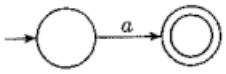
Calcolabilità e complessità (corso B) - Laurea in Informatica - a. a. 2021-2022
Prova scritta (2 ore) – 30 giugno 2022

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)).

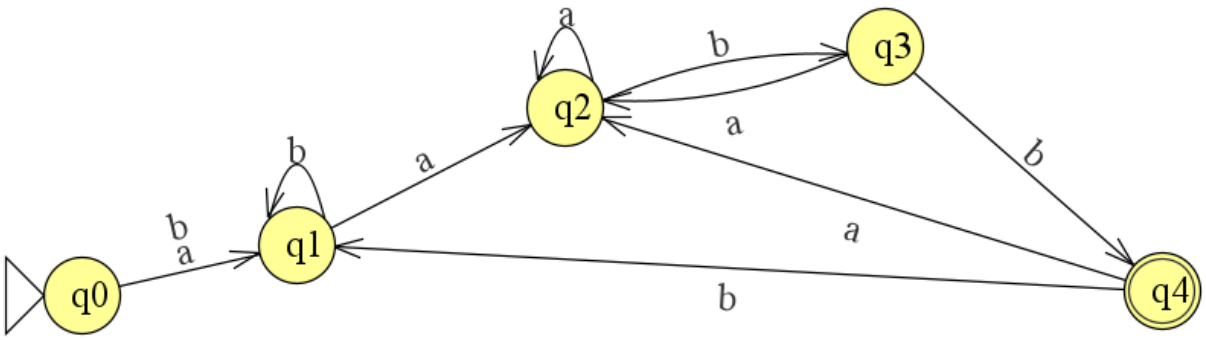
Pertanto, una macchina non deterministica è simulata da una deterministica utilizzando uno spazio che è quadratico.

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.