Algoritmi e principi dell'Informatica

Seconda prova in itinere – 24 Gennaio 2012, Sezione Pradella

Avvisi importanti

I punteggi attribuiti ai singoli esercizi hanno valore solo per chi sostiene la II prova completa del corso integrato.

Per chi deve sostenere solo il modulo di Informatica 3 o l'intero appello (riservato ai laureandi) il punteggio sarà diverso e valutato caso per caso.

Chi deve sostenere solo l'esame di Informatica 3 deve risolvere gli esercizi 2, 3, 4 in 2 ore.

Chi deve sostenere la II prova del corso integrato deve risolvere gli esercizi 1, 2, 3, 4 in 2 ore e 15 minuti.

Chi deve sostenere l'intero appello deve risolvere tutti gli esercizi in 3 ore e 30 minuti.

Su ogni foglio consegnato devono essere indicati chiaramente:

Cognome, nome, numero di matricola

Quale dei tre tipi di prova di cui sopra si sta sostenendo

Esercizio 1 (punti 4/18-esimi)
Si dica, giustificando brevemente ma precisamente la risposta, se la seguente funzione f è calcolabile:
Per i compreso tra 1 e 1000, j compreso tra 1 e 10000, f(i, j) = 1 se la i-esima MT si ferma computando il dato j, 0 altrimenti.

Esercizio 2 (punti 6/18-esimi)

1. Si descrivano, mediante opportuno pseudocodice, due algoritmi che simulino il comportamento di due rispettivi automi a pila che riconoscano i seguenti linguaggi:
   a. $L1 = \{ a^n b^n \} \cup \{ a^n b^{2n} \} | n \geq 1$
   b. $L2 = \{ww^R \mid w \in \{a,b\}^*\}$ (w^R indica, al solito, la stringa speculare di w)

   NB1: gli algoritmi non devono semplicemente decidere se una stringa appartiene al linguaggio dato, ma *simulare completamente* il comportamento dell'automa, ossia ripercorrere –senza necessariamente fornirle in output- le stesse computazioni che eseguirebbe l'automa.

   NB2: si può assumere che l'input dell'algoritmo sia una stringa di caratteri seguita da un terminatore di "fine stringa".

   NB3: la scelta dello pseudocodice è lasciata allo studente: può essere un linguaggio di tipo "mini-Pascal" o "mini-C" o in stile RAM; o anche in "linguaggio naturale strutturato". E' però importante che la descrizione degli algoritmi sia sufficientemente precisa da non lasciare dubbi sulla semantica delle operazioni elementari, sul loro costo e sul fatto che esse corrispondano alle operazioni astratte dei corrispondenti automi a pila.
2. Si valuti la complessità temporale di entrambi gli algoritmi nel caso pessimo, adottando il criterio di costo ritenuto più opportuno (e spiegandone brevemente la scelta)
3. E' possibile costruire MT deterministiche che riconoscano gli stessi linguaggi con complessità asintotica migliore di quella dei due rispettivi algoritmi? In caso positivo, quali complessità sono ottenibili? Spiegare brevemente le risposte fornite.

Esercizio 3 (punti 2/18-esimi)

Si consideri una tabella di hash con 18 posizioni. Si vuole gestirla usando il double hashing (h1(n)+i*h2(n)) mod 18 (con n numero da inserire e i numero del tentativo).

Dire quale tra queste coppie di funzioni si adotterebbe motivando brevemente la risposta

1) h1=n mod 9 e h2=n mod 3
2) h1=n mod 13 e h2=n mod 7

Esercizio 4 (punti 7/18-esimi)

Si ha un array di n liste; ognuna delle quali contenente al più m elementi che descrivono gli stipendi di una società. Ogni lista contiene i dati di uno dei reparti della società. Supponendo che ognuna delle liste contenga gli elementi in ordine crescente di stipendio, si progetti un algoritmo capace di generare un'unica lista ordinata per stipendio di tutti i dipendenti della società cercando di ottenere la miglior complessità temporale possibile e se ne fornisca una valutazione.
NB: si noti che n e m sono due parametri indipendenti che determinano la dimensione del problema.

Esercizio per il recupero della prima prova

1. Si progetti un automa A_1 che riconosca il linguaggio

   $L_1 = \{ a^n b^n \mid n \geq 3 \}$

   L'automa deve appartenere ad una classe C (tra FSA, PDA, TM) di automi a potenza minima tra quelli che riconoscono L_1.

