Informatica Teorica

Appello d'esame – 4 Febbraio 2011

Tempo a disposizione: 2h

Esercizio 1 (7 punti)

Si consideri il linguaggio L_p fatto di tutte e sole le stringhe sull'alfabeto {*a*, *b*} di lunghezza finita che, da un certo punto in poi, ripetono sempre gli stessi 3 caratteri. Tali 3 caratteri si devono ripetere almeno 2 volte.

Esempio di stringa che appartiene al linguaggio: *bbabaabaabaabaab*
Esempio di stringa che non appartiene al linguaggio: *bbabbaab*

Descrivere L_p mediante formule di logica del prim'ordine. Per descrivere il linguaggio si usi il predicato *car*(*x*,*i*,*c*) che è vero se e solo se in posizione *i* nella stringa *x* si trova il carattere *c*.

In alternativa, si scriva un automa che riconosce L_p. Tale automa deve essere è a potenza minima tra quelli che riconoscono il linguaggio L_p.

Esercizio 2 (12 punti)

Si consideri il problema di stabilire se, dato un generico sottoprogramma `P`, scritto in C, che ha un parametro `par` di tipo intero e ritorna un intero, e dato un valore in input `x` vale `P(x) = x`.
Sia data la seguente dimostrazione di indecidibilità del problema in questione, che fa uso della tecnica di riduzione:

> "A partire da un generico sottoprogramma `P` che ha un intero come parametro e restituisce un intero, costruiamo un programma `Pprime` che, a fronte di un input `x`, si comporta nel seguente modo:
>
> ```
> void Pprime(int x){
>     if (P(x) != x) {
>         while(1){printf("loop");}
>     }
> }
> ```
>
> Il sottoprogramma `Pprime` quindi termina su input `x` se e solo se `P(x) = x`. Poiché il problema di stabilire se un programma C termina a fronte di un generico input non è decidibile, allora non è decidibile neanche il problema di stabilire se `P(x) = x`."

1. La dimostrazione è corretta? Se no, spiegare perché è sbagliata.
2. Se la dimostrazione non è corretta, è decidibile il problema originario? Motivare adeguatamente la risposta.

Esercizio 3 (14 punti)

Si consideri il problema di calcolare, a partire da un valore *n* in input, il valore $n2^n$.

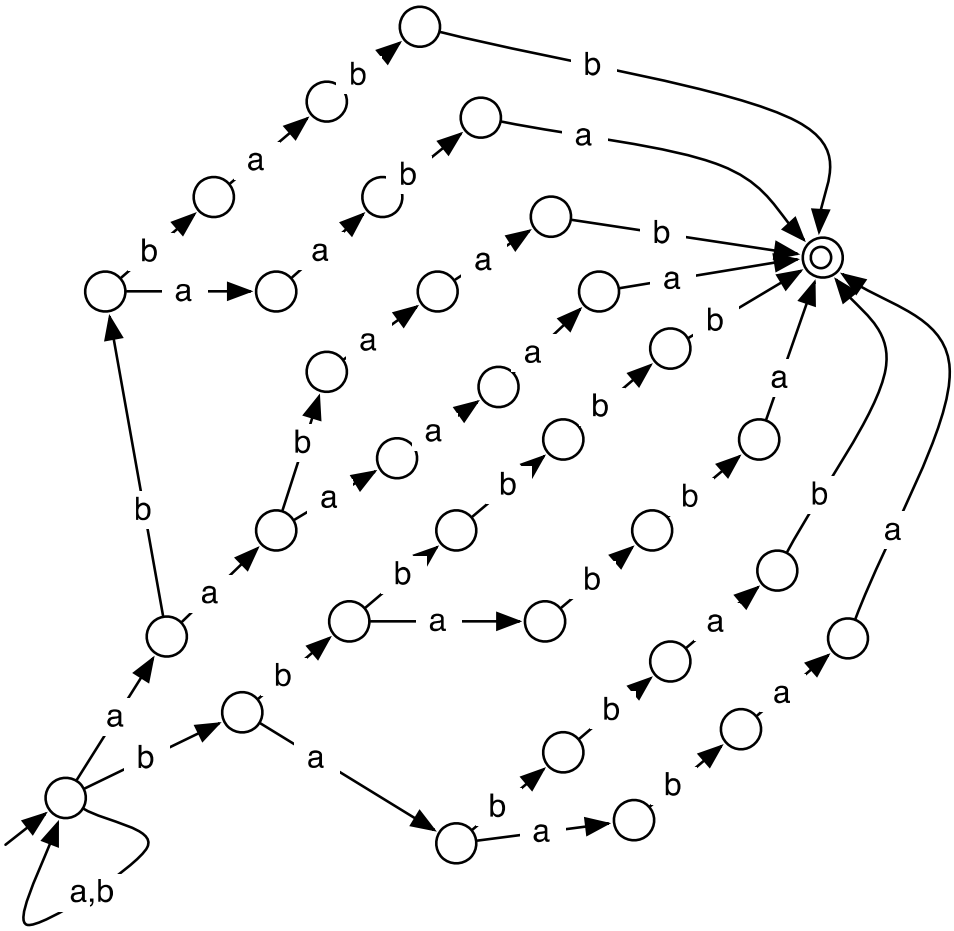
1. Si descriva in modo informale, ma sufficientemente preciso, una macchina di Turing a k nastri con alfabeto binario {0,1} (sia per il nastro di input che per i nastri di memoria) che calcola la funzione desiderata, supponendo che il valore *n* si trovi codificato in binario nel nastro di input. Si diano le complessità temporale e spaziale della macchina scritta.
2. Si descriva, in modo informale, ma sufficientemente preciso, una macchina di Turing a k nastri con alfabeto unario {A} (sia per il nastro di input che per i nastri di memoria) che calcola la funzione desiderata. Si diano le complessità temporale e spaziale della macchina scritta.
3. Si descriva (in pseudocodice) una macchina RAM che calcola la funzione desiderata. Si diano le complessità temporale e spaziale della macchina scritta, calcolate con criterio di costo logaritmico.

