Informatica Teorica

Seconda prova in itinere e appello del corso del Vecchio ordinamento

29 Giugno 2004

Coloro che sostengono la seconda prova in itinere svolgano gli esercizi 1,2,3 in un'ora e un quarto. Chi sostiene l'appello completo (o la parte di Informatica teorica del corso integrato Algebra + Inf. Teorica) svolga gli esercizi 1, 3 e 4 nello stesso tempo.

NB I punteggi assegnati agli esercizi 1,2,3 hanno valore solo con riferimento alla II prova in itinere.

Si suggerisce di affrontare per primo l'esercizio 3.

Esercizio 1 (Punti 6)

Si dica, giustificando brevemente la risposta, se le seguenti affermazioni sono vere o false:

a) Il problema di stabilire se un generico programma P, codificato in C, eseguito su un generico dato di ingresso x, non terminerà mai la sua esecuzione è semidecidibile.

b) Il problema di stabilire se un generico programma P, codificato in C, eseguito su un generico dato di ingresso x, "andrà in loop", nel senso stretto del termine, ossia ripeterà indefinitamente una stessa sequenza di "mosse" ritornando ogni volta nello stesso stato di memoria, è semidecidibile.

Esercizio 2 (Punti 4)

Il signor Furbetti sta realizzando un archivio mediante una tabella hash. I record dell'archivio hanno una chiave costituita da sequenze di al più 30 caratteri e la tabella ha una dimensione di 5 MB.
Furbetti è particolarmente soddisfatto perché ha trovato una funzione di hash h "perfetta" (ossia è riuscito a dimostrare che la sua funzione h è tale che $x \neq y$ implica $h(x) \neq h(y)$). Accingendosi però a implementare un algoritmo per il calcolo di h non riesce a trovarne uno. Egli viene allora assalito dal dubbio che la sua h non sia calcolabile.
Potete aiutare Furbetti a risolvere il suo dubbio?

Esercizio 3 (Punti 8)

a) Si descriva sinteticamente –senza necessariamente codificarlo in modo completo- un algoritmo per la macchina RAM per riconoscere il linguaggio $L = \{a^n b^n c^n \mid n >= 1\}$. Se ne valutino complessità spaziale e temporale –a meno della relazione Θ-mediante il criterio di costo logaritmico.
b) E' possibile ottenere le stesse complessità (sia spaziale che temporale) mediante una macchina di Turing? Giustificare brevemente la risposta.

Esercizio 4

Si scriva una grammatica che generi il linguaggio $L = \{a^n w \mid w \in \{b,c\}^* \wedge \#(b,w) = \#(c,w) = n, n >= 1\}$, dove la funzione #(i,x) denota il numero di occorrenze del carattere i nella stringa x.

La grammatica così ottenuta è a potenza minima, ossia non esiste una grammatica ad essa equivalente appartenente ad una classe di minor potenza generativa? Giustificare brevemente la risposta.

Soluzioni

Esercizio 1

a) Falsa. Infatti il problema complementare è notoriamente non decidibile ma semidecidibile. Quindi non può essere semidecidibile anche il problema della ***non***-terminazione.

b) Vera. E' concettualmente possibile registrare ogni stato dell'esecuzione del programma e verificare se lo stato attuale è identico a uno già attraversato. Se ciò accade, da quel momento in poi l'esecuzione si ripeterà identicamente all'infinito.

Esercizio 2

h è certamente calcolabile perché ha un dominio finito. Ciò però aiuta solo in parte Furbetti perché non fornisce indicazioni su come trovare l'algoritmo di calcolo di h.

Esercizio 3

a) La RAM aggiorna un contatore per ogni a letto (costo $\Theta(i)$, $i = 1, \ldots, n$). Salva il valore del contatore al termine della lettura degli a in due celle diverse. Decrementa la prima copia per ogni b letto e la seconda copia per ogni c letto successivamente. Al termine entrambi i contatori devono contenere esattamente 0. Ne deriva una complessità spaziale, a criterio logaritmico, $\Theta(\log(n))$ e una temporale $\Theta(n.\log(n))$

b) Sì. Una MdT può simulare il comportamento della RAM esattamente allo stesso modo, usando un nastro per ogni contatore e codificandone il contenuto in numerazione binaria. Per ogni carattere letto l'incremento o il decremento del nastro contatore richiede un numero di mosse al più proporzionale alla lunghezza del nastro, ossia $\Theta(\log(n))$. Perciò complessità spaziale e temporale coincidono con quelle della RAM.
Si noti che un MdT potrebbe riconoscere lo stesso linguaggio con complessità $\Theta(n)$ sia spaziale che temporale.

Esercizio 4

S → aSH | aH

H → BC

BC → CB

CB → BC

B → b

C → c

La grammatica di cui sopra non è non-contestuale (appartiene però alla classe delle grammatiche contestuali, una classe non considerata in questo corso di minor potenza delle grammatiche generali e di maggior potenza di quelle non-contestuali; questa classe genera linguaggi decidibili al contrario delle grammatiche generali). Non è possibile generare lo stesso linguaggio mediante una grammatica non-contestuale perché per riconoscere L occorre tenere aggiornati i conteggi sia delle b che delle c per confrontarli con il numero di a letto inizialmente.

Informatica Teorica

Sezione Mandrioli

Appello del 20 Luglio 2004

Coloro che recuperano la I prova risolvano gli esercizi 1 e 2 tra quelli indicati qui sotto entro un'ora.

Coloro che recuperano la II prova risolvano gli esercizi 3 e 4 tra quelli indicati qui sotto entro un'ora.

Coloro che recuperano entrambe le prove o che sostengono l'appello della semiunità di Informatica Teorica VO risolvano tutti gli esercizi entro due ore.

Coloro che sostengono l'esame del corso integrato con Algebra risolvano uno a scelta tra gli esercizi 1, 2, e l'esercizio 3 entro un'ora.

NB: i punteggi indicati per i vari esercizi fanno riferimento esclusivamente al corso di Informatica Teorica; non hanno valore per il corso integrato.

Attenzione!!!

I voti proposti verranno pubblicati sul sito seguente (e affissi nella bacheca del Dip. di Elettronica e Informazione in forma cartacea):

http://www.elet.polimi.it/upload/mandriol/Didattica/sitoinfteor1.html

Aggiornando il file originariamente esistente dopo le prove in itinere.

Gli studenti avranno tempo due giorni (48 ore, per la precisione) dalla data della pubblicazione per rifiutare il voto proposto, se sufficiente. L'eventuale rifiuto deve essere comunicato via email, facendo uso dell'indirizzo ufficiale del Poliself, non di indirizzo privato! Trascorsi i due giorni, i voti verranno registrati e trasmessi alla segreteria.

La rinuncia al voto proposto è *definitiva e totale*: chi rinuncerà dovrà sostenere il recupero dell'intero esame a settembre.

Così pure, chi risulterà insufficiente dopo il recupero di luglio *dovrà* sostenere a settembre il recupero dell'intero esame.

Esercizio 1 (punti 9/15)

Si specifichi, mediante una formula del prim'ordine un apparato che funziona nel modo seguente:

All'istante 0 esso emette un segnale s, che può essere uno 0 o un 1. Se, *dopo* l'emissione di s e *prima* dello scadere di 3 secondi, viene ricevuto lo stesso segnale in risposta, allora allo scadere del terzo secondo viene emesso un nuovo segnale invertendone il valore (0 se prima si era emesso 1 e viceversa); in caso contrario viene riemesso lo stesso segnale inviato in precedenza. Il comportamento dell'apparato si ripete poi periodicamente con periodo di 3 secondi applicando sempre la stessa regola.

L'apparato emette un segnale soltanto in istanti che sono multipli di 3 secondi.

Si suggerisce di usare i predicati emette(s,t) e riceve(s,t) per indicare, rispettivamente l'emissione e la ricezione del segnale s all'istante t.

NB: il domino temporale può essere sia discreto (ad esempio, l'insieme dei naturali) che continuo (ad esempio, l'insieme dei reali).

Esercizio 2 (punti 8/15)

Con riferimento all'esercizio precedente, si assuma un tempo discreto (la cui unità sia il secondo. Si costruisca un'opportuna macchina astratta, preferibilmente a potenza minima, che abbia come alfabeto di ingresso (rispettivamente, di uscita) la ricezione (risp., emissione) di 0, la ricezione (risp., emissione) di 1, oppure l'assenza di segnale e si comporti come specificato nell'esercizio 1.

NB: essendo il comportamento dell'apparato periodico, senza che sia prevista una sua terminazione, la macchina astratta, di conseguenza, non dovrà prevedere una situazione di arresto, a meno che non debba bloccarsi a causa di errori.

Esercizio 3 (punti 7/15)

Si dica, giustificando brevemente la risposta, quali di queste affermazioni sono vere e quali false:

1. La funzione g(y,x) = (1 se $f_y(x)$ è pari, 0 altrimenti) è calcolabile.
2. La funzione g(y,x) = (1 se $f_y(x)$ è pari, $\perp$ altrimenti) è calcolabile
3. La funzione g(y) = (1 se $f_y(x)$ è pari $\forall x$, 0 altrimenti) è calcolabile
4. La funzione g(y) = (1 se $f_y(x)$ è pari $\forall x$, $\perp$ altrimenti) è calcolabile

NB: Nel caso la risposta a qualcuna delle domande precedenti fosse "Falsa" e se ne fornisse una dimostrazione mediante tecnica diagonale, il punteggio acquisito verrebbe maggiorato di punti 3.

Esercizio 4 (punti 8/15)

Si descriva con sufficiente precisione, ma senza necessariamente specificare ogni dettaglio, come una Macchina di Turing a nastro singolo deterministica (si ricordi che una Macchina "a nastro singolo" è diversa da una macchina a k nastri con k = 1) possa riconoscere il linguaggio L = {ww, w $\in$ {a,b}*} analizzandone la complessità spaziale e temporale (a meno dell'ordine di grandezza determinato dalla Θ–equivalenza).

Soluzioni

Esercizio 1

$$\forall t \forall s(emette(s,t) \rightarrow ($$
$$(s = 0 \vee s = 1) \wedge (\exists k(k \in N \wedge t = 3k)) \wedge$$
$$\exists t_1(t < t_1 < t + 3 \wedge riceve(s,t_1)) \rightarrow emette(neg(s), t + 3) \wedge$$
$$\neg \exists t_1(t < t_1 < t + 3 \wedge riceve(s,t_1)) \rightarrow emette(s, t + 3)$$
$$))) \wedge (emette(0,0) \vee emette(1,0))$$

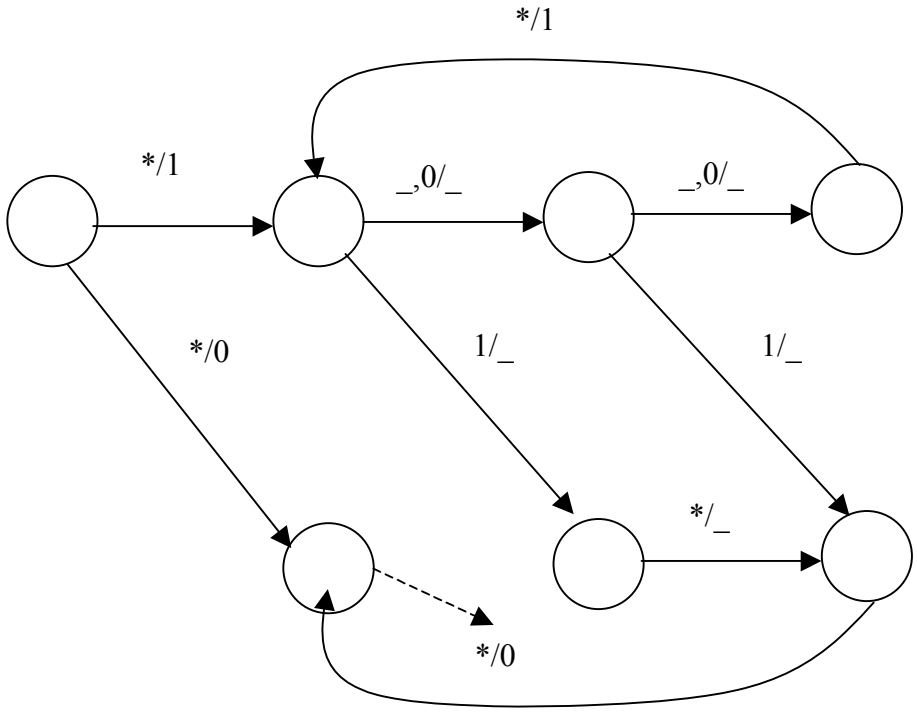
dove la lettera funzionale *neg*, che inverte i valori 0 e 1, è definita così:

$neg(s) = (s+1) \bmod 2$.

(mod è il ben noto operatore "resto della divisione intera").

Esercizio 2

La figura sottostante descrive parzialmente un automa a stati finiti che si comporta secondo le specifiche. La parte omessa, cui si accede attraverso la freccia tratteggiata, tratta il caso dell'emissione iniziale di uno 0 in modo simmetrico.



Legenda: * indica un qualsiasi simbolo; _ indica assenza di segnale; a, b/ etichetta una transizione che avviene in conseguenza dll'input a o dell'input b.

Esercizio 3

1. Falsa. E' possibile ridurre il problema della terminazione al problema dato assegnando al risultato della funzione il valore 2 a valle del calcolo di una funzione generica $f_z(w)$di cui si voglia stabilire la terminazione.

   Si può sfruttare per la dimostrazione anche il teorema di Rice nel seguente modo: si fissi ad arbitrio un valore intero k, e si definisca la funzione h(y) = (1 se g(y,k) è pari, 0 altrimenti). h(x) è la funzione caratteristica dell'insieme delle macchine di Turing che calcolano funzioni con valore pari nel punto k. Questo insieme non è né l'insieme vuoto, né l'insieme universo, e non è dunque decidibile, quindi la sua funzione caratteristica non è calcolabile. Se g(y,x) fosse calcolabile, lo sarebbe anche h(y), il che porterebbe a una contraddizione.

   La dimostrazione può anche essere costruita in forma diretta usando la tecnica diagonale nel modo seguente:
   Si definisca h(x) = Se $f_x(x)$ non è pari allora 2, $\perp$ altrimenti. Se g fosse calcolabile lo sarebbe anche h. Quindi h dovrebbe essere = f_{x0} per qualche x0. Ma h(x0) non pari implicherebbe h(x0) = 2 e h(x0) = 2 implicherebbe h(x0) non pari.
2. Vera: basta fare il run di $f_y(x)$.
3. Falsa grazie al teorema di Rice: infatti la g(y) è la funzione caratteristica dell'insieme di tutte (e sole) le macchine di Turing che computano funzioni totali con valore sempre pari. Questo insieme non è né l'insieme vuoto, né l'insieme universo, non è quindi decidibile, e la sua funzione caratteristica non è computabile.
4. Falsa. Infatti questa è la funzione semicaratteristica dell'insieme di tutte (e sole) le macchine di Turing che computano funzioni totali con valore sempre pari. Se tale funzione fosse calcolabile, ciò vorrebbe dire che tale insieme di funzioni è RE. Questo è tuttavia impossibile. Si prenda infatti una funzione calcolabile f(x) generica; si può facilmente calcolare la funzione f'(x) = 2f(x), la quale ha lo stesso dominio di f(x) ed ha, laddove è definita, valore pari. Di conseguenza, a partire dall'insieme di tutte le funzioni computabili a valori solo pari è possibile costruire l'insieme di tutte le funzioni computabili semplicemente dividendo per 2 le immagini. Da questo deriva che se l'insieme di tutte e sole le funzioni totali a valori pari fosse RE, anche l'insieme di tutte e sole le funzioni totali lo sarebbe. E' tuttavia ben noto (da lezione) che l'insieme di tutte e sole le funzioni totali non è RE, di conseguenza non lo è neanche l'insieme di funzioni di partenza.

Esercizio 4

La macchina M deve in primo luogo individuare la metà della stringa in ingresso. Per ottenere ciò, mediante una prima passata memorizza – ad esempio in unario – la lunghezza della stringa, alla sua destra (durante questa passata la macchina dovrà tenere traccia del carattere fino a dove si è contato, per esempio cambiandolo da a ad a', e da b a b', e riconvertendolo nel carattere originale prima di passare a contare il prossimo carattere). Ciò richiede un tempo $\Theta(n)$ per ogni carattere letto e quindi $\Theta(n^2)$ per l'intera stringa. Indi, per ogni coppia di caratteri della stringa che memorizza la lunghezza dell'input, sposta di una posizione un'opportuna marca all'interno della stringa di input (ad esempio sostituendo il carattere a con il carattere a' e b con b'). Al termine la marca si troverà a metà della stringa di ingresso. Anche questa macro-operazione richiede un tempo $\Theta(n^2)$ ($\Theta(n)$ per ogni coppia di caratteri).

A questo punto M può procedere a confrontare, uno per uno il primo carattere dell'input con il carattere in corrispondenza della marca che segna la metà; il secondo con il successivo. Anche questo confronto richiede un tempo $\Theta(n)$ per ogni carattere e quindi $\Theta(n^2)$ per l'intera stringa.

In conclusione l'intero procedimento ha una complessità temporale $\Theta(n^2)$ e spaziale $\Theta(n)$.

Informatica Teorica

Sezione Mandrioli

Appello del 15 Settembre 2004

Attenzione!!!

I voti proposti verranno pubblicati sul sito seguente:

http://www.elet.polimi.it/upload/mandriol/Didattica/sitoinfteor1.html

Aggiornando il file originariamente esistente dopo le prove in itinere.

Gli studenti avranno tempo due giorni (48 ore, per la precisione) dalla data della pubblicazione per rifiutare il voto proposto, se sufficiente. L'eventuale rifiuto deve essere comunicato via email, facendo uso dell'indirizzo ufficiale del Poliself, non di indirizzo privato! Trascorsi i due giorni, i voti verranno registrati e trasmessi alla segreteria.

Per evitare il ripetersi di recenti disguidi, probabilmente dovuti a malfunzionamenti del servizio di posta elettronica, il docente spedirà a ogni studente "rinunciatario" un esplicito messaggio di "ricevuta". In caso di mancata ricezione di tale ricevuta si consiglia di contattare il docente telefonicamente o di avvisare la segreteria didattica del DEI.

Chi risulterà insufficiente dopo il recupero di settembre o rinuncerà al voto proposto *dovrà* sostenere il recupero dell'intero esame nella sessione invernale, oppure ri-iscriversi al corso per l'anno accademico 2004/05.

Esercizio 1 (punti 8)

Si specifichi, mediante una formula del prim'ordine il ripetersi periodico di un segnale istantaneo (ossia che si verifica in un istante di tempo isolato) con periodo 2 unità di tempo a partire dall'istante 5. Prima dell'istante 5 e in tutti gli altri istanti il segnale non si verifica.

Esercizio 2 (punti 9)

Si costruisca un automa a pila equivalente alla grammatica seguente (ossia che riconosca il linguaggio da essa generato):

S → aAB

A → aAA | bAB | c

B → bBB | aBA | c

L'automa così costruito è deterministico? In caso negativo, è possibile costruirne uno equivalente deterministico?

Esercizio 3 (punti 7)

Si dica, giustificando brevemente la risposta, quali delle seguenti affermazioni sono vere e quali false:

1. L'insieme dei programmi C la cui terminazione è garantita a priori per ogni valore dei dati in ingresso è ricorsivamente enumerabile.
2. L'insieme dei programmi C la cui terminazione è garantita a priori per ogni valore dei dati in ingresso è ricorsivo.

Esercizio 4 (punti 7)

In passato alcuni studenti, nello svolgimento di prove d'esame, hanno costruito una valutazione di complessità per macchine di Turing dichiarando di utilizzare il criterio di costo logaritmico. Si spieghi brevemente perché ciò è un errore e perché invece tale criterio è utile e spesso necessario nella valutazione di complessità relativa a un modello di calcolo come la RAM.

Soluzioni

Esercizio 1

Codificando l'occorrenza del segnale all'istante t mediante il predicato s(t), la formula seguente specifica il comportamento desiderato:

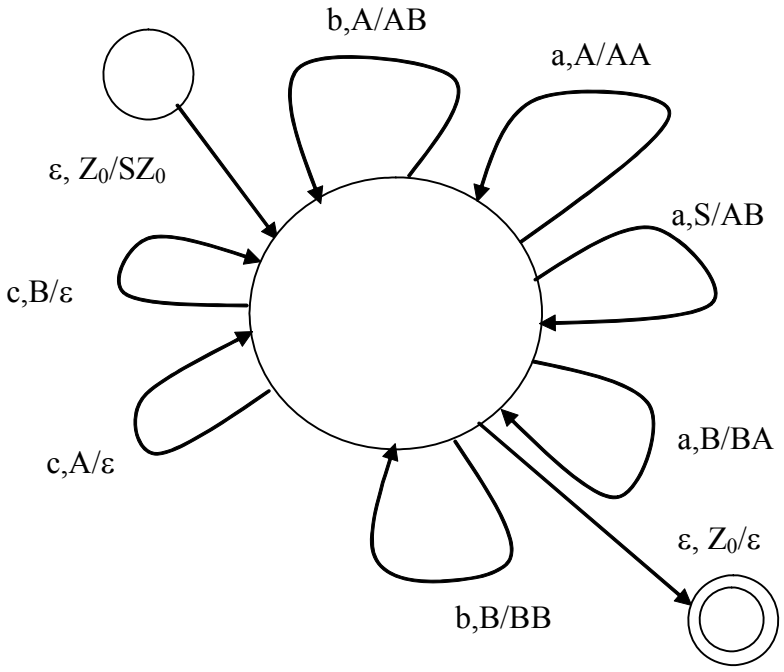
$$s(5) \wedge$$
$$\forall t((t < 5) \rightarrow \neg s(t) \wedge$$
$$(t \geq 5 \rightarrow (s(t) \leftrightarrow (\exists k(k \in N \wedge t = 5 + 2k)))))$$

Si noti che i primi termini possono essere omessi, dal momento che sono implicati dall'ultimo, ottenendo così la formula equivalente e più sintetica:

$$\forall t\ (\mathrm{s}(t)\ \leftrightarrow\ (\exists k\ (k \in \mathrm{N} \wedge t = 2k + 5)))$$

Esercizio 2

La figura sottostante descrive un automa a pila (deterministico) equivalente alla grammatica fornita.



Legenda: la stringa a destra del carattere / viene depositata sulla pila mettendo il carattere più a sinistra in alto.