2. Si progetti quindi a partire dall'automa A_1, un automa A_2 a potenza minima che riconosca il linguaggio $L_2 = L_1^+$.

3. La classe C è chiusa rispetto all'operazione "+" di chiusura transitiva ("più di Kleene")?

4. In caso affermativo, descrivere sinteticamente il procedimento che, dato un automa A appartenente alla classe C, consente di sintetizzare l'automa che riconosce il linguaggio $L(A)^+$. In caso negativo, dire come si può determinare se nessun automa della classe C può riconoscere $L(A)^+$.

5. Si dia la specifica logica del linguaggio L_2.

6. **(Parte opzionale, da svolgere solo dopo aver completato le precedenti)**

   Costruire una rete di Petri che accetti L_1.

Soluzioni schematiche

Esercizio 1

La funzione proposta ha un dominio finito (o, se la si volesse definire sull'intero N X N , sarebbe decidibile stabilire se una coppia <i,j> appartiene al suo dominio di definizione.).

Quindi essa è esprimibile come una matrice 1000 X 10000 in ogni elemento della quale si trova un 1 o uno 0. Esiste un numero finito di tali matrici; ognuna delle quali può essere calcolata da un'opportuna MT; quindi la funzione data è sicuramente calcolabile (anche se il ragionamento suddetto non porta a costruire la MT che la calcola.)

Esercizio 2

Entrambi i linguaggi sono nondeterminisitici.

a.

Nel primo caso l'automa riconoscitore compie una sola scelta nondeterministica (come prima mossa) e successivamente si comporta in modo deterministico.

Quindi un algoritmo può simularne il comportamento simulando inizialmente il riconoscimento mediante pila di { $a^n b^n$ } (o viceversa); se questo primo tentativo fallisce – assumendo che la stringa di ingresso sia stata memorizzata preliminarmente- passa alla simulazione dell'automa che riconosce { $a^n b^{2n}$ }.

Ovviamente in tal caso la complessità temporale dell'algoritmo (a costo costante) è $\Theta(n)$ (e $\Theta(n.\log(n))$ a costo logaritimico).

b.

Nel caso di L2 invece l'automa a pila "naturale" che lo riconosce decide nondeterministicamente ad ogni passo se impilare il simbolo letto o iniziare a confrontarlo con quello in cima alla pila; ovviamente, una volta deciso di iniziare il confronto il comportmento diventa deterministico e si conclude con successo solo se la pila viene completamente svuotata esattamente in corrispondenza della lettura dell'ultimo carattere.

Un algoritmo che simuli tale comportamento può consistere di due cicli: il primo scandisce e mette in pila ogni simbolo letto; un secondo ciclo, interno al primo, per ogni simbolo "impilato":

- Copia l'attuale contenuto della pila
- Riprende la lettura dal simbolo successivo a quello appena impilato e procede verificando deterministicamente l'eguaglianza tra ogni simbolo letto e quello in pila (ovviamente eliminandolo di volta in volta dalla pila).
- Se la pila viene svuotata esattamente in corrispondenza della lettura dell'ultimo carattere (all'uopo un normale algoritmo può usufruire di un carattere apposito (ETX)) la stringa viene accettata e l'esecuzione interrotta.
- Altrimenti si esce dal ciclo interno e si riprende l'esecuzione dall'ultimo carattere letto nell'esecuzione del ciclo esterno.

Per fare ciò può essere necessario –se l'unità di I/O non lo permette- aver inizialmente memorizzato l'intera stringa.

La complessità di un tale algoritmo è quadratica (a criterio di costo costante) perché nel caso pessimo il ciclo esterno viene eseguito n volte e quello interno i volte (più la copiatura della stringa impilata fino all'entrata, a sua volta lunga i) per l'i-esimo carattere letto nel ciclo esterno.

Anche in questo caso, l'uso del criterio di costo logaritmico non fa che aggiungere un fattore log(i) a ogni memorizzazione o eliminazione di un carattere dalla pila, producendo quindi una complessità $\Theta(n^2.\log(n))$

3.

Essendo la complessità asintotica del primo algoritmo lineare essa non è ovviamente migliorabile.

