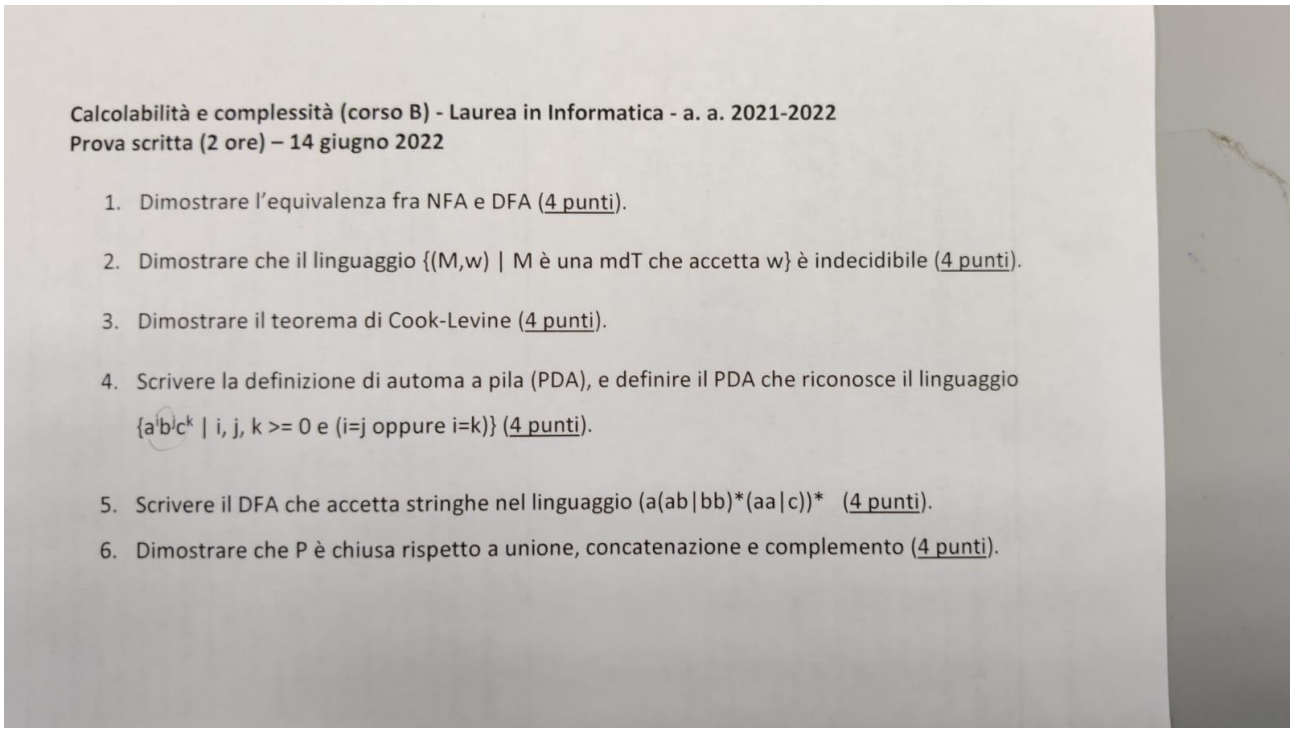


Calcolabilità e complessità (corso B) - Laurea in Informatica - a. a. 2021-2022
Prova scritta (2 ore) – 14 giugno 2022

1. Dimostrare l'equivalenza fra NFA e DFA (4 punti).
2. Dimostrare che il linguaggio {(M,w) | M è una mdT che accetta w} è indecidibile (4 punti).
3. Dimostrare il teorema di Cook-Levine (4 punti).
4. Scrivere la definizione di automa a pila (PDA), e definire il PDA che riconosce il linguaggio $\{a^i b^j c^k \mid i, j, k >= 0$ e (i=j oppure i=k)$\}$ (4 punti).
5. Scrivere il DFA che accetta stringhe nel linguaggio (a(ab|bb)*(aa|c))* (4 punti).
6. Dimostrare che P è chiusa rispetto a unione, concatenazione e complemento (4 punti).

1) Due automi sono equivalenti quando riconoscono lo stesso linguaggio.

Devo dimostrare che gli NFA contengono i DFA e che i DFA contengono gli NFA. Per quanto riguarda la prima implicazione, ovvero che gli NFA contengono i DFA, questa è sempre verificata, in quanto, ogni DFA può essere visto come un NFA e quindi accetta lo stesso linguaggio. Per quanto riguarda la seconda implicazione, ovvero che i DFA contengono gli NFA, devo attuare un procedimento di conversione che a partire da un NFA giungo a un DFA. Sia dato un automa NFA M = (Σ, Q, δ, q0, F), allora l'automa DFA M' = (Σ, Q', δ', q0', F') equivalente a M si costruisce nel seguente modo:

- Σ è l'alfabeto di input;
- Q' = 2^(Q);
- q0' = {q0};
- F' = {p contenuto in Q | p ∩ F ≠ Ø);
- δ': Q' × Σ -> Q', dove δ'(q', x) = δ'({q1, q2, …, qi}, x) = $\bigcup_{i=1}^{j} \delta'(qj, x)$ = $\bigcup_{q \in q'} \delta(q, x)$.

Pertanto, una volta attuato questo algoritmo di conversione ho dimostrato che il DFA risultante accetta lo stesso linguaggio accettato dall'NFA iniziale. Quindi ho dimostrato anche la seconda implicazione, pertanto ne consegue che i DFA e gli NFA riconoscono gli stessi linguaggi.

