Informatica Teorica

Prima prova in itinere - 11 Maggio 2007

Tempo a disposizione: 1h 45'

Esercizio 1 (punti 5/13)

Scrivere una grammatica che genera il linguaggio costruito sull'alfabeto {a,b,c} fatto di tutte e sole le stringhe x tali che esistano 2 sottostringhe disgiunte w e w', di lunghezza multiplo di 3, e $w' = w^R$ (es. w = ccacbb, w' = bbcacc).

NB: Il punteggio massimo verrà dato se la grammatica scritta sarà a potenza minima tra quelle che generano il linguaggio desiderato.

Esercizio 2 (punti 5/13)

Si abbia un impianto manifatturiero che produce 2 tipi di prodotti, p_1 e p_2, tali che ogni pezzo di tipo p_1 vale 20000 euro, ed ogni pezzo di tipo p_2 vale 30000 euro. L'impianto può produrre pezzi p_1 e p_2 in qualunque sequenza, fatto salvo che, ad ogni punto della sequenza di produzione, il ***valore totale*** dei pezzi p_2 prodotti deve essere maggiore o uguale del ***valore totale*** dei pezzi p_1 prodotti, e comunque la differenza tra i 2 valori totali non deve essere mai superiore ai 90000 euro.

Si scriva un automa che accetta tutte e sole le sequenze di produzione ammissibili dell'impianto sopracitato.

NB: Il punteggio massimo verrà dato se l'automa scritto sarà a potenza minima tra quelli che riconoscono le sequenze desiderate.

Esercizio 3 (punti 6/13)

E' noto che ogni automa trasduttore definisce una traduzione τ: I* $\longrightarrow$ O*. Per ogni famiglia di automi trasduttori (a stati finiti, a pila –deterministici e non–, di Turing) si dica, giustificando brevemente la risposta, se essa è chiusa rispetto alla composizione di traduzioni, ossia se, dati due automi A_1 e A_2 che definiscano rispettivamente le traduzioni τ_1 e τ_2, esista nella stessa famiglia un automa A che definisca la traduzione $\tau(x) = \tau_2(\tau_1(x))$.

Soluzioni

Esercizio1

Una grammatica (non-contestuale) che genera il linguaggio desiderato è la seguente.

$S \rightarrow QR_1Q$

$Q \rightarrow aQ \mid bQ \mid cQ \mid \varepsilon$

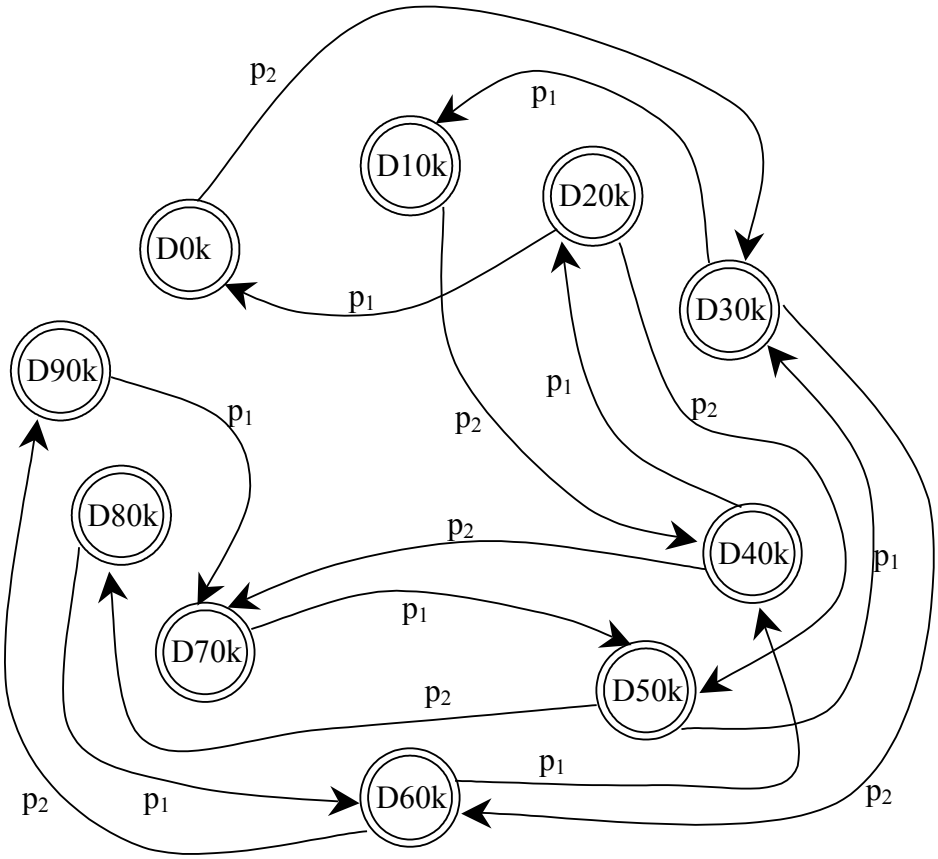
$R_1 \rightarrow aR_2a \mid bR_2b \mid cR_2c$