Esercizio 3

Entrambe le affermazioni sono false. Infatti è noto (Teorema 2.10 del testo) che un insieme di indici di Macchine di Turing che calcolino tutte e sole le funzioni calcolabili e totali non è ricorsivamente enumerabile. Se perciò l'insieme suddetto di programmi C fosse ricorsivamente enumerabile, potremmo automaticamente calcolare per ogni tale programma un corrispondente indice di MT che calcola la stessa funzione del programma, e quindi ricavare da una enumerazione ricorsiva dell'insieme dei programmi una corrispondente enumerazione ricorsiva di ***un*** insieme di indici di MT che calcolano tutte e sole le funzioni calcolabili e totali.

A maggior ragione, se tale insieme non è ricorsivamente enumerabile, non può essere ricorsivo.

Esercizio 4

Le operazioni astratte della MT (transizioni) sono effettivamente elementari e possono essere realizzate a "risorse costanti" con qualunque tecnologia di base. Al contrario le operazioni astratte della RAM, più di alto livello, sono realizzate mediante opportuno HW la cui complessità, intermini di circuiteria e utilizzata e tempo impiegato è funzione –appunto logaritmica- della dimensione dei dati in ingresso.

Informatica Teorica Sezione Mandrioli

Appello del 5 Febbraio 2005

Attenzione!!!

I voti proposti verranno pubblicati sul sito seguente:

http://www.elet.polimi.it/upload/mandriol/Didattica/sitoinfteor1.html

Gli studenti avranno tempo due giorni (48 ore, per la precisione) dalla data della pubblicazione per rifiutare il voto proposto, se sufficiente. L'eventuale rifiuto deve essere comunicato via email, facendo uso dell'indirizzo ufficiale del Poliself, non di indirizzo privato! Trascorsi i due giorni, i voti verranno registrati e trasmessi alla segreteria.

Per evitare il ripetersi di recenti disguidi, probabilmente dovuti a malfunzionamenti del servizio di posta elettronica, il docente spedirà a ogni studente "rinunciatario" un esplicito messaggio di "ricevuta". In caso di mancata ricezione di tale ricevuta si consiglia di contattare il docente telefonicamente o di avvisare la segreteria didattica del DEI.

Chi risulterà insufficiente dopo il recupero di febbraio o rinuncerà al voto proposto *dovrà* ri-iscriversi al corso per l'anno accademico 2004/05.

Esercizio 1 (punti 8)

Si specifichi, mediante una formula del prim'ordine il seguente comportamento di un ipotetico sistema:

Tutte le volte che, in un certo istante, si verifica l'evento A, dopo esattamente k unità di tempo si verifica l'evento B, a meno che, contemporaneamente ad A, non si verifichi anche l'evento C. In tal caso, B non si verifica dopo k unità di tempo, bensì dopo k+1.

Esercizio 2 (punti 8)

Si considerino i seguenti linguaggi:

$L1 = \{a^n b^{2m} \mid n, m >= 1\}$, $L2 = \{a^n b^{3n} \mid n >= 0\}$

Si costruisca una grammatica G che generi il linguaggio $L = L1 \cap L2$. E' preferibile una grammatica a potenza minima, ossia regolare se ne esiste una, non contestuale se ne esiste una ma non ne esiste una regolare, ecc. Nel caso, si spieghi brevemente perché non esstono grammatiche appartenenti a classi inferiori a quella proposta.

Esercizio 3 (punti 6)

Si dica, giustificando brevemente la risposta, se è decidibile il problema di stabilire se il linguaggio L di cui all'esercizio 2 è vuoto o no.

Esercizio 4 (punti 8)

Si dica, giustificando brevemente la risposta, qual è l'ordine di grandezza minimo per il riconoscimento del linguaggio L di cui all'esercizio 2 nei due casi in cui si usi

a) Una macchina di Turing.

b) Una RAM, assumendo il criterio di costo logaritmico.

Soluzioni

Esercizio 1

Codificando l'occorrenza di un generico evento E all'istante t mediante il predicato E(t), la formula seguente specifica il comportamento desiderato:

$$\forall t((A(t) \wedge \neg C(t)) \rightarrow B(t+k)) \wedge$$
$$((A(t) \wedge C(t)) \rightarrow (\neg B(t+k) \wedge B(t+k+1)))$$

Si noti che la formula di cui sopra porta ad una contraddizione se non solo all'istante *t*, ma anche all'istante *t-1* accadono sia A che C (cosa che la specifica a parole in principio non vieta). In questo caso, infatti, il secondo congiunto applicato a *t-1* vincola B ad accadere a *t+k*, mentre lo stesso congiunto riferito all'istante *t* proibisce che B accada a *t+k*. Una formula che evita questo fenomeno, reinterpretando la specifica a parole è la seguente:

$$\forall t((A(t) \wedge \neg C(t)) \rightarrow B(t+k)) \wedge$$
$$((A(t) \wedge C(t)) \rightarrow B(t+k+1))$$

Esercizio 2

Il linguaggio L = L1 $\cap$ L2 è l'insieme $\{a^{2n}b^{6n} | n >= 1\}$. Infatti, L1 è il linguaggio delle stringhe con un numero *pari* di 'b' che seguono un numero qualunque di 'a'. Tra le stringhe di L2, del tipo '$a^n b^{3n}$', quelle che appartengono anche a L1 sono esattamente quelle con un numero pari di 'b', ossia quelle nella forma '$a^{2n}b^{3\cdot(2n)} = a^{2n}b^{6n}$'.

L è generato dalla seguente grammatica non contestuale (e da nessuna grammatica regolare essendo necessaria una pila per il suo riconoscimento):

S → AB | ASB

A → aa

B → bbbbbb

Esercizio 3

L non è vuoto: contiene aabbbbbb, ecc. Quindi il problema L = ∅ è *deciso* e perciò *decidibile*.

Esercizio 4

Una MT che simuli un automa a pila deterministico può facilmente riconoscere L in tempo lineare. Una RAM deve necessariamente aggiornare un contatore (il cui contenuto è un numero proporzionale ad n) per ogni carattere letto. Perciò, a criterio di costo logaritmico, la sua complessità è necessariamente $\Theta(n.\log(n))$.

Informatica Teorica

Prima prova in itinere - 5 Maggio 2005

Esercizio 1 (5/13 punti)

Si consideri il linguaggio:

$L \subseteq \{a,b,c\}^*$ costituito dalle stringhe in cui il numero di a sia uguale al numero di b + 5 e il numero di c sia pari.

Si costruisca una macchina astratta che riconosca L. Tra le diverse macchine astratte che riconoscono L sono preferite macchine "a potenza minima" ossia appartenenti alla categoria di automi a minor potenza riconoscitiva possibile.

Esercizio 2 (4/13 punti)

E' noto che le due seguenti definizioni di grammatica regolare sono equivalenti, ossia generano la stessa classe di linguaggi:

1) $\forall\ \alpha \rightarrow \beta \in P, |\alpha| = 1, \beta \in \ V_N . V_T \ \cup V_T$

2) $\forall\ \alpha \rightarrow \beta \in P, |\alpha| = 1, \beta \in \ V_T . V_N \ \cup V_T$

La seguente ulteriore modifica della definizione è anch'essa equivalente alle precedenti?

3) $\forall\ \alpha \rightarrow \beta \in P, |\alpha| = 1, \beta \in \ V_T . V_N \ \cup V_T \cup V_N . V_T$

Giustificare brevemente la risposta.

Esercizio 3 (5/13 punti)

Si definisca in maniera matematicamente precisa la seguente versione arricchita di automa a pila.

L'automa, oltre a tutte le caratteristiche dei normali automi a pila (nondeterministici), può anche, ad ogni mossa, esaminare il simbolo contenuto nel fondo della pila e sostituirlo con un altro simbolo.

Se ne formalizzino anche le regole di funzionamento (per esempio mediante le normali nozioni di configurazione e transizione tra configurazioni) e di riconoscimento di stringhe.

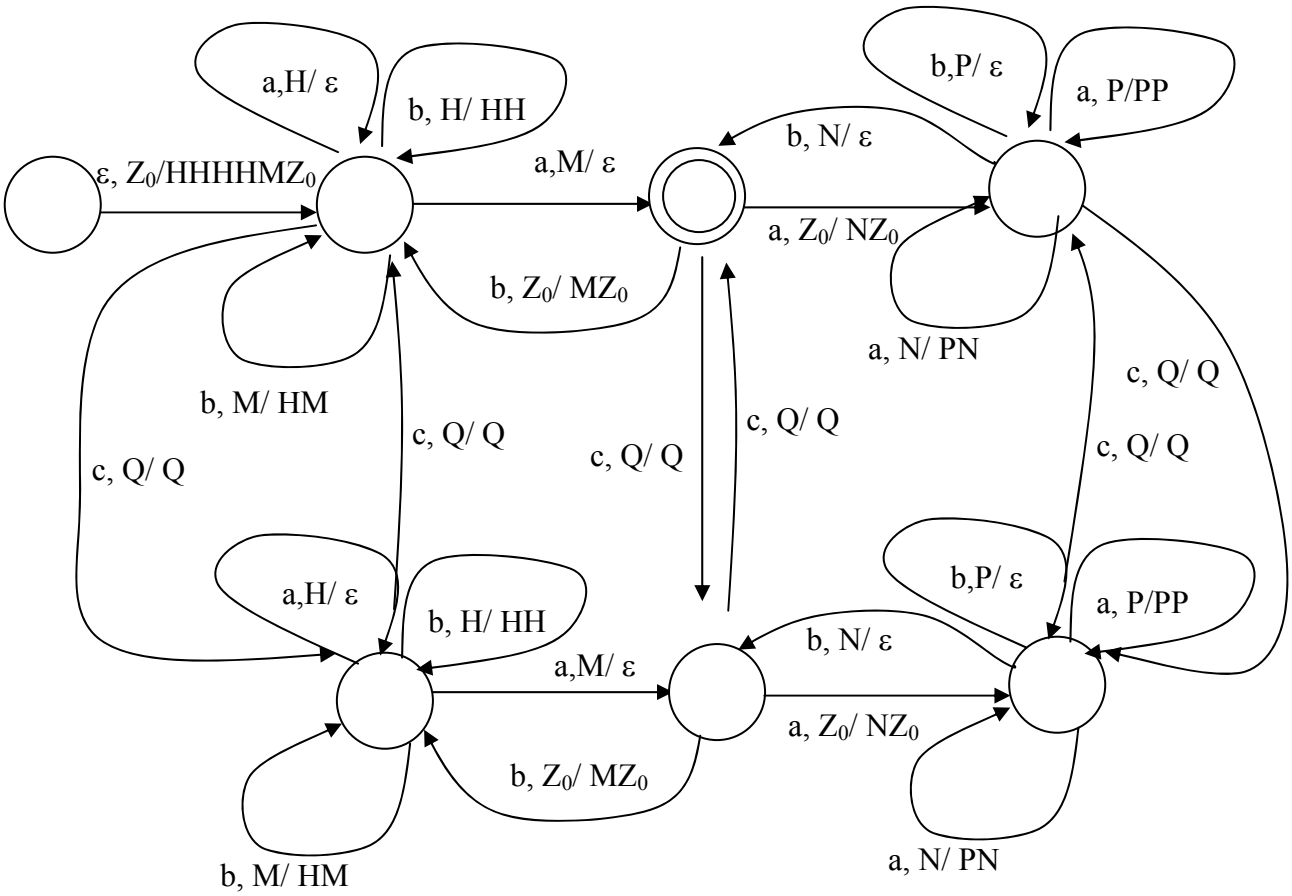
NB: nel caso la precedente definizione informale non risulti sufficientemente precisa in alcuni dettagli, si operino opportune scelte, spiegandone le ragioni, per giungere ad una formalizzazione precisa.

Si dica poi, giustificando brevemente la risposta, se l'automa, così arricchito aumenta la potenza riconoscitiva rispetto agli automi a pila tradizionali. In caso positivo, raggiunge la potenza delle macchine di Turing?

Soluzioni

Esercizio1

Il linguaggio L è riconosciuto dal seguente automa a pila deterministico:



Legenda: il simbolo Q indica un qualsiasi simbolo $\in \Gamma$

Evidentemente non è possibile riconoscere L con una macchina a stati finiti, a causa della necessità di conteggio illimitato sul numero di a e b.

Esercizio2

No

La grammatica seguente soddisfa la nuova definizione ma genera il linguaggio

$\{a^n b^n \mid n >= 1\}$ che notoriamente non è regolare.

S → aA

A → Sb | b

Esercizio 3

Un automa riconoscitore nondeterministico a pila, che possa accedere e modificare anche il fondo della pila può essere formalizzato mediante una 7-pla
<Q, I, Γ, δ, q_0, Z_0, F >in cui tutti i simboli hanno lo stesso significato di quelli relativi agli automi a pila tradizionali, con l'unica differenza che la funzione δ viene estesa nel modo seguente:

$$\delta : Q \times (I \cup \{\varepsilon\}) \times \Gamma \times \Gamma \to \wp_F (Q \times \Gamma^* \times \Gamma)$$

dove δ (q, i, A, B) = {<q', α, C>} significa che l'automa, trovandosi nello stato q, leggendo i dal nastro di ingresso (oppure ε nel caso di ε –mossa), A dalla cima della pila, B dal fondo della medesima, in maniera nondeterministica tra le diverse terne possibili, si porta in q', scrive α sulla cima della pila al posto di A e C sul fondo al posto di B.
Una configurazione c dell'automa è una 3-pla c = <q, x, γ>, come per l'automa a pila tradizionale; la relazione di transizione è, parzialmente, definita dalle regole seguentise | γ | ≥ 2

< q, i.y, AηH > |-- <q', y, α ηM> se e solo se <q', α, M> ∈ δ (q, i, A, H)

se | γ | = 1

<q, i.y, A> |-- <q', y, αM> se e solo se <q', αM, M> ∈ δ (q, i, A, A)

NB: se <q', αH, M> ∈ δ (q, i, A, A) con H ≠ M, la transizione non è applicabile. Sono però possibili anche altre definizioni: ad esempio che la macchina dia priorità alla riscrittura sulla cima della pila rispetto a quella sul fondo; oppure che comunque la transizione sia possibile solo se il contenuto della pila contiene almeno due caratteri.

(la parte rimanente della definizione della relazione di transizione viene data per scontata).

Infine x è accettata dall'automa se e solo se

< q_0, x, Z_0 > |-- <q', ε, γ>, per qualche γ, con q' ∈ F, come nel caso dell'automa a pila tradizionale.

La potenza dell'automa non aumenta rispetto all'automa a pila tradizionale, poiché l'informazione aggiuntiva che esso può trattare è un'informazione finita (il simbolo in fondo alla pila) e può quindi essere memorizzata attraverso gli stati finiti dell'organo di controllo.

Informatica Teorica

Seconda prova in itinere - 29 Giugno 2005

Esercizio 1 (punti 6/17-esimi)

Si *specifichi,* mediante opportune pre- e post-condizioni il seguente requisito per un frammento di programma FP:

Sia dato un array *a* di *n* < NMAX interi positivi; gli *n* interi sono memorizzati nelle prime posizioni di *a*, a partire da *a*[0]; dopo di essi si trova il valore convenzionale 0 usato come "terminatore".

FP deve produrre un nuovo array *b*, che sia il risultato dell'ordinamento di *a* in ordine crescente e dell'eliminazione di eventuali ripetizioni. FP usa *a* come variabile "read-only". Anche la sequenza dei valori di *b* deve essere terminata da uno 0.

Suggerimento: si consiglia di introdurre la funzione *lungh(z)* definita su array di NMAX interi che fornisce il numero di valori di *z* diversi da 0 che precedono il primo 0 (in caso di assenza di uno 0, *lungh(z)* = -1 per convenzione).

Esercizio 2 (punti 7/17-esimi)

Si consideri il programma seguente, codificato in un ipotetico linguaggio ispirato al C.

```
int g(int p); int f(char t);
main ()
{  char a[20]; int i, z;
   scanf(a); z = 0
   while (i = 0; g(i) != 20; i++) do z = z + f(a[i]);
   printf(z);
}
```

Si assuma che f e g siano due sottoprogrammi "esterni" la cui terminazione sia garantita per ogni valore dei parametri di ingresso.

Si dica, motivando brevemente la risposta, se è decidibile il problema di stabilire se il programma main di cui sopra terminerà o meno la sua esecuzione in corrispondenza di un generico –non fissato!- valore del dato di input a.

Esercizio 3 (punti 7/17-esimi)

Si consideri una sequenza S di elementi che contengano informazione codificabile in un numero limitato a priori di celle di memoria. Si assuma che S si trovi già memorizzata nella memoria di una macchina RAM secondo una delle due tradizionali tecniche:

a) come array di celle consecutive

b) come lista di elementi collegati tra loro mediante puntatori

Si valuti la complessità asintotica sia spaziale che temporale, a meno della relazione Θ, di un algoritmo di ricerca sequenziale nei due casi, facendo uso sia del criterio di costo costante che del criterio di costo logaritmico; si spieghi brevemente come si giunge ai risultati proposti, senza bisogno peraltro di codificare in dettaglio l'algoritmo. Si indichi con precisione il parametro rispetto al quale viene misurata la dimensione dei dati di ingresso, in funzione della quale viene fornita la funzione di complessità.

Esercizio 3-bis, Facoltativo (punti 4/17-esimi, valido solo se preceduto da soluzione corretta della parte obbligatoria)

Come cambiano le valutazioni di complessità di cui sopra immaginando che il medesimo algoritmo venga eseguito da una Macchina di Turing?

Soluzioni

Esercizio 1

In primo luogo si definisca la funzione lungh(z) mediante una formula del tipo

$$lungh(z) = n \leftrightarrow ((0 \le n < NMAX \wedge z[n]=0 \wedge \neg\exists i(0 \le i < n \wedge z[i] = 0)) \vee (n = -1 \wedge \neg\exists i(0 \le i < NMAX \wedge z[i] = 0)))$$

A questo punto il requisito richiesto può essere specificato dalla seguente coppia per-post-condizione:s

$$\{\exists n\,(lungh\,(a) = n \quad \wedge \forall i\,(0 \le i < n \rightarrow a[i] > 0))\}$$
$$FP$$
$$\{\exists m\,(lungh\,(b) = m \quad \wedge$$
$$\forall i\,(0 \le i < m-1 \rightarrow b[i] < b[i+1]) \quad \wedge$$
$$\forall i\,(0 \le i < lungh\,(a) \rightarrow \exists j\,((0 \le j < m) \wedge (a[i] = b[j]))) \quad \wedge$$
$$\forall j\,(0 \le j < m \rightarrow \exists i\,((0 \le i < lungh\,(a)) \wedge (a[i] = b[j])))\}$$

Esercizio 2

La risposta è positiva

Premessa. Perché il quesito sia significativo occorre ovviamente assumere una macchina astratta che presupponga un dominio totale di dati infinito: in questo caso, il tipo **int** deve essere l'insieme matematico degli interi. Altrimenti, come noto, la macchina astratta diventa automaticamente un automa a stati finiti e quindi la sua terminazione decidibile.

Il ***dominio*** dei dati di ingresso *del programma in oggetto* è però ***finito***, essendo l'insieme degli array di 20 caratteri. Per ognuno dei possibili valori di ingresso la risposta al quesito è semplicemente un sì o un no; esiste quindi una macchina di Turing che fornisce tale risposta, per ogni valore del dato di ingresso a; di conseguenza esiste un numero finito di macchine di Turing, che, opportunamente combinate tra loro, forniscono la risposta corretta per ogni valore del dato di ingresso a; la combinazione di tali macchine può dar luogo ad un'unica macchina che, sul dominio finito dei possibili valori di a, fornisce l'insieme finito delle risposte corrette. Ovviamente, ciò non garantisce la conoscenza di tale macchina; ne garantisce esclusivamente l'esistenza.

Equivalentemente e più sinteticamente si può osservare che il problema posto (la decidibilità dell'halt del programma rispetto ai diversi valori di ingresso) è formalizzato da una funzione a dominio finito (l'insieme degli array di 20 caratteri) e codominio {T, F}. Tale funzione è quindi riducibile a una tabella finita, e quindi calcolabile.

Si noti anche che la terminazione del programma, essendo garantita la terminazione di una singola esecuzione del corpo del ciclo, dipende solo dalla valutazione della funzione g, che, o non produce mai 20, oppure prima o poi deve produrre il valore 20 per qualche valore di i. Ciò è indipendente dal valore del dato di ingresso a. Quindi la tabella di cui sopra, non dipendendo dai dati di ingresso sarà costituita da tutti T o tutti F.

Esercizio 3

Si indichi con n il numero di elementi della sequenza. Poiché ogni elemento contiene informazione codificabile in un numero limitato a priori di celle di memoria, ossia in un numero fissato di byte, è del tutto indifferente assumere come parametro di dimensione dei dati in ingresso il numero n di elementi di S o le celle di memoria utilizzate per memorizzare S.

La memorizzazione mediante array richiede una quantità di memoria $\Theta(n)$ sia a criterio di costo costante che a criterio logaritmico. La memorizzazione mediante puntatori invece richiede di memorizzare un intero come puntatore per ogni elemento di S. Essendo il numero di elementi di S illimitato, una quantità di memoria richiesta sarà $\Theta(n)$ a criterio di costo costante ma $\Theta(n \cdot \log(n))$ a criterio logaritmico.

La complessità temporale risulterà invece $\Theta(n)$ a criterio di costo costante e $\Theta(n \cdot \log(n))$ a criterio logaritmico per entrambi i tipi di memorizzazione.

Esercizio 3 bis (parte facoltativa)

Simulare un array mediante un nastro di macchina di Turing richiede ovviamente uno spazio $\Theta(n)$. $\Theta(n)$ risulta pure la complessità temporale di un algoritmo di ricerca sequenziale.

La memorizzazione di un puntatore in un nastro di macchina di Turing richiede la codifica (ad esempio in binario) di un numero i e quindi un numero $\Theta(\log(i))$ di celle. La complessità spaziale risulta perciò $\Theta(n \cdot \log(n))$ e l'accesso a un generico elemento richiede pure $\Theta(n \cdot \log(n))$, rendendo perciò la complessità totale $\Theta(n^2 \cdot \log(n))$

Informatica Teorica

Recupero della prima prova in itinere - 29 Giugno 2005

Esercizio 1 (punti 6+3/13-esimi)

Si scrivano degli automi che riconoscano i seguenti linguaggi (k è fissato e > 1):

$L_1 = \{ x \in \{a,b\}^* \mid x = a^n b^{kn}, n > 0\}$

$L_2 = \{ x \in \{a,b\}^* \mid x = a^n b^{n^k}, n > 0\}$

Parte facoltativa. Si scriva un automa che riconosca il seguente linguaggio (k è fissato e > 1):

$L_3 = \{ x \in \{a,b\}^* \mid x = a^n b^{\log_k(n)}, n > 0\}$

NB: non è necessario che gli automi vengano disegnati completamente. Essi devono però essere descritti con sufficiente dettaglio da poter capire quali sono i loro stati e le loro transizioni.

