Informatica Teorica

Prima prova in itinere - 12 Maggio 2006

Si svolgano i seguenti esercizi. Il tempo complessivo a disposizione è di 1h30.

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.

- 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.
- Si **confronti** poi la potenza riconoscitiva degli APV con quella degli automi a stati finiti.
- **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 $\Sigma = \{a,b,c,p,z\}$, 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 $\Sigma = \{(,),*\}$; 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.

Soluzion1

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

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

con $q' \in \delta(q, a, \gamma)$ solo se $a \in I_{pop}$ oppure se $a \in I_{local}$, e $<q', \beta> \in \delta(q, a, \gamma)$ solo se $a \in I_{push}$.

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.

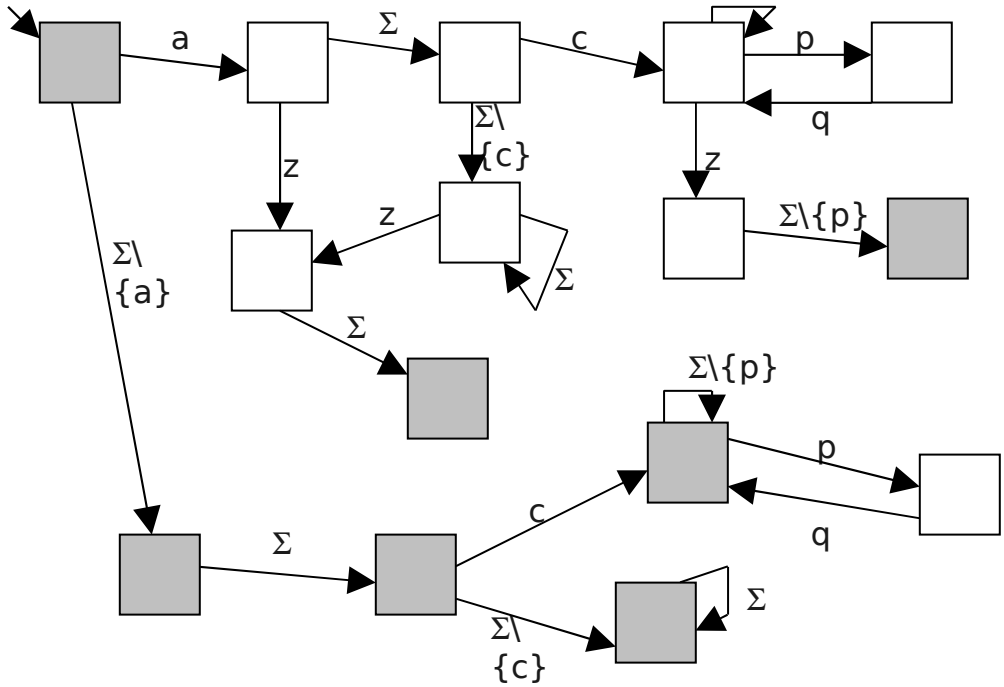
Il confronto con gli automi a pila standard è più complesso. Consideriamo il linguaggio $\{wcw^R \mid w \in \{a,b\}^*\}$, riconosciuto da un automa a pila deterministico standard (e dunque a maggior ragione da uno nondeterministico), mediante la seguente tecnica: si impilano i caratteri letti finchè 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).



Esercizio 3

La seguente grammatica noncontestuale genera L1.

S → B | Q

B → BB | (B) | ε

Q → A*A

A → *A | (A |)A | ε

Informatica Teorica

Seconda prova in itinere - 5 Luglio 2006

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.