$R_2 \rightarrow aR_3a \mid bR_3b \mid cR_3c$

$R_3 \rightarrow aR_1a \mid bR_1b \mid cR_1c \mid aQa \mid bQb \mid cQc$

Esercizio2

Un automa a stati finiti che accetta tutte e sole le sequenze di produzione desiderate è il seguente (laddove ogni stato rappresenta la differenza fino ad ora accumulata tra il totale del valore dei pezzi p_2 ed il totale del valore dei pezzi p_1):



Esercizio 3

1.

Le traduzioni a stati finiti sono chiuse rispetto alla composizione: un procedimento che, dati due traduttori $T_1 = <Q_1, I_1, \delta_1, q_{01}, F_1, \eta_1, O_1>$ e $T_2 = <Q_2, I_2, \delta_2, q_{02}, F_2, \eta_2, O_2>$ (con $O_1 = I_2$) produca un traduttore T che definisca $\tau(x) = \tau_2(\tau_1(x))$ può essere intuitivamente descritto con la seguente costruzione, derivata dalla costruzione per l'intersezione di automi.

T è una tupla $<Q, I_1, \delta, q_0, F, \eta, O_2>$ in cui $Q = Q_1 \times Q_2$ e le funzioni di transizione e output di T, $<\delta, \eta>$, sono definite secondo la seguente regola: se in T_1 esiste una coppia

$<\delta_1, \eta_1>(q_1, a) = <q_1', w_1>$, con $a \in I_1$ e $w_1 \in O_1^*$, e per T_2 si ha

$<\delta_2^*, \eta_2^*>(q_2, w_1) = <q_2', w_2>$, con $w_2 \in O_2^*$

si definisca la coppia $<\delta, \eta>(<q_1,q_2>, a) = <<q_1' \; q_2'>, w_2>$.

L'insieme di stati di accettazione F di T sono tutti e soli quelli nella forma $< q_1, q_2 >$ tali che $q_1 \in F_1$ e $q_2 \in F_2$. Lo stato iniziale di T è infine lo stato $<q_{01}, q_{02}>$.

2.

Le traduzioni a pila (sia deterministiche che non) non sono chiuse rispetto alla composizione.

Infatti la traduzione $\tau(w \cdot c) = w \cdot c \cdot w$, può essere ottenuta componendo le due traduzioni $\tau_1(w \cdot c) = w \cdot c \cdot w^R \cdot d$ e $\tau_2(w_1 \cdot c \cdot w_2 \cdot d) = w_1 \cdot c \cdot w_2^R$. Essa però non può essere ottenuta da un solo automa a pila, rigorosamente legato alla disciplina LIFO nella gestione del proprio input.

Un altro controesempio possibile è il seguente: la traduzione $\tau(a^n \cdot b^n \cdot c^n) = d$ può essere ottenuta componendo le due traduzioni $\tau_1(a^n \cdot b^n \cdot c^*) = b^n \cdot c^*$ e $\tau_2(b^n \cdot c^n) = d$. Anche in questo caso la traduzione, che necessita dell'accettazione del linguaggio $\{a^n \cdot b^n \cdot c^n\}$, non può essere effettuata da un automa a pila semplice.

Più in generale, si osserva che componendo "in serie" due trasduttori a pila è possibile fare in modo che la composizione dei due accetti un linguaggio uguale all'intersezione dei due linguaggi accettati da ognuno dei due trasduttori di partenza. Poiché è noto che la classe dei linguaggi accettati dagli automi a pila ***non*** è chiusa rispetto all'intersezione, in generale non esiste quindi un automa a pila equivalente alla composizione dei due di partenza.

3.

La composizione di due traduzioni effettuate da macchine di Turing può essere banalmente ottenuta da una macchina che memorizzi in un nastro di memoria l'output della prima e poi lo usi come fosse l'input della seconda.

