Risposte alle domande di automi del 08/09

1. Shuffle perfetto

Essendo A e B linguaggi regolari, esistono due DFA che li riconoscono. Essenzialmente essendo a_i, b_i simboli appartenenti all'alfabeto, si costruisce un automa E che contiene tutti gli stati degli automi E_A, E_B che accettano i linguaggi A e B, e che ogni input dispari (le a_i) si sposta nello stato corrispondente di E_A, e ad ogni input pari (le b_i) si sposta nello stato corrispondente di B. L'automa accetta quando attraversa sia lo stato finale di Ea che lo stato finale di E_B.

2. DROPOUT

Linguaggio regolare significa che esiste un DFA che lo riconosce. Tutte le parole di L sono di lunghezza > 1 (altrimenti non si potrebbe togliere una lettera). Togliendo una lettera ad una parola composta da una sola lettera otteniamo un set vuoto, che per definizione è un linguaggio regolare. Nel caso l'automa sia composto da due o più simboli dell'alfabeto dobbiamo semplicemente costruire un automa che sostituisce la transizione che accettava y e con una epsilon transizione allo stato successivo.

3. Regolarità dell'inversa

Dato il linguaggio A regolare formato da tutte le stringhe $w=w_1w_2w_3...w_n$ si può dimostrare che il suo inverso (quello formato da tutte le stringhe $w^R=w_n...w_3w_2w_1$) è ancora regolare in quanto è possibile partendo dal DFA che riconosce A costruire il DFA che riconosce A^R.

Procedimento

- Si girano tutte le transizioni del DFA che riconosce A (cambiare verso delle frecce)
- Lo stato iniziale diventa il nuovo stato finale (tutti gli altri stati diventano non finali)
- Creo il nuovo stato iniziale e da questo faccio partire tante epsilon transizioni che finiscono dove c'erano i vecchi stati finali

4. A/b regolare

Il linguaggio A/b riconosce tutte le stringhe che hanno la proprietà che se concateno b ad una qualsiasi stringa ottengo una stringa che appartiene al linguaggio regolare A.

Dato che i linguaggi regolari sono chiusi rispetto alla concatenazione, per ottenere una stringa appartenente ad A concatenando b a w significa che anche w deve appartenere ad un linguaggio regolare quindi A/b è un linguaggio regolare.

Si dimostra che i linguaggi regolari sono chiusi per concatenazione collegando lo stato finale di un DFA che riconosce un linguaggio regolare con un epsilon transizione allo stato iniziale di un DFA che riconosce un altro linguaggio regolare in modo da concatenare le due stringhe riconosciute da ciascun DFA.

5. A/B regolare

Il linguaggio A/B è formato dalle parole w ottenute a partire da una parola wx appartenente al linguaggio regolare A, alle quali viene rimossa la parte x, che è una parola del linguaggio B (linguaggio qualsiasi).
$A/B = \{w \mid wx \in A \text{ per qualche } x \in B\}$

w deve quindi essere un prefisso di A e per esserlo deve essere regolare.
Si può dimostrare che è regolare creando un DFA che accetta un prefisso di A.

Il DFA che riconosce un prefisso di A riceve w in input e rifiuta non appena un possibile simbolo in input non fa match con quello che il DFA si aspetta.

Nel caso in cui venga consumato tutto l'input (w) allora il DFA accetta andando in uno stato finale; se non fa match o se w è più lunga della stringa più lunga del linguaggio A allora rifiuta.

6. Pumping lemma

(a) 110* la lunghezza minima p è 3 perché si deve poter pompare almeno uno 0 per rientrare ancora nel linguaggio. Se scegliessi p=2 dovrei pompare un 1 il che non soddisfa la condizione per la quale devono esserci 2 uni iniziali infatti con la stringa di lunghezza minima 11 andrei a togliere uni o aggiungere uni.

(b) 1*0*1* la lunghezza minima p è 1 perché la stringa può iniziare con un numero arbitrario di 1 o di 0 quindi posso pompare 1 ottenendo 111111 che appartiene al linguaggio oppure 00000 che appartiene al linguaggio

(c) 0*1*0*1* + 10*1 la lunghezza minima p è 1
La lunghezza di 10*1 sarebbe 3 ma si nota che con l'espressione 0*1*0*1* si possono generare tutte le stringhe 10*1 quindi p=1 che è la minima tra le due espressioni

(d) (01)* la lunghezza minima p è 2 perché possiamo pompare almeno 01 un numero arbitrario di volte (anche nessuna) ed ottenere una stringa che appartiene al linguaggio (non potrei pompare solo lo 0 in quanto le stringhe sono formate da 01 concatenati)

(e) Ø la lunghezza minima p è 1

(f) 0*01*01* la lunghezza minima è 2 in quanto la stringa 00 può essere pompata.

