 3 (12 punti)

Sia A un array contenente numeri naturali (eventualmente anche ripetuti). La seguente procedura serve a costruire un array contenente, per ogni possibile partizione di A in due (o *bi-partizione*), una coppia costituita dalla somma dei valori degli elementi di ognuna delle due partizioni.

Per esempio, sia A l'array [2,3,1,8,1]. Una possibile bi-partizione è [2,1,1] e [3, 8], che sarà memorizzata, nell'array restituito dalla procedura, come coppia <4, 11>.

```
procedure sub_sum(A)
      n := size of A
      if n <  2 then return the empty array
      if n = 2 then return an array containing the pair <A[0], A[1]>
      first := A[0]
      B := sub_sum(A[1..n])
      <x, y> := B[0]
      C := array containing <first, x+y>
      C1 := array containing <k+first, k'> for each <k, k'> in B
      C2 := array containing <k, first+k'> for each <k, k'> in B
      return the array C concatenated with C1 and with C2
```

Si consideri una ragionevole implementazione della procedura con la macchina RAM (senza effettuare la traduzione di sub_sum in codice RAM).

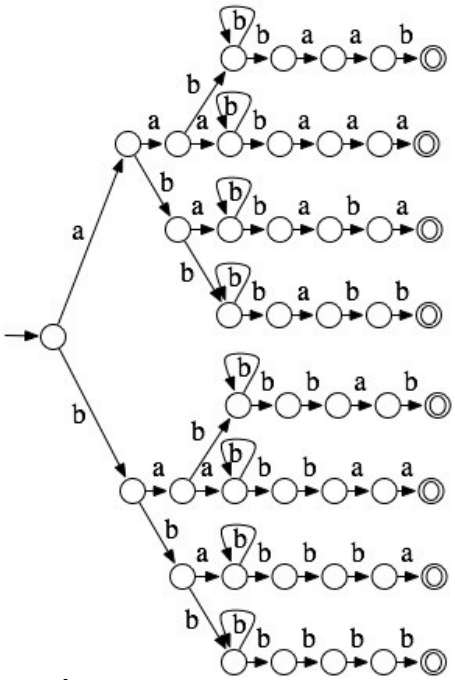
1) Si valutino la complessità spaziale e temporale di tale implementazione con il criterio del costo costante
2) Si valutino la complessità spaziale e temporale di tale implementazione con il criterio del costo logaritmico

Non è necessario dare la funzione esatta di complessità, ma è sufficiente ricavare la classe di complessità (le risposte devono essere adeguatamente motivate).

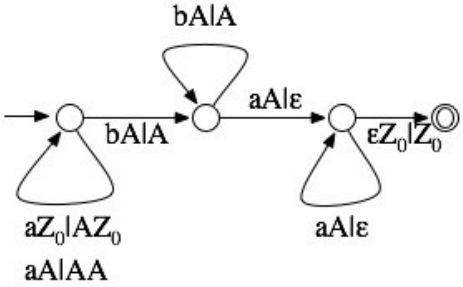
Soluzione

Esercizio 1

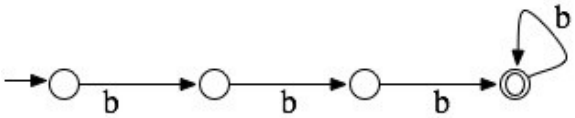
1.1 È sufficiente utilizzare un AF



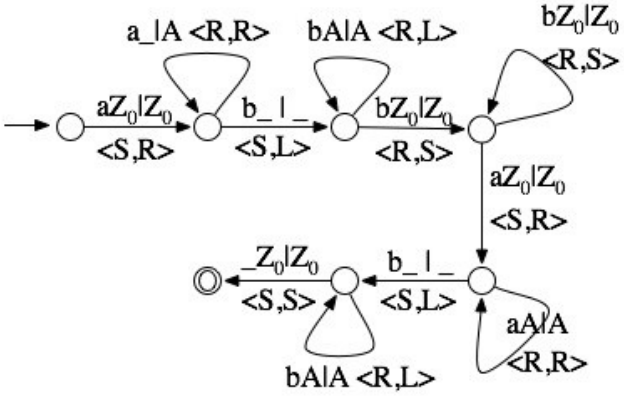
1.2 È necessario un AP che memorizzi il numero di ‘a’ in y per verificare che nella seconda occorrenza di y il numero di ‘a’ sia lo stesso che nella prima



1.3 Basta un AF che controlli che ci siano almeno 3 ‘b’



1.4 È necessaria una MT che memorizzi ‘m’



Esercizio 2

1. Si, infatti mediante una tipica enumerazione diagonale se esistono x, y, z tali per cui $f_y(x)\neq\perp \wedge\ f_z(x)\neq\perp \wedge f_y(x)=2*f_z(x)$ li trovo.
2. No. Il complemento di S, S', non è decidibile, quindi, essendo semidecidibile S, non è neanche semidecidibile. La non decidibilità di S' si può mostrare usando la riduzione e il teorema di Rice. S' è l'insieme

$$S'=\{<y, z> \mid \neg\exists x(f_y(x)\neq\perp \wedge\ f_z(x)\neq\perp \wedge f_y(x)=2*f_z(x)) \}$$
$$= \{<y, z> \mid \forall x\ (f_y(x)\neq\perp \wedge\ f_z(x)\neq\perp \rightarrow f_y(x)\neq 2*f_z(x)) \}$$

Consideriamo ora, ad esempio, una qualsiasi funzione totale e costante g(x) definita come $\forall x\ g(x)=1$, e sia h1 il suo numero di Goedel. Se S' fosse decidibile allora lo sarebbe anche l'insieme S1, derivato da S' prendendo z=h1,

$$S1= \{ y \mid \forall x\ (f_y(x)\neq\perp \rightarrow f_y(x)\neq 2) \}$$

cioè, l'insieme delle funzioni computabili che, dove sono definite, sono diverse da 2 sarebbe decidibile. Ma S1 non è decidibile per il Teorema di Rice, quindi non lo sarà neanche S'.

Esercizio 3

Sia n = |A|. Per prima cosa notiamo come il numero di elementi che saranno restituiti dalla procedura al termine della sua esecuzione sia $2^{(n-1)}$-1. Ad ogni chiamata ricorsiva, l'array che viene costruito per contenere la soluzione raddoppia come dimensioni (+ 1 elemento). La chiamata ricorsiva avviene su di un array (A[1..n]) che contiene un elemento in meno rispetto a quello originale, cioè il primo. Quindi si effettuano $\Theta(n)$ chiamate e, ad ogni chiamata, due cicli su di una struttura dati che raddoppia. Conseguentemente la complessità a tempo costante sarà $\Theta(2^n)$, come pure l'occupazione spaziale. Per quanto riguarda il criterio logaritmico, per prima cosa notiamo come il contenuto di A non cambi, come ordine di grandezza. Operare sui suoi dati costerà al più log m, dove m e` la somma di tutti gli elementi contenuti – dunque non cosidereremo nella valutazione questo aspetto, considerandolo una costante. Indirizzare l'array del risultato, contenente $\Theta(2^n)$ elementi, costa $\Theta(\log(2^n))$ cioè $\Theta(n)$. L'operazione di indirizzamento viene fatta su tutti gli elementi, dunque si ottiene $\Theta(n*2^n)$ per il tempo. La complessità spaziale è dominata dall'array che conterrà il risultato, dunque otteniamo $\Theta(2^n)$.