Informatica Teorica

Sezione Cremona+Como

Seconda prova in itinere - 3 Luglio 2007

Tempo a disposizione: 1h 30'

Esercizio 1 (punti 4/17)

La successione $2^{2^n}+1$, per *n* numero naturale, è detta dei *numeri di Fermat*. Il matematico Pierre de Fermat fu il primo a notare come i primi termini della successione (per n=0, 1, 2, 3) siano primi, dunque congetturò che lo siano tutti.

Si risponda alle seguenti domande, fornendo delle brevi, ma esaurienti, spiegazioni:

1.1. E' decidibile il problema "tutti i numeri di Fermat sono primi?"?

1.2. E' decidibile il problema "dato un numero naturale *n*, il numero $2^{2^n}+1$ è primo?"?

1.3. (nel caso si sia risposto positivamente alla domanda 1.2) E' decidibile l'insieme delle macchine di Turing che risolve il problema della domanda 1.2?

Esercizio 2 (punti 6/17)

2.1

Si descriva un algoritmo che, dato un numero naturale *n*, determini se l'n-esimo numero di Fermat è primo.

Si noti che:

- non è necessario che l'algoritmo sia efficiente o elegante, l'importante è che sia semplice;
- per descrivere l'algoritmo è possibile usare pseudocodice oppure un linguaggio ad alto livello.

2.2.

Se l'algoritmo definito al punto 2.1 viene eseguito su una macchina RAM, quale è il criterio di costo più opportuno per valutare la complessità di tale macchina RAM? Si motivi la risposta.

2.3.

Si usi il criterio individuato al punto 2.2 per valutare la complessità spaziale e temporale di una macchina RAM che esegue l'algoritmo definito.

Esercizio 3 (punti 9/17)

Si vuole usare la logica del prim'ordine (con il predicato di uguaglianza) per descrivere giocatori e squadre di uno sport a squadre (per esempio il calcio).

Si assuma quanto segue:

- le variabili (x, y, w, ...) denotano giocatori;
- viene introdotto il predicato `SameTeam(x,y)` il quale è vero se i giocatori x e y sono nella stessa squadra (laddove "essere nella stessa squadra" è una relazione di equivalenza).

3.1.

Si scrivano, usando *esclusivamente* i predicati di uguaglianza (=), disuguaglianza (≠) e `SameTeam`, delle formule logiche che formalizzano i seguenti vincoli:

a) tutte le squadre hanno almeno 3 giocatori;

b) esistono almeno 3 squadre.

3.2.

Si consideri una relazione di "rivalità personale" tra alcuni dei giocatori. A questo scopo si introduca il predicato `PersRivalry(x,y)`, il quale rappresenta il fatto che i giocatori x e y sono rivali (si assuma pure che `PersRivalry` sia simmetrica).

Si scriva una formula che formalizzi il fatto che non ci possono essere rivalità all'interno di una stessa squadra.

3.3.

Si assuma che periodicamente (per esempio ogni mese) si svolga un torneo tra le squadre, e che sia le squadre che le rivalità personali possano cambiare da un torneo all'altro.

Si assuma inoltre quanto segue:

- le variabili t, t', t'' denotano i momenti di svolgimento dei tornei, rappresentati come numeri interi.
- per descrivere squadre e rivalità si usano delle varianti temporizzate dei predicati introdotti in precedenza, `SameTeam(x,y,t)` e `PersRivalry(x,y,t)`, che indicano appartenenza alla stessa squadra / rivalità di x e y al momento del torneo t.

Si scrivano delle formule che formalizzino i seguenti vincoli:

a) due giocatori sono nella stessa squadra durante un torneo solo se non erano rivali in alcuno dei precedenti k tornei;

b) se, per un certo torneo, due giocatori della stessa squadra sono rivali, essi non saranno nella stessa squadra per i prossimi h tornei.

Soluzioni

Esercizio 1

1.1. Sì, si tratta di una domanda "chiusa" (la risposta è sì oppure no). Tra l'altro si conosce anche la risposta: Eulero dimostrò che per n=5 si ottiene un numero non primo.

1.2. Sì, basta scrivere una semplice procedura che, dato n, calcoli $2^{2^n}+1$ e poi ne controlli la primalità.

1.3. No, è una conseguenza del teorema di Rice (l'insieme delle MT che risolvono il problema 1.2 non è l'insieme vuoto, né tantomeno quello universo).

