Informatica Teorica

Prima prova in itinere - 3 Maggio 2010
Sezione Cremona+Como

Tempo a disposizione: 1h30

NB: il punteggio è espresso in 13-esimi e riflette il peso relativo della prova rispetto all'intero esame il cui punteggio è in 30-esimi. Come già in altre occasioni è tuttavia possibile ottenere un punteggio complessivo superiore a 13/13.

Esercizio 1 (7/13 punti)

1)
Si formalizzi la nozione di automa a pila (deterministico e non) AP3, in cui la differenza rispetto agli AP normali è che ogni mossa può scrivere sulla pila *al massimo 3 simboli*.

2)
Dire, giustificando la risposta, quale è la potenza espressiva del modello AP3 rispetto agli AP abituali (dire se è equipotente, strettamente più potente, strettamente meno potente, o nessuno di questi).

3)
Cambia qualcosa, *dal punto di vista della potenza espressiva*, se ad ogni mossa l'automa può scrivere sulla pila al massimo 2 simboli?
E se l'automa ad ogni mossa può scrivere al massimo un solo simbolo?

Esercizio 2 (7/13 punti)

Si consideri la grammatica G:
$S \rightarrow A \mid C$
$A \rightarrow A \text{ x } A \mid C$
$C \rightarrow c\ A\ b \mid c\ C\ b \mid a$

1) Si dica quale è il linguaggio L(G) generato da G.

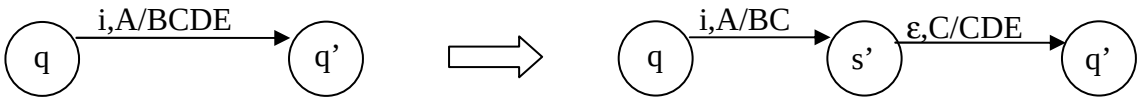
2) Se il L(G) è un linguaggio regolare, si scriva l'automa a stati finiti che lo riconosce; altrimenti si dimostri tramite il pumping lemma che L(G) non è regolare.

Soluzioni

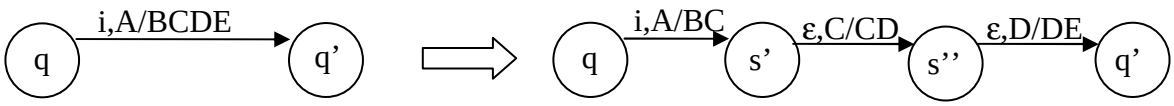
Esercizio 1

1)
L'unica cosa che cambia rispetto alla definizione degli AP usuali è la segnatura della funzione di transizione δ: $Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow (Q \times \cup^{3}_{i=0} \Gamma^{i})$ che, nel caso degli AP nondeterministici diventa δ: $Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow \wp(Q \times \cup^{3}_{i=0} \Gamma^{i})$ (si noti l'assenza del pedice F in $\wp$).
Il resto delle definizioni (configurazione, transizione tra configurazioni, accettazione, ecc.) rimane invariato.

2)
La potenza espressiva degli AP3 è analoga a quella degli AP. Infatti, gli AP3 sono banalmente un caso particolare degli AP3, quindi qualunque linguaggio riconosciuto da un AP3 è riconoscibile da un AP. Inoltre, dato un AP che abbia delle transizioni che scrivono su pila più di 3 caratteri sono banalmente trasformabili in AP3, come nel frammento seguente:



3)
Se invece che avere AP3 abbiamo AP2 (cioè automi che possono fare mosse che massimo scrivono 2 simboli sulla pila, dal punto di vista della potenza espressiva non cambia nulla, semplicemente, la trasformazione da AP in AP2 richiederà qualche stato intermedio in più, come nel caso seguente:



Le cose cambiano invece se l'automa può scrivere al massimo un solo simbolo sulla pila per ogni mossa. In questo caso il numero di simboli sulla pila non può crescere, quindi AP1 sono di fatto gli FSA, quindi strettamente meno potenti degli AP.

Esercizio 2

1)
Il linguaggio L(G) corrisponde alle espressioni ben parentetizzate (in cui i terminali *c* e *b* fungono da parentesi), con un operatore binario *x*, ed un letterale *a*.

