Argomenti trattati durante la lezione:

- (Continuazione) Riduzione mediante funzione
- Classe di complessità P.

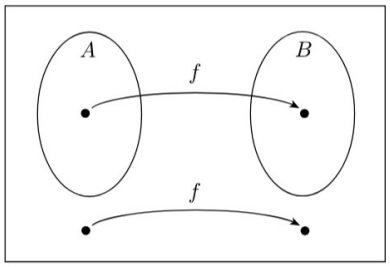
DEFINITION 5.20

Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f: \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***reduction*** from A to B.

The following figure illustrates mapping reducibility.



EXAMPLE 5.24

In Theorem 5.1 we used a reduction from A_{TM} to prove that $HALT_{\mathsf{TM}}$ is undecidable. This reduction showed how a decider for $HALT_{\mathsf{TM}}$ could be used to give a decider for A_{TM}. We can demonstrate a mapping reducibility from A_{TM} to $HALT_{\mathsf{TM}}$ as follows. To do so, we must present a computable function f that takes input of the form $\langle M, w\rangle$ and returns output of the form $\langle M', w'\rangle$, where

$$\langle M, w\rangle \in A_{\mathsf{TM}} \text{ if and only if } \langle M', w'\rangle \in HALT_{\mathsf{TM}}.$$

The following machine F computes a reduction f.

F = "On input $\langle M, w\rangle$:

1. Construct the following machine M'.
   M' = "On input x:
   1. Run M on x.
   2. If M accepts, *accept*.
   3. If M rejects, enter a loop."
2. Output $\langle M', w\rangle$."

THEOREM 5.22

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

N = "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$, then $f(w) \in B$ because f is a reduction from A to B. Thus, M accepts $f(w)$ whenever $w \in A$. Therefore, N works as desired.

The following corollary of Theorem 5.22 has been our main tool for proving undecidability.

COROLLARY 5.23

If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Now we revisit some of our earlier proofs that used the reducibility method to get examples of mapping reducibilities.

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

PROOF First we show that EQ_{TM} is not Turing-recognizable. We do so by showing that A_{TM} is reducible to $\overline{EQ_{\mathsf{TM}}}$. The reducing function f works as follows.

F = "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.
   M_1 = "On any input:
   1. *Reject*."
   M_2 = "On any input:
   1. Run M on w. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

Here, M_1 accepts nothing. If M accepts w, M_2 accepts everything, and so the two machines are not equivalent. Conversely, if M doesn't accept w, M_2 accepts nothing, and they are equivalent. Thus f reduces A_{TM} to $\overline{EQ_{\mathsf{TM}}}$, as desired.

To show that $\overline{EQ_{\mathsf{TM}}}$ is not Turing-recognizable, we give a reduction from A_{TM} to the complement of $\overline{EQ_{\mathsf{TM}}}$—namely, EQ_{TM}. Hence we show that $A_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$. The following TM G computes the reducing function g.

Facciamo subito esempi simili a quelli che avete sulle slide…

(File → Esercizi risolti: Da 5.22 a 5.25)

Passiamo ora ai prossimi set di slide.

- Sia M una TM deterministica che si ferma su tutti gli input.
- Il tempo di esecuzione (o complessità di tempo) di M è la funzione $f : \mathbb{N} \mapsto \mathbb{N}$ tale che $f(n)$ è il numero massimo di passi che M utilizza su un input di lunghezza n.
- Se $f(n)$ è il tempo di esecuzione di M, diciamo che M è una TM di tempo $f(n)$.
- Useremo n per rappresentare la lunghezza dell'input.
- Ci interesseremo dell'analisi del caso pessimo.

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

Let $t: \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, $\mathbf{TIME}(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

It shows that $A \in \text{TIME}(n \log n)$.

M_2 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

We can decide the language A in $O(n)$ time (also called ***linear time***) if the Turing machine has a second tape. The following two-tape TM M_3 decides A in linear time. Machine M_3 operates differently from the previous machines for A. It simply copies the 0s to its second tape and then matches them against the 1s.