Esercizio 2

2.1

```
boolean primeNthFermat(int n){
1. x = 2
2. for i from 1 to n do x = x*x
3. x = x+1
4. for i from 2 to squareRootOf(x) do
5.   if (x/i)*i==x then return false
6. return true
```

2.2-2.3.
Per il calcolo del numero di Fermat (righe 1-3) è necessario usare il criterio di costo logaritmico, per ottenere una valutazione di complessità realistica. Quindi si ottiene $\sum_{k=1}^{n} 2^k = \Theta(2^n)$ per il tempo e $\Theta(2^n)$ anche per lo spazio.

Il fattore di complessità dominante è però il secondo ciclo (righe 4-5): la complessità temporale che si ottiene per esso (naturalmente sempre a costo logaritmico) è:

$\Theta(\ 2^n\sqrt{2^{2^n}} \ ) = \Theta(\ 2^n 2^{2^{n-1}} \ ) = \Theta(\ 2^{n+2^{n-1}} \ )$.

Infatti la singola iterazione costa $\log(2^{2^n})$, ed il calcolo della radice quadrata non impatta sul comportamento asintotico della complessità.

Esercizio 3

3.1.

a) $\forall x \exists y \exists z(x \neq y \wedge y \neq z \wedge x \neq z \wedge \text{SameTeam}(x,y) \wedge \text{SameTeam}(y,z))$

b) $\exists x \exists y \exists z(x \neq y \wedge y \neq z \wedge x \neq z \wedge \neg\text{SameTeam}(x,y) \wedge \neg\text{SameTeam}(y,z) \wedge \neg\text{SameTeam}(x,z))$

3.2.

$\forall x \forall y(x \neq y \wedge \text{SameTeam}(x,y) \rightarrow \neg\text{PersRivalry}(x,y))$

3.3.

a) $\forall x \forall y \forall t(\text{SameTeam}(x,y,t) \rightarrow \forall t'(t-k \leq t' < t \rightarrow \neg\text{PersRivalry}(x,y,t')))$

b) $\forall x \forall y \forall t(\text{PersRivalry}(x,y,t) \wedge \text{SameTeam}(x,y,t) \rightarrow \forall t'(t < t' \leq t+h \rightarrow \neg\text{SameTeam}(x,y,t')))$

Informatica Teorica

Sezione Cremona+Como

Appello del 20 Luglio 2007

Il tempo a disposizione per lo svolgimento del tema d'esame è 2 ore.

Esercizio 1 (punti 6)

parte a.

Si scriva un automa che riconosce il linguaggio L_p, costruito sull'alfabeto {a, b, c, d}, composto di tutte e sole le stringhe che sono fatte nella seguente maniera: ogni volta che nella stringa compare un simbolo c, ci deve essere precedentemente un simbolo b, e tra questo e la c, se ci sono simboli, ci devono essere solo simboli d.

L'automa deve essere a potenza riconoscitiva minima tra quelli che riconoscono il linguaggio desiderato.

parte b.

Si scriva una grammatica che generi il linguaggio descritto al punto a.

NB: Non è necessario che la grammatica abbia potenza generativa minima tra quelle che generano il linguaggio: la grammatica sarà tanto meglio valutata quanto più piccolo sarà il suo numero di produzioni.

Esercizio 2 (punti 10)

Si vuole modellare mediante formule di logica del prim'ordine il comportamento di un segnale periodico s, il cui periodo di emissione può venire cambiato nel tempo mediante l'invio di un apposito comando.

Più precisamente, se il periodo è P, il segnale viene emesso esattamente ogni P istanti. Il comando di cambiamento del periodo comunica quale deve essere, a partire dal prossimo segnale (cioè tra il prossimo segnale e quelli successivi), il nuovo periodo di emissione (il quale rimarrà invariato fino al prossimo comando di cambiamento).

Il comando di cambiamento non può essere dato in un istante in cui deve essere emesso s. Inoltre, prima di poter di nuovo cambiare il periodo occorre aspettare l'emissione di almeno 2 segnali.

Il periodo iniziale è π.

Esercizio 3 (punti 10)

Sia FiniteC un linguaggio di programmazione di potenza espressiva equivalente agli FSA.

Si risponda, motivando brevemente (ma precisamente) le risposte, alle seguenti domande.