2) Un linguaggio è indecidibile se non esiste un processo di decisione per esso e quindi non esiste una mdT che accetta il linguaggio e prende una decisione per ogni stringa di input.

Il linguaggio corrispondente al membership problem è A_{TM}, dove:
$A_{TM} = \{<M, w> \mid M \text{ è una mdT che accetta } w\}$.

Quindi A_{TM} prende in input le coppie dove M è una mdT che accetta w, inoltre M può essere scritta come codice.

Suppongo per assurdo che A_{TM} è decidibile, quindi dato che è decidibile esiste una macchina H che data la coppia *<M, w>* decide se M accetta w; quindi, praticamente questa macchina prende il codice di M e la stringa w e decide se M accetta w.

Adesso definisco una macchina Diag che effettua l'esatto opposto di H, quindi se H accetta, Diag rigetta e se H rigetta, Diag accetta.

Diag, come H, ha due input, ovvero, il codice di M e la stringa w. Ora semplifico Diag e al posto della stringa w gli passo il codice di M; quindi Diag ha come input M, copia il codice di M e lo passa al decisore H e dunque possono accadere due cose: M accetta M o M rigetta M. Quindi risulta che:

- Diag accetta M se M rigetta M;
- Diag rigetta M se M accetta M;

Adesso al posto di M metto Diag perché è una mdT:

- Diag accetta Diag se Diag rigetta Diag;
- Diag rigetta Diag se Diag accetta Diag;

Siamo davanti ad un assurdo. L'assurdo deriva dall'aver supposto che A_{TM} fosse decidibile.

3) Il teorema di Cook-Levine dice che SAT è un linguaggio NP-completo.

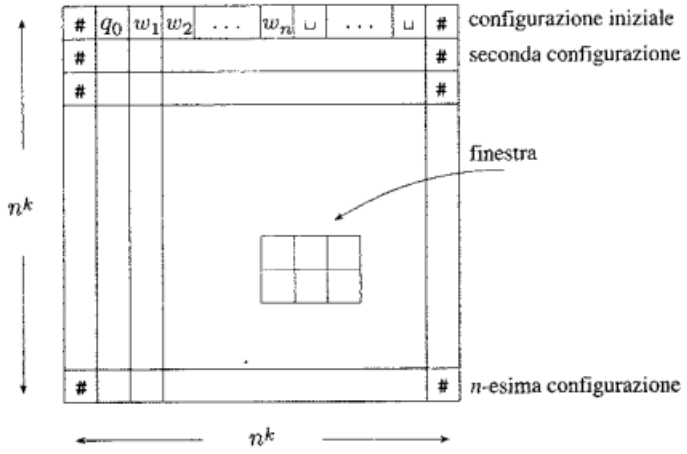
Per dimostrarlo deve risultare che:

- SAT è in NP;
- Devo ridurre tutti i linguaggi al problema SAT in tempo polinomiale.

Allora:

- SAT è in NP sia con il verificatore che in maniera non deterministica; infatti, con il verificatore la stringa c rappresentava i valori di verità che rendevano la formula vera;

- Sia dato un linguaggio L $\in$ NP.
  Devo definire una riduzione polinomiale a SAT.
  Dato che L $\in$ NP, allora esiste una mdT non deterministica che decide L in tempo polinomiale. Per ogni stringa w costruisco in tempo polinomiale un'espressione booleana tale che se w $\in$ L allora l'espressione è vera, quindi: ϕ(M, w) è soddisfacibile se w $\in$ L e non soddisfacibile se w $\notin$ L.
  Adesso vediamo come sono le computazioni della macchina non deterministica M.
  Dato che M non è deterministica ci saranno molte computazioni e dato che L $\in$ NP, allora l'albero delle computazioni avrà profondità al massimo n^k, dove n è la lunghezza della stringa in input.
  Adesso considero una computazione accettante che ovviamente sarà al massimo n^k e l'input occuperà al massimo n^k celle di memoria a sinistra; quindi, la massima area di calcolo sul nastro sarà $2n^k$.
  Adesso l'albero delle computazioni si può vedere sottoforma di una matrice chiamata tableaux delle configurazioni.



  Quindi l'alfabeto del tableau sarà: C = {{#} $\cup$ {insieme degli stati} $\cup$ {alfabeto del nastro}}.
  Per ogni cella con posizione i, j e per ogni simbolo dell'alfabeto s $\in$ C, creo la variabile $x_{i, j, s}$ in cui se nella posizione i e j vi è il simbolo s allora $x_{i, j, s}$ = 1, altrimenti è uguale a 0.
  Quindi ϕ(M, w) è costruita mediante la variabile $x_{i, j, s}$.
  $\phi(M, w) = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$

  ϕ_{cell} indica che ogni cella del tableau delle configurazioni contiene uno e un solo simbolo. Pertanto, per tutte le coppie di i e j esiste almeno un simbolo in ogni cella AND per tutti gli s e t, simboli dell'alfabeto, con s $\neq$ t deve succedere

che o è presente s o è presente t, quindi in ogni cella vi è solo un simbolo e vi è la certezza che questo simbolo c'è. Ha complessità $O(n^{2k})$.
Per quanto riguarda la dimensione si ha $2n^k + 3$ perché la massima area di calcolo è pari a $2n^k$, poi vi sono due #, uno all'inizio della riga e uno alla fine della riga e in più vi è lo stato, quindi $2n^k + 3$.