Esercizio 2 (punti 6/13-esimi)

Si scriva una grammatica che generi il seguente linguaggio:

$L = \{ x \in \{a,b,c\}^* \mid \#x_a = 2n, \#x_b = 4, \#x_c = n, n > 0\}$

dove con la scrittura $\#x_\alpha$ si intende il numero di occorrenze del carattere α nella stringa x.

La grammatica scritta è a potenza minima tra quelle che riconoscono L? Si giustifichi la risposta.

Soluzioni

Esercizio 1

Per riconoscere il linguaggio L_1 è sufficiente un automa a pila deterministico che funziona in questa maniera: per ogni a letta, esso mette k A sulla pila. Quando poi arriva a leggere le b, per ogni b letta cancella una B dalla pila. Se l'automa arriva in fondo alla stringa con solo il carattere Z_0 sulla pila, la stringa viene accettata, altrimenti viene rifiutata.

Per riconoscere il linguaggio L_2, invece, è necessaria una MT. Una MT (a 2 nastri) che riconosce L2 può essere fatta in questa maniera.
Essa innanzi tutto legge le a e memorizza nel primo nastro un numero di A pari al numero di a lette, mette nel secondo nastro altrettante B. Quindi ripete le seguenti operazioni k-1 volte:

- ad ogni B del secondo nastro sostituisce un numero di B pari al numero di A nel primo nastro (creando per ogni B da "moltiplicare" ogni volta lo spazio necessario nel secondo nastro, spostando le celle di n posti a partire dal fondo).

A questo punto legge le b dal nastro di ingresso, cancellando una B dal terzo nastro per ogni b letta. Se alla fine, quando si è letta l'ultima b, il terzo nastro è vuoto, la MT accetta la stringa, portandosi in uno stato finale.

Parte facoltativa.

Anche per riconoscere il linguaggio L_3 è necessaria una MT. La MT (a 2 nastri) può essere fatta in questa maniera.
Essa innanzi tutto legge le a e memorizza nel primo nastro un numero di A pari al numero di a lette. Quindi esegue passate successive sul primo nastro fino a che il primo nastro rimane vuoto, e ad ogni passata fa quanto segue:

- ad ogni k A del primo nastro sostituisce una sola A; se in fondo al nastro rimane un numero di A minore di k, le ultime A vengono eliminate e basta (di fatto ciò corrisponde a dividere il numero di A per k);
- scrive una B nel secondo nastro (a meno che nel primo nastro ci fossero meno di k A).

A questo punto legge le b dal nastro di ingresso, cancellando una B dal terzo nastro per ogni b letta. Se alla fine, quando si è letta l'ultima b, il terzo nastro è vuoto, la MT accetta la stringa, portandosi in uno stato finale.

Una maniera alternativa di risolvere il problema poteva essere di memorizzare sul primo nastro il numero di a lette codificando il numero in base k. $\log_k(n)$ a questo punto è banalmente il numero di cifre necessarie per codificare n in base k (meno uno), e quindi per verificare che ci siano $\log_k(n)$ b basta cancellare una cifra della codifica in base k per ogni b letta.

Esercizio 2

Una grammatica che genera il linguaggio L è la seguente:

$S \rightarrow AACS \mid X$

$X \rightarrow BBBB$

$AC \rightarrow CA$

$CA \rightarrow AC$

$AB \rightarrow BA$

$BA \rightarrow AB$

$CB \rightarrow BC$

$BC \rightarrow CB$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

Tale grammatica non è a potenza minima. Il linguaggio può essere riconosciuto da un automa a pila (deterministico), per cui la grammatica a potenza minima che lo genera è una grammatica non contestuale.

Una possibile grammatica non contestuale che genera il linguaggio L è la seguente:

$S \rightarrow Saac \mid Saca \mid Scaa \mid aSac \mid aSca \mid cSaa \mid aaSc \mid acSa \mid caSa \mid aacS \mid acaS \mid caaS \mid bS_1$

$S_1 \rightarrow S_1aac \mid S_1aca \mid S_1caa \mid aS_1ac \mid aS_1ca \mid cS_1aa \mid aaS_1c \mid acS_1a \mid caS_1a \mid aacS_1 \mid acaS_1 \mid caaS_1 \mid bS_2$

$S_2 \rightarrow S_2aac \mid S_2aca \mid S_2caa \mid aS_2ac \mid aS_2ca \mid cS_2aa \mid aaS_2c \mid acS_2a \mid caS_2a \mid aacS_2 \mid acaS_2 \mid caaS_2 \mid bS_3$

$S_3 \rightarrow S_3aac \mid S_3aca \mid S_3caa \mid aS_3ac \mid aS_3ca \mid cS_3aa \mid aaS_3c \mid acS_3a \mid caS_3a \mid aacS_3 \mid acaS_3 \mid caaS_3 \mid b$

Informatica Teorica

Appello del 7 – 7 2005

Esercizio 1 (punti 15)

Si consideri il seguente linguaggio L_{stud} formato dalle parole *p* così formate: $\{p = N^6V^k \mid N \in 0..9, V \in \{a, b, c, d\}, k <= 30\}$ (cioè composte da 6 cifre, seguite da al massimo 30 caratteri, in cui ogni carattere è o *a*, o *b*, o *c*, o *d*), in cui le 6 cifre rappresentato i numeri di matricola degli studenti immatricolati al Politecnico di Milano nell'anno 2001/2002, e i caratteri seguenti i voti conseguiti negli esami superati (*d* per un voto da 18 a 21, *c* per uno da 22 a 24, *b* per uno da 25 a 27, *a* per uno da 28 a 30).

Si dica, giustificando brevemente la risposta:

a. quale è l'automa a potenza minima in grado di riconoscere il linguaggio L_{stud};

b. se è decidibile il problema di stabilire se ci sono 2 studenti diversi con gli stessi voti;

c. quale è la complessità minima per risolvere il problema di appartenenza di una stringa al linguaggio L_{stud}.

Esercizio 2 (punti 8)

Si specifichi mediante opportune formule del prim'ordine il seguente comportamento di un ipotetico sistema.

Tutte le volte che, in un certo istante, si riceve il segnale *Start*, per i 10 istanti istanti successivi viene emesso un segnale intermittente (con intermittenza di 1 istante) *Blink* (per esempio, se viene ricevuto il segnale *Start* all'istante 6, il segnale *Blink* viene emesso agli istanti 8, 10, 12, 14 e 16). Due segnali di *Start* successivi sono a distanza l'uno dall'altro di almeno 15 istanti di tempo.

Esercizio 3 (punti 7)

Si costruisca una grammatica G che generi il seguente linguaggio L:

$L = \{a^n(bc)^m \mid n, m >= 1, m < n/2\}$

E' preferibile una grammatica a potenza minima, ossia regolare se ne esiste una, non contestuale se ne esiste una ma non ne esiste una regolare, ecc. Nel caso, si spieghi brevemente perché non esistono grammatiche appartenenti a classi inferiori a quella proposta.

Soluzioni

Esercizio 1

a. Il inguaggio è formato da un numero *finito* di parole, quindi può essere riconosciuto da un automa a stati finiti (addirittura, da un automa senza cicli).

b. Essendo il linguaggio finito, il problema è decidibile.

c. Il problema del riconoscimento di può risolvere con complessità O(1) (cioè costante) in quanto il linguaggio è finito.

Esercizio 2

Codificando la ricezione/emissione di un segnale S all'istante t con il predicato S(t), le formule seguenti specificano il comportamento desiderato:

$$\forall t(Start(t) \rightarrow \forall t_1(1 \leq t_1 \leq 5 \rightarrow Blink(t + 2t_1)))$$
$$\wedge$$
$$\forall t_2, t_3(Start(t_2) \wedge Start(t_3) \rightarrow |t_2 - t_3| \geq 15)$$

Esercizio 3

L è generato dalla seguente grammatica non contestuale (e da nessuna grammatica regolare essendo necessaria una pila per il suo riconoscimento):

S → aaaabc | ASB

A → aa

B → bc | ε

Informatica Teorica

Prova d'esame - 13 Luglio 2005

Esercizio 1 (punti 8/30-esimi)

Si scriva un automa che riconosca il linguaggio L le cui stringhe sono costruite sull'alfabeto A={0, 1} e sono fatte nella seguente maniera: le stringhe hanno lunghezza dispari; se il primo e l'ultimo carattere della stringa sono entrambi uguali ad '1', il carattere di centro è anch'esso uguale ad '1', altrimenti il carattere di centro è uguale a '0'.

NB: il punteggio massimo verrà assegnato solo se l'automa ideato sarà a potenza riconoscitiva minima tra quelli che riconoscono il linguaggio desiderato.

Esercizio 2 (punti 8/30-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.

Esercizio 3 (punti 8/30-esimi)

Scriva una formula logica che descrive un segnale fatto nella seguente maniera: dal momento in cui il segnale viene emesso la prima volta (che potrebbe anche non essere l'istante 0), esso viene emesso ad intervalli che si raddoppiano continuamente. L'intervallo tra i primi due istanti di emissione può essere qualunque.

In altre parole, se la distanza fra la (k-1)-esima emissione e la k-esima emissione è d, la distanza tra la k-esima e la (k+1)-esima emissione è 2d.

Un esempio possibile tra i tanti di evoluzione temporale del segnale è la seguente (vengono indicati gli istanti in cui il segnale viene emesso; si noti che l'intervallo tra le prime due emissioni è pari a 3 istanti di tempo):

2, 5, 11, 23, 47, 95, 191, ...

Esercizio 4 (punti 8/30-esimi)

parte a.

Siano date le seguenti pre- e post- condizioni di un metodo Java che ha un parametro in ingresso *arg* di tipo *String*, ritorna un *int*, e può sollevare un'eccezione *ComputationException*:

Pre: $\text{arg} \neq \textbf{null} \wedge 1 \leq \textbf{length}(\text{arg}) \leq 50 \wedge$
$\forall\ i, j(1 \leq i < j \leq \textbf{length}(\text{arg}) \rightarrow \text{arg}[i] \neq \text{arg}[j])$

Post: **result** $> 0 \vee$ **raise** ComputationException

laddove *result* indica il valore ritornato dal metodo (se questo ritorna senza sollevare eccezioni) e *raise ComputationException* indica il fatto che viene sollevata un'eccezione *ComputationException*; inoltre, *length*(*arg*) indica la lunghezza della stringa *arg*, e *arg*[*i*] indica l'i-esimo carattere della stringa.

E' decidibile il problema di stabilire se un'implementazione I soddisfa la specifica data sopra?

Si motivi adeguatamente la risposta.

parte b.

Sia data la seguente implementazione

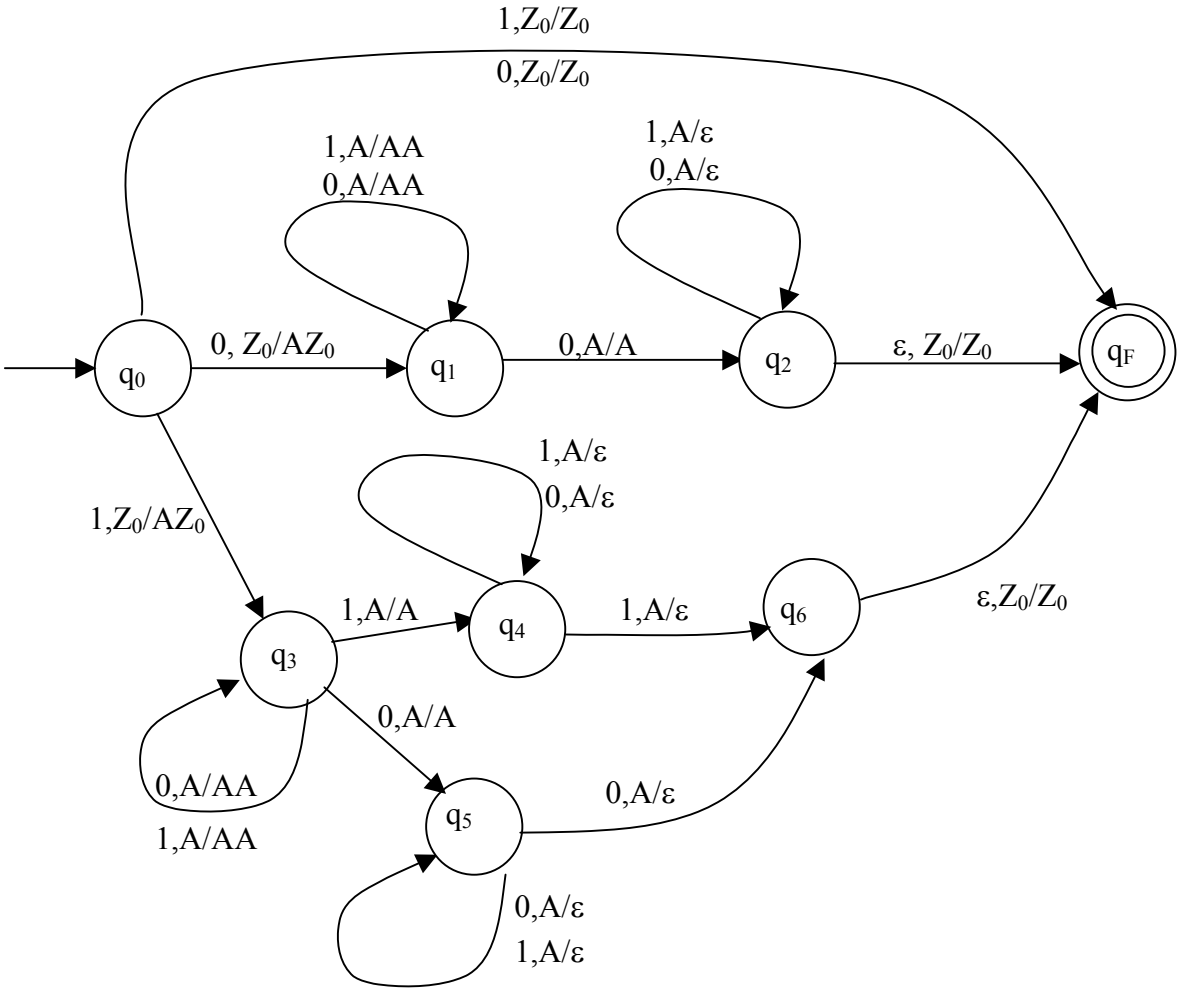
```
int myImpl(String arg) throws ComputationException {
  int sum = 0, i = 1;
  try {
     while(sum < (int)Math.pow(23.5, 4.21)){
       int diff = (int)(arg.charAt(i-1)-arg.charAt(0));
       sum += (int)(diff*Math.pow(4.67, 2.9));
       if (i == arg.length()) i = 1;
       else i++;
     }
  } catch (Exception e) {
    throw new ComputationException();
  }
  return sum;
}
```

E' decidibile il problema di stabilire se *myImpl* soddisfa la specifica descritta sopra?

Soluzioni

Esercizio 1

Il linguaggio L è riconosciuto dal seguente automa a pila nondeterministico:



Evidentemente non è possibile riconoscere L con una macchina a stati finiti, visto che le stringhe di L possono essere arbitrariamente lunghe, nè con un automa a pila deterministico, che non e' in grado di "indovinare" il punto centrale della stringa.

Esercizio 2

Una semplice macchina RAM che riconosce il linguaggio è fatta nella seguente maniera.

Innanzi tutto, essa legge una volta i caratteri in ingresso; ogni carattere letto viene memorizzato, e contemporaneamente si incrementa un contatore che mi dice quanti caratteri ho letto fino a quel momento. Alla fine della lettura, se ho contato un numero di caratteri pari, segnalo su nastro di uscita che la stringa non è accettata. Se invece ho contato un numero di caratteri dispari, vado a leggere il primo carattere, l'ultimo, e quello di mezzo (per recuperare l'offset del quale rispetto al primo carattere basta dividere per 2 il numero di caratteri letti). Se questi sono tutti uguali ad 1, oppure se quello di centro ed almeno uno tra il primo e l'ultimo sono uguali a 0, accetto la stringa, segnalandolo sul nastro di uscita, altrimenti rifiuto.

A costo costante, sia la complessità spaziale che quella temporale sono $\Theta(n)$.

A costo logaritmico, la complessità spaziale è ancora $\Theta(n)$, mentre quella temporale è $\Theta(n \cdot \log(n))$.

Esercizio 3

Indicando, come al solito, col predicato e(t) che il segnale viene emesso al tempo t, una possibile soluzione è:

$$\forall\, t_1, t_2\, (\, t_1 < t_2 \,\wedge\, e(t_1) \,\wedge\, e(t_2) \,\wedge\, \forall\, t'\, (\, t_1 < t' < t_2 \,\rightarrow\, \neg\, e(t')\,)$$

$$\rightarrow$$

$$e(t_2 + 2\,(t_2 - t_1)) \,\wedge\, \forall\, t'\, (\, t_2 < t' < t_2 + 2\,(t_2 - t_1) \,\rightarrow\, \neg\, e(t')\,)\,)$$

Esercizio 4

parte a.

Non è decidibile.

Una maniera per dimostrarlo potrebbe essere tramite il Teorema di Rice: l'insieme delle implementazioni (cioè delle MT, visto che si può tradurre un programma Java in una MT equivalente) che soddisfano la specifica non è certamente l'insieme vuoto (esiste almeno una funzione che soddisfa la specifica), e non è certamente neanche l'insieme universo (è banale pensare ad una funzione che non soddisfa la specifica), quindi il problema di stabilire se una data MT appartiene all'insieme non è decidibile.

Una maniera alternativa potrebbe riducendo il problema della terminazione al problema del soddisfacimento della specifica. Preso un programma qualunque P (facendo attenzione che in P non compaia il parametro *arg*) basta in effetti costruire un metodo siffatto:

```
int Pridotto(String arg) throws ComputationException {
  P
  return 1;
}
```

Tale metodo soddisfa la specifica se e solo se P termina.

Se sapessi risolvere il problema di partenza, saprei anche risolvere il problema della terminazione di un programma qualunque, che è un assurdo.

parte b.

In questo caso, il problema è banalmente decidibile.

Infatti, l'implementazione è in questo caso fissata a *myImpl*, quindi si danno 2 soli casi: o *myImpl* soddisfa le specifiche, oppure non le soddisfa. La risposta è quindi in questo caso binaria, o sì o no, e la MT che risolve il problema o è quella costante uguale ad 1, oppure è quella costante uguale a 0. In entrambi i casi essa è banalmente calcolabile.

Informatica Teorica (Sez. Mandrioli)

Appello del 13 Settembre 2005

Esercizio 1 (punti 10/30-esimi)

Si scrivano un automa e una grammatica che, rispettivamente, riconosca e generi il linguaggio L = $\{a^n a^m a^m a^n \mid n, m >= 1, n \text{ pari}, m \text{ dispari}\}$.

NB: il punteggio massimo verrà assegnato solo se l'automa ideato sarà a potenza riconoscitiva minima tra quelli che riconoscono il linguaggio desiderato e la grammatica avrà il minimo numero di produzioni.

Esercizio 2 (punti 10/30-esimi)

Si scriva una formula logica che descrive la seguente regola per controllare atterraggi e decolli in un aeroporto:

- Se c'è un aereo in fase di decollo nessun altro aereo può decollare entro 3 minuti dall'inizio del decollo e nessun altro aereo può atterrare entro 5 minuti dall'inizio del decollo dell'aereo precedente;
- Se c'è un aereo in fase di atterraggio nessun altro aereo può atterrare entro 3 minuti dall'inizio dell'atterraggio e nessun altro aereo può decollare entro 5 minuti dall'inizio dell'atterraggio dell'aereo precedente.

Esercizio 3 (punti 10/30-esimi)

Dire, giustificando brevemente la risposta, se la seguente funzione f: N → N è:

a) Computabile

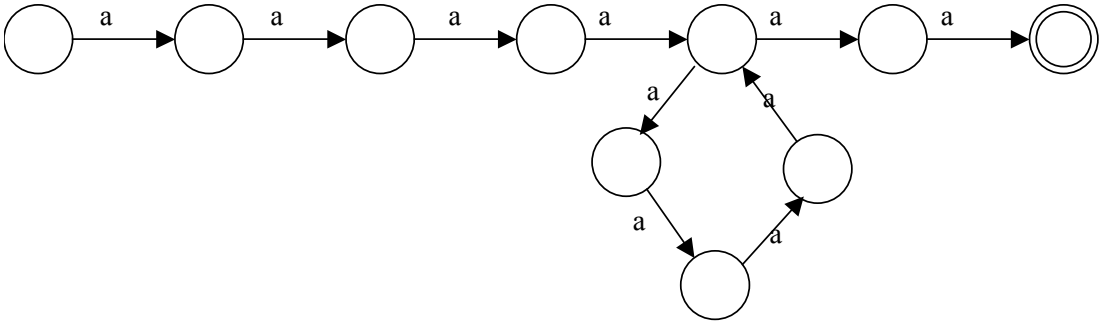
b) Totale.

f(n) = n-esimo numero primo. Per convenzione, si assuma f(0) = 1, e, di conseguenza, f(1) = 2, f(2) = 3, f(3) = 5, …

Soluzioni

Esercizio 1

La definizione di L equivale a L = {a^{2n}| n dispari e >= 2}, ossia L = {a^6 , a^{10}, a^{14}, a^{18}, …}, ossia, detto in altra maniera L = { $a^{2(2l+1)}$ | l ≥ 1 } = { a^{4l+2} | l ≥ 1 }. Quindi L può essere riconosciuto mediante il seguente semplice automa a stati finiti:



Da esso si può ricavare immediatamente una grammatica regolare equivalente, che può essere "compattata" nella seguente grammatica (non contestuale!) per minimizzare il numero di produzioni:

S → a^4A

A → a^4A| a^2

Esercizio 2

Si definscano i predicati StartTakeOff (x, t) e StartLand(x, t) per indicare, rispettivamente, che l'aereo x ha iniziato il decollo o l'atterraggio all'istante t.

Il requisito richiesto è allora formalizzato dalla formula seguente:

$$\forall t, x(\ StartTakeOff\ (x,t) \rightarrow$$
$$\neg\exists y, t_1 (y \neq x \wedge (t \leq t_1 \leq t+3) \wedge StartTakeOff\ (y,t_1)) \wedge$$
$$\neg\exists y, t_1 (y \neq x \wedge (t \leq t_1 \leq t+5) \wedge StartLand\ (y,t_1))\ )$$
$$\wedge$$
$$\forall t, x(\ StartLand\ (x,t) \rightarrow$$
$$\neg\exists y, t_1 (y \neq x \wedge (t \leq t_1 \leq t+3) \wedge StartLand\ (y,t_1)) \wedge$$
$$\neg\exists y, t_1 (y \neq x \wedge (t \leq t_1 \leq t+5) \wedge StartTakeOff\ (y,t_1))\ )$$

Altri predicati e formule, non strettamente necessari ai fini della formalizzazione del requisito ma utili a chiarire il significato dei termini potrebbero definire la fase di atterraggio/decollo dell'aereo x come quella compresa tra i relativi inizio (Start) e fine (End).