1. E' decidibile il problema di stabilire se due generici programmi F_1 ed F_2 scritti in FiniteC sono equivalenti?
2. Si indichino con τ_i delle funzioni che traducono dei programmi FiniteC in FSA. Più precisamente, se F è un programma scritto in FiniteC, $\tau_i(F)$, se è definito, è un FSA.

   E' decidibile il problema di stabilire se due programmi C generici T1 e T2 realizzano la stessa traduzione τ_i?
3. E' decidibile il problema di stabilire se, dato un generico programma F scritto in C che realizza una traduzione τ_F, l'automa che riconosce il linguaggio L = {a*b*} appartiene al codominio di τ_F?
4. E' semidecidibile il problema descritto al punto 3?

Esercizio 4 (punti 8)

parte a.

Si descriva (a parole, ma in modo sufficientemente dettagliato) il funzionamento di una MT a nastro *singolo* che, data una stringa in ingresso della forma $n_1\#n_2$, in cui n1 e n2 sono due numeri naturali codificati in binario, lascia sul nastro il valore -1, 0 o 1 a seconda che n_1 sia minore, uguale, o maggiore di n_2.

Quali sono le complessità temporale e spaziale della MT?

NB: Si supponga che le stringhe che rappresentano n_1 ed n_2 siano di lunghezza uguale, pari alla lunghezza della codifica binaria del maggiore dei due numeri.

parte b.

Si descriva (a parole, ma in modo sufficientemente dettagliato) il funzionamento di una MT a nastro *singolo* che, data una sequenza di numeri naturali n_1, ... n_N codificati mediante una stringa della forma $n_1\#n_2\#...\#n_N$, in cui i vari n_i (con $1 \le i \le N$) sono codificati in binario, ordina la sequenza mediante bubblesort.

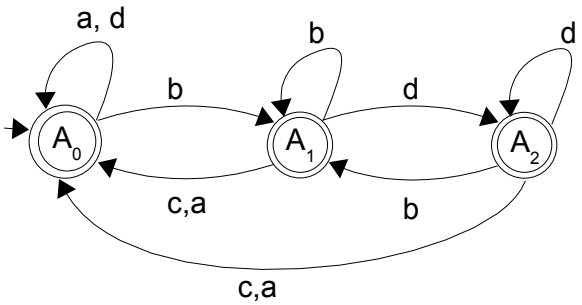
Quali sono le complessità temporale e spaziale della MT?

NB: Si supponga che le stringhe che codificano gli n_i siano tutte di lunghezza uguale, pari alla lunghezza della codifica binaria del maggiore dei numeri.

Soluzioni

Esercizio 1

parte a.



parte b.

$S \rightarrow aS \mid bS \mid dS \mid bDcS \mid \varepsilon$

$D \rightarrow \varepsilon \mid dD$

Esercizio 2

Si introducano i seguenti predicati e funzioni:

- s(t), un predicato che è vero se e solo se il segnale s è emesso all'istante t,
- per(t), una funzione che ritorna quale è il periodo all'istante t
- cambia_per(p, t), che è un predicato che è vero se e solo se all'istante t c'è il comando di cambio periodo che stabilisce che il nuovo periodo deve essere p; p è un valore reale e > 0.

Il sistema può essere descritto mediante le seguenti formule (si supponga che il dominio del tempo siano i reali non negativi, cioè incluso lo 0), che sono da intendere universalmente chiuse ed in congiunzione tra loro:

- Event_s $\wedge$ Event_cambia_per
- per(0) = π
- per(t) = p $\leftrightarrow \exists t' (t' < t \wedge$ cambia_per(p, t') $\wedge \forall t'' (t' < t'' < t \rightarrow \neg\exists p' ($cambia_per(p', t''))))$ $\vee$
  $p = \pi \wedge \forall t' (0 < t' < t \rightarrow \neg\exists p' ($cambia_per(p', t'')))
- cambia_per(p, t) $\rightarrow \neg$s(t)
- s(t) $\rightarrow$ s(t+per(t)) $\wedge \forall t' (t < t' < t+$per(t) $\rightarrow \neg$s(t'))
- s(π) $\wedge \forall t' (0 < t' < \pi \rightarrow \neg$s(t'))
- cambia_per(p, t) $\rightarrow \exists t', t'' (t' < t'' < t \wedge$ s(t') $\wedge$ s(t'') $\wedge$
  $\forall t''' (t' < t''' < t \rightarrow \neg\exists p' ($cambia_per(p', t''))))

Esercizio 3

1.