(g) 10(11*0)*0 la lunghezza minima è 4 perché la stringa di lunghezza 3 100 non può essere pompata e la stringa più corta dopo quella di lunghezza 3 generabile è di lunghezza 5 cioè 10100.
Posso mettere p=4 perché una lunghezza 4 non si presenterà mai perché non sono generabili e saranno tutte lunghezza almeno 5 che possono essere pompate.

(h) 101101 la lunghezza minima p è 7 perché la stringa di lunghezza minima 6 non può essere pompata. Mettere 7 mi permette di rispettare la condizione del lemma e di pompare il vuoto dopo l'unica stringa accettabile dal linguaggio (101101)

(i) {w∈a Σ | w != 101101} la lunghezza minima p è 1 perché posso pompare qualsiasi stringa di lunghezza uno e non ottenere mai 101101 perché formata da almeno due simboli diversi tra loro.

(j) 0001* la lunghezza minima di p è 4 altrimenti verrebbero ripetuti degli 0, andando quindi fuori dal linguaggio

(k) 0*1* la lunghezza minima è 1, l'unica condizione necessaria per soddisfare il pumping lemma in questo caso è che la y sia composta interamente di 0 o di 1

(l) 001+0*1* la lunghezza minima è 1, l'aggiunta di eventuali 0 (o anche di 1 per lunghezze > 2) non farebbe uscire la parola dal linguaggio

(m) ε non è possibile applicare il pumping lemma perché y deve essere diverso

(n) 1*01*01* la lunghezza minima è 1, il pumping lemma sarà soddisfatto se la y è fatta di soli 1

(o) 1011

(p) Σ*

7. Linguaggio D...

Let $\Sigma=\{0,1\}$ and let
D={w|w contains an equal number of occurrences of the substrings 01 and 10}.

Thus $101 \in D$ because 101 contains a single 01 and a single 10, but $1010 \notin D$ because 1010 contains two 10s and only one 01. Show that D is a regular language.

Solution:

This language is regular because it can be described by a regular expression and a FA (NFA):

Regular Expression

$$\left(1^{+}0^{*}1^{+}\right)^{*}+\left(0^{+}1^{*}0^{+}\right)^{*}$$

8. Linguaggi A regolare e B irregolare

L'espressione regolare che riconosce il linguaggio A è 00*(0+1)*00* oppure più semplicemente 0(0+1)*0.
Il linguaggio A è regolare perché qualsiasi stringa che inizia e termina con 0 appartiene al linguaggio (fonte). Pompando il linguaggio con degli 0 la parola risultante continuerà a far parte del linguaggio (perché la parte u non è regolare).

Il linguaggio B non è regolare perché un automa riconoscitore non sarebbe in grado di contare il numero di 0 all'inizio e alla fine della parola, nel caso in cui u sia una stringa vuota si può vedere chiaramente che pompando una parola si esce dal linguaggio.

9. Linguaggi non regolari

|xy| <= k, |y| > 0 |w| >= k

(a) $\{0^n 1^m 0^m$ con m,n >=0$\}$
Ponendo n=0 ottengo $w = 0^0 1^k 0^k$. |w| > k
$x = 1^a$, $y = 1^b$, $z = 1^c 0^k$ e sappiamo che a+b+c = k.
Se noi pompiamo y con i != 1 a + ib+ c != k, allora w non appartiene più al linguaggio, che non è regolare.

(b) $\{0^n 1^m \mid n != m\}$
$w = 0^{k-1}\ 0\ 1^{p+1}$, 0 = k, 1 = k+1

(c) {w ∈ {0, 1}* | w è palindoma}
$w = 0^k\ 111\ 0^k$
$x = 0^a$, $y= 0^b$, $z = 0^c\ 111\ 0^k$
Se io pompo y per i = 2 otteniamo che a+bi+c >k il che dimostra che $0^{a+bi+c}1110^k$ non è più palindroma, quindi non è regolare.

(d) {wtw | w,t ∈ {0,1}+}
G = wtw = $0^k1\ 11\ 0^k1$ w = 0^k1 t = 11
x = 0^a, y= 0^b, c = $0^c1\ 11\ 0^k1$
Se io pompo y per i = 2 otteniamo che a+bi+c > k rendendo la parola originariamente nella forma wtw nella forma $0^{a+bi+c}1\ 11\ 0^k1$ che non rispetta il linguaggio, quindi non è regolare.