M_3 = "On input string w:

1. Scan across tape 1 and *reject* if a 0 is found to the right of a 1.
2. Scan across the 0s on tape 1 until the first 1. At the same time, copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*.
4. If all the 0s have now been crossed off, *accept*. If any 0s remain, *reject*."

Possiamo fare di meglio?

- Dall'analisi che abbiamo fatto, sappiamo che

$$A = \{0^k 1^k \mid k \geq 0\}$$

appartiene alla classe di complessità di tempo $\text{TIME}(n^2)$, perché M_1 decide A in tempo $O(n^2)$

MA

- Esiste una macchina che decide A in modo asintoticamente più veloce?

Miglioriamo M_1 (cont.)

Idea: usiamo il secondo nastro per contare 0 e 1

M_2 = "Su input w:

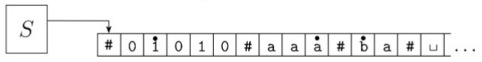
1. Scorre il nastro 1 e rifiuta se trova uno 0 a destra di un 1
2. Scorre i simboli 0 sul nastro 1 fino al primo 1. Contemporaneamente, copia ogni 0 sul nastro 2.
3. Scorre i simboli 1 sul nastro 1 fino alla fine dell'input. Per ogni 1 letto sul nastro 1, cancella uno 0 sul nastro 2. Se ogni 0 è stato cancellato prima di aver letto tutti gli 1, rifiuta.
4. Se tutti gli 0 sono stati cancellati, accetta. Se rimane qualche 0, rifiuta."

Singolo nastro vs. Multinastro

Theorem

Sia $t(n)$ una funzione tale che $t(n) \leq n$. Ogni TM multinastro di tempo $t(n)$ ammette una TM equivalente a nastro singolo di tempo $O(t^2(n))$.

- Ricordiamo la dimostrazione di come convertire una TM da multinastro a nastro singolo.



- Dobbiamo determinare quanto tempo ci vuole per simulare ogni passo della macchina multinastro sulla TM a nastro singolo.

THEOREM 7.8

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Let N be a nondeterministic Turing machine that is a decider. The ***running time*** of N is the function $f: \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n, as shown in the following figure.

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

The class P plays a central role in our theory and is important because

1. P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing machine, and
2. P roughly corresponds to the class of problems that are realistically solvable on a computer.

Per dimostrare che un problema/algoritmo è in P:

- Descrivi l'algoritmo per fasi numerate
- Dai un limite superiore polinomiale al numero di fasi che l'algoritmo esegue per un input di lunghezza n
- Assicurati che ogni fase possa essere completata in tempo polinomiale su un modello di calcolo deterministico ragionevole
- L'input deve essere codificato in modo ragionevole

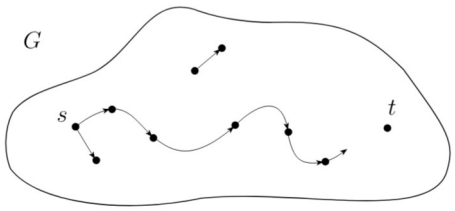
Raggiungibilità in un grafo

$PATH = \{\langle G, s, t\rangle \mid G$ grafo che contiene un cammino da s a $t\}$

Numeri relativamente primi

$RELPRIME = \{\langle x, y\rangle \mid 1$ è il massimo comun divisore di x e $y\}$

$PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$.



PROOF A polynomial time algorithm M for *PATH* operates as follows.

$M =$ "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

Now we analyze this algorithm to show that it runs in polynomial time. Obviously, stages 1 and 4 are executed only once. Stage 3 runs at most m times because each time except the last it marks an additional node in G. Thus, the total number of stages used is at most $1 + 1 + m$, giving a polynomial in the size of G.