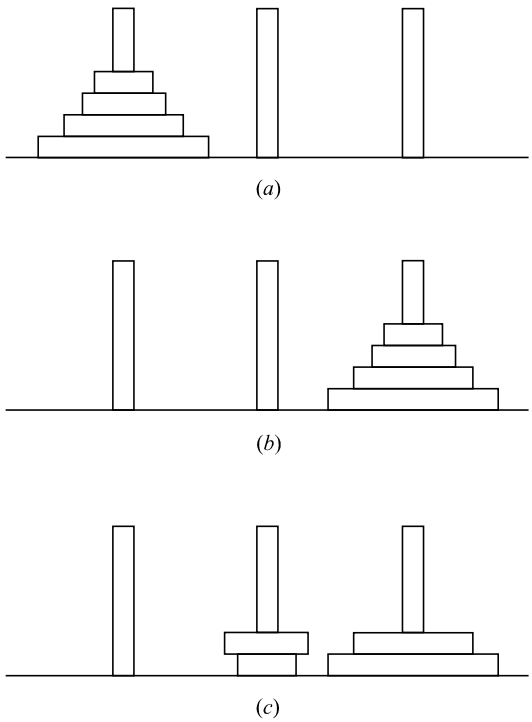


Figure (a), (b), (c) Il problema della torre di Hanoi: (*a*) situazione iniziale, (*b*) situazione finale richiesta, (*c*) situazione inammissibile.

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 (ma accuratamente) 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 3.bis (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. La durata della prova è di 1 ora e 30 minuti.

2. L'eventuale punteggio derivante dalla soluzione dell'Esercizio 1.bis sarà computato, ai fini della valutazione finale, solo se tutti gli altri esercizi saranno stati risolti in maniera ritenute soddisfacente.

3. I risultati complessivi delle prove in itinere, con eventuale voto positivo proposto e modalità di accettazione/rifiuto del voto proposto saranno pubblicati mediante il servizio online di verbalizzazione degli esami (per cui gli studenti riceveranno le necessarie comunicazioni tramite il loro indirizzo di posta "istituzionale", cioè quello @stud.polimi.it).

Soluzioni

Esercizio 1

Si indichi

- con D_i l'i-esimo disco: $1 <= i <= n$;
- con $dim(D_i)$ la dimensione del disco i-esimo;
  $(\forall i\ ((1 <= i <= n) \rightarrow (1 <= dim(D_i) <= n))) \quad \wedge$
  $(\forall i,j\ (((1 <= i, j <= n) \wedge (i \neq j)) \rightarrow (dim(D_i) \neq dim(D_j))))$
- 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.
- la funzione pos deve sottostare ai seguenti vincoli:
  - $\forall i,j\ (pos(D_i) = pos(D_j) \rightarrow D_i = D_j)$

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

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

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)

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

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

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

Soluzione alternativa per l'esercizio 1.Bis

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 */
)

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.

Soluzioni alternative per gli esercizi 1/1.Bis

Gli esercizi 1 ed 1.Bis possono essere svolti in molti modi diversi. Qui ne vengono proposti 2 che descrivono in maniera completa non solo i vincoli sul posizionamento dei dischi nei pioli e sul loro movimento, ma anche l'obiettivo del gioco.