10. SUFFIX

Dimostro che i linguaggi context-free sono chiusi rispetto all'operazione suffisso con un PDA che riconosce il suffisso di A:

Procedimento

Chiamiamo M il PDA che riconosce il suffisso di A.
Dato che A è context-free allora esiste un PDA che lo riconosce e lo chiamiamo M1.
Facciamo una copia di M1 e la chiamiamo M2, che ha esattamente gli stessi stati, stack, e transizioni di M1.

M1 ed M2 formeranno il PDA M.

Cambio le ε-transizioni di M2: se la transizione originale era a, b → ε la cambiamo in ε, b→ epsilon.
Per ogni stato di M2 aggiungo una transizione ε, ε → ε che va al corrispondente stato di M1.

Lo stato iniziale di M2 è lo stato iniziale di M.

Possiamo vedere che M costruirà lo stack appropriatamente ignorando l'input.

Ad un certo punto M non deterministicamente transita dallo stato M2 al corrispondente stato in M1, comincia a prendere il primo carattere del suffisso come input e continua le transizioni in M1 da questo punto in poi.

Quindi tutti i suffissi della stringa che appartengono ad A saranno accettati da M.

11. Classe CFL non chiusa all'intersezione

Se una stringa è accetta per A significa che ha lo stesso numero di b e di c - Se una stringa è accettata per B significa che ha lo stesso numero di A e b.

Se w è riconosciuto significa che #A = #B = #C. quindi $\{a^n + b^n + c^n \mid n \geq 0\}$ è facilmente dimostrabile che non è context free.

12. CFL

(a) S → 0T0 | 1T1
    T → 0T | 1T | ε
(b) S → 0S1 | 0S0 | 1S0 | 1S1 | 0
(c) S → 0S0 | 1S1 | 1 | 0 | ε
(d) S → T0T
    T → 1T0 | 0T1 | TT | 0T | ε
    oppure (S → 0 | SS | 0S1 | 1S0)?

(e) S → T1T0T | A | B
T → 1T | 0T | ε
A → 0A1 | A0 | 0
B → 0B1 | B1 | 1

(f) S → TA
T → 0T0 | 1T1 | #A
A → AA | 0 | 1 | ε

(g) Non CFL?

(h) Non CFL?

(i) a^ib^j con i != j e 2i != j
3 casi: i > j, i <j con 2i > j, 2i < j
S → G1 | G2 | G3
G1 → aG1b | aT
T → aT | ε
G3 → aG3bb | Tb
T → Tb| ε
G2 → aG2b | aTb
T → aTbb | abb

13. A ◦ B context free

Il linguaggio contiene tutte le stringhe che sono la concatenazione di una stringa di A con una stringa di B della stessa lunghezza dove A e B sono regolari.

I linguaggi regolari sono chiusi per concatenazione quindi A ◦ B è ancora un linguaggio regolare.

I linguaggi regolari sono anche linguaggi context free perché è possibile scrivere una grammatica che li genera o un PDA che li riconosce.

Ad esempio, per riconoscere un linguaggio regolare con un PDA basta creare un automa a pila che non utilizza la pila nelle sue transizioni e si ottiene l'equivalente DFA che riconosce il linguaggio regolare.

14. Dimostrare che le derivazioni dalla forma normale di Chomsky hanno lunghezza 2n-1

Se iniziamo la derivazione con un singolo non terminale S e creiamo la stringa mediante derivazioni successive ogni regola A→BC aggiunge un nuovo non terminale e ogni regola A→a converte un non terminale in terminale. Quindi impieghiamo n-1 passi per sviluppare l'originale non terminale in n non terminali e altri n passi per convertire i non terminali in terminali. Quindi 2n-1 passi

15. A decidibile

È decidibile perché il linguaggio s è formato da un set finito di elementi (0,1). La macchina che prende in input s non andrà mai in loop ma sarà sempre in grado di accettare o rifiutare.

16. Automa a coda

We use *queue automaton* to simulate any *Turing machine with left reset* introduced in *Exercise 3.12* (and proven to be equivalent with ordinary TM). First step of computation is to transfer the input string on queue, using right moves and pushing each read symbol. We also put a special symbol which signifies the left-hand end of tape.

Now we simulate right move of TM with pull of rightmost element from queue and pushing the new symbol according to transition function back to queue. In case the TM is on blank symbol after the input, we just push the new symbol.

Left reset of TM is simulated using pushing until we reach special symbol which denotes the left-hand end of tape, push and pull it as well, and then we are at the right position.