Esercizio 3

La funzione f è computabile: essendo infatti decidibile se un numero sia primo o no, dopo aver costruito l'n-esimo numero primo si può ottenere l'n+1-esimo enumerando tutti i numeri ad esso successivi fino ad ottenere il prossimo numero primo.

f è anche totale perché i numeri primi sono infiniti. Quindi, per ogni n, esiste l'n+1-esimo numero primo che verrà individuato dall'algoritmo suddetto.

Informatica Teorica (Nuovo e Vecchio Ordinamento)

Appello del 6 Febbraio 2006

Esercizio 1 (Punti 15)

Si formalizzino mediante formule logiche del prim'ordine le *regole* del gioco "Sudoku".

Informalmente le regole sono le seguenti:

E' dato un quadrato di 9 * 9 caselle, ognuna delle quali deve contenere un numero intero compreso tra 1 e 9. Il quadrato è suddiviso in 9 "sottoquadrati" di 3 * 3 caselle (il primo "copre" le caselle [1..3 * 1..3]; il secondo [4..6* 1..3], ecc.)

Occorre riempire le caselle del quadrato in modo che:

- Ogni riga e ogni colonna contenga tutti i numeri tra 1 e 9
- Ogni "sottoquadrato" contenga tutti i numeri tra 1 e 9.

Facoltativamente (ulteriori punti 1): potrebbe il problema essere generalizzato rispetto alle dimensioni del quadrato? Se sì, come?

Esercizio 2 (punti 7)

Si dica, giustificando brevemente la risposta, se il suddetto problema del Sudoku, ossia stabilire se, assegnati alcuni valori ad alcune caselle esiste un modo di riempire le altre in modo da soddisfare le regole, è decidibile o no.
Come cambia la risposta (se cambia), se il problema è espresso nella sua forma generalizzata?

Esercizio 3 (punti 8)

Si descrivano brevemente (senza necessariamente "codificarli" in dettaglio) algoritmi per il riconoscimento del linguaggio $\{a^n b^n \mid n >= 1\}$ con le seguenti caratteristiche:

- Una macchina di Turing che minimizzi la complessità temporale
- Una macchina di Turing che minimizzi la complessità spaziale
- Una RAM che minimizzi la complessità temporale
- Una RAM che minimizzi la complessità spaziale

E si confrontino le funzioni di complessità così ottenute.

Soluzioni

Esercizio 1

Si formalizzi la tabella del Sudoku mediante un array (si ricordi che, in termini matematici, un array altro non è che una funzione). Le regole del gioco sono allora formalizzate dalla formula seguente:

$$\forall i, j(1 \le i, j \le 9 \to 1 \le a[i, j] \le 9)$$
$$\wedge$$
$$\forall i, j, k(1 \le i, j, k \le 9 \to a[i, j] \ne a[i, k]$$
$$\wedge$$
$$\forall i, j, k(1 \le i, j, k \le 9 \to a[i, j] \ne a[k, j]$$
$$\wedge$$
$$\forall i, j, k, h((1 \le i, j, k, h \le 9 \wedge i/3 = k/3 \wedge j/3 = h/3 \wedge (i \ne k \vee j \ne h)) \to a[i, j] \ne a[k, h]$$

NB. il simbolo '/' indica la divisione a risultato intero.

La generalizzazione al caso di un quadrato k*k è banale se si ha l'accortezza di scegliere un valore di k che sia un quadrato perfetto. In tal caso k rimpiazza il numero 9 e $\sqrt{k}$ rimpiazza il numero 3 nella formula precedente.

Esercizio 2

Il problema è chiaramente decidibile in quanto è riconducibile del tutto a un problema finito. Infatti, ogni griglia 9 * 9 può essere riempita con numeri da 1 a 9 (rispettando le regole) in un numero finito di modi diversi. (Contare il numero esatto di schemi diversi non è banale, ma è sufficiente capire che esso è finito. Per curiosità, tale numero è stato calcolato essere uguale a 6 670 903 752 021 072 936 960). Pertanto un algoritmo che provi esaustivamente tutte le possibili soluzioni termina sicuramente.

Anche nal caso generalizzato, il problema rimane decidibile. Per ogni *n* fissato, ogni schema di n^2 * n^2 celle ha un numero finito di possibili completamenti (sicuramente meno di $n^{2n\hat{}4}$), per cui possono essere provati esaustivamente tutti.

Il problema è però computazionalmente oneroso, NP-completo per la precisione, come mostrato in: T. Yato, "Complexity and completeness of finding another solution and its application to puzzles". Master's thesis, University of Tokyo, Department of Information Sciences, 2003.

Esercizio 3

1. Macchina di Turing che minimizzi la complessità temporale.
   E' sufficiente "simulare" con la macchina di Turing un automa a pila, usando un nastro di memoria come una pila. La complessità temporale risultante sarebbe $\Theta(n)$, che è facile capire essere il meglio ottenibile.
2. Macchina di Turing che minimizzi la complessità spaziale.
   In questo caso conviene contare in codifica binaria (o equivalentemente, in altra base > 1) il numero di 'a' ricevute e poi decrementare il contatore binario per ogni 'b' ricevuta. La complessità spaziale sarebbe così $\Theta(\log n)$, ottenuta però al prezzo di un peggioramento di un fattore logaritmico della complessità temporale.
3. RAM che minimizzi la complessità temporale.
   Si può replicare la soluzione della macchina di Turing al punto 1 con una RAM, scrivendo un marker in n celle adiacenti. La complessità temporale risultante è la stessa $\Theta(n)$, con criterio di costo costante. Con criterio di costo logaritmico essa sale invece a $\Theta(n \log n)$, dal momento che anche se scrivo un valore costante (il marker) nelle celle adiacenti, ad ogni accesso devo manipolare un puntatore all'elemento corrente, che è un numero maggiorato da n.
4. RAM che minimizzi la complessità spaziale.
   In questo caso conviene utilizzare un singolo contatore intero in una singola cella di memoria. In questo modo il costo a criterio di costo costante è semplicemente $\Theta(1)$ (uso di un numero costante di celle di memoria), mentre a costo logaritmico diventa $\Theta(\log n)$, che è una caratterizzazione del fatto che la RAM di fatto "implementa" una codifica binaria del dato.

Informatica Teorica

Prima prova in itinere - 12 Maggio 2006

Si svolgano i seguenti esercizi. Il tempo complessivo a disposizione è di 1h e 30 minuti.

Esercizio 1 (Punti 6/13-esimi senza la parte opzionale c), che comporta ulteriori 3 punti)

Si consideri la seguente variante di automa a pila, detto **automa a pila visibile (APV)**. L'alfabeto di ingresso I dell'automa è ***partizionato*** in tre sottoinsiemi I_{push}, I_{pop}, I_{local}, che determinano indirettamente come la pila viene gestita durante il riconoscimento: se il carattere letto appartiene a I_{push}, allora l'automa deve effettuare una *push* (ossia aggiungere esattamente un carattere alla cima della pila, senza rimuoverne alcuno); se il carattere letto appartiene a I_{pop}, allora l'automa deve effettuare una *pop* (ossia rimuovere il carattere in cima alla pila, senza aggiungerne altri); se infine il carattere appartiene a I_{local}, l'automa *non* deve modificare il contenuto della pila. Si noti che ad ogni mossa è possibile modificare al più un carattere della pila. Per il resto l'automa si comporta come un normale automa a pila.

a) Si **formalizzi l'automa a pila visibile in versione nondeterministica**, nonché le regole di funzionamento e di riconoscimento di stringhe (mediante le usuali nozioni di configurazione, transizione, accettazione, ecc.). Per semplicità non si considerino le ε–mosse.

b) Si **confronti** poi la potenza riconoscitiva degli APV con quella degli automi a stati finiti.

c) **Opzionale:** si **confronti** la potenza riconoscitiva degli APV con quella degli automi a pila standard. (E' sufficiente una argomentazione non pienamente formale)

Esercizio 2 (Punti 4/13-esimi)

Si consideri il linguaggio L su alfabeto Σ = {a,b,c,p,q, z}, composto da tutte e sole le stringhe che: se iniziano per 'a', hanno 'z' come penultimo carattere (in questo caso sono lunghe almeno 3 caratteri), e se hanno 'c' come terzo carattere, allora hanno ogni occorrenza di 'p', successiva al terzo carattere, seguita immediatamente da una occorrenza di 'q'. Si scriva un automa a potenza minima tra quelli noti che accetta il linguaggio L.

Esercizio 3 (Punti 4/13-esimi)

Si scriva una grammatica che genera il linguaggio L1 definito sull'alfabeto Σ = {(,),*}; le stringhe di L1 o contengono esclusivamente parentesi tonde, e, in tal caso, devono essere ben parentetizzate, oppure, se contengono almeno un *, sono del tutto libere, cioè sono sequenze arbitrarie di (,),*. Anche in questo caso è preferibile una grammatica a potenza minima tra quelle che generano L1.

Soluzioni

Esercizio 1

Un automa riconoscitore nondeterministico a pila visibile (APV) è definito come una 7-pla

$$< Q, I, \Gamma, \delta, q_0, Z_0, F >$$

con tutti i simboli col significato degli automi a pila tradizionali. In più I è partizionato nei tre sottoinsiemi I_{push}, I_{pop}, I_{local}, ossia

$$I = I_{push} \cup I_{pop} \cup I_{local} \qquad \text{e} \qquad I_{push} \cap I_{pop} = I_{pop} \cap I_{local} = I_{push} \cap I_{local} = \varnothing$$

La funzione di transizione δ è definita

$$\delta: Q \times I \times \Gamma \rightarrow \wp(Q \cup Q \times \Gamma)$$

Transizione di configurazione:

$c = (q, a.x, \gamma.\eta) \vdash\!\!- c' = (q', x', \eta')$, con $a \in I$, $x \in I^*$, $\gamma \in \Gamma$, $\eta \in \Gamma^*$

- se $a \in I_{push}$ e $<q', \beta> \in \delta(q, a, \gamma)$, allora $\eta' = \beta.\eta$
- se $a \in I_{pop}$ e $q' \in \delta(q, a, \gamma)$, allora $\eta' = \eta$
- se $a \in I_{local}$ e $q' \in \delta(q, a, \gamma)$, allora $\eta' = \gamma.\eta$

La configurazione dell'automa, le relazione di transizione e la condizione di accettazione sono definite in maniera del tutto analoga al caso degli automi a pila standard (senza mosse ε).

La potenza espressiva degli APV è strettamente *maggiore* di quella degli *automi a stati finiti*; infatti, il linguaggio non regolare $\{a^n b^n \mid n > 0\}$ è accettato da un APV con $I_{push} = \{a\}$, $I_{pop} = \{b\}$, $I_{local} = \varnothing$, con l'accorgimento di avere un carattere extra di pila per poter accettare a pila non vuota. D'altro canto è evidente che ogni automa a stati finiti può essere espresso come un banale automa APV.

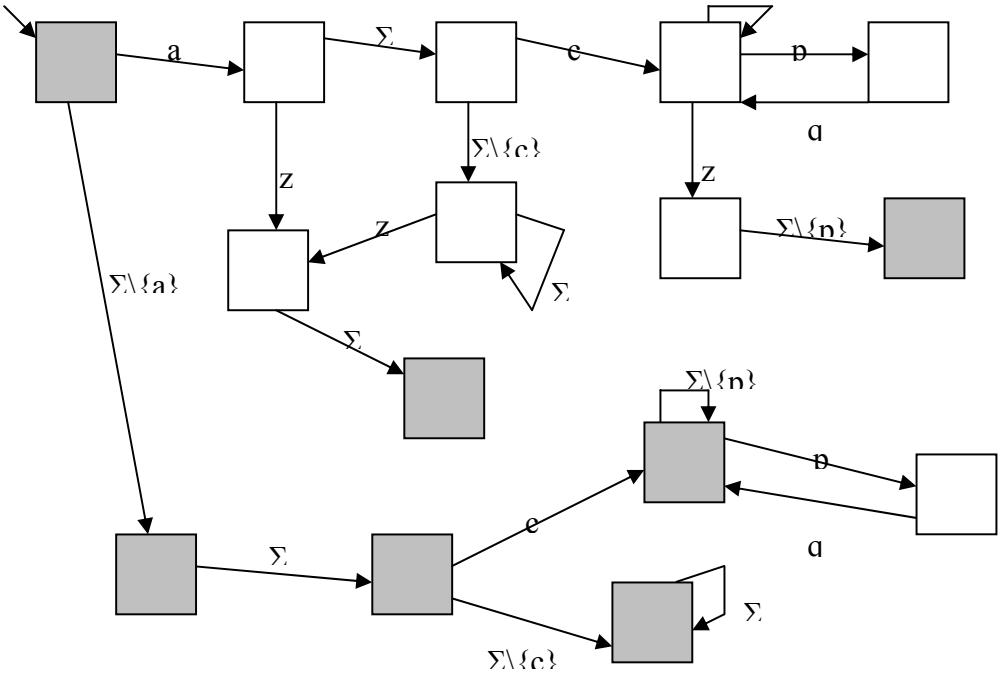
Il confronto con gli automi a pila standard è più complesso. Consideriamo il linguaggio $\{wcw^R \mid w \in \{a,b\}^*\}$, riconosciuto da un automa a pila deterministico standard (e dunque a maggior ragione da uno nondeterministico), mediante la seguente tecnica: si impilano i caratteri letti finchè si incontra il carattere c; a questo punto verifica che i rimanenti caratteri letti (ossia quelli di w^R) siano esattamente quelli che sono in pila, togliendoli dalla cima. Siccome ciascun carattere viene incontrato due volte – uno in w e uno in w^R – e per la prima occorrenza è necessario fare una push, mentre per la seconda una pop, non è possibile partizionare l'alfabeto di pila secondo quanto richiesto dagli APV per effettuare il riconoscimento richiesto.

In conclusione gli APV sono meno espressivi degli automi a pila standard (anche solo deterministici).

Gli APV (in inglese, visibly pushdown automata), sono stati introdotti in: Alur e Madhusadan: "Visibly pushdown languages", Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04), pp. 202-211, 2004.

Esercizio 2

Un automa a stati finiti è sufficiente per accettare L; eccone una versione nondeterministica (gli stati in grigio sono di accettazione).



Si noti che la definizione informale del linguaggio poteva ammettere anche un'interpretazione più complicata ma comunque corretta, ossia che la condizione "e se hanno 'c' come terzo carattere, allora hanno ogni occorrenza di 'p', successiva al terzo carattere, seguita immediatamente da una occorrenza di 'q'." si applicasse alle sole stringhe che iniziano per 'a'.

Esercizio 3

La seguente grammatica noncontestuale genera L1.

S → B | Q

B → BB | (B) | ε

Q → A*A

A → *A | (A |)A | ε

Anche in questo caso, sono ammissibili interpretazioni leggermente diverse, in particolare se la stringa nulla appartenga a L1 e se una stringa ben parentetizzata debba comunque avere un'unica coppia di parentesi più esterna.

Informatica Teorica

Seconda prova in itinere – 5 Luglio 2006, Sezione Mandrioli

Il problema della Torre di Hanoi è così definito. Vi sono *n* dischi simili ma di dimensioni diverse inseriti in un piolo, in modo tale che il disco più grande sia nel punto più basso e tutti gli altri siano sopra di esso in ordine decrescente. Esistono altri due pioli inizialmente liberi (si veda la Figura (*a*)).

Si vogliono trasportare tutti i dischi su un altro piolo, spostandoli uno alla volta da un piolo all'altro, e operando in modo tale che mai un disco sia appoggiato sopra uno più piccolo (ovviamente, per spostare un disco da un piolo all'altro occorre aver rimosso quelli sovrastanti, sempre rispettando le regole precedenti). Le Figure (*b*) e (*c*) completano l'illustrazione del problema.

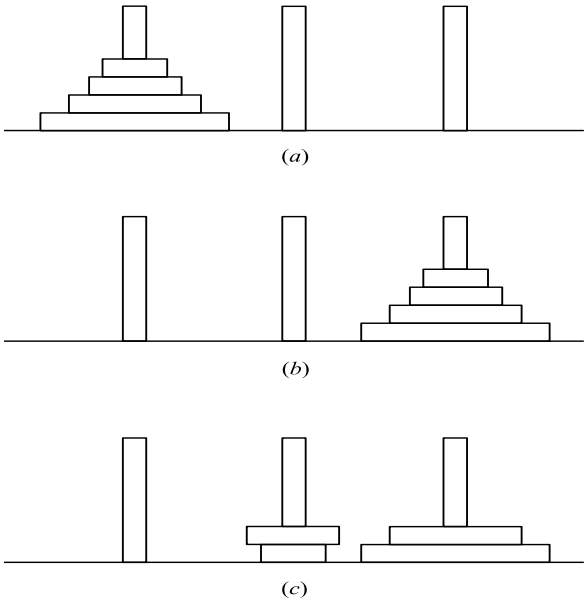


Figure (a), (b), (c) Il problema della torre di Hanoi: (*a*) situazione iniziale, (*b*) situazione finale richiesta, (*c*) situazione inammissibile.

Esercizio 1 (punti 8/17-esimi)

Si formalizzi mediante formule del prim'ordine una generica situazione ammissibile della posizione dei dischi nei pioli: ad esempio le figure (a) e (b) rappresentano due situazioni ammissibili al contrario della figura (c).

Esercizio 1.bis(da svolgere *facoltativamente* solo dopo aver svolto tutti gli altri esercizi. L'esercizio vale ulteriori punti 3/17-esimi)

Si formalizzi, sempre mediante formule del prim'ordine, la relazione che deve sussistere tra due diverse situazioni ammissibili perché l'una sia ottenibile dall'altra mediante una mossa lecita.

Esercizio 2 (punti 4/17-esimi)

Si dica se il seguente problema è decidibile: Dato un algoritmo A qualsiasi, stabilire se A risolve correttamente il problema della Torre di Hanoi. Giustificare brevemente la risposta.

Esercizio 3 (punti 5/17-esimi)

Dovendo valutare la complessità di un algoritmo per la soluzione del problema della Torre di Hanoi da eseguire mediante un normale calcolatore basato sulla classica architettura di von Neumann (macchina RAM), quale criterio di costo risulterebbe più adeguato e conveniente? Giustificare brevemente la risposta.

Esercizio 3bis, facoltativo (ulteriori punti 2/17-esimi)

Che relazione è "naturale" aspettarsi tra complessità (sia spaziale che temporale) di un algoritmo per la soluzione del problema, valutata a criterio costante e a criterio logaritmico? **NB**. Non si chiede di descrivere un algoritmo né di valutarne le complessità; bensì di stabilire come cambia la valutazione di complessità cambiando il criterio di costo.

Avvisi importanti

1.

Il tempo disponibile per lo svolgimento della prova è 1 ora e 30 minuti.

2.

L'eventuale soluzione dell'**Esercizio 1.bis** sarà presa in considerazione e valutata **solo se tutti gli altri esercizi** saranno stati risolti in maniera ritenuta **soddisfacente**.

3.

Si ricorda che possono svolgere questa prova in itinere solo coloro che abbiano ottenuto almeno 5 punti nella prima prova.

Eccezionalmente sono ammessi anche gli studenti laureandi, **che devono indicare esplicitamente la loro situazione nella testata del loro elaborato**. Come già comunicato, in caso di esito positivo nella presente prova, verranno loro comunicate le modalità di recupero della prima prova.

4.