Quale dei 3 meccanismi proposti è il più efficiente per eseguire il calcolo in questione?

Soluzioni

Esercizio 1

Un automa (a stati finiti) che riconosce il linguaggio L_p è il seguente.



Tale automa è ovviamente a potenza minima tra quelli che riconoscono il linguaggio desiderato.

Lo stesso linguaggio può essere specificato mediante le seguenti formule logiche, che fanno uso del predicato *car*(*x*,*i*,*c*)

$$x \in L_p \Leftrightarrow \exists n\ (n \geq 6 \wedge \forall i\ (1 \leq i \leq n \Leftrightarrow \exists c\ (car(x, i, c))) \wedge$$
$$\forall i\ (n\text{-}2 \leq i \leq n \Rightarrow \forall c\ (car(x, i, c) \Leftrightarrow car(x, i\text{-}3, c))))$$

Esercizio 2

1.

La dimostrazione è sbagliata, in quanto applica la riduzione al contrario.
Per dimostrare tramite riduzione che un problema ignoto P è indecidibile, occorre prendere un problema noto (ed indecidibile) P' e ridurre P' a P.
La dimostrazione offerta fa esattamente il contrario: prende il problema ignoto P e riduce P ad un problema noto (ed indecidibile) P'.

2.

Il problema originario è comunque indecidibile. Ci sono diversi modi per mostrarne l'indecidibilità.

Uno è quello di fissare *x* (per esempio, 10), e considerare il caso particolare di decidere se, preso un generico programma P, P(10) = 10. Questo problema è indecidibile per il Teorema di Rice, quindi il suo caso più generale è anche indecidibile.

Un altro modo per dimostrare l'indecidibilità del problema originario è applicando (correttamente) la riduzione.
Dato un generico programma `Pprime` si può costruire il seguente programma `P`:

```
int P(int par){
  Pprime(par);
  return par;
}
```

Questo programma è tale che `P(x) = x` se e solo se `Pprime` termina la sua computazione sull'input *x*. Questo problema è indecidibile, quindi lo deve essere anche quello originario, o si ha una contraddizione.

Esercizio 3

1.