For the other direction, ordinary TM can easily keep a snapshot of queue on its tape and simulate the appropriate actions.

17. TM a sola scrittura

Una macchina di Turing in sola scrittura equivalente ad una TM normale perché è possibile andare a scrivere i passaggi di ogni computazione su nastri diversi in modo da non scrivere mai sullo stesso nastro. Dato che una TM multinastro può essere trasformata in una macchina a singolo nastro separando ciascun nastro con un simbolo speciale si può affermare che una TM può compiere le stesse computazioni di una macchina che può compiere una sola scrittura.

18. k-PDA

2PDA sono più potenti dei 1PDA
si deve dimostrare che possono fare qualcosa in più infatti il linguaggio $0^n1^n0^n$ non è riconoscibile da un 1PDA perché non è un linguaggio context free.

Infatti, andando a scrivere sulla pila gli zeri per poi consumarli per mettere lo stesso numero di 1 perderei l'informazione per poter riconoscere la parte finale della stringa.

Con 2 pile posso usare la prima pila per memorizzare gli zeri poi consumarli e scrivere gli uni sulla seconda pila per poi usare la seconda pila per riconoscere la parte finale di zeri.

3PDA hanno la stessa potenza dei 2PDA uso il punto b per dimostrare c (2PDA visti come una TM)

i 2PDA possono simulare una TM

Una TM multinastro può simulare un k PDA perché utilizza il nastro come pila

dimostriamo che un 2PDA può simulare una TM

nella prima pila si mettono tutti i simboli prima della testina (all'inizio vuoto)

nella seconda pila mettiamo tutti gli elementi dopo la testina

la testina può spostarsi a destra e sinistra.

Questo può essere fatto spostando il simbolo in cima alla seconda pila nella prima pila

(la testina punta sempre al primo elemento della seconda pila)

se devo scrivere la cella sotto alla testina faccio il pop dell'elemento in cima alla seconda pila e il push del nuovo simbolo.

Dato che stiamo usando una memoria idealmente infinita per simulare il comportamento del nastro i 3PDA non sono più potenti dei 2PDA.

19. ALL_{DFA}

(c) $ALL_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA that recognizes } \Sigma^*\}$

Solution: A DFA A recognizes Σ^* iff all states that are reachable from the initial state are goal states. This can easily be checked by a Turing machine. Alternatively, we can use that EQ_{DFA} is decidable. Let M be a TM that decides this language. We can obviously construct a TM M' that decides ALL_{DFA} as follows:

M' = "On input $\langle A \rangle$, where A is a DFA:

1. Create a DFA B that consists only of the initial state q_0 which is a goal state. For each symbol of the alphabet there is a transition from q_0 to q_0.
2. Run M on $\langle A, B \rangle$.
3. If M accepts, accept; if M rejects, reject."

M' decides ALL_{DFA}: M accepts iff $L(A) = L(B)$, and by construction $L(B) = \Sigma^*$.

20. $A_{\varepsilon CFG}$

Solution: We know from the lecture that A_{CFG} = {hG, wi | G is a CFG that generates input string w} is decidable. Let M be a Turing machine that decides this language. We can obviously construct a TM M_0 that decides A_{CFG} as follows:

M_0 = On input hGi, where G is a CFG:
1. Run M on <G, i>
2. If M accepts, accept; if M rejects, reject.

21. S_{REX}

Assumo che il linguaggio sia decidibile quindi esiste una TM decisore

costruisco la macchina T che fa le seguenti operazioni:

prima di tutto prendo in input la stringa x. Essa deve essere una codifica valida dell'espressione regolare <R, S>. Se così non è rifiuto.

dato che R ed S sono regolari è possibile definire una codifica per i DFA. Questa è un'operazione che termina sempre la computazione.

A questo punto la macchina utilizza le codifiche ricevute per creare la codifica che rappresenta l'intersezione tra R ed S complementare (anche questa è un'operazione che impiega un tempo finito) e la dà in input alla macchina che riconosce il vuoto per i DFA. (si usa la macchina che decide il problema del vuoto come subroutine)

Sappiamo che il problema del vuoto per i linguaggi regolari è un problema decidibile quindi la macchina saprà sempre terminare la computazione.

Se la macchina che riconosce il vuoto accetta vuol dire che non c'è nessun elemento di R che non sta anche in S quindi R è sottoinsieme di S e la macchina T accetta altrimenti rifiuta.

Ho costruito un decisore di S_{REX} quindi il linguaggio è decidibile.

Controllo se x è valida
R → DFAR, S → DFAS
DFAS -> DFASC
Creo un DFA che intersechi DFAR e DFASC
Output nel DFA che riconosce il vuoto, se accetta allora R è sottoinsieme di S.