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right].$$

ϕ_{start} indica come è formata la prima riga del tableau, quindi come inizia la computazione. La ϕ_{start} ha dimensione $2n^k + 3$, in quanto lo spazio di ogni singola riga è pari a $2n^k + 3$ e, quindi, ha complessità pari a $O(n^k)$.

$$\begin{aligned} \phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}. \end{aligned}$$

ϕ_{accept} dà certezza che la computazione raggiunge uno stato di accettazione. La ϕ_{accept} ha dimensione $(2n^k + 3)^2$, quindi ha complessità pari a $O(n^{2k})$.

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}.$$

ϕ_{move} dà la sicurezza che ci sono computazioni valide ed è espressa mediante windows legali. Quindi la ϕ_{move} è pari a tutte le finestre(i, j) legali.
La dimensione di una window legale è pari a 6, il numero di possibili windows legali è pari a $|C|^6$ e il numero di possibili celle è dato dal numero di righe per il numero di colonne del tableau delle configurazioni, quindi $(2n^k + 3) * n^k$; quindi, risulta che la ϕ_{move} ha complessità pari a $O(n^{2k})$.

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} (\text{la finestra } (i,j) \text{ è lecita}).$$

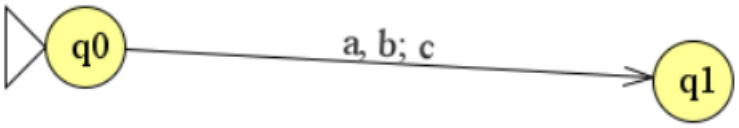
Risulta che la complessità di $\phi(M, w) = O(n^{2k}) + O(n^k) + O(n^{2k}) + O(n^{2k}) = O(n^{2k})$.

Quindi vi è complessità polinomiale in n e pertanto, data una mdT non deterministica, costruisco una formula $\phi(M, w)$ e ho che se $w \in L$ allora la formula risulta soddisfacibile.

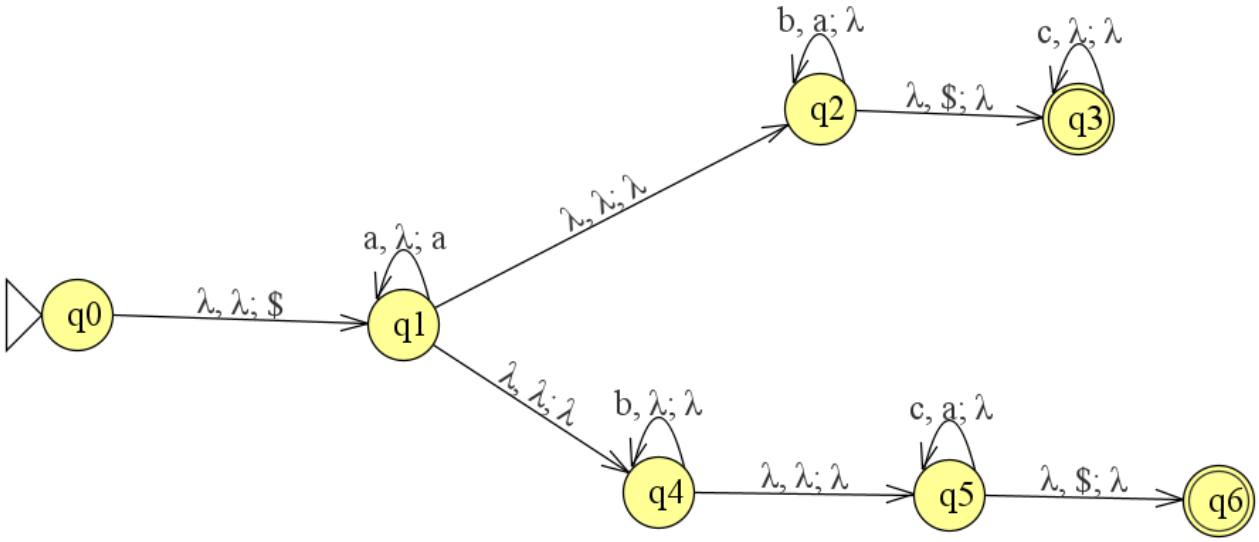
4) Un PDA è una settupla $P = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ tale che:

- Q è l'insieme degli stati;
- Σ è l'alfabeto;
- Γ è l'alfabeto dello stack;
- δ è la funzione di transizione;
- q_0 è lo stato iniziale;
- z è l'elemento iniziale dello stack;
- F è l'insieme degli stati finali;