Come in altre occasioni i risultati e le successive comunicazioni (in particolare le modalità di accettazione/rifiuto del voto proposto) saranno pubblicati nella pagina personale di Dino Mandrioli sul sito ufficiale del Dipartimento. Verrà data precedenza alla correzione dei compiti dei laureandi (che risultino tali dall'archivio ufficiale del CEDA)

Soluzioni

Esercizio 1

Si indichi

d) con D_i l'i-esimo disco: $1 <= i <= n$;

e) con $dim(D_i)$ la dimensione del disco i-esimo;
$(\forall i\ ((1 <= i <= n) \rightarrow (1 <= dim(D_i) <= n))) \wedge$
$(\forall i,j\ (((1 <= i, j <= n) \wedge (i \neq j)) \rightarrow (dim(D_i) \neq dim(D_j))))$

f) con $pos(D_i)$ la posizione del disco i-esimo:
$pos(D_i) = <h,p>$; $1 <= h <= n$; $1 <= p <= 3$; che indica che il disco si trova ad altezza h nel piolo p.

g) la funzione pos deve sottostare ai seguenti vincoli:

h) $\forall i,j\ (pos(D_i) = pos(D_j) \rightarrow D_i = D_j)$

$\wedge$

i) $\forall i,h, k, p\ ((pos(D_i) = <h,p> \wedge k < h) \rightarrow \exists j\ (pos(D_j) = <k,p>))$

$\wedge$

j) $\forall i,j, h, k, p\ ((pos(D_i) = <h,p> \wedge pos(D_j) = <k,p> \wedge k < h) \rightarrow (dim(D_j) > dim(D_i)))$

Esercizio 1.Bis

Si indichino con pos1 e pos2, rispettivamente, due funzioni che definiscono la posizione del disco i-esimo prima e dopo l'esecuzione di una mossa. Ovviamente sia pos1 che pos2 devono soggiacere ai vincoli formalizzati precedentemente. Inoltre esse soddisfare la relazione espressa dalle formule seguenti:

- Se un disco qualsiasi non si trova in cima a un piolo, la sua posizione rimane invariata nel passaggio da pos1 a pos2:

  $\forall i,h, p\ ((pos1(D_i) = <h,p> \wedge \exists\ k, j\ (k > h \wedge pos1(D_j) = <k,p>)) \rightarrow (pos2(D_i) = <h,p>))$

- **Se** (per ogni tripla di dischi D_{i1}, D_{i2}, D_{i3} e posizioni):

  $\forall i1,i2, i3, h1, h2, h3, p1, p2, p3$

  - il disco D_{i1} si trova in cima al piolo p1:

    $(\ (pos1(D_{i1}) = <h1,p1> \wedge \neg\exists\ k, j\ (k > h1 \wedge pos1(D_j) = <k,p1>))$

  - **e** il disco D_{i2} si trova in cima al piolo p2:

    $\wedge\ \ (pos1(D_{i2}) = <h2,p2> \wedge \neg\exists\ k, j\ (k > h2 \wedge pos1(D_j) = <k,p2>))$

  - **e** il disco D_{i3} si trova in cima al piolo p3:

    $\wedge\ \ (pos1(D_{i3}) = <h3,p3> \wedge \neg\exists\ k, j\ (k > h3 \wedge pos1(D_j) = <k,p3>))$

  - **e** i tre dischi Di1 sono diversi tra loro, e stanno sui tre pioli diversi

    $\wedge\ \ (i1 \neq i2 \wedge i2 \neq i3 \wedge i1 \neq i3\ \ \wedge\ \ p1 \neq p2 \wedge p2 \neq p3 \wedge p1 \neq p3)\ \ )$

- **allora**:

  $\rightarrow$

  - uno (e uno solo) tra Di1, Di2, e Di3 cambia piolo in pos2 (e, necessariamente per i vincoli su pos, viene a trovarsi in cima al nuovo piolo)

    $\exists\ h, p\ \ (\ (pos2(D_{i1}) = <h,p> \wedge p \neq p1 \wedge pos2(D_{i2}) = pos1(D_{i2}) \wedge pos2(D_{i3}) = pos1(D_{i3}))$

$\vee$ ($pos2(D_{i2}) = \langle h,p \rangle \wedge p \neq p2 \wedge pos2(D_{i1}) = pos1(D_{i1}) \wedge pos2(D_{i3}) = pos1(D_{i3})$)

$\vee$ ($pos2(D_{i3}) = \langle h,p \rangle \wedge p \neq p3 \wedge pos2(D_{i1}) = pos1(D_{i1}) \wedge pos2(D_{i2}) = pos1(D_{i2})$))

Soluzione alternativa

Definiamo che la differenza tra pos1 e pos2 riguarda un solo disco, che deve essere in cima ad un piolo in pos1 (e che sarà in cima ad un piolo in pos2 per i vincoli sui pioli):

$\exists$ D_i, h, p ($pos1(D_i) = \langle h,p \rangle \wedge pos2(D_i) \neq \langle h,p \rangle \wedge$ /* D_i cambia posizione */
$\neg\exists$ j ($pos1(D_j) = \langle h+1,p \rangle$) $\wedge$ /* D_i è in cima ad un piolo in pos1 */
$\forall j$ ($j \neq i \rightarrow pos1(D_j) = pos2(D_j)$) /* tutti gli altri dischi rimangono fermi */
)

Si noti che potremmo semplificare la condizione eliminando il vincolo che D_i in pos1 sia in cima ad un piolo, in quanto ciò è garantito dal fatto che in pos2 non ci possono essere "buchi", e dal fatto che l'unico disco ad essere spostato è D_i:

$\exists$ D_i, h, p ($pos1(D_i) = \langle h,p \rangle \wedge pos2(D_i) \neq \langle h,p \rangle \wedge$ /* D_i cambia posizione */
$\forall j$ ($j \neq i \rightarrow pos1(D_j) = pos2(D_j)$) /* tutti gli altri dischi rimangono fermi */
)

Se invece volessimo specificare, per maggiore chiarezza, che nella nuova posizione D_i deve essere in cima ad un piolo, potremmo modificare la condizione come segue:

$\exists$ D_i, h, p ($pos1(D_i) = \langle h,p \rangle \wedge pos2(D_i) \neq \langle h,p \rangle \wedge$ /* D_i cambia posizione */
$\neg\exists$ j ($pos1(D_j) = \langle h+1,p \rangle$) $\wedge$ /* D_i è in cima ad un piolo in pos1 */
$\forall h', p'$ ($pos1(D_i) = \langle h',p' \rangle$)
$\rightarrow$
$\neg\exists$ j ($pos2(D_j) = \langle h'+1,p' \rangle$)) $\wedge$ /* D_i è in cima ad un piolo anche in pos2 */
$\forall j$ ($j \neq i \rightarrow pos1(D_j) = pos2(D_j)$) /* tutti gli altri dischi rimangono fermi */
)

*Le due seguenti **ulteriori soluzioni alternative** descrivono in maniera completa non solo i vincoli sul posizionamento dei dischi nei pioli e sul loro movimento, ma anche l'obiettivo del gioco.*

Esercizio 1 + 1.Bis, alio modo

Si introduca la seguente funzione:

- piolo(p, h, t) = d se e solo se *d* è la dimensione del disco che si trova ad altezza *h* sul piolo *p* al tempo *t*

Una possibile formalizzazione del problema della Torre di Hanoi è la seguente (con implicite quantificazioni universali esterne):

- vincoli sui domini:
  $piolo(p, h, t) = d \rightarrow 1 \leq p \leq 3 \wedge 1 \leq h \leq n \wedge 1 \leq d \leq n$
- unicità del disco:
  $piolo(p_1, h_1, t) = piolo(p_2, h_2, t) \rightarrow p_1 = p_2 \wedge h_1 = h_2$
- assenza di "buchi" sul piolo:
  $piolo(p, h, t) = d \wedge h > 1 \rightarrow \exists d_1(piolo(p, h-1, t) = d_1)$
- vincolo sull'ordinamento dei dischi sul piolo:
  $piolo(p, h_1, t) > piolo(p, h_2, t) \rightarrow h_1 < h_2$

Si introducano poi i seguenti predicati e funzioni:

- altezza(p, t) = h se e solo se *h* è l'altezza del piolo *p* al tempo *t*
- $muovi(p_1, p_2, t)$ se e solo se all'istante t il disco che si trova in cima al piolo p_1 è spostato in cima al piolo p_2

Una possibile formalizzazione dell'evoluzione del gioco della Torre di Hanoi è la seguente (con implicite quantificazioni universali esterne):

- definizione della funzione "altezza", che deriva da "piolo":
  $altezza(p, t) = h \leftrightarrow (h > 0 \wedge \exists d(piolo(p, h, t) = d) \wedge \neg\exists d(piolo(p, h+1, t) = d)) \vee (h = 0 \wedge \neg\exists d(piolo(p, 1, t) = d)))$
- una è mossa possibile solo se il piolo di partenza non è vuoto e se il piolo di arrivo è diverso da quello di partenza:
  $muovi(p_1, p_2, t) \rightarrow altezza(p_1, t) > 0 \wedge p_1 \neq p_2$
- non è possibile effettuare 2 mosse contemporaneamente:
  $muovi(p_1, p_2, t) \wedge muovi(p_3, p_4, t) \rightarrow p_1 = p_3 \wedge p_2 = p_4$
- effetto di una mossa sullo stato di un piolo:
  $$piolo(p, h, t) = d \leftrightarrow (p = 1 \wedge h = n - d + 1 \wedge \neg\exists t_1, p_1, p_2(t_1 < t \wedge muovi(p_1, p_2, t_1))) \vee \exists t_1 (t_1 < t \wedge altezza(p, t_1) = h-1 \wedge \exists p_1(muovi(p_1, p, t_1) \wedge piolo(p_1, altezza(p_1, t_1), t_1) = d \wedge \forall t_2 (t1 < t2 < t \wedge altezza(p, t_2) = h \rightarrow \neg\exists p_1 (muovi(p, p_1, t_2)))))$$
- stato iniziale del gioco:
  $\forall i(1 \leq i \leq n \rightarrow piolo(1, i, 0) = n - i + 1) \wedge altezza(2, 0) = 0 \wedge altezza(3, 0) = 0$
  - si noti che la seconda e terza condizione sono superflue, in quanto se tutti i dischi sono sul primo piolo, sugli altri non ci può essere nulla
- stato finale del gioco:
  $\exists t (\forall i(1 \leq i \leq n \rightarrow piolo(3, i, t) = n - i + 1) \wedge altezza(1, t) = 0 \wedge altezza(2, t) = 0)$

Esercizio 1 + 1.Bis, yet another

Insiemi: D = {1,2,..,n} dischi; P = {1,2,3} pioli; tempo N.

Variabili: $t \in N$; $d,d',d'' \in D$; $p,p',p'' \in P$.

Predicati:

piolo(p,d,t): il disco d è sul piolo p al tempo t

mossa(p,p',t): si muove dal piolo k a k' al tempo t

on(d,d',t): il disco d è immediatamente sopra d' al tempo t (serve solo per descrivere la situazione iniziale)

Formule:

unicità dischi su pioli:

$$piolo(p,d,t) \rightarrow \neg\exists p' \, (p' \neq p \wedge piolo(p',d,t))$$

definizione di disco più in alto:

$$top(p,d,t) \leftrightarrow$$
$$(piolo(p,d,t) \wedge \neg\exists d' \, (d' < d \wedge piolo(p,d',t))$$
$$\vee$$
$$d = n+1 \wedge \neg\exists d' \, piolo(p,d',t))$$

situazione iniziale:

$$init(t) \leftrightarrow$$
$$(1 =< d < n \rightarrow on(d,d+1,t)) \wedge$$
$$(on(d,d',t) \rightarrow piolo(1,d,t) \wedge piolo(1,d,t'))$$

situazione finale:

$$end(t) \leftrightarrow \forall d \; piolo(3,d,t)$$

mossa:

$$mossa(p,p',t) \leftrightarrow$$
$$top(p,d,t) \wedge top(p',d',t) \wedge d' > d \wedge$$
$$piolo(p',d,t+1) \wedge$$
$$\forall d'',p'' \, (d'' \neq d \wedge piolo(p'',d'',t) \rightarrow piolo(p'',d'',t+1))$$

partita:

$$init(0) \wedge \forall t \, (\neg end(t) \rightarrow \exists p,p' \; mossa(p,p',t))$$

Esercizio 2

Si tratta di un classico caso di applicazione del teorema di Rice. Infatti, il problema può essere visto come una specifica funzione (il cui valore, per una data configurazione iniziale, potrebbe essere la sequenza di mosse, se esiste, che porta alla desiderata configurazione finale). Tale funzione è ovviamente calcolabile. Quindi considerando l'insieme costituito esattamente da tale funzione, non è decidibile, in base al teorema di Rice, se un generico algoritmo calcola esattamente la funzione considerata.

Esercizi 3 e 3 bis

Ovviamente il criterio di costo logaritmico fornisce sempre la garanzia di risultati realistici. In questo caso, tuttavia esso risulterebbe superfluo e quindi inutilmente oneroso da applicare. Si osservi infatti che una quantità di memoria limitata linearmente dal numero n di dischi è sufficiente per rappresentare qualsiasi configurazione del sistema (ad esempio un array che indichi la posizione di ogni disco costerebbe Θ (n) a criterio di costo costante e Θ (n.log(n)) a criterio logaritmico). L'applicazione di ogni mossa richiederebbe perciò un tempo costante a criterio di costo costante e al più Θ (log(n)) a criterio di costo logaritmico. Ne consegue che qualsiasi sia la complessità T(n) di un algoritmo (ovviamente che non usi la memoria in modo scioccamente inefficiente) calcolata a criterio di costo costante, la corrispondente complessità calcolata a criterio di costo logaritmico sarebbe automaticamente Θ (T(n).log(n)).

Si noti tuttavia che la funzione T(n) invece sarà probabilmente di un ordine di grandezza molto elevato data la natura "a ricorsione fortemente nidificata" dei normali algoritmi per risolvere il problema.

Informatica Teorica

Appello del 19 Luglio 2006

Esercizio 1 (Punti 10)

Si consideri il linguaggio seguente:

L = {wcW | w, W $\in$ {a, b}*; W essendo una stringa che differisce dalla stringa riflessa, w^R, di w per al più 3 caratteri}

Si scrivano una grammatica e un automa, preferibilmente a potenza minima (ossia appartenenti a una categoria di grammatiche e automi tale che una categoria di minor potenza generativa non possa definire L) che generi e riconosca L, rispettivamente.

Esercizio 2 (Punti 13)

Si consideri il seguente problema: data una *f(x)* qualsiasi (con dominio e codominio l'insieme *N*) di cui sia noto a priori che è computabile, stabilire se esistono un polinomio *P* a coefficienti interi non negativi e un valore $\bar{x} \in N$ tali che $P(\bar{x}) = f(\bar{x})$.

Si dica, giustificando brevemente la risposta, se il problema è decidibile, semidecidibile, o neanche semidecidibile.

Come cambia, se cambia, la risposta assumendo di sapere a priori che *f* non è totalmente indefinita (ossia indefinita per ogni valore del suo dominio)?

Esercizio 3 (Punti 10)

Si consideri il linguaggio L presentato nell'esercizio 1. Si descriva nel dettaglio un algoritmo per una macchina basata sull'architettura di Von Neumann (ad es. la macchina RAM), che restituisca in uscita la stringa wcw^R, se la stringa in ingresso x = wcW appartiene a L, cc altrimenti. Si supponga che la stringa in ingresso NON sia già disponibile in memoria.
Se ne valuti la complessità (spaziale e temporale, con entrambi i criteri di costo).

Soluzioni

Esercizio 1

L è evidentemente un linguaggio non contestuale.

Una G che genera L è la seguente:

S → c | aSa | bSb | aAb | bAa

A → c | aAa | bAb | aBb | bBa

B → c | aBa | bBb | aCb | bCa

C → c | aCa | bCb

Similmente, un automa a pila che riconosca L può operare nel modo seguente:

Inizialmente impila i caratteri di w fino ad incontrare c.

Dopo aver letto c inizia a svuotare la pila rimanendo nello stesso stato finché il simbolo letto corrisponde al simbolo in cima alla pila. Se la pila si svuota quando si giunge a fine stringa in questo stato l'ingresso viene accettato.

Non appena viene letto un carattere no corrispondente al carattere in pila si cambia stato una prima volta; indi si prosegue lo svuotamento della pila nel modo usuale.

Sono possibili al più 3 cambiamenti di stato durante la lettura di W.

Esercizio 2

E' noto che per qualsiasi valore y esistono un polinomio P e un valore z tale che P(z) = y (ad esempio il polinomio costante y). Quindi il fatto che esistano un polinomio P a coefficienti interi non negativi e un valore $\bar{x} \in N$ tali che $P(\bar{x}) = f(\bar{x})$ è equivalente a stabilire se esiste un $\bar{x} \in N$ tale che $f(\bar{x})$ sia definito, ossia se *f* sia totalmente indefinita o meno.

Questo è un problema notoriamente indecidibile (tra i tanti modi di dimostrarlo si può utilizzare il teorema di Rice); tuttavia esso è semidecidibile: infatti se un tale $\bar{x}$ esiste, mediante una tipica enumerazione diagonale (simulo x mosse di una macchina che calcola f sul valore y, ecc.) esso viene individuato.

Se invece si sapesse a priori che *f* non è totalmente indefinita, allora l'esistenza di $\bar{x}$ sarebbe già garantita e quindi anche il problema di partenza sarebbe risolto.

Esercizio 3

Un semplice algoritmo che risolve il problema utilizza una lista di caratteri la cui lunghezza massima sarà pari alla lunghezza di w, una struttura dati LIFO (cioè una pila) ed un contatore che memorizza il numero di "errori" commessi dalla stringa W rispetto alla w^R (quest'ultimo è inizializzato a zero). Ogni volta che un carattere viene letto dall'input, si procede in questa maniera:

1. se non si è ancora letto il separatore 'c', si copia il carattere letto in coda alla lista e si impila lo stesso carattere sulla pila
2. se si legge il carattere 'c', si procede al carattere successivo
3. se si è già letto il separatore 'c', allora
    a. se il carattere appena letto corrisponde a quello in cima alla pila, si elimina quest'ultimo dalla pila e si procede alla lettura del carattere successivo
    b. altrimenti, si incrementa il contatore degli errori e
        i. se il contatore errori è maggiore di 3, si scrive 'cc' in output e si termina,
        ii. altrimenti, si procede con la lettura del carattere successivo
4. se la stringa è stata letta completamente dall'input e la pila è vuota, si utilizza la lista riempita al passo 1. per scrivere in output wcw^R. Per fare ciò, si scrive prima w scorrendo la lista a partire dalla testa, si inserisce 'c' , e si ripercorre la lista in direzione opposta, ottenendo quindi w^R in output. Altrimenti, si scrive 'cc' in output.

Chiamando n la lunghezza della stringa in ingresso, l'occupazione in memoria dell'algoritmo proposto è proporzionale a questa lunghezza. Infatti, per ciascun carattere letto, è necessario un numero costante di celle di memoria (nel caso peggiore, una posizione nella lista e una posizione sulla pila). Per cui, la complessità spaziale dell'algoritmo sarà $\Theta(n)$ sia a costo costante che a costo logaritmico. Per quanto riguarda la complessità temporale, i passi 1., 2. e 3. hanno un costo costante per singola esecuzione. Nel caso peggiore, 1. e 3. vengono eseguiti n volte. Analogamente, il passo 4. richiede un numero di operazioni direttamente proporzionale ad n nel caso peggiore. Di conseguenza, la complessità temporale dell'algoritmo valutata a costo costante sarà $\Theta(n)$, mentre valutata a costo logaritmico sarà $\Theta(n \log(n))$.

Informatica Teorica

Sezione Mandrioli

Appello del 13 Settembre 2006

Attenzione

I voti proposti verranno pubblicati sul sito seguente:

http://www.elet.polimi.it/upload/mandriol/Didattica/sitoinfteor1.html

Verrà data precedenza ai laureandi, che devono indicare questa loro qualifica nel proprio elaborato.

Gli studenti avranno tempo due giorni (48 ore, per la precisione) dalla data della pubblicazione per rifiutare il voto proposto, se sufficiente. L'eventuale rifiuto deve essere comunicato via email, facendo uso dell'indirizzo ufficiale del Poliself, non di indirizzo privato! Trascorsi i due giorni, i voti verranno registrati e trasmessi alla segreteria.

Il docente spedirà a ogni studente "rinunciatario" un esplicito messaggio di "ricevuta". In caso di mancata ricezione di tale ricevuta si consiglia di contattare il docente telefonicamente o di avvisare la segreteria didattica del DEI.

Il tempo a disposizione per lo svolgimento del tema d'esame è 1h e 30 minuti.

Esercizio 1 (punti 12)

Si considerino le seguenti regole di attraversamento di un incrocio regolato da un semaforo.

- Il semaforo è verde per 4 unità di tempo; poi passa a giallo per un'unità di tempo e rosso per 4 ulteriori unità.
- Se un veicolo attraversa l'incrocio con semaforo verde o giallo nessuna azione viene eseguite nei suoi confronti.
- Se un veicolo attraversa l'incrocio con semaforo rosso entro 1 unità di tempo da quando è diventato rosso, viene elevata contravvenzione per Euro 50.
- Se un veicolo attraversa l'incrocio con semaforo rosso dopo 1 unità di tempo da quando è diventato rosso, viene elevata contravvenzione per Euro 200.
- Per semplicità si consideri un solo veicolo alla volta; si può anche assumere che la contravvenzione sia contemporanea all'infrazione.

Si formalizzino le suddette regole in ***una sola*** delle versioni proposte nel seguito.

Versione A

Si utilizzi un'opportuna macchina astratta assumendo un tempo discreto in cui l'unità corrisponde all'unità di tempo indicata sopra.

Versione B

Si utilizzino formule del primo ordine. In tal caso si può adottare un dominio temporale sia discreto che continuo.

NB

E' vietato consegnare entrambe le soluzioni che in tal caso non sarebbero valutate.

Esercizio 2 (punti 10)

Si consideri il sottoinsieme del linguaggio C in cui siano escluse tutte le istruzioni di controllo del flusso dell'esecuzione ad eccezione dell'istruzione condizionale **if** e dell'istruzione ciclica **for**.

Si dica, giustificando brevemente la risposta, se è decidibile o no il problema di stabilire se un generico programma scritto in tale sottoinsieme del C terminerà sempre la sua esecuzione (ossia per ogni valore dei dati di ingresso) o no.

Esercizio 3 (punti 10)

Si supponga di implementare un algoritmo di merge-sort mediante una macchina RAM (non è necessario scrivere effettivamente il codice!).

Quale sarebbe la sua complessità temporale valutata a criterio di costo costante e a criterio di costo logaritmico? Giustificare brevemente la risposta. Per semplicità si può assumere che i dati da ordinare occupino *singolarmente* una quantità limitata di memoria (ad esempio, 32 bit).

Parte facoltativa (ulteriori punti 5)

Si valutino anche la complessità spaziale dell'implementazione mediante RAM e la complessità sia spaziale che temporale di una possibile implementazione mediante macchina di Turing, assumendo la stessa ipotesi semplificativa della parte precedente.

Soluzioni

Esercizio 1

Versione A (traccia per la costruzione di un automa)

Assumendo un dominio temporale discreto, conviene associare ogni scatto di transizione a un'unità di tempo. Quindi si può costruire un automa a 9 stati per descrivere l'evoluzione del semaforo. Ogni transizione è etichettata da un input attr ("il veicolo attraversa durante l'unità di tempo corrente") oppure n_attr ("il veicolo non attraversa durante l'unità di tempo corrente"). La transizione etichettata attr uscente dal primo stato rosso comporta l'uscita multa50 e le successive transizioni uscenti dagli stati rossi comportano l'uscita multa200.

Versione B

Si introducano i seguenti predicati:

- v(t), (g(t), r(t), s(t)) Il semaforo è verde (rispettivamente, giallo, rosso, spento) all'istante t
- attr(t) Il veicolo attraversa l'incrocio all'istante t
- multa50(t) Viene elevata contravvenzione per 50 euro all'istante t
- multa200(t) Viene elevata contravvenzione per 200 euro all'istante t

La formula seguente formalizza le regole richieste, assumendo che il semaforo inizi a funzionare all'istante 0 (con il verde) e che la multa venga notificata immediatamente.

$$
\begin{aligned}
&\forall t \\
&(t < 0 \rightarrow s(t)) \wedge (0 \bmod 9 \leq t < 4 \bmod 9 \rightarrow v(t)) \wedge \\
&(4 \bmod 9 \leq t < 5 \bmod 9 \rightarrow g(t)) \wedge (5 \bmod 9 \leq t < 9 \bmod 9 \rightarrow v(t)) \\
&\wedge \\
&((attr(t) \wedge 0 \bmod 9 \leq t < 4 \bmod 9) \rightarrow (\neg multa50(t) \wedge \neg multa200(t)) \\
&\wedge \\
&((attr(t) \wedge 4 \bmod 9 \leq t < 5 \bmod 9) \rightarrow (\neg multa50(t) \wedge \neg multa200(t)) \\
&\wedge \\
&((attr(t) \wedge 5 \bmod 9 \leq t < 6 \bmod 9) \rightarrow (multa50(t) \wedge \neg multa200(t)) \\
&\wedge \\
&((attr(t) \wedge 6 \bmod 9 \leq t < 9 \bmod 9) \rightarrow (\neg multa50(t) \wedge multa200(t))
\end{aligned}
$$

Commento

Si noti che le formule di cui sopra, semplici e sistematiche, in realtà non formalizzano *esattamente* le regole enunciate; bensì formalizzano un insieme di "comportamenti" che garantiscono la soddisfazione delle regole: da queste formule, usate come *assiomi*, è possibile *derivare come teorema*, ad esempio, il fatto che *Se un veicolo attraversa l'incrocio con semaforo rosso dopo 1 unità di tempo da quando è diventato rosso, viene elevata contravvenzione per Euro 200.*

Esercizio 2

In C, diversamente da altri linguaggi, come il Pascal, il ciclo **for** ha sufficiente generalità da permettere di simulare anche il ciclo **while**. E' noto che istruzione if e ciclo while (oppure l'istruzione if e la possibilità di usare la ricorsione nella chiamata di sottorpogrammi) permettono a un normale linguaggio di programmazione di avere la stessa potenza delle Macchine di Turing. Quindi il problema posto è equivalente a stabilire se una generica MT calcola una funzione totale o no. Questo problema è notoriamente indecidibile.