22. TM che non modifica l'input

3. Let $X = \{\langle M, w\rangle |$ where M is a single tape TM that never modifies the portion of the tape containing the input w $\}$. Show that X is not decidable by a reduction from A_{TM}.

   Hint: you may find it easier to work with $\overline{X}$.

   Solution: We show $A_{\text{TM}} \leq_m \overline{X}$. Here is the reduction: $f(\langle M, w\rangle) = \langle M'_w, w\rangle$, where M'_w is defined as follows: on any input it moves past the input, writes \$ followed by w, and then it simulates M on w never going to the left of \$. If M accepts w, M'_w moves to the left of \$, and writes anything on the original input, and accepts.

   The correctness of the reduction follows from the fact that if M does not accept w, M' will never move to the left of \$, and if M does accept w, then M' writes something on the original input.

Faccio una riduzione (tramite funzione di riduzione) A_{TM} a X complemento. La riduzione prende input M, w andando a creare M', w con M' definita così:

M' si sposta dopo l'input (a destra), ci mette un $ e ricopia l'input. Esegue tutte le operazioni che deve fare e, se M accetta, si sposta dopo il $ e scrive un qualsiasi carattere sull'input. Se rifiuta, ce ne freghiamo.

Essendo che se M accetta, M' scriverà sopra l'input crea così un assurdo (perché M accetta se non scrive niente sopra l'input).

ciaociao$ciaociao

23. Complemento di E_{TM} riconoscibile

4. Consider the emptiness problem for Turing machines:

$$E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a Turing machine with } L(M) = \emptyset \}.$$

Show that E_{TM} is co-Turing-recognizable. (A language L is co-Turing-recognizable if its complement $\overline{L}$ is Turing-recognizable.) Note that the complement of E_{TM} is

$$\overline{E_{\text{TM}}} = \{ \langle M \rangle \mid M \text{ is a Turing machine with } L(M) \neq \emptyset \}.$$

(Actually, $\overline{E_{\text{TM}}}$ also contains all $\langle M \rangle$ such that $\langle M \rangle$ is not a valid Turing-machine encoding, but we will ignore this technicality.)

Answer: We need to show there is a Turing machine that recognizes $\overline{E_{\text{TM}}}$, the complement of E_{TM}. Let $s_1, s_2, s_3, \ldots$ be a list of all strings in Σ^*. For a given Turing machine M, we want to determine if any of the strings $s_1, s_2, s_3, \ldots$ is accepted by M. If M accepts at least one string s_i, then $L(M) \neq \emptyset$, so $\langle M \rangle \in \overline{E_{\text{TM}}}$; if M accepts none of the strings, then $L(M) = \emptyset$, so $\langle M \rangle \notin \overline{E_{\text{TM}}}$. However, we cannot just run M sequentially on the strings $s_1, s_2, s_3, \ldots$. For example, suppose M accepts s_2 but loops on s_1. Since M accepts s_2, we have that $\langle M \rangle \in \overline{E_{\text{TM}}}$. But if we run M sequentially on $s_1, s_2, s_3, \ldots$, we never get past the first string. The following Turing machine avoids this problem and recognizes $\overline{E_{\text{TM}}}$:

R = "On input $\langle M \rangle$, where M is a Turing machine:

1. Repeat the following for $i = 1, 2, 3, \ldots$.
2. Run M for i steps on each input $s_1, s_2, \ldots, s_i$.
3. If any computation accepts, *accept*.

Dobbiamo dimostrare che la E_{TM} complementare è Turing riconoscibile cioè una TM che riconosca il linguaggio L(M) != ε. Il che significa creare una TM che sia in grado di farlo.

Su input di stringhe, si itera ogni stringa in modo da riconoscere se s∈L(M). (Se vogliamo farlo decidibile, basta seguire la foto qui sopra).

24. A decidibile

3. ($\Rightarrow$) If $A \leq_m A_{TM}$, then A is Turing-recognizable because A_{TM} is Turing recognizable.

($\Leftarrow$) If A is Turing-recognizable, then there exists some TM R that recognizes A. That is, R would receive an input w and accept if w is in A (otherwise R does not accept). To show that $A \leq_m A_{TM}$, we design a TM that does the following: On input w, writes $\langle R, w \rangle$ on the tape and halts. It is easy to check that $\langle R, w \rangle$ is in A_{TM} if and only if w is in A. Thus, we get a mapping reduction of A to A_{TM}.

Se A è Turing riconoscibile allora esiste una TM in grado di accettare A.