2)
Il linguaggio non è regolare. Infatti, un sottoinsieme di questo linguaggio è dato dalle stringhe della forma $c^{n}ab^{n}$. Ora, un qualunque FSA A in grado di riconoscere tutte le stringhe della forma $c^{n}ab^{n}$, avrà un qualche insieme di stati Q di cardinalità |Q|. Per il pumping lemma, per un qualunque n>|Q| deve esistere una sottostringa *w* (non vuota) di $c^{n}ab^{n}$ leggendo la quale A esegue un ciclo.
Se $w = c^{k}$, A riconosce anche stringhe della forma $c^{n-k+kr}ab^{n}$, (con r≥0) che non appartengono a L(G). Dualmente se $w = b^{k}$.
Se invece $w = c^{k}ab^{t}$, allora A riconoscerà anche stringhe della forma $c^{n-k}(c^{k}ab^{t})^{r}b^{n-t}$ per un qualunque r≥0, che altrettanto non appartengono a L(G) (i casi in cui $w = ab^{t}$, $w = c^{k}a$, o *w* = a sono analoghi).
Di conseguenza, nessun FSA può esistere che riconosce tutte e sole le stringhe di L(G).

Informatica Teorica

Seconda prova in itinere - 28 Giugno 2010

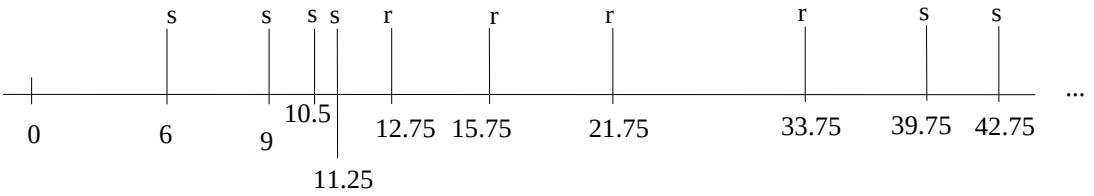
Tempo a disposizione: 2h

NB: il punteggio è espresso in 17-esimi e riflette il peso relativo della prova rispetto all'intero esame il cui punteggio è in 30-esimi. Come già in altre occasioni è tuttavia possibile ottenere un punteggio complessivo superiore a 17/17.

Esercizio 1 (6/17 punti)

Specificare mediante formule di logica del prim'ordine il comportamento di sequenze di segnali *s* e *r* descritte nel seguito.
Le sequenze iniziano con una serie di segnali *s* in cui la distanza tra un segnale *s* ed il successivo si dimezza, fino a che non diventa minore di 1; a quel punto inizia una sequenza di segnali *r*, nella quale la distanza tra due segnali *r* consecutivi si raddoppia, fino a che non diventa maggiore di 10; in seguito c'è una serie di segnali *s* simile alla precedente, a così via infinitamente al futuro.

La figura seguente mostra una esempio di sequenza di segnali.



Si noti che nella transizione da una serie di segnali *s* ad una serie di segnali *r* la distanza tra il primo *r* e l'ultimo *s* è due volte la distanza tra i due ultimi segnali *s*; in modo analogo, nella transizione da una serie di segnali *r* ad una serie di segnali *s* la distanza tra il primo *s* e l'ultimo *r* è la metà della distanza tra i due ultimi segnali *r*.
Si noti inoltre che i segnali *s* ed *r* sono mutuamente esclusivi, nel senso che durante una serie di emissioni di *s*, il segnale *r* non è emesso, e vice versa.

Per descrivere il comportamento di cui sopra si usino solo i predicati $s(t)$ e $r(t)$, in cui $s(t)$ (risp. $r(t)$) è vero esattamente in quegli istanti t in cui *s* (risp. *r*) è emesso.

Suggerimento: si scriva una formula logica per ognuno dei seguenti punti.

- Il segnale *s* è emesso la prima volta in un istante maggiore di 1. La seconda emissione del segnale *s* accade ad una distanza dalla prima che è la metà della distanza tra il primo segnale e 0.
- La distanza tra un'occorrenza del segnale *s* e il segnale precedente (sia esso *s* o *r*) è dimezzata ad ogni emissione, fino a che non diventa meno di 1. Quando la distanza tra un segnale *s* e il precedente è meno di 1, un segnale *r* è emesso ad una distanza dall'ultimo segnale che è due volte la distanza tra i due ultimi segnali *s*.
- (in modo analogo al punto precedente) La distanza tra un'occorrenza del segnale *r* e il segnale precedente (sia esso *r* o *s*) è raddoppiata ad ogni emissione, fino a che non diventa maggiore di 10. Quando la distanza tra un segnale *r* e il precedente è più di 10, un segnale *s* è emesso ad una distanza dall'ultimo segnale che è la metà della distanza tra i due ultimi segnali *r*.