Esercizio 3

La normale analisi di complessità temporale dell'algoritmo di merge-sort assume implicitamente un criterio di costo costante. E' noto che essa produce un risultato $\Theta(n.\log(n))$. Assumendo il criterio di costo logaritmico occorre tener presente che ogni accesso a un singolo elemento costa un fattore log(n) (in questa valutazione interviene solo il costo dell'indirizzamento, poiché la memorizzazione dei singoli dati richiede un numero di bit limitato a priori). Quindi la complessità temporale valutata a criterio di costo logaritmico è $\Theta(n.\log^2(n))$.

Parte facoltativa

Per valutare la complessità spaziale dell'implementazione mediante RAM occorre tener presente che sono possibili diverse realizzazioni –ricorsive e non- dell'algoritmo di merge-sort. La più efficiente dal punto di vista della complessità spaziale fa uso alternato di due array (o file) e risulta quindi $\Theta(n)$ in entrambi i criteri.

Un'analoga implementazione mediante MT richiederebbe anch'essa una quantità di memoria $\Theta(n)$, mentre avrebbe una complessità temprale $\Theta(n.\log(n))$ grazie alla natura tipicamente sequenziale sia dell'algoritmo di merge-sort che della MT (risparmiando così il fattore log(n) per l'accesso al singolo dato).

Informatica Teorica

Appello dell'8 Febbraio 2007

Il tempo a disposizione per lo svolgimento del tema d'esame è 1h e 30'.

Esercizio 1 (punti 10)

Sia dato il linguaggio L sull'alfabeto {a, b, c} contenenti tutte e sole le stringhe che hanno le seguenti caratteristiche:

k) la stringa inizia e termina con *esattamente* lo stesso numero di 'a'

l) la stringa contiene almeno un carattere diverso da 'a'

Per esempio, le seguenti stringhe appartengono a L:

aabcaa, aca, aacabbabaa, aaaacaaaa, ...

Le seguenti stringhe, invece, **non** appartengono ad L:

aaabaa, aaaa, aaacaccaaaaaa, ecc.

Si scriva un automa *oppure* una grammatica che riconosca o generi il linguaggio L.

NB1: il punteggio massimo verrà dato se il formalismo scelto è a potenza minima tra quelli che riconoscono/generano L.

NB2: si scelga *una sola* tra le due possibilità: si scriva o un automa, oppure una grammatica, ma *non* entrambi.

Esercizio 2 (11)

L'algoritmo di ordinamento per conteggio serve ad ordinare un array di valori interi fornito in ingresso. Esso per funzionare si basa sulla ipotesi fondamentale che i valori da ordinare siano degli interi appartenenti all'insieme [0..k], con k prefissato. L'algoritmo sfrutta per funzionare un array ausiliario C di k+1 elementi, che serve a contenere il conteggio effettuato dall'algoritmo: in pratica C[j] contiene il numero di elementi presenti nell'array in ingresso con valore pari a j.

Si implementi un algoritmo di ordinamento per conteggio, sapendo che ogni singolo dato da ordinare è sempre un intero non negativo memorizzabile in 8 bit. Quali sono le sue complessità temporale e spaziale valutate a criterio di costo costante e a criterio di costo logaritmico? Giustificare brevemente la risposta.

NB: Si consiglia per comodità di descrivere l'algoritmo in pseudocodice, o mediante un linguaggio di alto livello.

Esercizio 3 (punti 11)

Il professor Touring ha preparato, per il prossimo appello del suo esame di Informatica 1, i 3 seguenti esercizi:

- Sia dato il seguente frammento di codice C:

```
1.  void DivisoreComune (int i1, int i2){
2.    int min, max;
3.
4.    if(i1 <= i2){
5.      min = i1;
6.      max = i2;
7.    } else {
8.      min = i2;
9.      max = i1;
10.   }
11.
12.   for(d = 1; d <= min; d++){
13.     if(min % d == 0 && max % d == 0)
14.       return d;
15.   }
16.
17.   return 0;
18. }
```

  Dire se ci sono errori di sintassi e, se ce ne sono, indicare quali.

- Si scriva un sottoprogramma che prende in ingresso due parole `p1` e `p2` di al massimo 20 caratteri l'una, e ritorna `true` se sono una l'anagramma dell'altra, `false` altrimenti.

- Siano P, Q, R, tre puntatori a interi e x, y due variabili intere. Si dica quanto valgono rispettivamente x, y, *P, *Q, *R dopo l'esecuzione della seguente sequenza di istruzioni.

```
x = 3; y = 5;
P = &x; Q = &y; R = P;
*R = 10; y = x + *Q; x = x + *P;
Q = R; P = Q
```

Per ognuno di questi esercizi dire, motivando adeguatamente la risposta, se è decidibile il problema di stabilire se la soluzione proposta da uno studente è corretta oppure no.

Soluzioni

Esercizio 1

La seguente grammatica noncontestuale genera il linguaggio L.

S → aSa | BXB | B

X → aX | bX | cX | ε

B → b | c

Esercizio 2

L'algoritmo puo' operare come segue. Si mantengono due puntatori a, c rispettivamente all'elemento corrente dell'array A da ordinare e all'elemento corrente dell'array C dei contatori.

1. Fase di input, che supponiamo venga terminata dalla lettura di -1, e di scrittura dei contatori:

```
for (c = 0 to 255)
   C[c] := 0
c := read();
a := 0;
while( c ≠ -1 ) {
   A[a] := c;
   C[c] := C[c]+1;
   c := read();
   a := a+1;
}
A[a] := -1;
```

2. Fase di scrittura dell'output, che scandisce C e sovrascrive A:

```
a := 0
for (c = 0 to 255) {
  while ( C[c] > 0 ) {
   A[a] := c;
   C[c] := C[c]-1;
   a := a+1; }}
```

Sia *n* il numero di elementi dell'array da ordinare.

- A **costo costante**, la *complessita' spaziale* e' $\Theta(n)$ in quanto memorizzo un array di dimensione n e uno di dimensione costante (infatti considero interi a 8 bit, quindi 256 possibili valori).
  La *complessita' temporale* e' pure $\Theta(n)$. Infatti il passo 1 consta di 1 ciclo eseguito $\Theta(n)$ volte. Nel passo 2, invece, il ciclo piu' interno e' eseguito un numero costante di volte, mentre quello esterno e' eseguito al piu' *n* volte. Quindi tutti i passi sono di complessita' temporale $\Theta(n)$, e lo stesso e' la loro composizione sequenziale.

- A **costo logaritmico**, il costo di memorizzazione dell'array da ordinare e' sempre $\Theta(n)$, in quanto in ogni cella ho un valore limitato da 256. L'array di contatori consta invece di 256 celle, ognuna delle quali costa $\Theta(\log(n))$. Quindi in tutto la *complessita' spaziale* e' ancora limitata superiormente da $\Theta(n)$.
  Per quanto riguarda la *complessita' temporale*, si manipolano dei contatori che sono limitati

superiormente da n. Dunque, in costo logaritmico, questo implica in fattore log(n) in ciascuna esecuzione dei cicli, che si traduce in una complessita' temporale complessiva di $\Theta(n \log(n))$.

Esercizio 3

1. La risposta si riduce a una lista, sicuramente finita visto che il frammento di programma e' finito, di potenziali errori di sintassi. Pertanto verificare la risposta corrisponde semplicemente a confrontare due liste di dimensioni finita, problema chiaramente decidibile.
2. Verificare la correttezza della risposta corrisponde a decidere se il programma/soluzione fornito dallo studente implementa la funzione richiesta oppure no. Come noto, decidere se un dato programma implementa una certa funzione e' un problema non decidibile. Si noti che il fatto che l'input del programma e' ristretto ad un numero finito di elementi (le stringhe di 20 caratteri) non cambia la risposta, dal momento che il formalismo usato per descrivere la soluzione (il codice) e' comunque Turing-completo.
3. In questo caso la risposta consiste semplicemente in 3 valori interi per *P, *Q, e *R. Controllare la validita' della risposta e' pertanto decidibile, in quanto si riduce al confrontro tra 3 coppie di interi.

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.

$S \rightarrow QR_1Q$

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

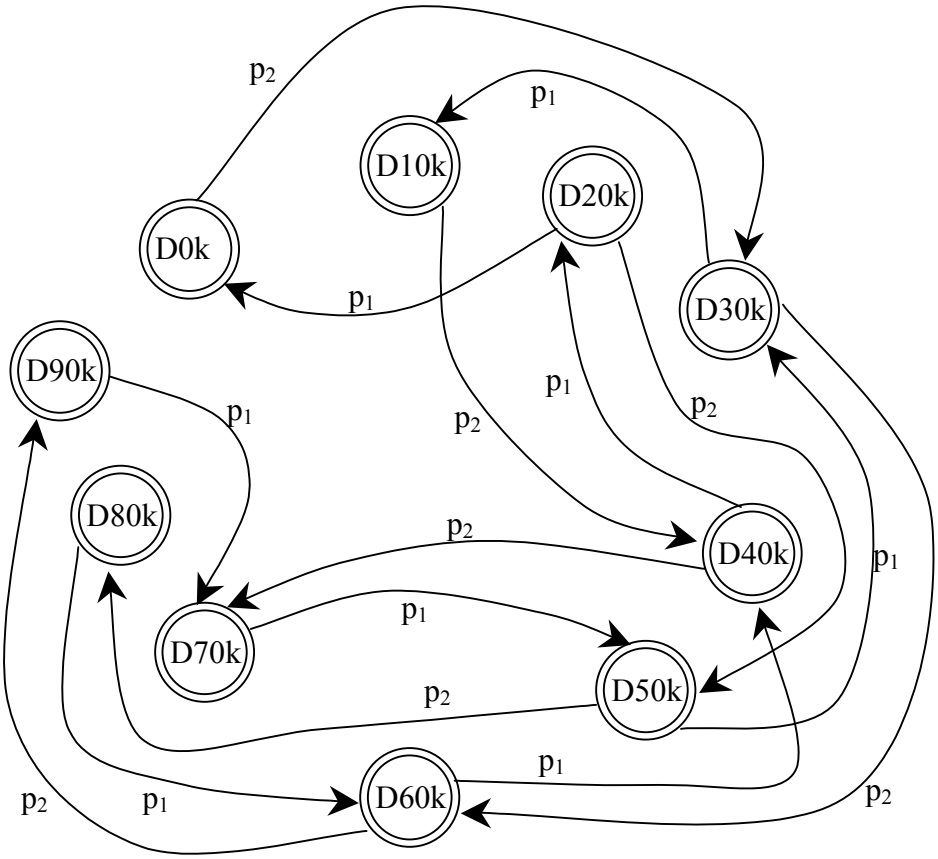
$R_1 \rightarrow aR_2a \mid bR_2b \mid cR_2c$

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

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

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

Seconda prova in itinere - 3 Luglio 2007

Avvisi importanti

- Il tempo disponibile per lo svolgimento della prova è **1 ora e 30 minuti**.
- Gli studenti laureandi **devono indicare esplicitamente la loro situazione nella testata del loro elaborato**.
- Come in altre occasioni i risultati e le successive comunicazioni (in particolare le modalità di accettazione/rifiuto del voto proposto) saranno pubblicati nella pagina personale del docente sul sito ufficiale del Dipartimento. Verrà data precedenza alla correzione dei compiti dei laureandi (che risultino tali dall'archivio ufficiale del CEDA)

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:

m) non è necessario che l'algoritmo sia efficiente o elegante, l'importante è che sia semplice;
n) 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:

$\sum_{k=1,\ldots,n} 2^k = \Theta(2^n)$ 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:

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

Infatti la singola iterazione costa log (2^{2n}) 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

Appello del 17 Luglio 2007

Avvisi importanti

- Il tempo disponibile per lo svolgimento della prova è **2 ore**.
- Come in altre occasioni i risultati e le successive comunicazioni (in particolare le modalità di accettazione/rifiuto del voto proposto) saranno pubblicati nella pagina personale del docente sul sito ufficiale del Dipartimento.

Esercizio 1 (punti 12)

Parte A

Si considerino le grammatiche seguenti:

G1: $S \rightarrow SS \mid ASB \mid BSA \mid \varepsilon$

$A \rightarrow a$

$B \rightarrow b$

G2: $S \rightarrow SBA \mid \varepsilon$

$BA \rightarrow AB$

$AB \rightarrow BA$

$A \rightarrow a$

$B \rightarrow b$

G3: $S \rightarrow ABCS \mid$ e tutte le permutazioni della stringa ABCS $\mid \varepsilon$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

G4: $S \rightarrow ABCS \mid \varepsilon$

$BA \rightarrow AB$

$AB \rightarrow BA$

$CB \rightarrow BC$

$BC \rightarrow CB$

$AC \rightarrow CA$

$CA \rightarrow AC$

$A \rightarrow a$

$B \rightarrow b$

C → c

Si dica, motivando brevemente la risposta, se G1 e G2, e G3 e G4 sono tra loro equivalenti.

Parte B

Si dica, motivando brevemente la risposta, se i linguaggi generati da G1, G2, G3 e G4 sono riconoscibili da automi a pila (anche nondeterministici).

(continua sul retro)

Esercizio 2 (Punti 12)

Per una funzione $F:N \rightarrow N$, un elemento x del suo dominio di definizione è detto *mimimo globale di F* se F(x) è definita e, per ogni valore y in cui F(y) è definita, $F(x) \leq F(y)$, dove $\leq$ definisce l'usuale relazione di "minore o uguale" tra numeri naturali.

1. Si determini se il problema di trovare se una generica funzione computabile f ha un minimo globale è decidibile.
2. Si determini se lo stesso problema è semidecidibile.
3. Come cambia, se cambia, la risposta ai quesiti precedenti se si considerano solo funzioni totali e computabili?

Esercizio 3 (Punti 10)

Si consideri il problema della ricerca di un elemento x in un array S di lunghezza n. In tutto l'esercizio, consideriamo solo la complessità temporale e il criterio di costo costante.

Come noto, l'algoritmo più immediato per risolvere il problema è quello della ricerca sequenziale, che scandisce tutto l'array sequenzialmente alla ricerca dell'elemento x, e raggiunge una complessità di $\Theta(n)$ nel caso pessimo.

Si consideri ora il problema di trovare un algoritmo per risolvere lo stesso problema che sia *asintoticamente peggiore* della ricerca sequenziale, ossia raggiunga una qualche complessità $\Theta(p(n))$, con p(n) funzione totale e computabile, tale che $\lim_{n\to\infty} \frac{n}{p(n)} = 0$.

1. Si descriva un algoritmo di ricerca *asintoticamente peggiore* della ricerca sequenziale, e se ne valuti la complessità temporale, mostrando che è effettivamente asintoticamente peggiore.
2. Un algoritmo A che risolve un problema P con complessità $T_A(n)$ è detto *pessimo* se ogni altro algoritmo B che risolve P con complessità $T_B(n)$ è tale che: o A è asintoticamente peggiore di B, oppure $T_A(n) \quad \Theta(T_B(n))$.
   Per un generico problema P, esiste sempre un algoritmo pessimo che lo risolve? Esiste un problema P che ammette un algoritmo pessimo per la sua soluzione? Motivare brevemente le risposte.

Soluzioni

Esercizio 1

Parte A

G1 e G2 sono equivalenti. Infatti entrambe generano tutte e sole le stringhe con ugual numero di a e b.

G3 e G4 non lo sono. Infatti G3, pur generando stringhe con ugual numero di a, b, c non può generare, ad esempio stringhe del tipo $a^n b^n c^n$.

Parte B

Poiché G1 e G3 sono non contestuali, i linguaggi da esse generati sono certamente riconoscibili da automi a pila. Essendo G2 equivalente a G1, lo stesso vale per L(G2).

Invece G4 richiede il riconoscimento di stringhe del tipo $a^n b^n c^n$ che devono essere distinte, ad esempio, da stringhe come $a^n b^n c^{2n}$. Ciò richiede una capacità di "conteggio illimitato doppio" che, come ben noto, non rientra tra le possibilità dell'automa a pila.

Esercizio 3

1. Il problema non è decidibile, in quanti riducibile al problema di determinare se una generica funzione computabile è definita per almeno un valore del suo dominio, problema che notoriamente non è decidibile.
   Infatti, sulla base della definizione data, tenendo conto che l'insieme dei numeri naturali è discreto e limitato inferiormente, una funzione computabile $f: N \to N$ ha minimo globale se e solo se è definita in almeno un punto.
2. Sia f una funzione totale e computabile, e sia $I \subseteq N$ la sua immagine. Essendo I un insieme limitato inferiormente (da 0), è sicuramente definito l'estremo inferiore $i = \inf I$. Inoltre, essendo I un insieme discreto, i appartiene a I, quindi $i = \min I$. Il valore x tale che $f(x) = i$ è il minimo di f, che dunque esiste sempre per funzioni totali e computabili a valori in N. Dunque il problema è deciso, e quindi a maggior ragione decidibile.
   Per ripassare un pò di matematica, ecco un'altra giustificazione che I ha sicuramente minimo. Un insieme S è detto ben ordinato (well-ordered) sse ogni suo sottinisieme non vuoto ha un elemento minimo. Per assurdo, sia I *non* ben ordinato. Allora esiste un suo sottoinsieme $\varnothing \neq J \subseteq I$ che non ammette minimo. Sia K il complemento, rispetto a N, di J. Se $0 \in J$, esso è anche minimo; quindi $0 \notin J$ e dunque $0 \in K$. Per induzione, tutti i naturali appartengono a K e dunque J è vuoto, contrariamente all'ipotesi. Dunque I è ben ordinato, e dunque ha minimo.

Esercizio 4

1.

Data una qualunque funzione computabile p(n) tale che $\lim_{n\to\infty} \frac{n}{p(n)} = 0$, si può generare un algoritmo di ricerca che abbia complessità Ω(p(n)) semplicemente "peggiorando" un normale algoritmo di ricerca sequenziale aggiungendo l'esecuzione di un numero di istruzioni pari a p(n), ove n è la lunghezza dell'array ove si sta cercando. In pseudocodice:

```
ricerca_p_peggiore(x, S):
1     n = |S|
2     max = p(n)
3     for i:=1 to max do
4           continue;
5     return ricerca_sequenziale(x, S)
end
```

La complessità dell'istruzione 5 è chiaramente pari a Θ(n). Sia c(n) la complessità dell'istruzione 2, ossia il costo di calcolare p(n). Infine il ciclo **for** delle istruzioni 3-4 ha complessità Θ(p(n)). In totale quindi, ricordando che p(n) cresce asintoticamente più di n, si ha una complessità Θ(c(n) + p(n)) che è Ω(p(n)).

2.

Nella soluzione di 1. abbiamo mostrato come la complessità si possa "peggiorare" arbitrariamente. Dunque per qualsiasi problema non ha senso cercare la complessità pessima perchè è sempre possibile scegliere una funzione che cresce più velocemente e peggiorare l'algoritmo di quel fattore.

Informatica Teorica

Appello dell'11 Settembre 2007

Avvisi importanti

- Il tempo disponibile per lo svolgimento della prova è 2 ore.
- Come in altre occasioni i risultati e le successive comunicazioni (in particolare le modalità di accettazione/rifiuto del voto proposto) saranno pubblicati nella pagina personale del docente sul sito ufficiale del Dipartimento.

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:

S → ADaCD | DAbDC

DC → CD | AA

A → a | SSAD

C → ADC | c

SA → AS | ε

G2:

S → aADCD | DbADC

DC → CD | AA

A → a | SSAD | ε

C → ADC | c

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

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.

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

3. 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

1) 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.

2) 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

Prova d'esame - 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)

- 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.
- 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.

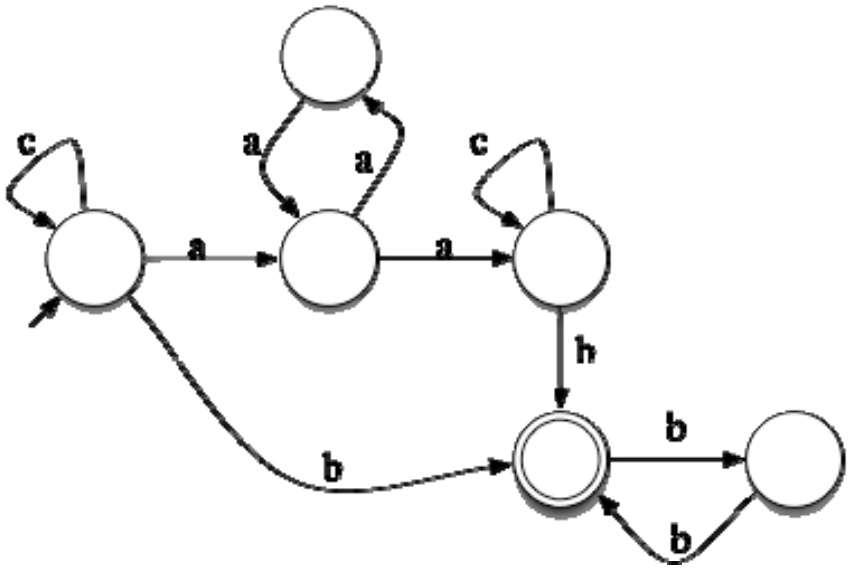
Soluzioni

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 Θ(K) e complessità temporale Θ(n) sia a cirterio di costo costante che a criterio di costo logaritmico.

Esercizio 3

1. Il problema è indecidibile in quanto è il calassico 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).

Informatica Teorica

Prima prova in itinere - 8 Maggio 2008

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 (10/13 punti)

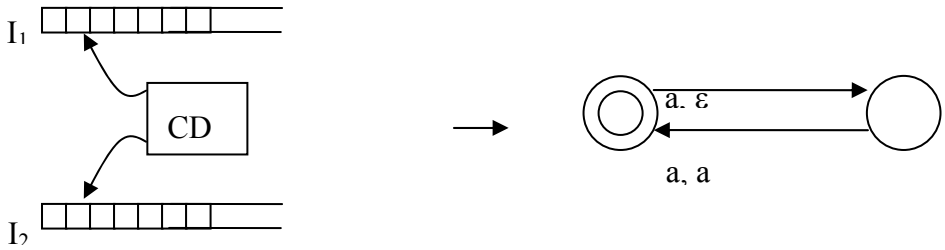
Parte 1

Si formalizzi la nozione di automa trasduttore finito deterministico FTε1 che, al contrario del modello di trasduttore finito presentato a lezione (chiamiamolo FT), può anche effettuare ε-mosse ma può emettere, a ogni transizione, al più un solo simbolo sul nastro di uscita.

