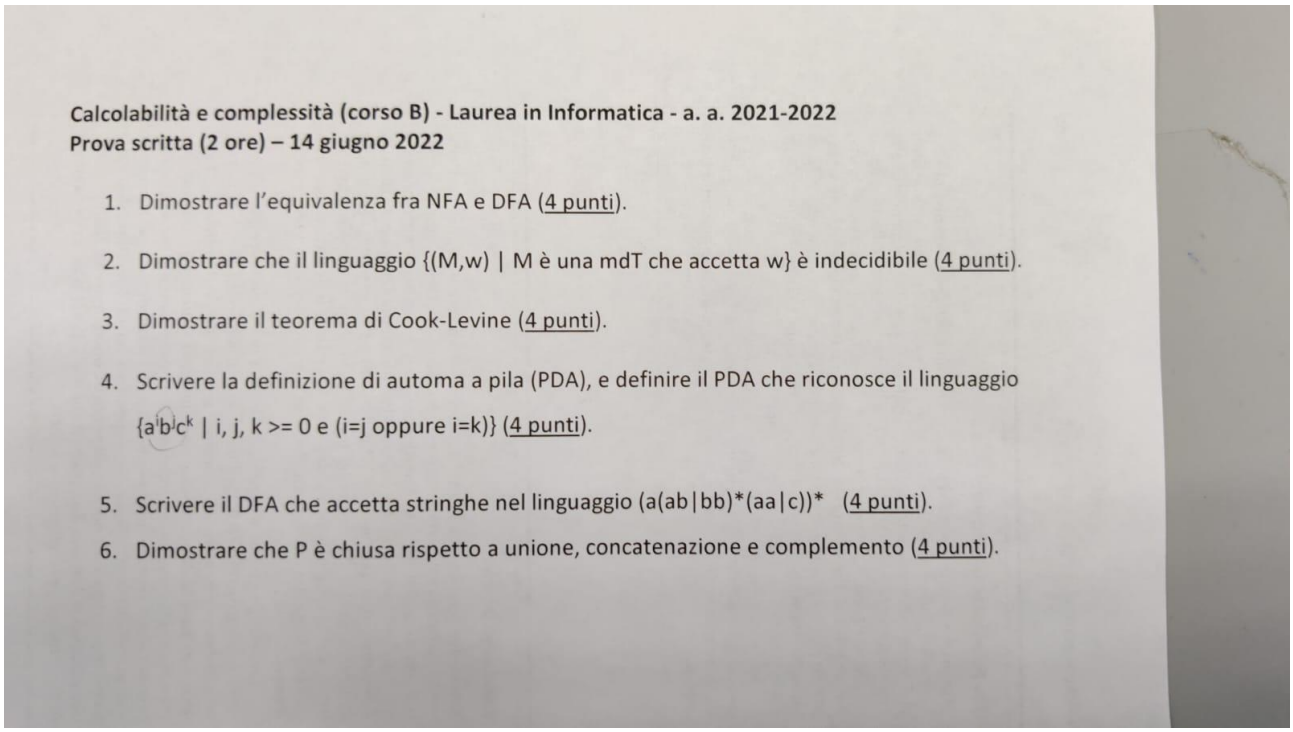


Calcolabilità e complessità (corso B) - Laurea in Informatica - a. a. 2021-2022
Prova scritta (2 ore) – 14 giugno 2022

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.

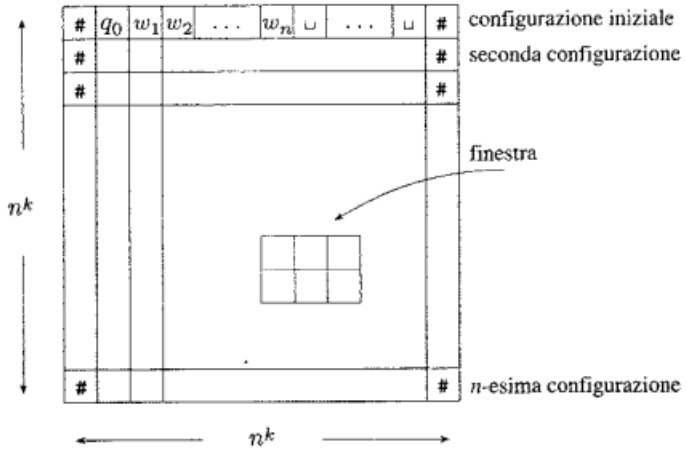
Per dimostrarlo deve risultare che:

- 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 $O(n^{2k})$.