Stages 1 and 4 of M are easily implemented in polynomial time on any reasonable deterministic model. Stage 3 involves a scan of the input and a test of whether certain nodes are marked, which also is easily implemented in polynomial time. Hence M is a polynomial time algorithm for *PATH*.

both are divisible by 2. Let *RELPRIME* be the problem of testing whether two numbers are relatively prime. Thus

$$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}.$$

Instead, we solve this problem with an ancient numerical procedure, called the ***Euclidean algorithm***, for computing the greatest common divisor. The ***greatest common divisor*** of natural numbers x and y, written $\gcd(x, y)$, is the largest integer that evenly divides both x and y. For example, $\gcd(18, 24) = 6$. Obviously, x and y are relatively prime iff $\gcd(x, y) = 1$. We describe the Euclidean algorithm as algorithm E in the proof. It uses the mod function, where $x \bmod y$ is the remainder after the integer division of x by y.

THEOREM 7.15

RELPRIME $\in$ P.

PROOF The Euclidean algorithm E is as follows.

$E =$ "On input $\langle x, y\rangle$, where x and y are natural numbers in binary:

1. Repeat until $y = 0$:
2. Assign $x \leftarrow x \bmod y$.
3. Exchange x and y.
4. Output x."

Algorithm R solves *RELPRIME*, using E as a subroutine.

$R =$ "On input $\langle x, y\rangle$, where x and y are natural numbers in binary:

1. Run E on $\langle x, y\rangle$.
2. If the result is 1, *accept*. Otherwise, *reject*."

Vediamo subito alcuni esempi…

4. A **triangle** in an undirected graph is a 3-clique. Show that $TRIANGLE \in$ P, where $TRIANGLE = \{\langle G\rangle \mid G \text{ contains a triangle}\}$.

   Answer: Let $G = (V, E)$ be a graph with a set V of vertices and a set E of edges. We enumerate all triples (u, v, w) with vertices $u, v, w \in V$ and $u < v < w$, and then check whether or not all three edges (u, v), (v, w) and (u, w) exist in E. Enumeration of all triples requires $O(|V|^3)$ time. Checking whether or not all three edges belong to E takes $O(|E|)$ time. Thus, the overall time is $O(|V|^3\,|E|)$, which is polynomial in the length of the input $\langle G\rangle$. Therefore, $TRIANGLE \in$ P.

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

7. **Exercise 7.10** Show that ALL_{DFA} is in P.

   Solution: ALL_{DFA} is decided by the following deterministic Turing machine.

   $T =$ "On input $\langle M\rangle$ where M is a DFA

   1. Mark the start state of M.
   2. Repeat until no new states are marked.
   3. Mark any state that has a transition coming into it from a marked state.
   4. If every marked state is an accept state, *accept*.

   Steps 1 and 4 are done once. Each time through the loop in Steps 2 and 3 except the last time, an unmarked state is marked, so the loop is executed no more than m times, where m is the number of states in M. Thus, the total number of steps executed is no more than $2m + 2$ and this is polynomial in the size of $\langle M\rangle$.

   Step 1 involves finding the start state in the list of states and marking it. Step 3 involves processing each transition once and for each transition checking if it goes from a marked state to an unmarked state. Step 4 involves checking if each marked state is in the list of accept states. Each of these steps can be implemented in time polynomial in the size of $\langle M\rangle$, so the algorithm runs in polynomial time.

Let $MODEXP = \{\langle a, b, c, p\rangle \mid a, b, c, \text{ and } p \text{ are positive binary integers such that } a^b = c \pmod p\}$

Show that $MODEXP \in$P

Because b is a binary integer, let $b = b_1b_2...b_{n-1}b_n$. So the decimal representation of b is $\sum_{k=1}^{n} b_{n-k+1}2^{k-1}$. By the hint, we observe that $a^{(10)_2} = a^2$ and $a^{(1000)_2} = ((a^2)^2)^2$.

Construct the following algorithm to decide $MODEXP$:

A = "On input $\langle a, b, c, p\rangle$ where a, b, c, and p are positive binary integers:
1. Let $T = 1$ (and $n = \lceil log_2 b\rceil$).
2. For $i = 1$ to n,
   a. if $b_i = 1$, $T = (a(T^2) \pmod p)$,
   b. if $b_i = 0$, $T = (T^2 \pmod p)$;
3. Return $T \pmod p$.
4. If $T = c \pmod p$, accept; otherwise reject."