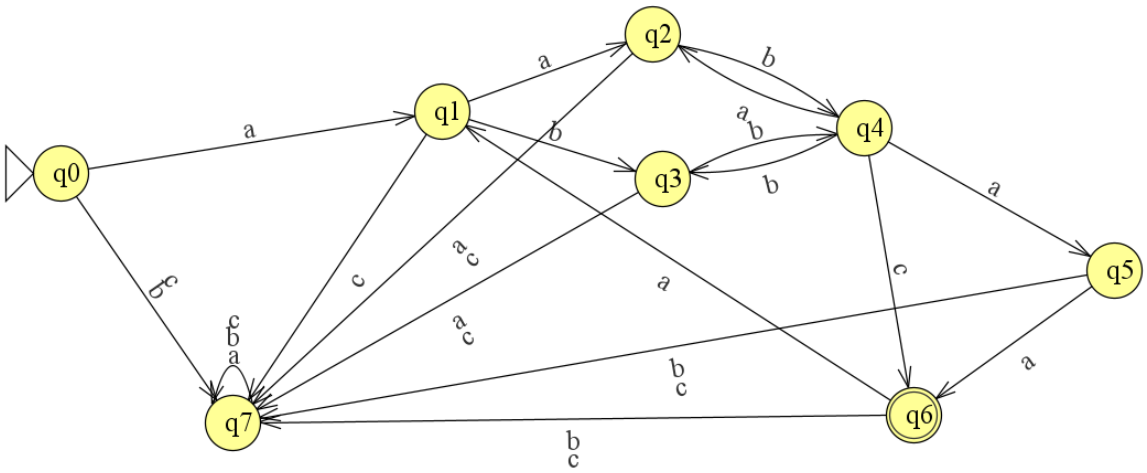
La funzione di transizione è fatta nel modo seguente:
$\delta(q_0, b, c) = \{(q_1, c)\}$



PDA che riconosce il linguaggio $\{a^i b^j c^k \mid i = j \text{ oppure } i = k\}$



5)



6) P è chiusa rispetto all'unione.
Per due qualsiasi linguaggi P, L_1 ed L_2, siano M_1 ed M_2 le macchine di Turing che li decidono in tempo polinomiale. Costruiamo una macchina di Turing M che decide l'unione di L_1 ed L_2 in tempo polinomiale:
M = "Su input <w>:

1. Avvia M_1 su w, se l'accetta allora accetta Altrimenti continua.
2. Avvia M_2 su w, se l'accetta allora accetta. Altrimenti rifiuta."

M accetta L_1 ed L_2 solo se M_1 o M_2 accettano w; quindi, M accetta l'unione di L_1 ed L_2. Visto che sia M_1 che M_2 vengono eseguite in tempo polinomiale, allora M avrà tempo polinomiale.

P è chiusa rispetto alla concatenazione.
Per due qualsiasi linguaggi P, L_1 ed L_2, siano M_1 ed M_2 le macchine di Turing che li decidono in tempo polinomiale. Costruiamo una macchina di Turing M che decide la concatenazione di L_1 ed L_2 in tempo polinomiale:
M = "Su input <w>:

1. Dividi w in due sottostringhe in tutte le maniere possibili ($w = w_1w_2$).
2. Esegui M_1 su w_1 ed M_2 su w_2. Se entrambe accettano, accetta.
3. Se w non è accettata dopo aver provato tutte le possibili combinazioni di sottostringhe, rifiuta."

M accetta w se e solo se può essere scritto in due come w_1w_2 tale che M_1 accetti w_1 e M_2 accetti w_2. Quindi M accetta la concatenazione di L_1 ed L_2. Dato che il punto 2

viene eseguito in tempo polinomiale ed è ripetuto al massimo O(n) volte, l'algoritmo viene eseguito in tempo polinomiale.

P è chiusa rispetto al complemento.
Per ogni linguaggio P, L, sia M la macchina di Turing che lo decide in tempo polinomiale. Costruiamo una macchina di Turin M' che decide il complemento di L in tempo polinomiale:
M' = "Su input <w>:

1. Esegui M su w.
2. Se M accetta, rifiuta. Se rifiuta, accetta."

M' decide il complemento di L. Dato che M viene eseguita in tempo polinomiale, anche M' viene eseguita in tempo polinomiale.

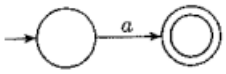
Calcolabilità e complessità (corso B) - Laurea in Informatica - a. a. 2021-2022
Prova scritta (2 ore) – 30 giugno 2022

1. Dimostrare che un linguaggio è regolare se e solo se esiste almeno una espressione regolare che lo descrive (4 punti).
2. Dato il linguaggio { M | M è una mdT e L(M)=∅ }, dimostrare che il suo complemento è Turing-riconoscibile (4 punti).
3. Dimostrare il teorema di Savitch (4 punti).
4. Scrivere la definizione di DFA, di NFA, e definire il DFA che riconosce il linguaggio (a|b)*abb (4 punti).
5. Fornire un esempio di problema in NP (4 punti).
6. Dimostrare che P è chiusa rispetto a intersezione e concatenazione (4 punti).

1) Questo teorema va dimostrato in entrambe le direzioni. Partiamo dalla prima: Se un linguaggio è descritto da un'espressione regolare, allora esso è regolare.

Trasformiamo l'espressione regolare R in un NFA N. Consideriamo i sei casi nella definizione di espressione regolare.

1. R = a per qualche a $\in \Sigma$. Allora L(R) = {a} e il seguente NFA riconosce L(R).

   Formalmente, N = ($\{q_1, q_2\}$, Σ, δ, q_1, $\{q_2\}$), dove descriviamo δ dicendo che $\delta(q_1, a) = \{q_2\}$ e $\delta(r, b) = \emptyset$ per $r \neq q_1$ o $b \neq a$.
2. R = ε. Allora L(R) = {ε} e il seguente NFA riconosce L(R).

   Formalmente, N = ($\{q_1\}$, Σ, δ, q_1, $\{q_1\}$), dove $\delta(r, b) = \emptyset$ per ogni r e b.
3. R = ∅. Allora L(R) = ∅, e il seguente NFA riconosce L(R).