Dimostrare che il modello FTε1 è almeno altrettanto potente del modello FT deterministico (ossia che qualsiasi FT può essere simulato da un opportuno FTε1).

Parte 2

Formalizzare poi la nozione di automa finito deterministico a due ingressi, 2FA, tratteggiato in figura a sinistra.



2FA può fare ε-mosse, non facendo avanzare la testina di ingresso su uno o entrambi i nastri di ingresso I_1 e I_2. L'automa 2FA può essere visto come traduttore nel modo seguente: la stringa di ingresso x viene accettata e tradotta in $\tau(x)$ se e solo se la coppia di stringhe $<x, \tau(x)>$ sui due nastri I_1 e I_2 è accettata. Per esempio, l'automa 2FA nella figura a destra definisce la traduzione $\tau(a^n)=a^{n/2}$ per $n\geq 0$ e pari. Mostrare che il modello 2FA è almeno altrettanto potente di FT deterministico, nel senso che se un FT deterministico accetta una stringa x e la traduce nella stringa $\tau(x)$, allora esiste un opportuno 2FA che accetta la coppia di stringhe $<x, \tau(x)>$ sui due nastri I_1 e I_2.

Valutare se il modello 2FA ha *esattamente* la stessa potenza espressiva di FT, cioè se qualsiasi traduzione definita da un 2FA nel modo sopra indicato può essere calcolata da un opportuno FT.

Esercizio 2 (4/13 punti)

Scrivere una grammatica G che generi il linguaggio $\{a^n b^n$ se n è pari >= 0, $a^n c^n$ se n è dispari$\}$

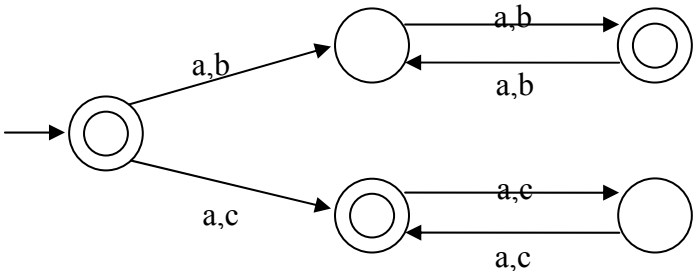
Soluzioni

Esercizio 1

- Un FTε1 è una tupla (Q, I, O, δ, q_0, F) dove Q, I, O, q_0, e F hanno lo stesso significato che negli automi trasduttori tradizionali. La funzione di transizione δ: $Q \times I \cup \{\varepsilon\} \rightarrow Q \times O \cup \{\varepsilon\}$ definisce, per ogni coppia (q, i) di stato corrente e carattere in input (oppure ε) lo stato prossimo q' e il singolo carattere scritto in output (oppure ε), cioè δ(q,i) = (q', o). Affinchè sia deterministico si richiede che se δ(q,ε) è definito, allora δ(q,i) è indefinito per tutti gli $i \in I$.
  Conviene poi fornire la definizione di configurazione, transizione tra configurazioni, e trasduzione seguendo la prassi standard usata per automi a pila e macchine di Turing.
- Dato un FT = (Q, I, O, δ, q_0, F) deterministico, è sempre possibile definite un FTε1 = (Q', I, O, δ', q_0, F) che definisce la stessa traduzione facendo in modo che, quando FT emette una stringa di lunghezza >1, l'FTε1 corrispondente emetta in sequenza i caratteri della stringa, uno alla volta, mediante una serie di ε-mosse che lo fanno passare attraverso una serie di stati aggiuntivi, diversi da tutti gli altri, introdotti a questo scopo. Q' e δ' sono quindi definiti in questo modo. Q' contiene tutti gli stati che sono in Q. Inoltre, per ogni δ(q,i) = (q', s) con $s \in O^*$, se $|s| \leq 1$ si ha semplicemente δ'(q,i) = (q', s); se $|s| > 1$si aggiungono |s| stati q^s_1, q^s_2, ..., $q^s_{|s|}$ a Q' e si pone: (1) δ'(q,i) = (q^s_1, s[1]); (2) per ogni $2 \leq k \leq |s|$: δ'(q^s_{k-1},ε) = (q^s_k, s[k]); (3) δ'($q^s_{|s|}$,ε) = (q', ε);.
- Un 2FA è una tupla (Q, I_1, I_2, δ, q_0, F) dove I1 e I2 sono gli alfabeti dei due nastri di ingresso, Q, q_0, e F hanno il significato convenzionale. La funzione di transizione δ: $Q \times I_1 \cup \{\varepsilon\} \times I_2 \cup \{\varepsilon\} \rightarrow Q$ definisce per ogni tupla (q, x, y) di stato corrente q, carattere sul primo nastro x (o ε), e carattere sul secondo nastro y (o ε) lo stato prossimo δ(q, x, y). Si richiede per il determinismo che se δ(q, x, ε) (rispettivamente δ(q, ε, y)) è definito allora δ(q, x, y) è indefinito per ogni $y \in I_2$ (rispettivamente per ogni $x \in I_1$).
  La definizione di configurazione e transizione tra configurazioni sono definite di conseguenza nel solito modo (includendo nella configurazione il contenuto dei due nastri dalla posizione della testina in poi).
- Il modello 2FA è almeno altrettanto potente del modello FTε1 (e quindi, transitivamente, del modello FT): dato un qualsiasi FTε1, esiste un 2FA che simula ogni sua transizione, leggendo dal secondo nastro di ingresso lo stesso carattere che l'FTε1 emette sul nastro di uscita ed effettuando ε-mosse sul primo o sul secondo nastro di ingresso quando, rispettivamente, l'FTε1 fa una ε-mossa (i.e., non consuma ingresso) o non emette alcun simbolo in uscita. Formalmente, dato un FTε1 = (Q, I, O, δ, q_0, F) si definisce il 2FA = (Q, I, O, δ', q_0, F) dove δ'(q, x, y) = q' se e solo se δ(q, x)=(q',y), $\forall q,q' \in Q$ e $\forall x \in I \cup \{\varepsilon\}$ e $\forall y \in O \cup \{\varepsilon\}$.
- Il modello 2FA è strettamente più potente del modello FT, perchè permette di definire una traduzione, accettandola come stringa in ingresso sul secondo nastro, anche nel caso in cui la traduzione dipenda da una proprietà della stringa di ingresso (quella sul primo nastro) che può essere decisa solo al termine della scansione, dopo un numero di passi illimitato a priori. Ciò non può essere fatto dal modello FT, a causa del vincolo di memoria finita che lo obbliga a emettere la traduzione "in tempo reale", durante la scansione della stringa in ingresso. Come esempio di ciò si consideri la seguente traduzione: $\tau(a^n)=b^n$ se n è pari, $\tau(a^n)=c^n$ se n è dispari. Nessun FT può calcolare questa traduzione, perchè il riconoscimento della stringa come avente lunghezza pari o dispari avviene solo al termine della scansione,

mentre l'emissione della traduzione deve avvenire durante la scansione. Il 2FA mostrato in figura invece *definisce* la traduzione in questione accettando il linguaggio L={(a^n,b^n) | n≥0, n pari }∪{(a^n,c^n) | n dispari }

.



Esercizio 2

S → X | Y

X → aAb | ε

A → aXb

Y → aBc

B → aYc | ε

Theoretical Computer Science

(English version, with slightly more detailed solutions)

Midterm exam – May 8th 2008

NB: the total score is 13, reflecting the relative weight of this part of the exam w.r.t. the total of 30 for the whole course. Just like in previous occasions it is possible, however, to get a grade higher than 13.

Exercise 1 (10/14 points)

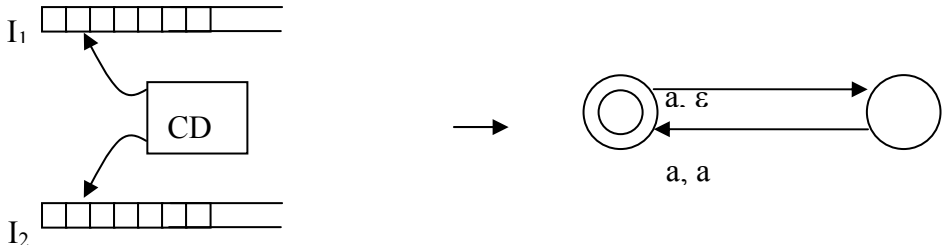
Part 1

Please formalize the notion of a deterministic finite transducer FTε1 that, unlike the finite transducer model presented at lesson (let us call that one FT), can also have ε-moves but can emit, at every transition step, at most one symbol on the output tape.

Prove that the FTε1 model is at least as powerful as the deterministic FT (i.e., any possible deterministic FT can be simulated by a suitable FTε1).

Part 2

Please formalize the notion of a two-input deterministic finite automaton 2FA, outlined in the figure on the left.



The 2FA may make ε-moves, i.e., it may leave the scanning head still on one or both of the input tapes I_1 and I_2. The 2FA automaton can be viewed as a transducer in the following way: the input string x is accepted and translated into $\tau(x)$ if and only if the string pair $<x, \tau(x)>$ is accepted on the two input tapes I_1 and I_2. For instance, the 2FA automaton in the figure on the right defines the translation $\tau(a^n)=a^{n/2}$ for n≥0 and n even.

Please show that the 2FA model is at least as powerful as the deterministic FT, i.e., if a deterministic FT accepts a string x and translates it into the string $\tau(x)$, then there exists a suitable 2FA that accepts the string pair $<x, \tau(x)>$ on the two input tapes I_1 and I_2.

Please determine if the 2FA model has *exactly* the same expressive power as the FT model, i.e., if any translation defined by a 2FA in the manner indicated above can be computed by a suitable FT.

Exercise 2 (4/14 points)

Write a grammar G that generates the language {$a^n b^n$ if *n* is even >= 0, $a^n c^n$ if *n* is odd}

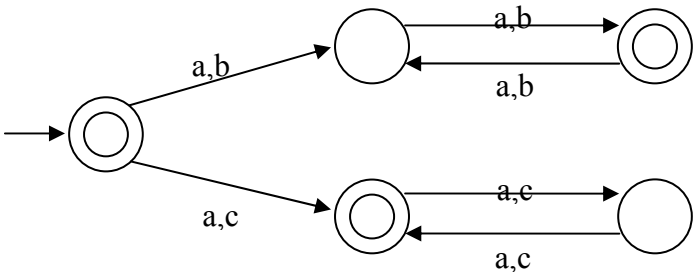
Solutions

Exercise 1

- An FTε1 is a tuple (Q, I, O, δ, q_0, F) where Q, I, O, q_0, and F have the same meaning as in traditional finite transducers. The transition function δ: $Q \times I \cup \{\varepsilon\} \rightarrow Q \times O \cup \{\varepsilon\}$ defines, for each pair (q, i) of current state and input character (or ε) the next state q' and the unique character written on the output (or ε), i.e., δ(q, i) = (q', o). To make the automaton deterministic it is required that, if δ(q, ε) is defined, then δ(q, i) is not defined for all $i \in I$. Following the usual notations adopted for Stack automata and Turing machines, the configuration of a FTε1 is defined as a tuple $(q, x, y) \in Q \times I^* \times O^*$, q being the current state, x the string still to be read on the input, and y the string output so far. The transition relation |-- among configurations is defined as follows. If δ(q, ε)=(q', o) then (q, x, y)|--(q', x, y.o); if δ(q, i)=(q', o) then (q, i.x, y)|--(q', x, y.o); the relation |-*- is defined as the reflexive transitive closure of |--; a string $x \in I_1$ is accepted if and only if, for some $q_f \in F$ and $y \in O^*$, (q_0, x, ε) |-*- (q_f, ε, y); in that case the translation performed by FTε1 on x is define as $\tau(x) = y$.
- Given a deterministic FT = (Q, I, O, δ, q_0, F), it is always possible to define a FTε1 = (Q', I, O, δ', q_0, F) that defines the same translation by making sure that, when the FT emits a string of length >1, the corresponding FTε1 emits in a sequence the characters composing the string, one at the time, by means of a series of ε-moves that make it go through a sequence of additional states, different from all the other ones and suitably introduced for this purpose. Q' and δ' are hence defined as follows. $Q \subseteq Q'$ and besides, for each δ(q,i) = (q', s) with $s \in O^*$, if $|s| \leq 1$ then simply δ'(q,i) = (q', s); if $|s| > 1$ the |s| new states are added to Q', q^s_1, q^s_2, ..., $q^s_{|s|}$ and: (1) δ'(q,i) = (q^s_1, s[1]); (2) for each $2 \leq k \leq |s|$: δ'(q^s_{k-1},ε) = (q^s_k, s[k]); (3) δ'($q^s_{|s|}$,ε) = (q', ε).
- A 2FA is a tuple (Q, I_1, I_2, δ, q_0, F) where I_1 and I_2 are the alphabets of the two input tapes, Q, q_0, and F have the usual meaning. The transition function δ: $Q \times I_1 \cup \{\varepsilon\} \times I_2 \cup \{\varepsilon\} \rightarrow Q$ defines, for each tuple (q, x, y) of current state q, input character on the first tape x (or ε), and input character on the second tape y (or ε) the next state δ(q, x, y). To make the device deterministic it is required that if δ(q, x, ε) (respectively, δ(q, ε, y)) is defined, then δ(q, x, y) is not defined for every $y \in I_2$ (respectively, for every $x \in I_1$).
  The configuration of a 2FA is defined as a tuple $(q, x, y) \in Q \times I_1^* \times I_2^*$, q being the current state, x and y the strings still to be read on the first and second input. The transition relation |-- among configuration is defined as follows. (q, x, y) |-- (q', w, z) in the following cases: δ(q, ε, ε) = q' and w=x and z=y, or δ(q, i_1, ε)=q' with $i_1 \in I_1$ and x=i_1.w and z=y, or δ(q, ε, i_2)=q' with $i_2 \in I_2$ and w=x and y = i_2.z, or δ(q, i_1, i_2)=q' with $i_1 \in I_1$ and with $i_2 \in I_2$ and x=i_1.w and y = i_2.z. The relation |-*- is defined as the reflexive transitive closure of |--; a pair of strings (x, y) is accepted if and only if, for some $q_f \in F$, (q_0, x, y) |-*- $(q_f, \varepsilon, \varepsilon)$.
- The 2FA model is at least as powerful as the FTε1 model (and hence, transitively, as the FT model): given any FTε1, there exists a 2FA that simulates all its transitions, by reading from the second input symbol the same character that the FTε1 writes on its output tape, and performs ε-moves on the first or second input tape when, respectively, the FTε1 makes an ε-moves (i.e., it does not consume its input) or it does not emit any output symbol. Formally, given an FTε1 = (Q, I, O, δ, q_0, F) one can define the 2FA = (Q, I, O, δ', q_0, F) where δ'(q, x, y) = q' if and only if δ(q, x)=(q',y), $\forall q,q' \in Q$ and $\forall x \in I \cup \{\varepsilon\}$ and $\forall y \in O \cup \{\varepsilon\}$.

- The 2FA model is *strictly more* powerful than the FT model, because it allows one to define a translation, by accepting the string result of the translation as input on the second tape, also in the case when the translation depends on a property of the input string (the string on the first input tape) that can be decided only at the end of its scanning, after a number of steps that is *a priori* unlimited. This cannot be done by the FT, due to the finite memory constraint that forces it to emit the translation "in real time", during the scanning of the input string. As an example, let us consider the following translation: $\tau(a^n)=b^n$ if n is even and $n\geq 0$, $\tau(a^n)=c^n$ if n is odd. No FT can compute this translation, because the decision that the string has even or odd length can be taken only at the end of the scanning of the input, while the translation must be written during the scanning of the input. Instead, the 2FA shown in the figure below *defines* the translation by accepting the language $L=\{(a^n,b^n) \mid n\geq 0, n \text{ even}\} \cup \{(a^n,c^n) \mid n \text{ odd}\}$.

.



Exercise 2

S → X | Y

X → aAb | ε

A → aXb

Y → aBc

B → aYc | ε

Informatica Teorica

Seconda prova in itinere – 30 Giugno 2008, Sezione Mandrioli

Il tempo a disposizione è di 2 ore e 30 minuti

Esercizio 1 (punti 8/17-esimi)

Si formalizzi mediante formule del prim'ordine il seguente comportamento di un sistema di allarme a tempo continuo:

- L'allarme si attiva e disattiva inserendo la chiave nell'apposito alloggiamento; mentre la disattivazione è immediata, l'attivazione avviene dopo 12 secondi dall'introduzione della chiave; un eventuale reinserimento della chiave durante questo periodo non ha effetto; l'estrazione della chiave dall'alloggiamento non ha alcun effetto e può essere trascurata.
- Ad allarme attivato l'ingresso di un corpo estraneo nel volume sotto controllo determina lo scatto dell'allarme dopo 12 secondi, a meno che durante tale intervallo non venga inserita la chiave nell'alloggiamento per disinserirlo.

Si suggerisce di usare i seguenti predicati logici (dall'ovvio significato), tutti quanti aventi un parametro a valori reali: *attiva_allarme*(*t*), *disattiva_allarme*(*t*), *allarme_attivo*(*t*), *inserimento_chiave*(*t*), *rilevamento_corpo_estraneo*(*t*), *scatto_allarme*(*t*).

Esercizio 2 (punti 6/17-esimi)

Si dica, giustificando brevemente la risposta, se i seguenti problemi sono decidibili o semidecidibili:

o) Stabilire se, data una generica macchina di Turing a nastro singolo e due configurazioni della medesima, le due configurazioni sono tra loro in relazione di transizione immediata.

p) Stabilire se, data una generica macchina di Turing a nastro singolo e due configurazioni della medesima, le due configurazioni sono tra loro in relazione di transizione non necessariamente immediata.

Esercizio 3 (punti 6/17-esimi)

- Si supponga che una macchina di Turing deterministica simuli il comportamento di un automa a stati finiti nondeterministico nel seguente modo: ricevendo in ingresso una stringa, essa riproduce, una dopo l'altra, tutte le possibili esecuzioni che l'automa nondeterministico può svolgere su tale stringa. Si indichi la complessità minima (col criterio del caso pessimo) che tale simulazione richiede, sia per il tempo che per lo spazio, in funzione della lunghezza *n* della stringa in ingresso *x*.
- Si supponga che una macchina di Turing deterministica abbia in ingresso la descrizione di un automa a stati finiti nondeterministico, e una stringa, da intendere come ingresso per l'automa, e scriva in uscita il valore 1 se la stringa appartiene al linguaggio accettato dall'automa, 0 altrimenti. Si valutino, giustificando brevemente la risposta:
    - la complessità temporale e quella spaziale minime (col criterio del caso pessimo) come funzione del numero *m* degli stati dell'automa in ingresso;
    - la complessità temporale e quella spaziale minime (sempre col criterio del caso pessimo) come funzione della lunghezza *n* della stringa in ingresso *x*.

Soluzioni schematiche

Esercizio 1

inserimento_chiave, *attiva_allarme*, *disattiva_allarme*, *rilevamento_corpo_estraneo*, *scatto_allarme* sono definiti a priori come eventi, ossia istantanei, *allarme_attivo* è invece definito a priori come uno stato che ha quindi una durata.

La formula seguente esprime i requisiti richiesti mediante la sintassi del calcolo dei predicati.

$$\forall t\left(\begin{array}{l} attiva\_allarme(t) \leftrightarrow \\ \quad inserimento\_chiave(t-12) \wedge \neg allarme\_attivo(t-12) \wedge \\ \quad \forall t_1(t-12 < t_1 < t \rightarrow \neg attiva\_allarme(t_1)) \end{array}\right)$$

$$\wedge$$

$$\forall t\left(disattiva\_allarme(t) \leftrightarrow inserimento\_chiave(t) \wedge allarme\_attivo(t)\right)$$

$$\wedge$$

$$\forall t\left(\begin{array}{l} allarme\_attivo(t) \leftrightarrow \\ \quad \exists t_1\left(\begin{array}{l} t_1 < t \wedge attiva\_allarme(t_1) \wedge \\ \forall t_2(t_1 < t_2 < t \rightarrow \neg disattiva\_allarme(t_2)) \end{array}\right) \end{array}\right)$$

$$\wedge$$

$$\forall t\left(\begin{array}{l} scatto\_allarme(t) \leftrightarrow \\ \quad rilevamento\_corpo\_estraneo(t-12) \wedge allarme\_attivo(t-12) \wedge \\ \quad \forall t_1(t-12 < t_1 < t \rightarrow \neg inserimento\_chiave(t_1)) \end{array}\right)$$

Esercizio 2

Quesito 1

Il problema è sicuramente decidibile: è infatti immediato costruire un semplice algoritmo che scandisca la configurazione iniziale e determini il simbolo in lettura e lo stato della macchina, individui la mossa definita dalla funzione δ della macchina e verifichi se la seconda configurazione coincide con quella che si otterrebbe applicando la δ.

Quesito 2

Questo problema è invece non decidibile. Infatti il classico problema dell'HALT è un caso particolare del problema posto: stabilire cioè se da una configurazione iniziale si può giungere a una configurazione di HALT. NB: per la precisione, visto che in generale vi possono essere diverse di configurazioni di HALT, per ricondurre il problema dell'HALT esattamente al problema in questione è necessario modificare la macchina in modo tale che essa abbia un'unica configurazione di HALT (ad esempio, nastro vuoto e stato q_F.)

Il problema è invece semidecidibile in quanto è sufficiente "far girare" la macchina a partire dalla prima configurazione: se la seconda è raggiungibile, prima o poi la si raggiunge e si risolve il problema. Non vale ovviamente il viceversa.

Esercizio 3

Quesito 1

Il meccanismo di simulazione della MT è quello classico di visita dell'albero delle computazioni; quindi la complessità temporale è nel caso pessimo $\Theta(2^n)$ (per semplicità assumiamo che l'albero sia binario). Se la macchina costruisce l'intero albero delle computazioni anche la complessità spaziale è dello stesso ordine. Tuttavia la macchina potrebbe limitarsi a memorizzare solo l'ultimo cammino visitato, ad esempio, in preordine sinistro; in tal modo potrebbe generarli ed esaminarli tutti in sequenza (ad esempio:

S, S, S, S, S -→ S,S,S,S,D → S, S, S, D, S → S, S, S, D, D → S, S, D, S, S ….)

senza costruire l'intero albero. In tal caso la complessità spaziale risulta $\Theta(n)$.

Quesito 2

Consideriamo una macchina di Turing che costruisce l'automa deterministico equivalente a quello non deterministico di partenza, e quindi effettua il riconoscimento della stringa simulando direttamente l'automa deterministico.

a.