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

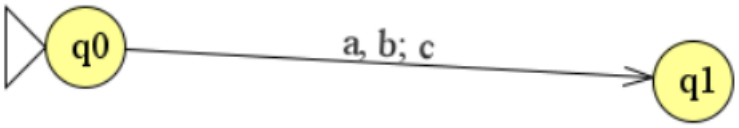
Risulta che la complessità di $\phi(M, w) = O(n^{2k}) + O(n^k) + O(n^{2k}) + O(n^{2k}) = O(n^{2k})$.

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.

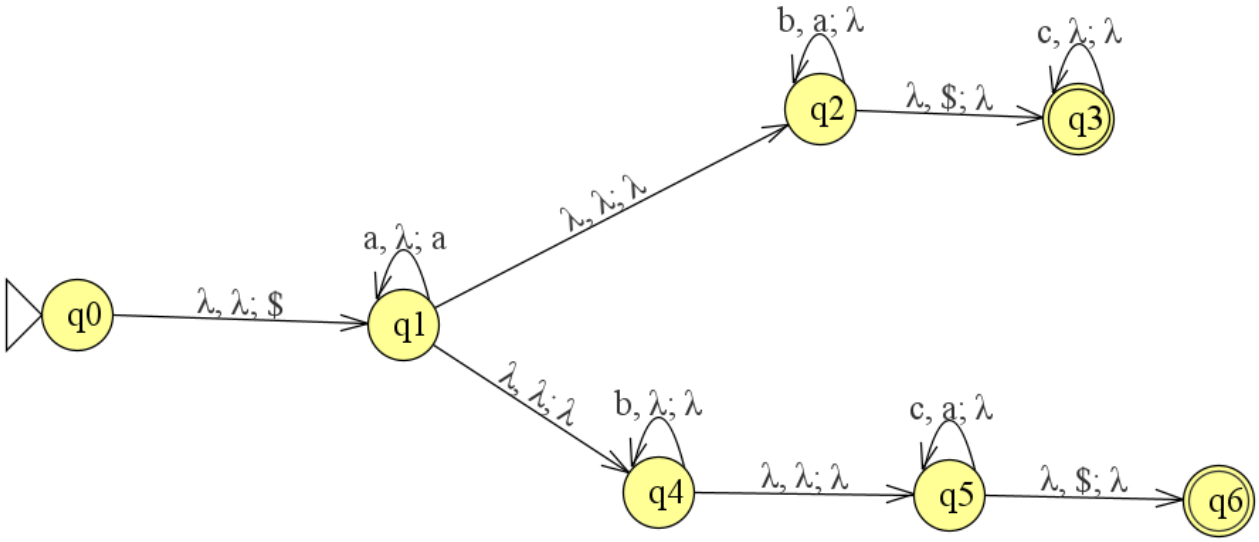
4) Un PDA è una settupla $P = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ tale che:

- 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;

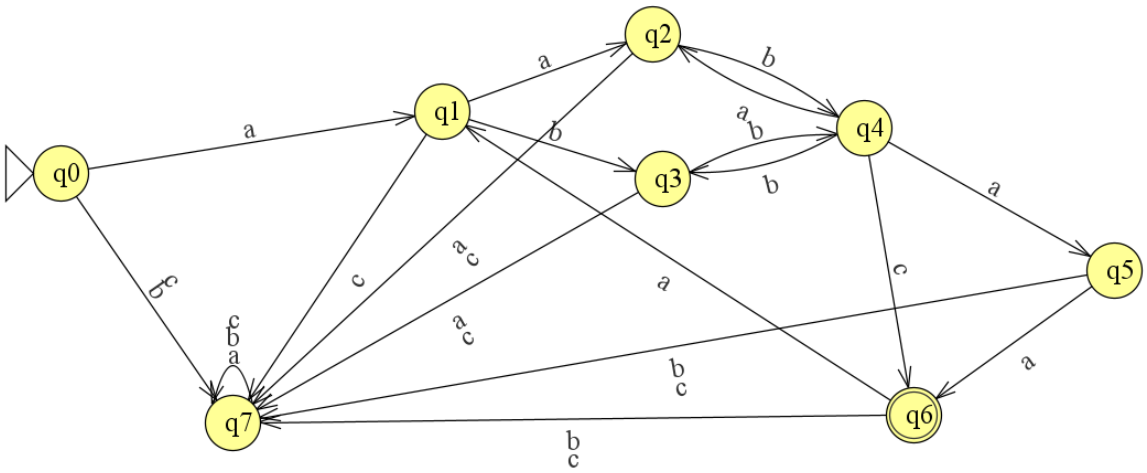
La funzione di transizione è fatta nel modo seguente:
$\delta(q_0, b, c) = \{(q_1, c)\}$



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.