(continua dietro)

Esercizio 2 (6/17 punti)

Si dica se le seguenti funzioni sono computabili, motivando adeguatamente la risposta:

1) $g'(x,y) = 0$ se $f_x(y) \neq \perp$ e $f_x(y) = 2\ f_y(x)$; $g'(x,y) = f_x(y)+f_y(x)$ altrimenti;
2) $g''(x) = 0$ se $\exists y : f_x(y) = f_y(x)$; $g''(x) = 1$ altrimenti.

NB: Si adotta la convenzione che, per una qualunque operazione aritmetica $\oplus$, il risultato dell'espressione $x \oplus y$ è $\perp$ se uno qualunque dei degli operandi è $\perp$.

Esercizio 3 (7/17 punti)

La seguente funzione C calcola il valore del polinomio `c[0]+c[1]·x+c[2]·x`^{2}`+...+c[n]·x`^{n} a partire da un array `c` contenente i coefficienti, dal grado `n` del polinomio, e dal valore della variabile `x`.

```
int evalPolyn (int c[], int n, int x) {
  int pow, val;
  int i, j;

  val = 0;
  for (i = 0; i <= n; i++){
    pow = 1.0;
    for (j=1; j<=i; j++)
        pow = pow * x;
    val = val + c[i]*pow;
  }
  return val;
}
```

1. Si valutino la complessità temporale T(n) e la complessità spaziale S(n) del programma, in funzione del grado del polinomio, usando il criterio di costo costante.
2. Si valutino la complessità temporale T(n, x) e la complessità spaziale S(n, x) in funzione del grado del polinomio e del valore della variabile x, usando il criterio di costo logaritmico. Non è necessario dare l'espressione esatta di T(n, x) e S(n, x), ma è sufficiente fornire una valutazione approssimata che tiene in considerazione i fattori dominanti.

Soluzioni

Esercizio 1

$\exists t\ (t>1 \wedge s(t) \wedge \forall t'\ (0 \le t' < (3/2)t \wedge t' \neq t \rightarrow \neg s(t')) \wedge s((3/2)t)\ \wedge \forall t''\ (0 \le t'' \le (3/2)t \rightarrow \neg r(t'')))$

$\forall t,t'\ (t'>t \wedge s(t') \wedge (s(t) \vee r(t)) \wedge \forall t''\ (t < t'' < t' \rightarrow \neg s(t'') \wedge \neg r(t'')) \rightarrow$
$((t' - t \ge 1) \rightarrow s(t'+(t'-t)/2) \wedge \neg r(t'+(t'-t)/2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)/2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$
$\wedge$
$((t' - t < 1) \rightarrow r(t'+(t'-t)*2) \wedge \neg s(t'+(t'-t)*2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)*2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$

$\forall t,t'\ (t'>t \wedge r(t') \wedge (s(t) \vee r(t)) \wedge \forall t''\ (t < t'' < t' \rightarrow \neg s(t'') \wedge \neg r(t'')) \rightarrow$
$((t' - t \le 10) \rightarrow r(t'+(t'-t)*2) \wedge \neg s(t'+(t'-t)*2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)*2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$
$\wedge$
$((t' - t > 10) \rightarrow s(t'+(t'-t)/2) \wedge \neg s(t'+(t'-t)/2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)/2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$

Esercizio 2

Le funzioni sono entrambe computabili.
Definiamo un algoritmo per computare g'. Utilizzando la usuale enumerazione *E*, otteniamo dapprima due MT M_x e M_y. Poi calcoliamo a := $f_x(y)$, e b := $f_y(x)$. Se a = 2b, viene restituito 0. altrimenti, a+b. Si noti che anche se $f_x(y)$, o $f_y(x)$, o entrambe ($f_x(y)$ e $f_y(x)$) sono indefinite, l'algoritmo rispetta la definizione di g'.
Per g'' è sufficiente notare che possiamo sempre prendere y = x. Quindi g'' è la funzione costante 0, che è ovviamente computabile.