Il problema è decidibile, in quanto è noto che il problema dell'equivalenza di 2 automi a stati finiti è decidibile (una facile conseguenza del pumping lemma).

2.

Il problema in questo caso non è decidibile, in quanto è riconducibile al problema di stabilire se 2 programmi C calcolano la stessa funzione $N \rightarrow N$. In effetti, sia i programmi FiniteC che gli FSA sono enumerabili, cioè è possibile assegnare univocamente sia ad ogni programma FinteC che ad ogni FSA un numero naturale (e viceversa). Di conseguenza, si può vedere un programma C che trasforma un programma FiniteC in una FSA (non necessariamente equivalente al programma FiniteC di partenza) come un programma che calcola una funzione $N \rightarrow N$ (e viceversa).

Siccome, analogamente che per le MT, non è decidibile il problema di stabilire se 2 programmi C che calcolano funzioni $N \rightarrow N$ sono equivalenti, anche il problema di partenza non è decidibile.

3.

Anche questo problema non è decidibile, in quanto, per quanto detto al punto 2, è analogo al problema di stabilire se un particolare valore (o un insieme di particolari valori) appartiene al codominio di una MT, che, per il teorema di Rice, non è un problema decidibile (l'insieme di funzioni con un particolare codominio non è l'insieme vuoto, né è l'insieme universo).

4.

Il problema di cui al punto precedente è invece semidicidibile. Per vedere questo basta costruire un algoritmo che simula il programma F per i vari input (enumerabili) e per numeri di passi variabili (secondo la classica tecnica di "simulazione diagonale" vista a lezione). Se per una certa coppia <n, N> (laddove n è il numero di passi di simulazione ed N è il valore di input) il programma F termina, è possibile controllare se l'FSA di output riconosce il linguaggio desiderato (problema che è decidibile, come anche indicato al punto 1). Dato questo algoritmo, se l'FSA desiderato appartiene al codominio della funzione calcolata da F, prima o poi verrà trovato, altrimenti la computazione andrà avanti all'infinito.

Esercizio 4

parte a.

Supponiamo che i 2 valori n_1 ed n_2 siano scritti con la cifra più significativa a sinistra (il caso duale è analogo). E' sufficiente scrivere una MT che scorre avanti e indietro il nastro, leggendo la prima cifra (cioè la più significativa) di n_1, memorizzandola nello stato, quindi confrontandola con la prima di n_2, quindi ripetendo la cosa per la seconda, la terza, ecc., fino a che o si trova una cifra diversa (in questo caso il numero maggiore è quello con la cifra

maggiore), oppure si arriva in fondo al nastro (in questo caso i numeri sono uguali). A questo punto il nastro viene cancellato interamente, lasciando solo -1, 0 o 0 a seconda dei casi.

La complessità spaziale della MT è $\Theta(m)$, con m il numero di cifre binarie necessarie per codificare i due numeri (cioè $m = \log_2(\max(n_1, n_2))$).

La complessità temporale è invece $O(m^2)$, in quanto, nel caso peggiore (quello in cui i numeri siano uguali) occorre scorrere il nastro (metà di esso, per la precisione, il che richiede $\Theta(m)$ mosse) m volte.

parte b.

In questo caso la MT può implementare l'algoritmo classico di bubblesort, quello in cui si deve confrontare il primo elemento con il secondo, scambiarli se necessario, quindi passare al confronto tra il secondo e il terzo, e così via fino ad arrivare in fondo alla sequenza. Se nell'arrivare in fondo si è scambiata qualche coppia di numeri, occorre ricominciare dall'inizio della sequenza, e ripetere così fino a che nello scorrimento della sequenza non si scambia alcuna coppia di numeri. Per il confronto tra 2 numeri adiacenti si può usare l'algoritmo descritto al punto a. Invece, per lo scambio di 2 numeri adiacenti n_i ed n_{i+1} è sufficiente leggere la prima cifra di n_i, memorizzarla nello stato, andare alla prima cifra di n_{i+1}, leggerla, memorizzarla nello stato, e sovrascriverla con quella letta da n_i in precedenza, quindi tornare sulla prima cifra di n_i e sovrascriverla con la prima di n_{i+1}, e ripetere il tutto con la seconda, e così via.