Prima soluzione alternativa per gli esercizi 1/1.Bis

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

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*
- $\text{muovi}(p_1, p_2, t)$ 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":
  $\text{altezza}(p, t) = h \leftrightarrow \quad (h > 0 \wedge \exists d(\text{piolo}(p, h, t) = d) \wedge \neg\exists d(\text{piolo}(p, h+1, t) = d)) \vee$
  $(h = 0 \wedge \neg\exists d(\text{piolo}(p, 1, t) = d)))$
- 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:
  $\text{muovi}(p_1, p_2, t) \rightarrow \text{altezza}(p_1, t) > 0 \wedge p_1 \neq p_2$
- non è possibile effettuare 2 mosse contemporaneamente:
  $\text{muovi}(p_1, p_2, t) \wedge \text{muovi}(p_3, p_4, t) \rightarrow p_1 = p_3 \wedge p_2 = p_4$
- effetto di una mossa sullo stato di un piolo:
  $\text{piolo}(p, h, t) = d \leftrightarrow \quad (p = 1 \wedge h = n - d + 1 \wedge \neg\exists t_1, p_1, p_2(t_1 < t \wedge \text{muovi}(p_1, p_2, t_1)))$
  $\vee$
  $\exists t_1 (t_1 < t \wedge \text{altezza}(p, t_1) = h-1 \wedge$
  $\exists p_1(\text{muovi}(p_1, p, t_1) \wedge$
  $\text{piolo}(p_1, \text{altezza}(p_1, t_1), t_1) = d \wedge$
  $\forall t_2 (t1 < t2 < t \wedge \text{altezza}(p, t_2) = h$
  $\rightarrow$
  $\neg\exists p_1 (\text{muovi}(p, p_1, t_2)))))$
- stato iniziale del gioco:
  $\forall i(1 \leq i \leq n \rightarrow \text{piolo}(1, i, 0) = n - i + 1) \wedge \text{altezza}(2, 0) = 0 \wedge \text{altezza}(3, 0) = 0$
  - 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:
  $\exists t (\forall i(1 \leq i \leq n \rightarrow \text{piolo}(3, i, t) = n - i + 1) \wedge \text{altezza}(1, t) = 0 \wedge \text{altezza}(2, t) = 0)$

Seconda soluzione alternativa per gli esercizi 1/1.Bis

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:

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

Informatica Teorica

Appello del 19 Luglio 2006

Esercizio 1 (Punti 10)

Si consideri il linguaggio seguente:

L = {wcW | w, W ∈ {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 sequenza 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:

- se non si è ancora letto il separatore 'c', si copia il carattere letto in coda alla sequenza e si impila lo stesso carattere sulla pila
- se si legge il carattere 'c', si procede al carattere successivo
- se si è già letto il separatore 'c', allora
    - 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
    - altrimenti, si incrementa il contatore degli errori e
        - se il contatore errori è maggiore di 3, si scrive 'cc' in output e si termina,
        - altrimenti, si procede con la lettura del carattere successivo
- se la stringa è stata letta completamente dall'input e la pila è vuota, si utilizza la sequenza creata 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 sequenza 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 sequenza 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 Cremona+Como

Appello del 13 Settembre 2006

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 + 5 per la parte facoltativa)

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

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.

$$\forall t$$
$$(t < 0 \rightarrow s(t)) \wedge (0 \bmod 9 \leq t < 4 \bmod 9 \rightarrow v(t)) \wedge$$
$$(4 \bmod 9 \leq t < 5 \bmod 9 \rightarrow g(t)) \wedge (5 \bmod 9 \leq t < 9 \bmod 9 \rightarrow v(t))$$
$$\wedge$$
$$((attr(t) \wedge 0 \bmod 9 \leq t < 4 \bmod 9) \rightarrow (\neg multa50(t) \wedge \neg multa200(t))$$
$$\wedge$$
$$((attr(t) \wedge 4 \bmod 9 \leq t < 5 \bmod 9) \rightarrow (\neg multa50(t) \wedge \neg multa200(t))$$
$$\wedge$$
$$((attr(t) \wedge 5 \bmod 9 \leq t < 6 \bmod 9) \rightarrow (multa50(t) \wedge \neg multa200(t))$$
$$\wedge$$
$$((attr(t) \wedge 6 \bmod 9 \leq t < 9 \bmod 9) \rightarrow (\neg multa50(t) \wedge multa200(t))$$

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 $\Theta(n \cdot \log(n))$. 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 $\Theta(n \cdot \log^2(n))$.

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 $\Theta(n \cdot \log(n))$ 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

Sezione Cremona+Como

Appello dell'8 Febbraio 2007

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

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:

- la stringa inizia e termina con *esattamente* lo stesso numero di 'a'
- 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: Scegliere *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.

$S \rightarrow aSa \mid BXB \mid B$

$X \rightarrow aX \mid bX \mid cX \mid \varepsilon$

$B \rightarrow b \mid c$

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.

8.1 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;
```

8.2 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

- 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.
- 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.
- 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.