Esercizio 3

1. Il fattore dominante della complessità deriva dall'istruzione `pow=pow*x;` nel ciclo interno. Tale istruzione viene eseguita un numero di volte pari a $\sum_{i=0}^{n}\sum_{j=1}^{i}1$, che è $\Theta(n^2)$. La complessità spaziale è invece dominata dalla dimensione dell'array c, ed è quindi $\Theta(n)$.
2. Con il criterio del costo logaritmico, dobbiamo considerare che l'esecuzione dell'istruzione `pow=pow*x;` corrisponde a eseguire le seguenti 3 istruzioni della macchina RAM:
   ```
   LOAD w;
   MULT y;
   STORE w;
   ```
   dove `w` e `y` sono gli indirizzi delle variabili `pow` e `x`, rispettivamente. Alla j-esima iterazione del ciclo interno, la complessità dell'istruzione `LOAD w` è $l(w)+l(x^{j-1}) \cong (j-1)l(x)$, la complessità dell'istruzione `MULT y` è $l(y)+l(x)+l(x^{j-1}) \cong j \cdot l(x)$, e quella dell'istruzione `STORE w` è $l(w)+l(x^j) \cong j \cdot l(x)$. Quindi la complessità di `pow=pow*x;` è $\cong (3j-1)l(x)$. La complessità dell'intero programma è quindi dominata dal fattore $\sum_{i=0}^{n}\sum_{j=1}^{i}(3j-1)l(x) \cong 3 \cdot l(x)\sum_{i=0}^{n}\sum_{j=1}^{i} j$. Poichè $\sum_{j=1}^{i} j$ è $\Theta(i^2)$ e $\sum_{i=0}^{n} i^2$ è $\Theta(n^3)$, abbiamo che $T(n,x) \cong K \cdot l(x) \cdot n^3$ per qualche valore costante K.
   Per il calcolo della complessità spaziale con il criterio del costo logaritmico dobbiamo invece tenere conto anche dell'occupazione delle celle di memoria. I fattori in gioco sono l'array dei coefficienti, le cui celle contengono valori costanti e quindi la complessità spaziale non cambia rispetto al criterio del costo costante e la cella che conterrà il valore massimo, cioè quella che contiene il risultato. In particolare `val` raggiungerà il valore $\sum_{i=0}^{n} Kx^i$, che è $\Theta(x^n)$ quindi l'occupazione della cella è $\Theta(\log(x^n))$, cioè è $\Theta(n\log(x))$. La complessità spaziale è quindi è $\Theta(n\log(x))$.

Informatica Teorica

Appello d'esame - 13 Luglio 2010
Tempo a disposizione: 2h

Esercizio 1 (14 punti)
Si consideri il seguente linguaggio costruito sull'alfabeto I={a,b}. Le stringhe del linguaggio sono tali che ogni simbolo *a* è seguito da almeno *n*=3 simboli, ed il terzo simbolo seguente è una *b*.

1. Si scriva un automa a stati finiti deterministico che accetta il linguaggio descritto sopra.
2. Si descriva come tale automa a stati finiti deterministico potrebbe essere modificato per trattare il caso in cui *n*=4, o in generale un qualunque numero maggiore di 3. In particolar modo, si dica precisamente quanti stati ha l'automa a stati finiti generalizzato in dipendenza dal valore di *n*.
3. Si consideri ora la nozione di automa a stati finiti *universalmente nondeterministico*. Questo è definito come un usuale automa a stati finiti nondeterministico, salvo per il fatto che una stringa *x* è accettata se e solo se *tutte* le possibili computazioni nondeterministiche accettano *x*, cioè terminano in uno stato finale. Formalmente, il linguaggio accettato *L* è definito, invece che dalla formula $x \in L \leftrightarrow (\delta^*(q_o,x) \cap F \neq \emptyset)$ come negli abituali automi a stati finiti nondeterministici, dalla seguente formula: $x \in L \leftrightarrow (\delta^*(q_o,x) \subseteq F)$.
   Si definisca un automa a stati finiti *universalmente nondeterministico* che accetta il linguaggio descritto sopra con *n*=3. Si sfrutti questa forma di nondeterminismo per definire un automa con il minor numero di stati possibile.
4. Si descriva come l'automa stati finiti universalmente nondeterministico potrebbe essere modificato per trattare il caso in cui *n*=4, o in generale un qualunque numero maggiore di 3. In particolar modo, si dica precisamente quanti stati ha tale automa a stati finiti universalmente nondeterministico generalizzato in dipendenza del valore di *n*.