Assume that a, b, c and p are at most m bits (so $n \leq m$). It is known that two m-bit numbers cost $O(m^2)$ unit time to do multiplication and division (and hence modular), so each i costs $O(m^2)$ time. The total cost of the for loop is $O(m^2) \times n = O(m^3)$, which is the dominant cost of all steps. The time complexity of the algorithm A is polynomial in the length of its input.

Passiamo ora ad altre categorie di esercizio…

1. *Una* macchina di Turing bidimensionale *utilizza una griglia bidimensionale infinita di celle come nastro. Ad ogni transizione, la testina può spostarsi dalla cella corrente ad una qualsiasi delle quattro celle adiacenti. La funzione di transizione di tale macchina ha la forma*

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\uparrow, \downarrow, \rightarrow, \leftarrow\},$$

dove le frecce indicano in quale direzione si muove la testina dopo aver scritto il simbolo sulla cella corrente.

Dimostra che ogni macchina di Turing bidimensionale può essere simulata da una macchina di Turing deterministica a nastro singolo.

$S =$ "su input w:

1. Sostituisce w con la configurazione iniziale $\#\#w\#\#$ e marca con $\widehat{\ }$ il primo simbolo di w.
2. Scorre il nastro finché non trova la cella marcata con $\widehat{\ }$.
3. Aggiorna il nastro in accordo con la funzione di transizione di B:
   - Se $\delta(r, a) = (s, b, \rightarrow)$, scrive b non marcato sulla cella corrente, sposta $\widehat{\ }$ sulla cella immediatamente a destra. Poi sposta di una cella a destra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un $\#$, S scrive un blank marcato al posto del $\#$ e sposta il contenuto del nastro da questa cella fino al $\#\#$ finale, di una cella più a destra.
   - Se $\delta(r, a) = (s, b, \leftarrow)$, scrive b non marcato sulla cella corrente, sposta $\widehat{\ }$ sulla cella immediatamente a sinistra. Poi sposta di una cella a sinistra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un $\#$, S scrive un blank marcato al posto del $\#$ e sposta il contenuto del nastro da questa cella fino al $\#\#$ iniziale, di una cella più a sinistra.
   - Se $\delta(r, a) = (s, b, \uparrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\widehat{\ }$ sulla prima cella marcata con un pallino posta a sinistra della cella corrente. Se questa cella marcata non esiste, aggiunge una nuova riga composta solo da blank all'inizio della configurazione.
   - Se $\delta(r, a) = (s, b, \downarrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\widehat{\ }$ sulla prima cella marcata con un pallino posta a destra della cella corrente. Se questa cella non esiste, aggiunge una nuova riga composta solo da blank alla fine della configurazione.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di B, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di B allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

1. **(12 punti)** Una *Macchina di Turing con eliminazione* è una macchina di Turing deterministica a nastro singolo che può eliminare celle dal nastro. Formalmente la funzione di transizione è definita come

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, D\}$$

dove L, R indicano i normali spostamenti a sinistra e a destra della testina, e D indica l'eliminazione della cella sotto la posizione corrente della testina. Dopo una operazione di eliminazione, la testina si muove nella cella che si trovava immediatamente a destra della cella eliminata.

Dimostra che qualsiasi macchina di Turing con eliminazione può essere simulata da una macchina di Turing deterministica a nastro singolo.

Soluzione. Mostriamo come convertire una macchina di Turing con Inserimento M in una TM deterministica a nastro singolo S equivalente. La simulazione usa il simbolo speciale # per segnare le celle che vengono eliminate.