Al contrario un semplice macchina di Turing che riconosca L2 in tempo lineare può operare nel modo seguente:

- Memorizza la stringa di ingresso su due nastri;
- Pone le due testine, rispettivamente all'inizio e alla fine della stringa;
- Verifica che i caratteri letti dalle due testine coincidano, spostandole successivamente a destra e sinistra, rispettivamente;
- NB: ai fini della complessità asintotica non è necessario –ma è possibile per migliorare la costante di proporzionalità- verificare che le testine giungano a metà stringa.

Esercizio 3

Nella scelta 1) sia h1 che h2 sono divisori di 18, quindi favoriscono una "clusterizzazione" dei dati con rischio di rapida saturazione di alcune posizioni.

E' quindi preferibile la seconda coppia di funzioni proposta.

Esercizio 4

Un algoritmo "banale" potrebbe costruire incrementalmente l'unica lista globale "fondendo" (operazione di merge) le varie liste ad una ad una. Il merge di due liste lunghe al più h richiede un tempo O(h); quindi la complessità totale di un tale algoritmo sarebbe

$$O(\sum_{i=1}^{n}(i.m+m)) = O(n^2.m)$$

Un algoritmo un po' meno banale, invece, potrebbe trarre ispirazione dal merge-sort e fondere le liste a due a due in diverse passate:

La prima passata produrrebbe quindi n/2 liste di lunghezza (al più) 2.m ciascuna, in tempo O(n.m); la seconda n/4 liste di lunghezza 4.m, sempre in tempo O(n.m), ecc.

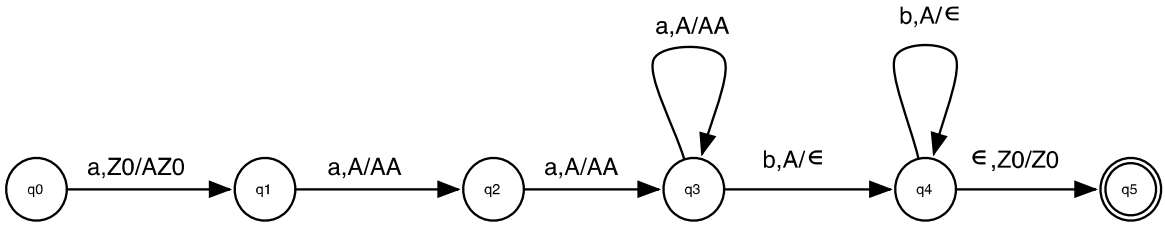
Il numero di passate, risulterebbe, al solito O(log(n)) per una complessità totale O(n.m.log(n)).

Esercizio per il recupero della prima prova

Quesito 1

Per riconoscere il linguaggio L_1 occorre la capacità di "contare" numeri arbitrariamente grandi. Il formalismo degli automi a stati finiti non è pertanto sufficiente (come si può dimostrare, ad esempio, tramite il *pumping lemma*).

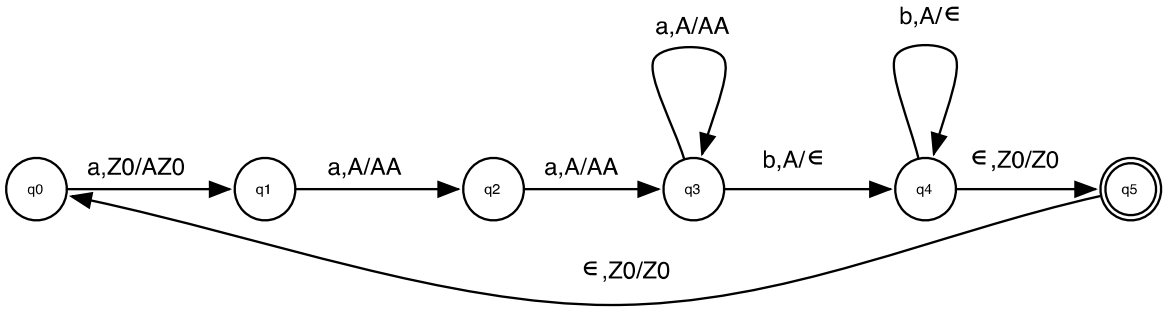
Un semplice automa a pila che riconosce L_1 è il seguente.



La classe degli automi a pila (deterministici) è pertanto a potenza minima, per il problema dato, tra quelle specificate nel testo.

Quesito 2

Mostriamo ora un automa a pila che riconosce L_2.