E' noto che nel caso pessimo l'automa deterministico, equivalente a quello in input di m stati, ha $\Theta(2^m)$ stati e quindi la complessità spaziale della macchina è dominata da questo fattore. Allo stesso modo, la complessità temporale è dominata dalla costruzione di tale automa.

b.

Una volta costruito l'automa deterministico la MT può semplicemente simulare quest'ultimo con la stessa sua complessità, ossia $\Theta(n)$ per la complessità temporale e $\Theta(1)$ per quella spaziale, tenendo conto ovviamente che la memoria finita costituita dagli $\Theta(2^m)$ stati non è funzione della lunghezza della stringa di ingresso *x*.

Soluzione alternativa del Quesito 2:

Considerando i quesiti a e b come *indipendenti*, possiamo scegliere un algoritmo diverso in ciascuno dei due casi, puntando unicamente a minimizzare la complessità rispetto al parametri considerato nel punto in questione.

Per il punto **b** la soluzione basata sulla determinizzazione suggerita nella soluzione precedente rimane la migliore dal punto di vista della complessità ottenibile.

Per il punto **a**, invece, possiamo utilizzare di nuovo l'algoritmo di simulazione del punto 1. La complessità spaziale è quella necessaria a memorizzare l'ultimo cammino visitato; essendo un cammino lungo n ed essendo necessari log m bit per memorizzare (sinteticamente in binario) l'informazione su uno degli m stati, abbiamo una complessità spaziale di $\Theta(\log(m))$ visto che n non è funzione di m. Similmente, la complessità temporale è quella necessaria ad esplorare l'albero delle 2^n computazioni, dove ogni passo costa l'aggiornamento dello stato corrente, ossia log(m). Quindi anche la complessità temporale è $\Theta(\log(m))$. Ovviamente questa valutazione di complessità è formalmente corretta ma irrealistica perchè ignora completamente la dimensione di parte dell'input (e i suoi effetti sulla complessità).

Informatica Teorica

Appello d'esame del 16 luglio 2008, Sezione Mandrioli

Si consideri la grammatica G seguente:

S → BAcB | ε

A → aAa | acBa

B → CcB | ε

C → aC | a

Esercizio 1 (punti 11)

Si fornisca un automa che riconosca il linguaggio L(G) generato da G.

L'automa deve appartenere alla classe di minima potenza riconoscitiva possibile per riconoscere L(G).

Si spieghi brevemente perché la classe dell'automa scritto è quella a potenza minima possibile.

Esercizio 2 (punti 12)

q) Si dica, giustificando brevemente la risposta, se è decidibile il problema di stabilire se L(G) intersecato con il linguaggio {(a*c)*} produce il linguaggio vuoto o no.

r) Si dica, giustificando brevemente la risposta, se è decidibile il problema di stabilire se, dato un generico automa a stati finiti A, il linguaggio L(A) riconosciuto da A è uguale al linguaggio L(G) oppure no.

s) Si dica, giustificando brevemente la risposta, se, ***fissata*** una grammatica G', è decidibile il problema di stabilire se, dato un generico automa a stati finiti A, il linguaggio L(A) riconosciuto da A è uguale al linguaggio L(G') generato da G' oppure no.

t) Si dica, giustificando brevemente la risposta, se è decidibile il problema di stabilire se, dati una ***generica*** grammatica G' ed un generico automa a stati finiti A, il linguaggio L(A) riconosciuto da A è uguale al linguaggio L(G') generato da G' oppure no.

Esercizio 3 (punti 10)

Si descrivano a grandi linee il funzionamento di una macchina di Turing deterministica e di una RAM che riconoscano L(G) e se ne valutino le complessità secondo la classe Θ, usando per la RAM il criterio di costo logaritmico.

Soluzioni schematiche

Esercizio 1

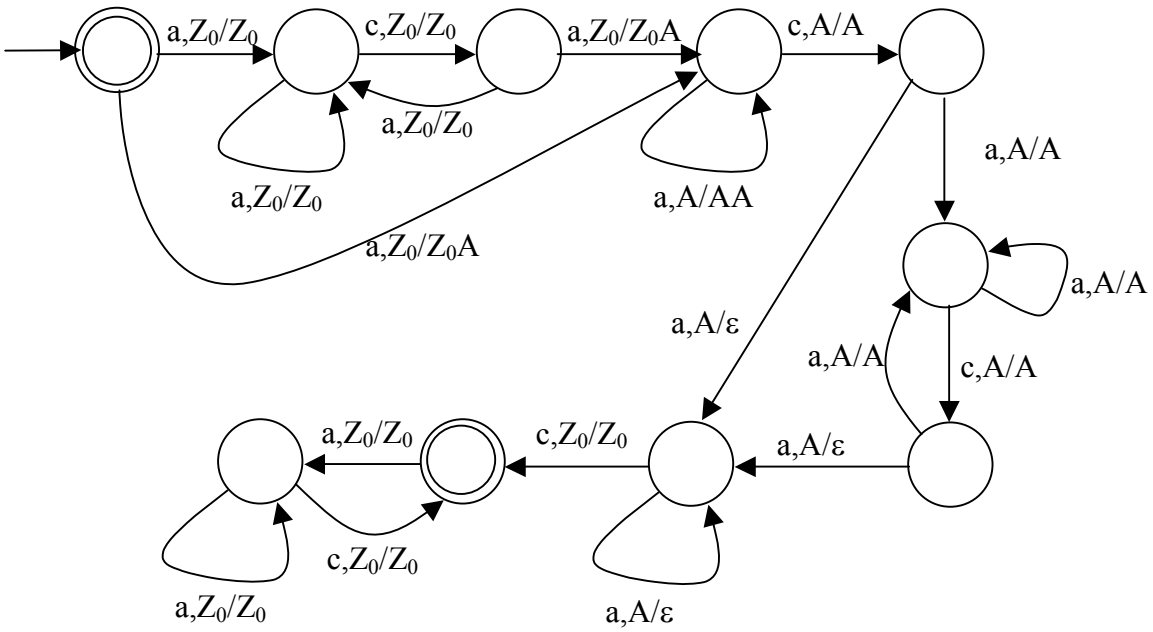
La grammatica genera il linguaggio:

$$L(G) = \{ (a^+c)^*a^nc(a^+c)^*a^nc(a^+c)^* \mid n > 0 \} \cup \varepsilon,$$

dove $x^+ = x^* - \{\varepsilon\}$.

Per riconoscere L(G) è necessario e sufficiente un automa a pila nondeterministico. La pila è necessaria per effettuare il conteggio delle sottostringhe a^n. Anche il nondeterminsmo è necessario perchè con il solo determinismo non è possibile stabilire in quale punto delle sottostringhe fatte di 'a' è necessario iniziare ad usare la pila per contare tali 'a'. In altre parole, il nondeterminismo serve a discernere le sottostringhe del tipo $(a^+c)^*$, per le quali è sufficientemente un riconoscimento a stati finiti senza pile, da quelle del tipo a^n per le quali la pila è invece ovviamente necessaria.

Un possibile ND PDA che riconosce L(G) è il seguente:



Esercizio 2

- Decidibile, la risposta è chiusa (Si/No). Anzi, in questo caso il problema è deciso, in quanto l'intersezione non è vuota.
- Decidibile: L(G) non è un linguaggio regolare, quindi la risposta sarà sempre No, qualunque sia A.
- Decidibile: se L(G') non è un linguaggio regolare la risposta sarà sempre No; se invece L(G') è regolare, siccome è decidibile il problema di stabilire se FSA riconoscono lo stesso linguaggio, ci sarà sempre una MT che, fissato un linguaggio regolare, determina se questo è uguale a quello riconosciuto da un FSA in input. Ora, fissato L(G'), non è detto che io sappia esattamente quale è questa MT (i linguaggi regolari sono un'infinità enumerabile, e

non è detto che io capisca quale di questi è quello generato da G'), ma essa di certo esiste, rendendo il problema decidibile.

- Indecidibile, in quanto riconducibile al problema di stabilire se una generica MT calcola una certa funzione (in questo caso, quella corrispondente all'FSA), oppure no.

Esercizio 3

Una macchina di Turing (deterministica) che riconosce L(G) può operare nella seguente maniera (al solito indichiamo con n la lunghezza dell'input):

- In primo luogo effettua una scansione completa dell'input per verificare che sia una stringa del tipo $(a^+c)^k$ con $k > 1$. Questa è una condizione necessaria perchè sia in L(G), e può essere verificata in tempo $\Theta(n)$.
- Se il test precedente è passato con successo, l'input appartiene a L(G) se e solo se esistono due sottostringhe (massimali, ossia racchiuse tra due 'c') di sole 'a' della stessa lunghezza. Questa verifica può essere fatta nella seguente maniera: per ogni sottostringa di 'a' incontrata sequenzialmente nell'input:
  - la sottostringa di 'a' viene ricopiata su un nastro di memoria;
  - si prosegue la scansione dell'input confrontando ogni successiva sottostringa di 'a' con quella presente sul nastro di memoria.

  Nel caso pessimo questo comporta una scansione completa dell'input per ogni sottostringa di 'a' incontrata. Possono esservi fino a $\Theta(n)$ sottostringhe di questo tipo, quindi la complessità temporale di caso pessimo è $\Theta(n^2)$, che è anche la complessità complessiva dell'algoritmo. La complessità spaziale è invece chiaramente $\Theta(n)$.

Una macchina RAM può usare un algoritmo simile a quello descritto per la macchina di Turing, usando ad esempio dei contatori per memorizzare lunghezza delle sottostringhe di 'a' incontrate. Si avrebbero dunque $\Theta(n)$ contatori, ciascuno memorizzante un intero $\Theta(n)$. Adottando il criterio di costo logaritmico, i $\Theta(n^2)$ confronti da effettuare porterebbero pertanto a una complessità temporale complessiva di $\Theta(n^2 \log n)$. La complessità spaziale, secondo le stesse considerazioni, sarebbe di $\Theta(n \log n)$.

Informatica Teorica

Appello d'esame dell'8 Settembre 2008, Sezione Mandrioli

Il tempo a disposizione è di 2 ore

Esercizio 1 (Punti 13)

Si considerino le seguenti regole relative al funzionamento di un apparecchio per l'uso del Bancomat:

1. L'apparecchio si trova inizialmente in uno stato di riposo
2. L'introduzione della tessera provoca la transizione in uno stato di immissione codice
3. Il codice consta di 5 cifre
4. L'immissione del codice è ultimata se l'utente ha digitato 5 cifre e successivamente premuto il tasto ENTER, oppure dopo che sono trascorsi 5 secondi dall'immissione della quinta cifra.
5. Una volta ultimata l'immissione del codice l'apparecchio passa in uno stato di verifica.
6. Se, nello stato di immissione, trascorrono più di 5 secondi senza che si digiti una cifra e non è stata ancora digitata la quinta cifra, viene espulsa la tessera e l'apparecchio torna nello stato di riposo.

Si formalizzino mediante formule del prim'ordine solo le regole 4, 5 e 6 (le altre servono da contesto ma non è necessario che vengano formalizzate esplicitamente). Si suggerisce di utilizzare i seguenti predicati che denotano possibili *stati* in cui l'apparecchio si trova in un generico istante t:

- Riposo(t)
- ImissioneDati(t)
- Verifica(t)
- CifreImesse(t, k) indica che all'istante t le cifre immesse sono k; lo stato ha senso solo durante lo stato ImmissioneDati.

e i seguenti predicati che denotano possibili *eventi* che si verificano in un generico istante t:

- PremiCifra(t)
- PremiEnter(t)
- EspulsioneTessera(t)

Esercizio 2 (Punti 11)

Dire se i seguenti problemi sono decidibili, solo semidecidibili, o né uno né l'altro:

1. Stabilire se, data una generica Macchina di Turing M, il linguaggio riconosciuto da M è vuoto.
2. Stabilire se, data una generica Macchina di Turing M che non ha cicli (cioè il cui grafo corrispondente è aciclico), il linguaggio riconosciuto da M è vuoto.
3. Stabilire se, data una generica Macchina di Turing M che non ha autoanelli (cioè il cui grafo corrispondente non ha archi che puntano al nodo da cui originano; un autoanello è un caso particolare di ciclo), il linguaggio riconosciuto da M **non è** vuoto.

Esercizio 3 (Punti 10)

Si consideri la traduzione di una stringa di ingresso $w \in \{a,b,c\}^+$ in una stringa $a^{n/2}b^m$, dove n è il numero di a nella stringa w e m è il numero di b nella stringa w.

1. Si descriva a parole una Macchina di Turing a k nastri che realizza la traduzione indicata, e se ne diano le complessità spaziale e temporale.
2. Si descriva a parole una Macchina RAM che realizza la traduzione indicata, e se ne diano le complessità spaziale e temporale usando il criterio di costo logaritmico.
3. Si descriva a parole una Macchina di Turing a nastro singolo che realizza la traduzione indicata, e se ne diano le complessità spaziale e temporale.

NB: Il punteggio conseguito sarà tanto più alto tanto migliori saranno le complessità delle macchine ideate.

Soluzioni

Esercizio 1

Come già in altre occasioni, per un generico stato S, Up_to_now_S(t) è una notazione abbreviata per la formula seguente

$$\exists\delta(\forall t_1(t-\delta < t_1 < t) \rightarrow S(t_1)) \qquad (\text{sottinteso}: \delta > 0)$$

Similmente From_now_on_S(t) è una notazione abbreviata per la formula seguente:

$$\exists\delta(\forall t_1(t \leq t_1 < t+\delta) \rightarrow S(t_1))$$

Ciò premesso le regole 4 e 6 possono essere formalizzate nel modo seguente, sottintendendo per ognuna la quantificazione universale ∀t, k:

Regole 4 e 5

(

Up_to_now_CifreImmesse(t, 5) ∧ PremiEnter(t)

→

From_now_on_¬ImmissioneDati(t) ∧ From_now_on_Verifica(t)

)

∧

(

Up_to_now_CifreImmesse(t, 5) ∧ PremiCifra(t-5) ∧

(∀t1 (t-5 < t1 < t) → (¬PremiEnter(t) ∧ ¬ PremiCifra(t)))

→

From_now_on_¬ImmissioneDati(t) ∧ From_now_on_Verifica(t)

)

Regola 6

Up_to_now_ImmissioneDati(t) ∧ Up_to_now_CifreImmesse(t, k) ∧ (0 ≤ k <5)

∧

(∀t1 (t-5 < t1 < t) → (¬PremiEnter(t) ∧ ¬ PremiCifra(t)))

→

From_now_on_¬ImmissioneDati(t) ∧ From_now_on_Riposo(t) ∧ EspulsioneTessera(t)

Esercizio 2

1.

Il problema non è né decidibile, né semidecidibile. Non è decidibile per il teorema di Rice (l'insieme delle MT che riconoscono il linguaggio vuoto non è l'insieme vuoto, né è l'insieme universo).

Non è neanche semidecidibile, in quanto è semidecidibile il suo complemento: enumerando opportunamente le stringhe in ingresso, con la solita tecnica diagonale è possibile simulare le esecuzioni della MT in modo che, se una stringa viene accettata, prima o poi questa viene trovata; in caso contrario, la computazione prosegue all'infinito.

2.

Il problema è decidibile (e quindi anche semidecidibile). Il numero di computazioni di una MT senza cicli è finito ed ognuna di esse è fatta di un numero finito di passi. E' quindi possibile enumerarle tutte, e determinare se almeno una di queste si porta in stato finale oppure no.

3.

Il problema non è decidibile, ma è semidecidibile. Data una MT qualunque (anche con autoanelli) è possibile costruirne in modo algoritmico una che sia invece priva di autoanelli (è sufficiente "sdoppiare" gli stati su cui ci sono autoanelli). Di conseguenza, è facile ridurre il problema della *emptiness* del linguaggio di una MT generica (cioè il determinare se il linguaggio accettato è vuoto o no) a quello di una MT senza autoanelli.

Se quindi fosse decidibile il problema in oggetto, lo sarebbe anche quello della *emptiness* di una MT generica (che è in effetti il complemento di quello in oggetto), che non è decidibile per quanto detto al punto 1.

Il problema è invece semidecidibile, in quanto, sempre per quanto detto al punto 1, per una generica MT (e quindi a maggior ragione per una senza autoanelli) è possibile, enumerando le stringhe in ingresso e le varie computazioni con meccanismo diagonale, determinare se questa accetta almeno una stringa (in caso contrario il procedimento algoritmico non termina).

Esercizio 3

1.

E' possibile scrivere una MT che effettua la traduzione desiderata senza fare uso di nastri di memoria, e in tempo $\Theta(n)$.

Tale macchina deve semplicemente scorrere il nastro di input 2 volte: una per produrre in uscita $a^{n/2}$, ed una per produrre in uscita b^m. Più precisamente, nella prima passata, la macchina produce, ogni 2 a incontrate, una a in uscita. Arrivata in fondo al nastro di input, essa lo scorre al contrario, producendo una b per ogni b letta.

2.

Una macchina RAM, non potendo scorrere al contrario il nastro di input, dovrà necessariamente ricorrere alla memoria per effettuare la traduzione desiderata.

Più precisamente, essa si può comportare nella seguente maniera: legge la stringa in ingresso e, per ogni a (risp. b) letta, incrementa un opportuno contatore memorizzato in M[1] (risp. M[2]). Alla fine della lettura divide M[1] per 2, quindi ripete M[1] volte la scrittura in uscita di a, ed M[2] volte la scrittura in uscita di b.

Le complessità (a criterio di costo logaritmico) di una simile macchina sono $\Theta(n \log(n))$ per quella temporale, e $\Theta(\log(n))$ per quella spaziale.

3.

Una Macchina di Turing a nastro singolo può realizzare la traduzione desiderata con un meccanismo simile a quello dell'insertion sort per portare prima tutte le a in testa alla stringa, farle seguire dalle b, e quindi chiudere con le c.

Una volta fatto questo, è sufficiente cancellare le c in coda per ottenere la traduzione desiderata.

La complessità temporale di una simile macchina di Turing è, come per l'algoritmo di insertion sort, $\Theta(n^2)$; quella spaziale è $\Theta(n)$.

Informatica Teorica

Appello d'esame del 3 febbraio 2009, Sezioni Mandrioli-Pradella

Il tempo a disposizione è di 1h30.

Esercizio 1 (punti 8)

Sia data la seguente grammatica G1:

$S \rightarrow AcS \mid bBS \mid bac$

$A \rightarrow ba$

$B \rightarrow ac$

1) Si dica quale è il linguaggio generato da G1.

2) Si scriva un automa che riconosce il linguaggio generato da G1. L'automa deve appartenere alla classe di potenza riconoscitiva minima tra quelle che riconoscono il linguaggio generato da G1.

Esercizio 2 (punti 12)

Si dica, giustificando brevemente la risposta, se è decidibile l'equivalenza tra:

1. Un automa a stati finiti e una macchina di Turing.
2. Una grammatica non contestuale e una macchina di Turing.
3. Una macchina di Turing e la grammatica G1 dell'esercizio 1.
4. Due automi a stati finiti.
5. Un automa a stati finiti e un automa a pila determinitico.

(**Suggerimento**: si tenga presente che l'intersezione tra il linguaggio riconosciuto da un automa a stati finiti e quello riconosciuto da un automa a pila deterministico è riconoscibile da un automa a pila deterministico costruibile algoritmicamente sulla base dei due automi dati.)

Esercizio 3 (punti 11)

Si descriva a grandi linee il funzionamento di una macchina di Turing deterministica che riconosce il linguaggio $L = \{w^4 \mid w \in \{a, b, c\}+\}$ (laddove $w^4 = wwww$).

Si valutino le complessità spaziale e temporale della macchina di Turing descritta.

La macchina di Turing deve essere tale da minimizzare la complessità spaziale.

Soluzioni schematiche

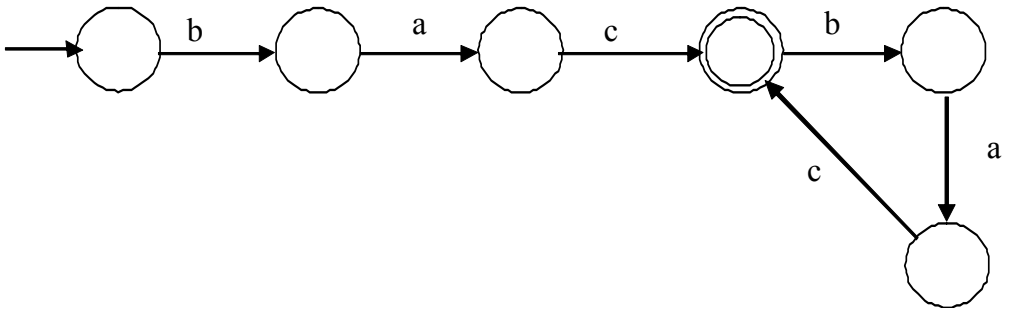
Esercizio 1

La grammatica genera il linguaggio:

$$L(G1) = \{ (bac)^+ \}$$

L(G1) è un linguaggio regolare, cioè riconoscibile da un automa a stati finiti.

Un automa a stati finiti che riconosce il linguaggio è il seguente:



Esercizio 2

Le prime 3 domande hanno risposta negativa sulla base di normali ragionamenti in applicazione al teorema di Rice.

La domanda 4 ha risposta positiva, grazie alla chiusura del formalismo rispetto alle operazioni booleane: $L1 \equiv L2 \Leftrightarrow (L1 \cap \neg L2) = \varnothing \wedge (\neg L1 \cap L2) = \varnothing$.

Alla stessa maniera, sfruttando l'informazione fornita nel suggerimento e il fatto che sia i linguaggi regolari che quelli noncontestuali deterministici sono chiusi rispetto al complemento, si dimostra decidibile anche il quesito 5.

Esercizio 3

Una macchina di Turing (deterministica) a 2 nastri che riconosce il linguaggio desiderato può operare nella seguente maniera (al solito indichiamo con n la lunghezza dell'input):

1- conta gli elementi dell'input in binario (usando il nastro T1)
2- se il numero su T1 è divisibile per 4 (deve terminare con 2 zeri), allora elimina gli ultimi 2 zeri per dividere per 4
3- torna all'inizio dell'input
4- copia il contenuto di T1 nel nastro T2, tenendo traccia nello stato del simbolo letto
5- si sposta sul nastro di input decrementando T2 ad ogni mossa
6- quando T2 è a zero, confronta il simbolo sotto la testina con quello memorizzato, se sono diversi, si blocca
7- si sposta di una cella in avanti sul nastro di input; se è in fondo al nastro, stop con successo

8- torna indietro di T1 elementi sul nastro (usando ancora T2 come contatore)
9- ripete dal passo 4

La complessità spaziale della macchina è log(n).

La complessità temporale, invece, è $n^2 log(n)$.