   Formalmente, N = ($\{q\}$, Σ, δ, q, ∅), dove $\delta(r, b) = \emptyset$ per ogni r e b.
4. R = $R_1 \cup R_2$
5. R = $R_1 \circ R_2$
6. R = R_1*

Adesso passiamo alla seconda direzione:
Se un linguaggio è regolare, allora è descritto da un'espressione regolare.

Definiamo formalmente un GNFA. Un GNFA è simile ad un NFA tranne che per la funzione di transizione che ha la forma δ: $(Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \mathcal{R}$.

Il simbolo $\mathcal{R}$ è la collezione di tutte le espressioni regolari sull'alfabeto Σ, e q_{start} e q_{accept} sono gli stati iniziale e accettante. Se $\delta(q_i, q_j) = R$, l'arco dallo stato q_i allo stato q_j ha come etichetta l'espressione regolare R. Il dominio della funzione di transizione è $(Q - \{q_{accept}\}) \times (Q - \{q_{start}\})$ perché un arco collega ogni stato ad un qualsiasi altro stato, tranne che nessun arco è uscente da q_{accept} e uscente da q_{start}.

2) Sia $s_1, s_2, \ldots$ la lista di tutte le stringhe di Σ^*. La seguente mdT riconosce il complemento di $\{M \mid M \text{ è una mdT e } L = \emptyset\}$:

"Su input <M>, dove M è una mdT:

1. Ripete per i = 1, 2, 3, ….
2. Esegue M per i passi su ogni input, $s_1, s_2, \ldots, s_i$.
3. Se M ne ha accettato almeno uno, allora accetta. Altrimenti continua."

3) Il teorema di Savitch dice che una mdT deterministica simula una mdT non deterministica utilizzando uno spazio quadratico.

Una prima idea per dimostrare questo teorema sarebbe quella di simulare ogni ramo di computazione della macchina non deterministica ma, facendo in questo modo, non potrei sovrascrivere aree di memoria siccome per passare da un ramo di computazione ad un altro ho bisogno di memorizzare le scelte sul ramo di computazione. Pertanto, utilizzando questo approccio, avrei bisogno di uno spazio esponenziale.
Per dimostrare questo teorema utilizzo lo Yeldability Problem, il quale verifica se una mdT non deterministica con input con input w può passare da una configurazione iniziale a una finale in numero di passi che è minore o uguale a un dato t, dove t rappresenta il numero massimo di operazioni che la macchina non deterministica effettua per accettare la stringa w.
Quindi siano c_1 e c_2 rispettivamente le configurazioni iniziale e finale che utilizzano al massimo uno spazio pari a f(n) e sia t una potenza del 2.
Allora vado a definire una funzione ricorsiva CANYELD(c_1, c_2, 2^t), allora possono accadere le seguenti casistiche:

1) t = 0, allora vuol dire che $c_1 = c_2$, oppure che si passa da c_1 a c_2 in un solo step e, quindi, accetto, altrimenti rigetto;
2) t > 0, allora, evidentemente, si passa dalla configurazione iniziale a quella finale mediante delle configurazioni intermedie c_m; quindi, vado a effettuare due chiamate ricorsive della funzione CANYELD;
3) CANYELD(c_1, c_m, 2^{t-1});
4) CANYELD(c_m, c_2, 2^{t-1})
   (2^{t-1} perché devo passare prima da c_1 a c_m e poi da c_m a c_2);
5) Se i passi 3) e 4) accettano allora la computazione viene accettata;
6) Se uno solo tra i passi 3) e 4) non accetta, allora la computazione viene rigettata.

Tutte le computazioni utilizzano uno spazio che al massimo è f(n).
Ora vado a definire la mdT deterministica che simula quella non deterministica.

La mdT non deterministica prima di accettare pulisce il nastro e si riporta all'inizio del nastro dove entra in una configurazione di accettazione c_{accept}.

Ora denoto con d una costante tale che la mdT non deterministica effettua al massimo d * f(n) configurazioni per accettare la stringa w.

Pertanto, con queste assunzioni, risulta che la computazione viene accettata solo se si passa dalla configurazione iniziale a quella finale in al massimo $2^{d\,f(n)}$ passi.

L'output è raffigurato dal risultato della funzione ricorsiva
CANYELD(c_{start}, c_{accept}, $2^{d*f(n)}$)
Ogni computazione utilizza al massimo f(n) spazio.
Quindi la macchina deterministica utilizzerà per ogni chiamata ricorsiva O(f(n)) spazio per il numero delle chiamate ricorsive che è O(f(n)), quindi la mdT deterministica utilizzerà uno spazio pari a: O(f(n)) * O(f(n)) = O(f^2(n)).

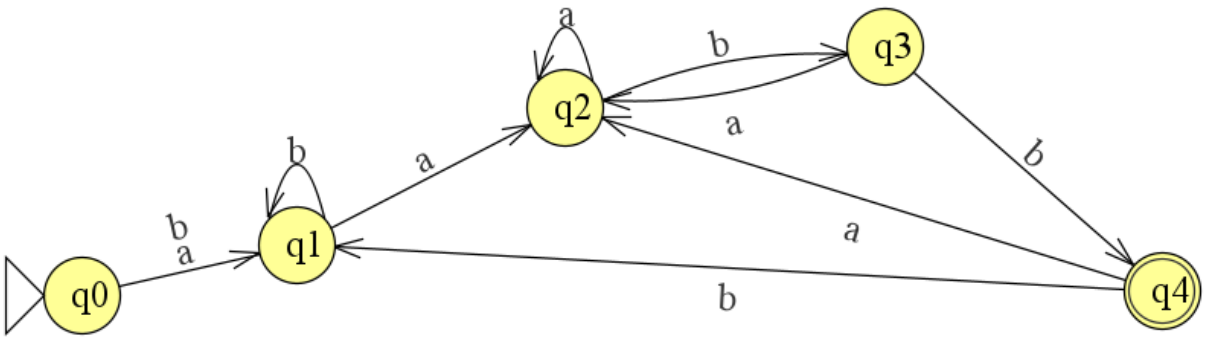
Pertanto, una macchina non deterministica è simulata da una deterministica utilizzando uno spazio che è quadratico.