E' noto che il bubblesort su una sequenza di N elementi richiede, nel caso peggiore, N^2 confronti. Se m è il numero di cifre necessarie per codificare ognuno degli N numeri (cioè $m = \log_2(\max\{n_i\})$, ognuno di questi confronti costa $O(m^2)$, per quanto visto prima, Inoltre, anche lo scambio di 2 numeri adiacenti costa $O(m^2)$, quindi, in definitiva, il costo temporale complessivo dell'algoritmo è $O(N^2 \cdot m^2)$.

Il costo spaziale è invece semplicemente $\Theta(N \cdot m)$, perché tale è il numero di celle occupate del nastro della MT.

Informatica Teorica

Sezione Cremona+Como

Appello dell'11 Settembre 2007

Il tempo a disposizione per lo svolgimento del tema d'esame è 2 ore.

Esercizio 1 (punti 14)

Si consideri un modello di calcolo consistente in una Macchina di Turing a nastro singolo e dotata di h testine di lettura/scrittura.

1. Si formalizzi tale modello.
   NB, non è necessario formalizzarne il comportamento, ossia la relazione di transizione; è sufficiente formalizzare la struttura della macchina ed eventualmente la configurazione.
2. Si dica, spiegandone brevemente i motivi, se una tal macchina dotata di k > h testine ha una potenza di calcolo (e.g. capacità di riconoscere linguaggi) maggiore di quella dotata di sole h testine.
3. Si dica, spiegandone brevemente i motivi, se una tal macchina dotata di k > h testine ha la capacità di risolvere problemi (e.g. di riconoscere linguaggi) con complessità strettamente inferiori (secondo la relazione Θ) rispetto a quella dotata di sole h testine.

Esercizio 2 (punti 10)

Si considerino le due grammatiche seguenti:

G1:	G2:
S → ADaCD | DAbDC	S → aADCD | DbADC
DC → CD | AA	DC → CD | AA
A → a | SSAD	A → a | SSAD | ε
C → ADC | c	C → ADC | c
SA → AS | ε	SA → AS | ε

Si dica, giustificando brevemente le risposte, se i seguenti problemi sono decidibili:

1. G1 è equivalente a G2?
2. Data una generica grammatica regolare G, G è equivalente a G1?

Esercizio 3 (punti 10)

Si formalizzi mediante una formula del prim'ordine il linguaggio L costituito da stringhe del tipo `wcu`, in cui `w` e `u` denotano stringhe non nulle consistenti dei soli caratteri '`a`' e '`b`' tali che almeno in una posizione a partire dal loro inizio si trovi lo stesso carattere.
Per esempio la stringa `ababcaabaaa` appartiene a L, `aaaaacbb` non vi appartiene, ecc.

Soluzioni

Esercizio 1

- Partendo dalla normale formalizzazione delle macchina a nastro singolo basta modificare la funzione δ nel modo seguente:

  $\delta: Q \times A^h \rightarrow Q \times A^h \times M^h \mid M \in \{ R, L, S\}$
  La definizione di configurazione dovrà tener conto della posizione delle h testine, ad esempio mediante apposite marche.

- La potenza di calcolo ovviamente non aumenta, essendo già massima la potenza della macchina con una sola testina.

- Aumenta invece la capacità di riconoscere linguaggi con minor complessità, come dimostrato dal seguente ragionamento intuitivo:
  Il linguaggio $\{wcw \mid w \in \{a,b\}^*\}$ non è riconoscibile in tempo lineare da una macchina a nastro singolo con una sola testina ma è facilmente riconoscibile con 2 testine in tempo O(n); con una naturale estrapolazione il linguaggio
  $\{wcwcw \ldots \mid w \in \{a,b\}^*$ ripetuto h volte $\}$
  può essere riconosciuto in tempo lineare mediante h+ 1 testine ma non con sole h testine.

Esercizio 2

a. Senza bisogno di rispondere al quesito se le due grammatiche siano equivalenti o meno, si può comunque affermare che il problema di trovare tale risposta è decidibile, dato che la risposta non può che essere SI o NO indipendentemente da qualsiasi altro fattore. Quindi una tra le macchine di Turing che forniscono risposta costante SI e quella che forniscono risposta costante NO risolve il problema di stabilire se le due grammatiche siano equivalenti.

b. Senza analizzare la regolarità del linguaggio di G1, o G1 genera un linguaggio regolare, oppure non lo genera. Se non lo genera, la risposta è sempre negativa, dunque decidibile. Se lo genera, allora il problema si riconduce all'equivalenza tra automi a stati finiti, che è decidibile.