$S =$ "Su input w:

1. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, L)$ procede come nella TM standard: S scrive b sul nastro e muove la testina di una cella a sinistra. Se lo spostamento a sinistra porta la testina sopra un #, allora continua a spostare la testina a sinistra finché non si arriva in una cella che contiene un simbolo diverso dal #.
2. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, R)$ procede come nella TM standard: S scrive b sul nastro e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra un #, allora continua a spostare la testina a destra finché non si arriva in una cella che contiene un simbolo diverso dal #.
3. Per simulare una mossa del tipo $\delta(q, a) = (r, b, D)$ la TM S scrive # nella cella corrente e e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra un #, allora continua a spostare la testina a destra finché non si arriva in una cella che contiene un simbolo diverso dal #.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M, allora S termina con accettazione. Se in qualsiasi momento la simulazione raggiunge lo stato di rifiuto di M, allora S termina con rifiuto. Negli altri casi continua la simulazione dal punto 2."

2. **(12 punti)** Data una Turing Machine M, definiamo

$$\text{HALTS}(M) = \{w \mid M \text{ termina la computazione su } w\}.$$

Considera il linguaggio

$$F = \{\langle M \rangle \mid \text{HALTS}(M) \text{ è un insieme finito}\}.$$

Dimostra che F è indecidibile.

Soluzione. La seguente macchina G calcola una riduzione $\overline{A_{TM}} \leq_m F$:

$G =$ "su input $\langle M, w \rangle$, dove M è una TM e w una stringa:

1. Costruisci la seguente macchina M':

   $M' =$ "Su input x:
   1. Esegue M su input w.
   2. Se M accetta, *accetta*.
   3. Se M rifiuta, va in loop."

2. Ritorna $\langle M' \rangle$."

Mostriamo che G calcola una funzione di riduzione g da $\overline{A_{TM}}$ a F, cioè una funzione tale che

$$\langle M, w \rangle \in \overline{A_{TM}} \text{ se e solo se } M' \in F.$$

- Se $\langle M, w\rangle \in \overline{A_{TM}}$ allora la macchina M su input w rifiuta oppure va in loop. In entrambi i casi la macchina M' va in loop su tutte le stringhe, quindi $\text{HALTS}(M) = \emptyset$ che è un insieme finito. Di conseguenza $M' \in F$.
- Viceversa, se $\langle M, w\rangle \notin \overline{A_{TM}}$ allora la macchina M accetta w. In questo caso la macchina M' accetta tutte le parole, quindi $\text{HALTS}(M) = \Sigma^*$ che è un insieme infinito. Di conseguenza $M' \notin F$.

2. (12 punti) Data una Turing Machine M, definiamo

$$\text{HALTS}(M) = \{w \mid M \text{ termina la computazione su } w\}.$$

Considera il linguaggio

$$I = \{\langle M\rangle \mid \text{HALTS}(M) \text{ è un insieme infinito}\}.$$

Dimostra che I è indecidibile.

Soluzione. La seguente macchina F calcola una riduzione mediante funzione $A_{TM} \leq_m I$:

$F =$ "su input $\langle M, w\rangle$, dove M è una TM e w una stringa:

1. Costruisci la seguente macchina M':

   $M' =$ "Su input x:
   1. Esegue M su input w.
   2. Se M accetta, *accetta.*
   3. Se M rifiuta, va in loop."

2. Ritorna $\langle M'\rangle$."

Mostriamo che F calcola una funzione di riduzione f da A_{TM} a I, cioè una funzione tale che

$$\langle M, w\rangle \in A_{TM} \text{ se e solo se } M' \in I.$$

- Se $\langle M, w\rangle \in A_{TM}$ allora la macchina M accetta w. In questo caso la macchina M' accetta tutte le parole, quindi $\text{HALTS}(M) = \Sigma^*$ che è un insieme infinito. Di conseguenza $M' \in I$.
- Viceversa, se $\langle M, w\rangle \notin A_{TM}$, allora la macchina M su input w rifiuta oppure va in loop. In entrambi i casi la macchina M' va in loop su tutte le stringhe, quindi $\text{HALTS}(M) = \emptyset$ che è un insieme finito. Di conseguenza $M' \notin I$.