A è riducibile al suo complementare in modo che dato l'input della macchina che accetta A è possibile ottenere l'input di una macchina che accetta A complementare.

L'input va trasformato in modo che quando la TM che accetta A accetta la TM che accetta A complementare vada in loop e quando la TM che accetta A va in loop la macchina che accetta A complementare accetta.

Ciò equivale a dire che esiste una macchina che sa accettare A e una che sa rifiutare A.

Esiste quindi un decisore di A.

Il decisore di A deve simulare le due macchine in parallelo in modo che se una va in loop l'altra posso terminare la computazione.

25. A riconoscibile

1. Let C be a language. Prove that C is Turing-recognizable iff a decidable language D exists such that $C = \{x | \exists y(\langle x, y\rangle \in D)\}$

Solution:

We need to prove both directions. To handle the easier one first, assume that the decidable language D exists. A TM recognizing C operates on input x by going through each possible string y and testing whether $\langle x, y\rangle \in D$. If such a y is ever found, accept; if not, just continue searching.

For the other direction, assume that C is recognized by TM, denoted by M. Define the language D to be $\{\langle x, y\rangle \mid M \text{ accepts } x \text{ within } |y| \text{ steps}\}$. Language D is decidable since one can run M for y steps and accept iff M has accepted. If $x \in C$, then M accepts x within some number of steps, so $\langle x, y\rangle \in D$ for some sufficiently long y, but if $x \notin C$ then $\langle x, y\rangle \notin C$ for any y.

Si devono dimostrare entrambe le direzioni dell'implicazione.

Se B è decidibile ALLORA A è Turing-riconoscibile. =>

Assumiamo che il linguaggio decidibile B esista. Una TM che riconosce A su input x controlla ogni possibile stringa y e controlla se <x, y>∈B. Se y la trova, accetta, altrimenti continua a cercare.

Se A è Turing-riconoscibile ALLORA B è decidibile <=

Nel secondo verso assumiamo che A sia riconosciuto da una TM chiamata M. Definiamo il linguaggio B {<x, y> | M accetta x entro |y| passi}. Il linguaggio B è decidibile dal momento che M può eseguire al massimo per |y| passi e accetta se e solo se M ha accettato. Se x∈A, allora M accetta x entro un numero di passi, quindi <x, y>∈B per un qualche sufficientemente lungo y, ma se x !∈ A, allora <x, y> !∈ A per qualsiasi y.

26. A regolare

1. If $A \leq_m B$ and B is a regular language, does that imply that A is a regular language?

   Answer: No. For example, define the languages $A = \{0^n1^n \mid n \geq 0\}$ and $B = \{1\}$, both over the alphabet $\Sigma = \{0, 1\}$. Define the function $f : \Sigma^* \to \Sigma^*$ as

   $$f(w) = \begin{cases} 1 & \text{if } w \in A, \\ 0 & \text{if } w \notin A. \end{cases}$$

   Observe that A is a context-free language, so it is also Turing-decidable. Thus, f is a computable function. Also, $w \in A$ if and only if $f(w) = 1$, which is true if and only if $f(w) \in B$. Hence, $A \leq_m B$. Language A is nonregular, but B is regular since it is finite.

Forniamo un esempio. Sia A = $\{0^n1^n \mid n \geq 0\}$ e B = {1}, tutti e due sull'alfabeto ∑ = {0, 1}. Definiamo la funzione f: ∑* → ∑* così:

f(input) = {1 se w∈A, 0 se w!∈A}

Osserviamo che A è un linguaggio context free e quindi anche Turing-Decidibile. F è una funzione di riduzione. w∈A se e solo se f(w) = 1, che è vero se e solo se f(w)∈B, Quindi A funzione riduzione di B. Linguaggio A è non regolare ma B è regolare perché è un linguaggio finito.

27. Uguale alla 24

28. J non Turing-riconoscibile

29. Uguale alla 28

30. Circuito Hamiltoniano/Toniano/quasi Hamiltoniano

31. SetPartitioning e SubsetSum

SetPartitioning è un problema NP perché si riduce a SubsetSum.
Possiamo costruire una TM che calcoli t (la somma dei numeri di uno dei sottoinsiemi S_1 o S_2) sommando tutti i numeri in S e dividendo il totale per 2 (se la somma è dispari allora non è possibile dividere questo insieme in 2 parti e la TM rifiuta). A questo punto la TM dovrà dare in pasto a SubsetSum<S, t>, il primo insieme S_1 sarà composto da S' ritornato da SubsetSum, mentre i numeri rimanenti faranno parte di S_2.