Esercizio 2 (10 punti)
Sia *f(y,x)* la funzione che restituisce il valore computato dalla *y*-esima Macchina di Turing quando questa in ingresso ha il valore *x*.

Per ognuna della seguenti formule logiche dire, spiegando in modo adeguato il perché, se esiste una Macchina di Turing che è in grado di calcolare il valore di verità delle formule al variare del valore delle variabili che appaiono in esse.

1) $\exists y,z \; \forall x \; (f(y,x) = f(z, x^2+1))$

2) $\forall x \; (f(y,x) = f(z,x+1))$

3) $\exists z \; (z \neq y \wedge \forall x (f(y,x) = f(z,x)))$

4) $\forall x (f(y,x) = f(z,x))$
   $\wedge$
   $\forall x (\; (f(y,x) = 1 \wedge \forall w (f(x,w) \neq \perp))$
   $\vee$
   $(f(y,x) = 0 \wedge \exists w (f(x,w) = \perp)) \;)$

(continua dietro)

Esercizio 3 (10 punti)
Si consideri una macchina di Turing che accetta il seguente linguaggio: array bidimensionali (quadrati o rettangolari) di zeri e uni in cui esattamente una riga e una colonna sono composti di caratteri 1, mentre il resto è fatto da caratteri 0. La figura viene fornita in ingresso alla macchina riga per riga, con il carattere $ come separatore di riga.
Dato un ingresso di questo genere, la macchina:

- controlla che le righe siano tutte della stessa lunghezza;
- controlla che ci siano esattamente una riga e una colonna composte da soli 1.

Se il dato in ingresso soddisfa le condizioni, la macchina emette in uscita il numero complessivo di 1 in esso presenti, altrimenti si ferma in uno stato non finale senza emettere nulla.

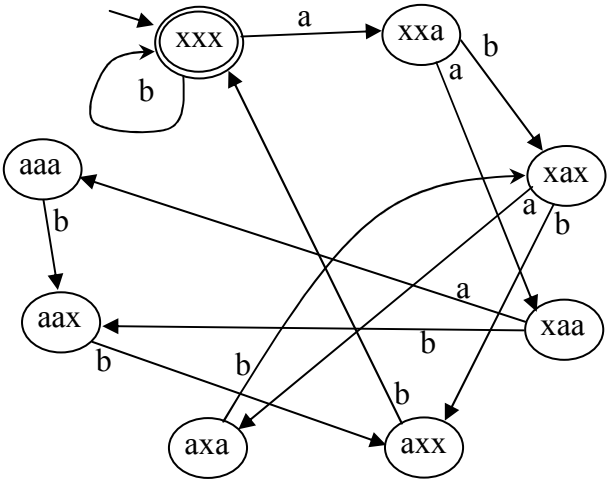
Si descriva in maniera sufficientemente accurata ma ad alto livello il funzionamento di una MT che effettui la precedente traduzione, minimizzandone il più possibile la complessità spaziale. Se ne valutino inoltre sia la complessità temporale che quella spaziale.

NB: non è necessario fornire il diagramma delle transizioni della macchina, basta per esempio una descrizione in pseudocodice, includendo comunque la valutazione esplicita della complessità spaziale e temporale.