4) Un automa a stati finiti è una quintupla M = (Q, Σ, δ, q_0 F) dove:

- Q è l'insieme degli stati;
- Σ è l'alfabeto;
- δ è la funzione di transizione;
- q_0 è lo stato iniziale;
- F è l'insieme degli stati finali.

Un automa a stati finiti deterministico (DFA) è un automa a stati finiti dove per ogni coppia di stato e simbolo in ingresso c'è una ed una sola transizione allo stato successivo.

Un automa a stati finiti non deterministico (NFA) è una macchina a stati finiti dove per ogni coppia stato-simbolo in input possono esservi più stati di destinazione. Al contrario degli automi a stati finiti deterministici, gli NFA possono cambiare stato indipendentemente dal simbolo letto, tramite epsilon-transizioni. Gli automi che presentano questo tipo di transizioni sono anche detti ε-NFA.



5) Un esempio di problema in NP può essere quello della clique. Una clique in un grafo non orientato è un sottografo, in cui ogni due nodi sono collegati da un arco. Una k-clique è una clique che contiene k-nodi. Il problema della clique consiste nel determinare se un grafo contiene una clique di una dimensione specificata.

Sia CLIQUE = {<G, k> | G è un grafo non orientato contenente una k-clique}.

Per dimostrare che CLIQUE è in NP si può usare un verificatore V:

V = “su input <<G, k>, c>:

1. Controlla se c è sottografo di G con k nodi;
2. Controlla se G contiene tutti gli archi tra i nodi in c;
3. Se entrambe le condizioni sono verificate accetta, altrimenti rifiuta.”

6) P è chiusa rispetto all’intersezione.
Per due qualsiasi linguaggi P, L_1 ed L_2, siano M_1 ed M_2 le macchine di Turing che li decidono in tempo polinomiale. Costruiamo una macchina di Turin M che decide

l'intersezione di L_1 ed L_2 in tempo polinomiale:
M = "Su input <w>:

1. Avvia M_1 su w, se l'accetta allora continua. Altrimenti rifiuta.
2. Avvia M_2 su w, se l'accetta allora accetta. Altrimenti rifiuta."

M accetta L_1 ed L_2 solo se sia M_1 che M_2 accettano w; quindi, M accetta l'intersezione di L_1 ed L_2. Visto che sia M_1 che M_2 vengono eseguite in tempo polinomiale, allora M avrà tempo polinomiale.

P è chiusa rispetto alla concatenazione.
Per due qualsiasi linguaggi P, L_1 ed L_2, siano M_1 ed M_2 le macchine di Turing che li decidono in tempo polinomiale. Costruiamo una macchina di Turin M che decide la concatenazione di L_1 ed L_2 in tempo polinomiale:
M = "Su input <w>:

1. Dividi w in due sottostringhe in tutte le maniere possibili ($w = w_1w_2$).
2. Esegui M_1 su w_1 ed M_2 su w_2. Se entrambe accettano, accetta.
3. Se w non è accettata dopo aver provato tutte le possibili combinazioni di sottostringhe, rifiuta."

M accetta w se e solo se può essere scritto in due come w_1w_2 tale che M_1 accetti w_1 e M_2 accetti w_2. Quindi M accetta la concatenazione di L_1 ed L_2. Dato che il punto 2 viene eseguito in tempo polinomiale ed è ripetuto al massimo O(n) volte, l'algoritmo viene eseguito in tempo polinomiale.

Calcolabilità e complessità (corso B) - Laurea in Informatica - a. a. 2021-2022
Prova scritta (2 ore) – 15 luglio 2022

1. Dimostrare che la classe dei linguaggi regolari è chiusa rispetto all'operazione di unione. (senza usare automi non deterministici - 4 punti).
2. Dato il linguaggio { ww | w in {0,1}* }, dimostrare che non è regolare (4 punti).
3. Fornire un esempio di linguaggio non Turing-riconoscibile e dimostrare perchè (4 punti).
4. Dimostrare che SAT è NP-completo (4 punti).
5. Un triangolo in un grafo non orientato è una clique di dimensione 3. Dimostrare che TRIANGLE è in P, con TRIANGLE = {<G> | G contiene un triangolo} (4 punti).
6. Scrivere il DFA che accetta il linguaggio {w in {0,1}* | w non contiene mai più di due 0 consecutivi} (4 punti).

1) Supponiamo che M_1 riconosca A_1, dove $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$, e che M_2 riconosca A_2, dove $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$. Costruiamo quindi $M = (Q, \Sigma, \delta, q_0, F)$ che riconosce $A_1 \cup A_2$.