32. QuasiPartitioning

5. Il problema SetPartitioning chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2. Sappiamo che questo problema è NP-completo.

Considerate la seguente variante del problema, che chiameremo QuasiPartitioning: dato un insieme di numeri interi S, stabilire se può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 *meno 1*.

(a) Dimostrare che il problema QuasiPartitioning è in NP fornendo un certificato per il *Sì* che si può verificare in tempo polinomiale.

Il certificato è dato da una coppia di insiemi di numeri interi S_1, S_2. Per verificarlo occorre controllare che rispetti le seguenti condizioni:

- i due insiemi $S1, S_2$ devono essere una partizione dell'insieme S;
- la somma degli elementi in S_1 deve essere uguale alla somma degli elementi in S_2 meno 1.

Entrambe le condizioni si possono verificare in tempo polinomiale.

(b) Dimostrare che il problema QuasiPartitioning è NP-hard, mostrando come si può risolvere il problema SetPartitioning usando il problema QuasiPartitioning come sottoprocedura.

Dimostrare che QuasiPartitioning è NP-hard, usando SetPartitioning come problema di riferimento richiede diversi passaggi:

1. Descrivere un algoritmo per risolvere SetPartitioning usando QuasiPartitioning come subroutine. Questo algoritmo avrà la seguente forma: data un'istanza di SetPartitioning, trasformala in un'istanza di QuasiPartitioning, quindi chiama l'algoritmo magico black-box per QuasiPartitioning.
2. Dimostrare che la riduzione è corretta. Ciò richiede sempre due passaggi separati, che di solito hanno la seguente forma:
   - Dimostrare che l'algoritmo trasforma istanze "buone" di SetPartitioning in istanze "buone" di QuasiPartitioning.
   - Dimostrare che se la trasformazione produce un'istanza "buona" di QuasiPartitioning, allora era partita da un'istanza "buona" di SetPartitioning.
3. Mostrare che la riduzione funziona in tempo polinomiale, a meno della chiamata (o delle chiamate) all'algoritmo magico black-box per QuasiPartitioning. (Questo di solito è banale.)

Una istanza di SetPartitioning è data da un insieme S di numeri interi da suddividere in due. Una istanza di QuasiPartitioning è data anch'essa da un insieme di numeri interi S'. Quindi la riduzione deve trasformare un insieme di numeri S input di SetPartitioning in un altro insieme di numeri S' che diventerà l'input per la black-box che risolve QuasiPartitioning.

Come primo tentativo usiamo una riduzione che crea S' aggiungendo un nuovo elemento a S di valore 1, e proviamo a dimostrare che la riduzione è corretta:

$\Rightarrow$ sia S un'istanza buona di SetPartitioning. Allora è possibile partizionare S in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2. Se aggiungiamo il nuovo valore 1 ad S_1 otteniamo una soluzione per QuasiPartitioning, e abbiamo dimostrato che S' è una istanza buona di QuasiPartitioning.

$\Leftarrow$ sia S' un'istanza buona di QuasiPartitioning. Allora è possibile partizionare S' in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 meno 1. Controlliamo quale dei due sottoinsiemi contiene il nuovo elemento 1 aggiunto dalla riduzione:

- se $1 \in S_1$, allora se tolgo 1 da S_1 la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2. Abbiamo trovato una soluzione per SetPartitioning con input S.
- se $1 \in S_2$, allora se tolgo 1 da S_2 quello che succede è che la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2 meno 2. In questo caso abbiamo un

problema perché quello che otteniamo *non è una soluzione di* SETPARTITIONING*!*

Quindi il primo tentativo di riduzione non funziona: ci sono dei casi in cui istanze cattive di SETPARTITIONING diventano istanze buone di QUASIPARTITIONING: per esempio, l'insieme $S = \{2, 4\}$, che non ha soluzione per SETPARTITIONING, diventa $S' = \{2, 4, 1\}$ dopo l'aggiunta dell'1, che ha soluzione per QUASIPARTITIONING: basta dividerlo in $S_1 = \{4\}$ e $S_2 = \{2, 1\}$.