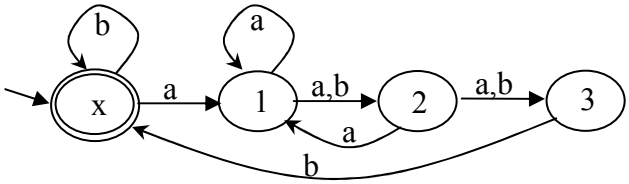
Soluzioni

Esercizio 1

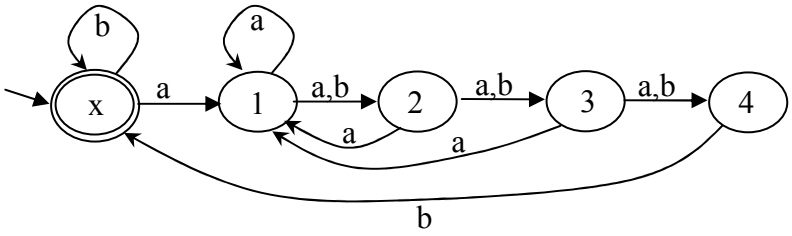
1. Nella soluzione proposta, le *a* che compaiono negli stati corrispondono ai simboli incontrati nella porzione di stringa analizzata fino a quel punto che devono ancora avere una *b* corrispondente.



2. Per trattare il caso generale di *n* simboli, l'automa dovrebbe avere 2^n stati, e tenere traccia di tutte le *a* che possono essere state incontrate negli ultimi *n* simboli della stringa analizzata fino a quel punto.
3. Con il nondeterminismo universale bastano *n+1* stati. Per ogni *a* incontrata nella stringa l'automa deve avere una computazione in cui una *b* si trova *n* posizioni dopo la *a*. Lo stato etichettato con *k* (k=1, 2, ... , n) indica che il *k*-esimo simbolo alla sinistra del simbolo corrente era una *a*. Lo stato etichettato con *x* indica che ogni simbolo *a* precedentemente letto a distanza >*n* da quello corrente ha avuto la *b* corrispondente. Questa è la soluzione per n=3...



4. ... e per n=4



Esercizio 2

1)
La MT esiste, in quanto la formula è chiusa, e quindi o è sempre vera o è sempre falsa, indipendentemente dai valori di *y*, *z*, *x*; di conseguenza, la MT in questione o è quella che ritorna sempre 1 (vero) o sempre 0 (falso).

2)
In questo caso la formula è vera per tutte e sole le coppie *y*,*z* di (indici di) MT tali che f_z corrisponde a f_y scalata di 1. Se esistesse una MT che calcola il valore di verità di questa formula al variare di *y* e *z*, allora

sarebbe decidibile il problema di stabilire se, data una qualunque coppia y, z di (indici di) MT, esse sono tali che f_z corrisponde a f_y scalata di 1. Tale problema però non è decidibile (è possibile farlo vedere fissando un particolare valore di y, ed applicando il teorema di Rice), quindi la MT in questione non esiste.

3)
La formula dice che, preso un generico indice di MT y, è possibile trovare una MT di indice z diversa dalla prima che calcola la stessa funzione. Questo è sempre vero (data una qualunque MT, è banale costruirne una diversa che calcola la stessa funzione, per esempio introducendo uno stato iniziale fittizio nel quale la nuova MT non fa nulla), quindi una MT che calcola la funzione desiderata è semplicemente una che come risultato dà sempre 1 (vero).

4)
In questo caso la formula è vera per tutte le coppie di (indici di) MT y, z che calcolano la stessa funzione, ma tali che la funzione calcolata da y (e quindi da z) è quella che, preso un qualunque indice di MT x, ritorna 1 se f_x è una funzione totale, 0 altrimenti. Una tale funzione però non è calcolabile, quindi l'insieme di coppie y,z suddetto è vuoto, e la MT richiesta è semplicemente quella che dà sempre come risultato falso (0).

Esercizio 3

Sia:

- m: numero di righe
- n: numero di colonne
- LR: contatore binario per lunghezza di ogni riga (lg n)
- IC: contatore binario per indice di colonna (lg n)
- NR: contatore binario per la somma di 1 (lg (n + m))
- X: contatore binario temporaneo (lg n)

in cui la complessità spaziale è indicata tra parentesi

Gli stati dell'organo di controllo servono ad assicurarsi che ad ogni riga si trova un unico 1, e che una unica riga è fatta di 1.