Per risolvere in modo efficiente il problema dato con una MT, basta considerare che, per numeri codificati in binario, moltiplicare per 2 corrisponde ad aggiungere uno 0 come cifra meno significativa. Per questo, per ottenere il valore $n2^n$ è sufficiente aggiungere *n* volte 0 alla fine della stringa binaria che rappresenta *n* (supponendo che la cifra più significativa sia a sinistra).
Quindi, una MT (traduttrice) che esegue il calcolo desiderato può avere 1 nastro di memoria e comportarsi come segue.

- Innanzi tutto copia *n* (in binario) sul nastro T_1. Contemporaneamente scrive *n* (sempre in binario) in uscita.
- Con un ciclo, decrementa via via il contatore in T_1, e ad ogni decremento appende uno 0 in fondo al nastro di uscita.

Una tale macchina di Turing deve ripetere il ciclo *n* volte, ed ogni ripetizione costa log(*n*) per decrementare il contatore in T_1. In totale, quindi, la complessità temporale è $n \cdot \log(n)$, con *n* valore del numero in input. La complessità spaziale è invece log(*n*) (vengono contati solo i nastri di memoria).
Si noti che se, come è abitudine, per la MT consideriamo la lunghezza *m* della stringa in input come "dimensione del dato", essendo come noto $m = \log(n)$ (cioè $n = 2^m$) allora la complessità temporale diventa $m2^m$, e quella spaziale *m*.

2.

Se la MT ha solo un alfabeto unario, non è possibile memorizzare *n* in codifica binaria, e per rappresentare $n2^n$ servono esattamente $n2^n$ celle.
Per calcolare il valore desiderato, poi, è sufficiente mantenere un contatore (unario) nel nastro T_1, da inizializzare al valore *n*. Il nastro T_2 viene inizialmente inizializzato con un singolo simbolo A, che viene poi raddoppiato *n* volte per produrre 2^n (per effettuare il raddoppio si può usare un terzo nastro T_3, nel quale ogni volta si memorizza un numero doppio di A rispetto a quelle in T_2). Dopo che si è memorizzato in T_2 un numero 2^n di A, viene fatto un ciclo per cui, ad ogni iterazione, in output vengono ricopiate un numero *n* di A. Tale ciclo viene ripetuto tante volte quante sono le A su T_2, cioè 2^n volte, per ottenere il valore desiderato.
La complessità temporale di una simile MT è $n2^n$, e quella spaziale è 2^n (sempre perché vengono contati solo i nastri di memoria). Si noti che in questo caso il numero di celle della stringa in input coincide con il valore *n*, in quanto la codifica del numero è in unario.

3.

Il seguente pseudocodice risolve il problema in questione.

```
1 read(n)
2 ris := 1
3 for i := 1 to n
4   ris := ris*2
5 ris := n*ris
6 write(ris)
```

Il ciclo 3-4 viene eseguito *n* volte, e ad ogni iterazione *ris* vale 2^i. Il suo costo, a criterio di costo logaritmico, è quindi $\Sigma^n_{i=1} \log(2^i)$ cioè $\Sigma^n_{i=1} i$, che è $\Theta(n^2)$.

Dei 2 meccanismi di calcolo presentati, quindi, quello realizzato dalla MT è il più efficiente.
Tuttavia, è possibile scrivere una macchina RAM che realizza il calcolo desiderato, ma con complessità migliore.
L'algoritmo è il seguente

```
1  read(n)
2  m := n
3  ris := 1
4  temp := 1
5  while m > 0
6    if m mod 2 = 1
7      ris := ris * temp
8    temp := temp * temp
9    m := m div 2
10 ris := ris*n
11 write(ris)
```

In questo caso il ciclo 5-9 viene eseguito solo log(*n*) volte, e l'operazione più onerosa del ciclo è la 7 (che nel caso pessimo, quello in cui *n* sia una potenza di 2 meno 1, per esempio 15 viene eseguita ad ogni iterazione del ciclo), che costa $\log(2^{2^i} \cdot 2^{2^i})$, cioè $\log(2^{2^{i+1}})$, cioè 2^{i+1}. Quindi, il costo del ciclo 5-9 è $\Sigma^{\log n}_{i=1} 2^{(i+1)}$ che è $2^{(\log(n)+2)}$, che è $\Theta(n)$.
In realtà, con un meccanismo per certi versi analogo a quello descritto ora per la macchina RAM, è possibile ottenere una complessità $\Theta(n)$ anche per la MT.