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
   Questo è il prodotto cartesiano degli insiemi Q_1 e Q_2 ed è l'insieme di tutte le coppie di stati, il primo in Q_1 e il secondo in Q_2.
2. Σ, l'alfabeto è lo stesso sia per M_1 che per M_2 assumendo, per semplicità, che abbiano uguale alfabeto.
3. δ, la funzione di transizione è definita come segue. Per ogni $r_1, r_2 \in Q$ e ogni $a \in \Sigma$, sia $\delta((r_1, r_2)\ a) = (\delta_1(r_1, a), \delta_2(r_2, a))$. Quindi δ riceve uno stato di M, con un simbolo di input, e restituisce lo stato successivo di M.
4. q_0 è la coppia (q_1, q_2).
5. F è l'insieme delle coppie in cui l'uno o l'altro elemento è uno stato accettante di M_1 o di M_2. Possiamo scriverlo come $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$. Questa espressione è uguale a $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

Questo conclude la costruzione dell'automa finito M che riconosce l'unione di A_1 e A_2.

2) Dobbiamo dimostrare con il pumping lemma per i linguaggi regolari che il linguaggio $A = \{ww \mid w \in \{0, 1\}\}$ non è regolare.

Il pumping lemma dice che se A è un linguaggio regolare allora esiste un numero p tale che se s è una qualsiasi stringa in A di lunghezza almeno p, allora s può essere divisa in tre parti, s = xyz, soddisfacenti le seguenti condizioni:

1. $\forall i \geq 0, xy^iz \in A$;
2. $|y| > 0$;
3. $|xy| \leq p$.

Assumiamo quindi per assurdo che A sia regolare. Sia p la lunghezza del pumping data dal pumping lemma. Sia s la stringa 0^p10^p1. Poiché s è un elemento di A ed s ha lunghezza maggiore di p, il pumping lemma assicura che s può essere divisa in tre parti, s = xyz, che verificano le tre condizioni del lemma. Mostriamo che questo risultato è impossibile considerando quattro casi distinti.

1. La stringa y contiene solo 0 allora ci sarà un numero troppo elevato di 0 e la stringa risultante xyyz non farà parte di A perché non divisibile in due stringhe uguali;
2. La stringa y contiene 1 (ponendo z uguale a stringa vuota) e in questo caso ci sarà un 1 in più per ogni volta che si pompa la y, facendo sì che anche in questo caso la stringa risultante xyyz non faccia parte di A;
3. La stringa y contiene 1 seguito da p 0 e in questo caso la stringa risultante xyyz farà parte di A ma continuando ad iterare ci si accorgerà che xyyyz non fa parte di A, deducendo che i deve essere necessariamente dispari.
4. La stringa y è uguale a 0^p10^p1 (ponendo x e z stringhe vuote) osservando che effettivamente $\forall i \geq 0, xy^iz \in A$ però questo viola la condizione numero 3.

Osservando la condizione 3 ci si accorge che ci saremmo potuti fermare direttamente al primo caso, visto che è l'unico a non violarla. Quindi il linguaggio A non è regolare.

3) Un esempio di linguaggio non Turing-riconoscibile è il complemento del linguaggio A_{TM} = {(M, w), M è una mdT e che accetta w}.

A_{TM} è Turing-riconoscibile visto che è riconosciuto dalla seguente mdT:
"Su input <M, w>, dove M è una TM e w è una stringa:

1. Simula M su input w;
2. Se durante la computazione M entra nello stato di accettazione, accetta; se M entra nello stato di rifiuto, rifiuta."

Essendo Turing-riconoscibile ed essendo non decidibile, il suo complemento è necessariamente non Turing-riconoscibile.

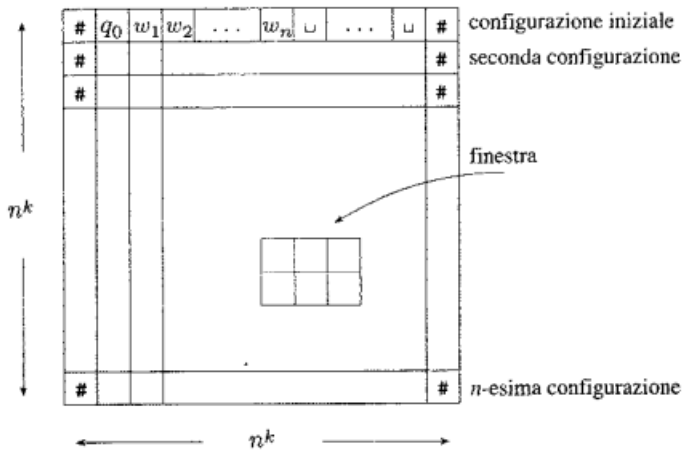
4) Il teorema di Cook-Levine dice che SAT è un linguaggio NP-completo.

Per dimostrarlo deve risultare che:

- SAT è in NP;
- Devo ridurre tutti i linguaggi al problema SAT in tempo polinomiale.

Allora:

- SAT è in NP sia con il verificatore che in maniera non deterministica; infatti, con il verificatore la stringa c rappresentava i valori di verità che rendevano la formula vera;
- Sia dato un linguaggio $L \in NP$.
  Devo definire una riduzione polinomiale a SAT.
  Dato che $L \in NP$, allora esiste una mdT non deterministica che decide L in tempo polinomiale. Per ogni stringa w costruisco in tempo polinomiale un'espressione booleana tale che se $w \in L$ allora l'espressione è vera, quindi: $\phi(M, w)$ è soddisfacibile se $w \in L$ e non soddisfacibile se $w \notin L$.
  Adesso vediamo come sono le computazioni della macchina non deterministica M.
  Dato che M non è deterministica ci saranno molte computazioni e dato che $L \in NP$, allora l'albero delle computazioni avrà profondità al massimo n^k, dove n è la lunghezza della stringa in input.
  Adesso considero una computazione accettante che ovviamente sarà al massimo n^k e l'input occuperà al massimo n^k celle di memoria a sinistra; quindi, la massima area di calcolo sul nastro sarà $2n^k$.
  Adesso l'albero delle computazioni si può vedere sottoforma di una matrice chiamata tableaux delle configurazioni.