La MT si comporta nel seguente modo (le complessità dei singoli passi sono indicate tra parentesi):

```
LR := 0
leggi fino a $, incrementando LR ad ogni passo - quando trovi 1, copia LR in IC. (n lg n)
NR := 0
per ogni riga successiva: (m)
        incrementa NR (lg (n + m))
        X := 0 (lg n)
        se riga normale: leggi la riga e incrementa X ad ogni passo, fino all'1;
                se X = IC continua e alla fine controlla che X = LR    (n lg n)
        se riga di 1: leggi la riga e incrementa X e NR ad ogni passo, fino a $;
                poi controlla che X = LR  (n lg (n + m))
restituisci NR in uscita (lg (n + m))
```

complessità spaziale S = Θ(lg (n + m))
complessità temporale T = Θ(m n lg (n)+(m+n)lg(m+n))= Θ(m n lg (n))

Informatica Teorica

Appello d'esame - 3 Settembre 2010
Tempo a disposizione: 2h

Esercizio 1 (8 punti)
Si definisca un automa in grado di riconoscere il seguente linguaggio:

$$L=\{bin(i)\$[bin(i+1)]^R | i>=0\}$$

dove *bin(k)* indica la rappresentazione in binario (con il minor numero di bit possibile) del numero naturale k e l'apice R il riflesso, cioè nelle parole che fanno parte di L il primo numero (bin(i)) è scritto con la cifra più significativa a sinistra, mentre il secondo (bin(i+1)) con la cifra più significativa a destra.
L'automa deve essere a potenza minima tra quelli che riconoscono il linguaggio L.

Esempi di stringhe che appartengono al linguaggio: 1111$00001, 1000$1001, 100101$011001.
Esempi di stringhe che non appartengono al linguaggio: 001$010, 100$111, 0011$0010.

Esercizio 2 (9 punti)
Si introduca il predicato *nextS(t1,t2)* che è vero se e solo se

- *t2 > t1*
- all'istante *t1* viene emesso il segnale S
- la volta successiva a *t1* che il segnale S viene emesso è all'istante *t2*.

Utilizzando il solo predicato *nextS* (ed eventualmente relazioni e operazioni aritmetiche quali >, <, =, +, -, ecc.) si formalizzino mediante formule logiche le seguenti proprietà (tra di loro indipendenti):

1) Il segnale S viene emesso al massimo una volta sola.
2) Il segnale S viene emesso la seconda volta all'istante 10.
3) Il segnale S viene emesso un numero finito di volte, maggiore o uguale a 2.
4) Il segnale S viene emesso un'infinità di volte.

Il tempo è da intendersi discreto.

Esercizio 3 (8 punti)
Si considerino gli insiemi:

$S = \{i \mid \forall x\ (f_i(x) = \perp \vee f_i(x) \leq x)\}$;
$T = \{i \mid \exists x\ \exists k\ (k > 0 \wedge f_i(x) = x+k)\}$.

Si dica se i seguenti insiemi sono decidibili, semidecidibili o altro:
1) S
2) T
3) $S \cap T$.

Esercizio 4 (9 punti)
Si consideri il linguaggio L fatto di stringhe di parentesi tonde ben parentetizzate terminate dal simbolo *$*.
Esempi di stringhe appartenenti ad L sono: ()()$, (())$, ((())()(()))()(())$.
Esempi di stringhe *non* appartenenti ad L sono: ())$,)()($, ((()))($.

1) Si delinei la più semplice Macchina di Turing a k nastri che accetta il linguaggio L, e se ne valutino le complessità spaziale e temporale in funzione della lunghezza *n* della stringa in input.