Dobbiamo quindi trovare un modo per "forzare" l'elemento 1 aggiuntivo ad appartenere ad S_1 nella soluzione di QUASIPARTITIONING. Per far questo basta modificare la riduzione in modo che S' contenga tutti gli elementi di S *moltiplicati per* 3, oltre all'1 aggiuntivo. Formalmente:

$$S' = \{3x \mid x \in S\} \cup \{1\}.$$

Proviamo a dimostrare che la nuova riduzione è corretta:

- ⇒ sia S un'istanza buona di SETPARTITIONING. Allora è possibile partizionare S in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2. Se moltiplichiamo per 3 gli elementi di S_1 ed S_2, ed aggiungiamo il nuovo valore 1 ad S_1 otteniamo una soluzione per QUASIPARTITIONING, e abbiamo dimostrato che S' è una istanza buona di QUASIPARTITIONING.
- ⇐ sia S' un'istanza buona di QUASIPARTITIONING. Allora è possibile partizionare S' in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 meno 1. Vediamo adesso che, a differenza della riduzione precedente, non è possibile che $1 \in S_2$: se così fosse, allora se tolgo 1 da S_2 quello che succede è che la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2 meno 2. Tuttavia, gli elementi che stanno in S_1 ed S_2 sono tutti quanti multipli di 3 (tranne l'1 aggiuntivo). Non è possibile che due insiemi che contengono solo multipli di 3 abbiano differenza 2. Quindi l'1 aggiuntivo non può appartenere a S_2 e deve appartenere per forza a S_1. Come visto prima, se tolgo 1 da S_1 la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2. Abbiamo trovato una soluzione per SETPARTITIONING con input S.

In questo caso la riduzione è corretta. Per completare la dimostrazione basta osservare che per costruire S' dobbiamo moltiplicare per 3 gli n elementi di S ed aggiungere un nuovo elemento. Tutte operazioni che si fanno in tempo polinomiale.

33. k-COLOR

(a) Per dimostrare che il problema è in NP va dimostrato che è possibile avere un verificatore del problema che lavora in tempo polinomiale. Questo è possibile perché dato un grafo di dimensioni finite, basterà passare tutti i nodi del grafo e comparare il loro colore a quello dei nodi adiacenti. Se per nessun nodo il colore sarà lo stesso dei nodi adiacenti, allora il verificatore accetterà, altrimenti rifiuterà.

(b) …

(c) 3-COLOR è riducibile a k-COLOR con k ≥ 3 assegnando più volte i 3 colori ai k colori?

(d) Per dimostrare che ALMOST3COLOR è NP-completo, si deve dimostrare che è sia NP che NP-HARD.
È NP perché esiste un certificato del sì che si può verificare in tempo polinomiale.
Il certificato è dato da una coppia di insiemi di vertici e di archi.
Per verificarlo, occorre controllare che:
1. Il numero di archi, i cui vertici sono di colore uguale, sia al più 8939

2. Il numero di colori usati in tutto il grafo sia 3
Entrambe le condizioni si possono verificare in tempo polinomiale.
È NP-HARD, perché si può mostrare come si può risolvere il problema 3COLOR usando il problema ALMOST3COLOR come sotto procedura.
1- ALGORITMO
Dato un grafo G (che è istanza di 3COLOR), costruiamo in tempo polinomiale un nuovo grafo, che è istanza di ALMOST3COLOR, aggiungendo 8939 x (2 vertici dello stesso colore e 1 arco che li collega).
Il nuovo grafo contiene 8939 coppie di vertici uguali.
2- DIMOSTRAZIONE
=>Sia S un'istanza buona di 3COLOR.
Allora, è possibile aggiungere 8939 coppie di vertici uguali, collegati da un arco (2 a 2). La parte del grafo senza colori uguali adiacenti rimane intatta (S). Vengono soddisfatti 3COLOR e ALMOST3COLOR.
<= Sia S' un'istanza buona di ALMOST3COLOR. Una volta individuate tutte le coppie di vertici, si devono contare quali sono i colori adiacenti uguali. Dato che le coppie con colori uguali sono state aggiunte a quelle diverse, quelle rimanenti sono 3 COLOR.
3- TEMPI POLINOMIALI
Dato che la costruzione del nuovo grafo implica l'aggiunta di 8939 coppie di vertici (collegate da un arco), il tempo richiesto è polinomiale

Definizioni da sapere (incompleto)

- Complementare di un linguaggio
- Pumping lemma
- Verificatore di un problema NP

Crediti

Vittorio Santagiuliana, Stefano Lazzaroni, Valton Tahiraj, Mirko Stella, Daniele Giachetto, Marco Canovese, Andrea Breggion, Damiano Bertoldo, Martina Garon & many others.

Chiunque volesse completare/migliorare/correggere il documento si senta libero di farlo.

Link al documento

https://unipdit-my.sharepoint.com/:w:/g/personal/vittorio_santagiuliana_studenti_unipd_it/EfgpaHJ31kpBit4C7WzhoSEB_sPmQCxM2AMHNrJ0CqBg4g?e=Kgm82h