Quesiti 3 e 4

La classe dei linguaggi riconosciuti da automi a pila deterministici *non* è chiusa rispetto all'operazione "+": si consideri ad esempio il linguaggio $\{ a^n b^n \ c \ a^m b^{2m} \mid n, m \geq 0\}$
Lo è invece quella riconosciuta da automi a pila *non*deterministici. Il procedimento consiste nel "cortocircuitare", mediante una mossa spontanea che non alteri il contenuto della pila, gli stati finali con lo stato iniziale.

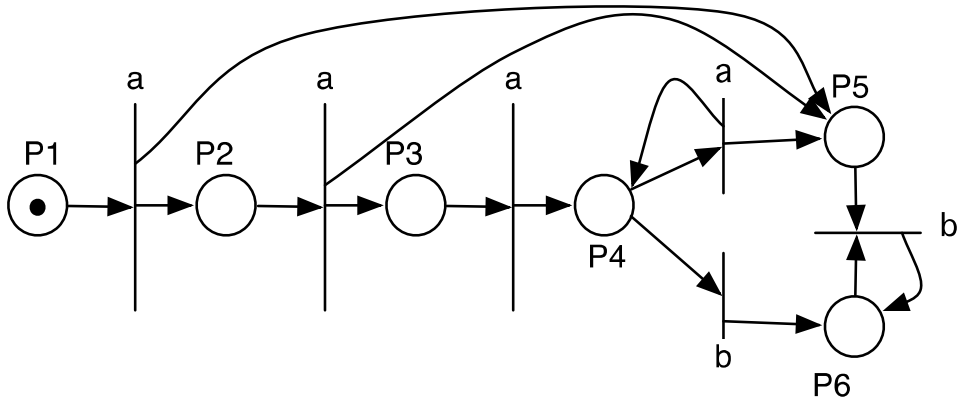
Quesito 5

Specifica di L_1: $\forall x(x \in L_1 \leftrightarrow \exists n(n \geq 3 \wedge x = a^n.b^n))$
Specifica di L_2: $\forall x(x \in L_2 \leftrightarrow x \in L_1 \vee \exists y \exists z(y \in L_1 \wedge z \in L_2 \wedge x = y.z))$

Soluzione alternativa (senza ricorrere alla specifica intermedia di L_1):
$\forall x(x \in L_2 \leftrightarrow \exists n(n \geq 3 \wedge x = a^n.b^n) \vee \exists z \exists n(z \in L_2 \wedge n \geq 3 \wedge x = a^n.b^n.z))$

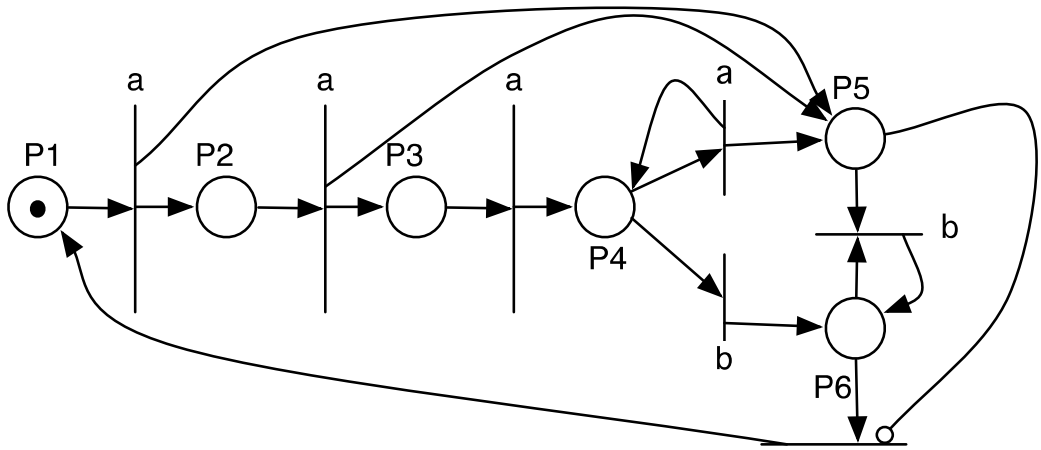
Quesito 6

Un esempio di rete di Petri che riconosce il linguaggio L_1 è il seguente, in cui la marcatura iniziale è mostrata in figura e l'insieme di marcature finali è $F=\{M_F\}$ con $M_F(P_6)=1$ e $M_F(P_i)=0$ per $i=1,\ldots,5$.