Quindi l'alfabeto del tableau sarà: C = {{#} ∪ {insieme degli stati} ∪ {alfabeto del nastro}}.
Per ogni cella con posizione i, j e per ogni simbolo dell'alfabeto s ∈ C, creo la variabile $x_{i, j, s}$ in cui se nella posizione i e j vi è il simbolo s allora $x_{i, j, s} = 1$, altrimenti è uguale a 0.
Quindi ϕ(M, w) è costruita mediante la variabile $x_{i, j, s}$.
$\phi(M, w) = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$

ϕ_{cell} indica che ogni cella del tableau delle configurazioni contiene uno e un solo simbolo. Pertanto, per tutte le coppie di i e j esiste almeno un simbolo in ogni cella AND per tutti gli s e t, simboli dell'alfabeto, con s ≠ t deve succedere che o è presente s o è presente t, quindi in ogni cella vi è solo un simbolo e vi è la certezza che questo simbolo c'è. Ha complessità $O(n^{2k})$.
Per quanto riguarda la dimensione si ha $2n^k + 3$ perché la massima area di calcolo è pari a $2n^k$, poi vi sono due #, uno all'inizio della riga e uno alla fine della riga e in più vi è lo stato, quindi $2n^k + 3$.

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right].$$

ϕ_{start} indica come è formata la prima riga del tableau, quindi come inizia la computazione. La ϕ_{start} ha dimensione $2n^k + 3$, in quanto lo spazio di ogni singola riga è pari a $2n^k + 3$ e, quindi, ha complessità pari a $O(n^k)$.

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge$$
$$x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge$$
$$x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}.$$

ϕ_{accept} dà certezza che la computazione raggiunge uno stato di accettazione. La ϕ_{accept} ha dimensione $(2n^k + 3)^2$, quindi ha complessità pari a $O(n^{2k})$.

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}.$$

ϕ_{move} dà la sicurezza che ci sono computazioni valide ed è espressa mediante windows legali. Quindi la ϕ_{move} è pari a tutte le finestre(i, j) legali.
La dimensione di una window legale è pari a 6, il numero di possibili windows legali è pari a $|C|^6$ e il numero di possibili celle è dato dal numero di righe per il numero di colonne del tableau delle configurazioni, quindi $(2n^k + 3) * n^k$; quindi, risulta che la ϕ_{move} ha complessità pari a $O(n^{2k})$.

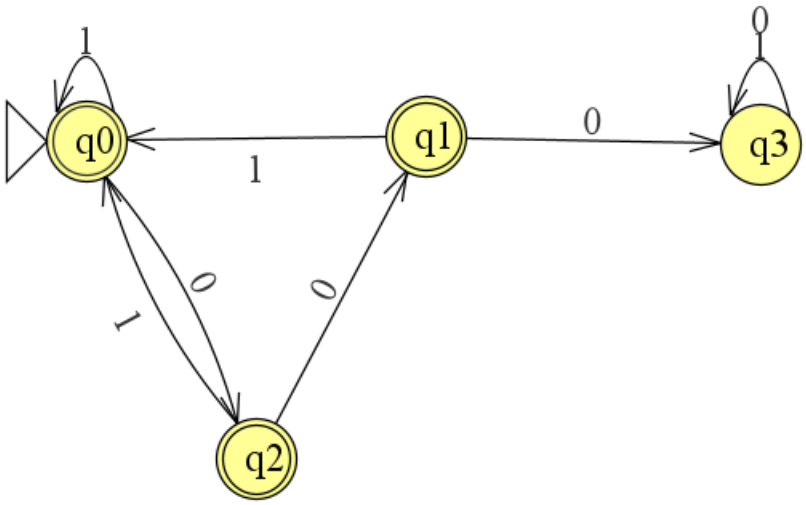
$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} (\text{la finestra } (i,j) \text{ è lecita}).$$

Risulta che la complessità di $\phi(M, w) = O(n^{2k}) + O(n^k) + O(n^{2k}) + O(n^{2k}) = O(n^{2k})$.

Quindi vi è complessità polinomiale in n e pertanto, data una mdT non deterministica, costruisco una formula $\phi(M, w)$ e ho che se $w \in L$ allora la formula risulta soddisfacibile.

5) TRIANGLE è in P perché può essere deciso in tempo polinomiale. Si può utilizzare un algoritmo che, con input <G>, esamina tutte le terne di nodi. Se n = |V|, quindi n uguale al numero dei vertici, allora ci sono $\frac{n(n-1)(n-2)}{3 \cdot 2 \cdot 1}$ triple. Per ciascuna di queste triple, controlla se tutti e tre i bordi sono collegati, cioè formano un triangolo. Appena viene trovato un triangolo, accetta. Se alla fine non viene trovato nessun triangolo, rifiutare. L'algoritmo richiede passi $O(n^3)$, quindi è polinomiale.

6)


q0
q1
q2
q3
1
0
1
0
1
0
0