Esercizio 3

Seguendo una schema consueto si usino simboli di variabili per indicare generiche stringhe o generici caratteri, a, b, c essendo invece simboli di costanti corrispondenti ai caratteri dell'alfabeto. Usando inoltre le normali abbreviazioni '.' e '| |' per rappresentare la funzione binaria concatenazione e la funzione "lunghezza di una stringa" la cui definizione viene data qui per scontata, la formula seguente caratterizza tutte e sole le strnghe del linguaggio:

$x \in L \leftrightarrow (\exists w, u, v, z (x = wfucvfz \wedge |w| = |v| \wedge (f = a \vee f = b) \wedge (w,u,v,z \in \{a,b\}^*)$.

Informatica Teorica

Sezione Cremona+Como

Appello del 13 Febbraio 2008

Esercizio 1 (punti 10/30-esimi, 7 se l'automa non è a potenza minima)

Si consideri la seguente grammatica G:

$S \rightarrow AB$

$A \rightarrow \varepsilon \mid CAC \mid D$

$D \rightarrow aDa \mid \varepsilon$

$C \rightarrow cC \mid Cc \mid \varepsilon$

$B \rightarrow bbB \mid b$

Si scriva un automa, preferibilmente a potenza minima, che riconosca il linguaggio L generato da G.

Esercizio 2 (punti 10/30-esimi)

Descrivere (senza necessariamente codificarla nei dettagli) una macchina RAM che riconosce il linguaggio L descritto nell'esercizio 1, e se ne dia la complessità spaziale e temporale a costo costante e a costo logaritmico. E' preferita una macchina che minimizzi entrambe le complessità, a meno della relazione di equivalenza Θ.

Esercizio 3 (punti 10/30-esimi)

1. Dire se è decidibile il problema di stabilire se, data una generica macchina RAM, questa riconosce il linguaggio generato dalla grammatica G dell'esercizio 1.
2. Dire se è decidibile il problema di stabilire se la macchina RAM definita nell'esercizio 2 riconosce il linguaggio L generato dalla grammatica G dell'esercizio 1 oppure no.

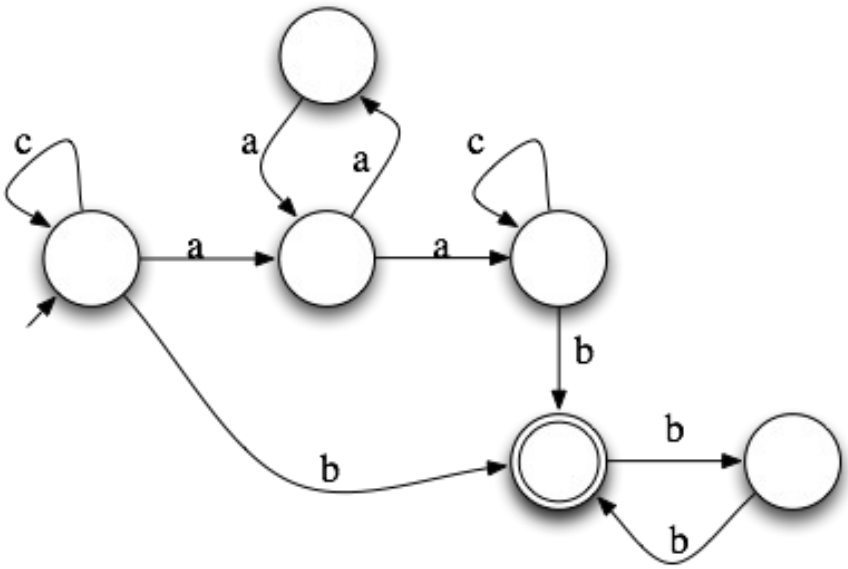
Soluzioni

Esercizio 1

Il linguaggio L(G) è l'insieme

$\{c^*a^{2n}c^*b^{2m+1} \mid n, m >= 0 \}$

che è un linguaggio regolare, anche se G non lo è. Quindi un automa a potenza minima che riconosce L(G) è il seguente automa a stati finiti.



Esercizio 2

Una macchina RAM può simulare un automa a stati finiti con complessità spaziale $\Theta(K)$ e complessità temporale $\Theta(n)$ sia a criterio di costo costante che a criterio di costo logaritmico.

Esercizio 3

1. Il problema è indecidibile in quanto è il classico problema della correttezza.

2. Il problema è decidibile, in quanto la macchina RAM in questione è fissata (e quindi la risposta è chiusa, o SI, o NO).