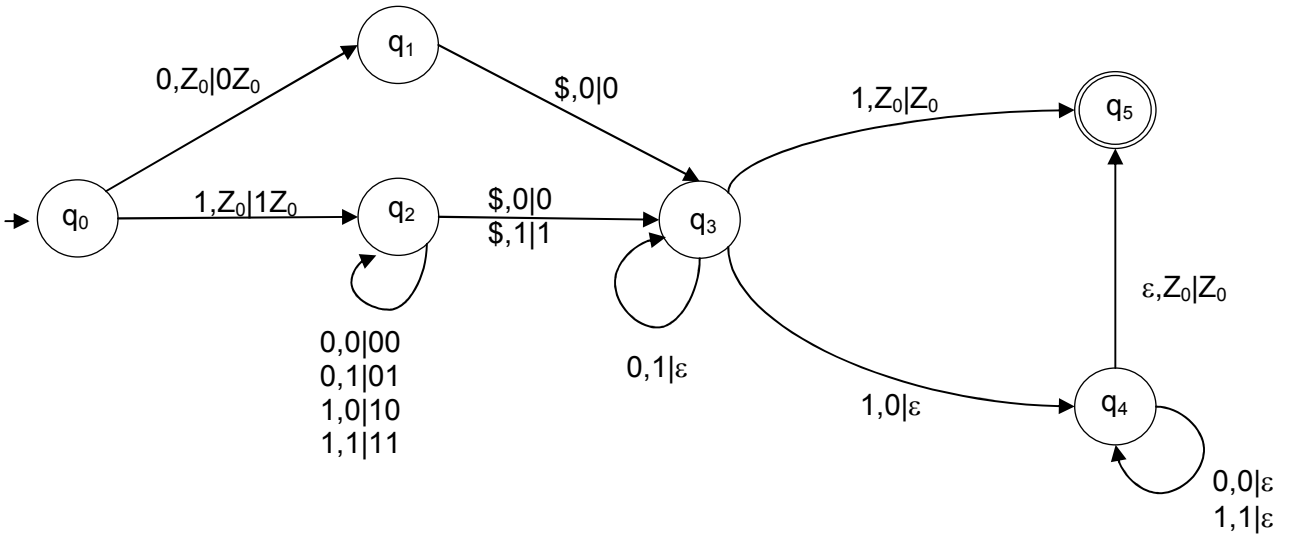
   1a) Come cambiano, se cambiano, le complessità se si usa una Macchina di Turing *a nastro singolo* invece di una a k nastri?

2) Si delinei la più semplice macchina RAM che accetta il linguaggio L, e se ne valutino le complessità temporale e spaziale in funzione della lunghezza *n* della stringa di input, usando il criterio di *costo logaritmico*.

Soluzioni

Esercizio 1

Per riconoscere L è sufficiente il seguente AP deterministico (in cui, per rappresentare la pila, si adotta la convenzione sinistra/alto, destra/basso):



Esercizio 2

1) $\neg\exists$t1,t2 (nextS(t1,t2))
2) $\exists$t1 (nextS(t1,10) $\wedge$ $\neg\exists$t' (nextS(t',t1))
3) $\exists$t1,t2 (nextS(t1,t2) $\wedge$ $\neg\exists$t'(nextS(t2,t'))
4) $\exists$t1,t2 (nextS(t1,t2)) $\wedge$ $\forall$t3,t4(nextS(t3,t4) $\rightarrow$ $\exists$t'(nextS(t4,t')))

Esercizio 3

1) Indecidibile, non semidecidibile (si vede facilmente che il suo complemento è semidecidibile, con la solita tecnica di esecuzione un passo alla volta di $f_i(x)$ a partire da x=0).
2) Semidecidibile (stesso approccio).
3) Decidibile, visto che è l'insieme vuoto.

Esercizio 4

Una macchina di Turing a 1 nastro può leggere l'input e mantenere un contatore unario (per esempio memorizzando degli '*') del livello corrente di annidamento, incrementando il contatore ogni volta che si incontra un simbolo '(' e decrementandolo ogni volta che si incontra un simbolo ')'. La macchina scrive in output NO se il contatore diventa negativo, o SI se raggiunge la fine della stringa con il contatore 0. I costi temporale e spaziale sono entrambi $\Theta(n)$.

Una macchina di Turing a nastro singolo può usare una porzione vuota del suo nastro per memorizzare il contatore in forma unaria. La complessità spaziale rimane invariata, ma quella temporale aumenta: nel caso pessimo la MT deve fare $\Theta(n)$ mosse per aggiornare il contatore per ogni simbolo letto, quindi la complessità temporale è $\Theta(n^2)$.

Una macchina RAM può realizzare lo stesso algoritmo della MT a 1 nastro, ma memorizzando il contatore in forma binaria in una cella di memoria singla. Usando il cirterio di costo logaritmico, il caso pessimo è quello di una stringa di lunghezza $n=2\cdot k*1$ fatta di k simboli '(' seguiti da k ')', e quindi il simbolo '$'. Per tale stringa la complessità spaziale (la dimensione della cella che memorizza il contatore) è $\Theta(\log(n))$; per la complessità temporale il fattore dominante è collegato alle operazioni di incremento e decremento del contatore. Esse sono entrambe proporzionali a $\sum_{i=1}^{n}\log(i)=\log(n!)=n\cdot\log(n)$, quindi la complessità temporale è $\Theta(n\cdot\log(n))$.