Si noti che la classe delle reti di Petri è a potenza *minimale* – non *minima* – rispetto alle precedenti perché le due classi PDA e PN sono tra loro non confrontabili.

A titolo di approfondimento si osservi che non si può costruire una rete di Petri che riconosca L_2. L'unica possibilità è di utilizzare archi inibitori, raggiungendo pertanto il potere espressivo delle MT. Una rete di Petri con archi inibitori che riconosce L_2 è mostrata di seguito.



Più in generale, è stato dimostrato che la classe dei linguaggi riconosciuti da rete di Petri non è chiusa rispetto all'operazione "+", se non facendo ricorso agli archi inibitori. La dimostrazione formale di questo fatto è complessa. Dal punto di vista intuitivo, con riferimento all'esempio in questione, oltre a spostare il gettone da P_6 a P_1, occorre anche sincerarsi che il posto P_5 sia vuoto, il che richiede un arco inibitore.

Algoritmi e principi dell'Informatica

Seconda prova in itinere – 24 Gennaio 2012, Sezione Mandrioli

English version

Delivery time: 2 hours and 15 minutes after start.

Exercise 1 (max score: 4/18 points)

Consider the function f defined as follows:

for $1 \leq i \leq 1000$, $1 \leq j \leq 10000$,

f(i, j) = **if** the i-th Turing machine halts when computing on the input j **then** 1

else 0

Is f computable? Give a short but precise explanation for your answer.

Exercise 2 (max score: 6/18 points)

Consider the following languages:

a. $L1 = \{ a^n b^n \} \cup \{ a^n b^{2n} \} | n \geq 1$

b. $L2 = \{ww^R \mid w \in \{a,b\}^*$ (as usual, w^R denotes the mirror image of w)

1. For each one of L1 and L2, provide an algorithm that simulates the behavior of a pushdown automaton recognizing L. The algorithm should *not* –only- accept a string iff it is accepted by the corresponding automaton, but should execute a sequence of steps corresponding to the single automaton transitions; in particular it should use its memory as a stack and manage it just as the formal automaton would do, by means of suitable push and pop –abstract- operations.

   You are free to describe the language in any pseudo-code you feel appropriate: could be a "mini-C" or a RAM, or even "structured natural language", provided your description is precise enough.

2. Evaluate the asymptotic complexity of both algorithms by applying the cost criterion you feel more appropriate; briefly explain your answer.

3. Is it possible to build deterministic Turing machines that accept the same languages with a better time complexity? In the positive case, how? Which complexities can you obtain? Briefly explain your answer.

Exercise 3 (max score: 2/18 points)

Consider a hash table with 18 placeholders. We want to manage it in a double-hashing fashion: h1(n)+i*h2(n) (where n denotes the value of the key to be inserted and i denotes the iteration of the attempt to insert the element.

Which one between the following pairs of function would you choose? Why?
1) h1=n mod 9 e h2=n mod 3
2) h1=n mod 13 e h2=n mod 7

Exercise 4 (max score: 7/18 points)

Assume you have an array consisting of *n* linked lists. Each list contains at most *m* elements (say, describing the salaries of the employees of a section of a firm). Assume also that each list is ordered in increasing order. Design an algorithm that, starting from the above data structure produces a single ordered list containing –all and only- the elements of the *n* original lists; try to optimize the time complexity of your algorithm and evaluate it (with respect to the order of magnitude Θ).

Supplement exercise for those who must repeat the first midterm test

1. Build an automaton A_1 that recognizes the following language

   $L_1 = \{ a^n b^n \mid n \geq 3 \}$

   A_1 must belong to a class C (among FSA, PDA, TM) of automata with minimal power among those that recognize L_1.

2. Starting from A_1, build a new automaton A_2 (with minimal power) that recognizes the language $L_2 = L_1^+$.

3. Is class C closed under transitive closure (a.k.a. "+" or "Kleene plus")?

4. If it is, you should sketchily describe the procedure allowing you to build the automaton that recognizes the language $L(A)^+$, given an automaton A belonging to class C. Otherwise you should explain how you can determine whether no automaton of class C can recognize $L(A)^+$.

5. Specify language L_2 in first-order logic.

6. **Optional part, to be executed only after having completed the previous parts and the other exercises.**
   Build a Petri net that recognizes L_1.