3. (12 punti) Data una Turing Machine M, considera il problema di determinare se esiste un input tale che M scrive "$xyzzy$" su cinque celle adiacenti del nastro. Puoi assumere che l'alfabeto di input di M non contenga i simboli x, y, z.

(a) Formula questo problema come un linguaggio $MAGIC_{\text{TM}}$.

(b) Dimostra che il linguaggio $MAGIC_{\text{TM}}$ è indecidibile.

Soluzione.

(a) $MAGIC_{\text{TM}} = \{\langle M \rangle \mid M \text{ è una TM che scrive } xyzzy \text{ sul nastro per qualche input } w\}$

(b) La seguente macchina F calcola una riduzione $A_{\text{TM}} \leq_m MAGIC_{\text{TM}}$:

$F =$ "su input $\langle M, w \rangle$, dove M è una TM e w una stringa:

1. Verifica che i simboli x, y, z non compaiano in w, né nell'alfabeto di input o nell'alfabeto del nastro di M. Se vi compaiono, sostituiscili con tre nuovi simboli X, Y, Z nella parola w e nella codifica di M.
2. Costruisci la seguente macchina M':

   $M' =$ "Su input x:
   1. Simula l'esecuzione di M su input w, senza usare i simboli x, y, z.
   2. Se M accetta, scrivi $xyzzy$ sul nastro, altrimenti rifiuta senza modificare il nastro.

3. Ritorna $\langle M' \rangle$."

Mostriamo che F calcola una funzione di riduzione da A_{TM} a $MAGIC_{\text{TM}}$, cioè una funzione tale che

$$\langle M, w \rangle \in A_{\text{TM}} \text{ se e solo se } \langle M' \rangle \in MAGIC_{\text{TM}}.$$

- Se $\langle M, w \rangle \in A_{\text{TM}}$ allora la macchina M accetta w. In questo caso la macchina M' scrive $xyzzy$ sul nastro per tutti gli input. Di conseguenza $\langle M' \rangle \in MAGIC_{\text{TM}}$.
- Viceversa, se $\langle M, w \rangle \notin A_{TM}$ allora la macchina M rifiuta o va in loop su w. Per tutti gli input, la macchina M' simula l'esecuzione di M su w senza usare i simboli x, y, z (perché sono stati tolti dalla definizione di M e di w se vi comparivano), e rifiuta o va in loop senza scrivere mai $xyzzy$ sul nastro. Di conseguenza $\langle M' \rangle \notin MAGIC_{\text{TM}}$.

2. (12 punti) Considera il seguente problema: dato un DFA D e un'espressione regolare R, il linguaggio riconosciuto da D è uguale al linguaggio generato da R?

(a) Formula questo problema come un linguaggio $EQ_{\text{DFA,REX}}$.

(b) Dimostra che $EQ_{\text{DFA,REX}}$ è decidibile.

Soluzione.

(a) $EQ_{\text{DFA,REX}} = \{\langle D, R \rangle \mid D \text{ è un DFA, } R \text{ è una espressione regolare e } L(D) = L(R)\}$

(b) La seguente macchina N usa la Turing machine M che decide EQ_{DFA} per decidere $EQ_{\text{DFA,REX}}$:

$N =$ "su input $\langle D, R \rangle$, dove D è un DFA e R una espressione regolare:

1. Converti R in un DFA equivalente D_R
2. Esegui M su input $\langle D, D_R \rangle$, e ritorna lo stesso risultato di M."

Mostriamo che N è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Sappiamo che esiste un algoritmo per convertire ogni espressione regolare in un ε-NFA, ed un algoritmo per convertire ogni ε-NFA in un DFA. Il primo step di N si implementa eseguendo i due algoritmi in sequenza, e termina sempre perché entrambi gli algoritmi di conversione terminano. Il secondo step termina sempre perché sappiamo che EQ_{DFA} è un linguaggio decidibile. Quindi N termina sempre la computazione.

Vediamo ora che N dà la risposta corretta:

- Se $\langle D, R \rangle \in EQ_{\text{DFA,REX}}$ allora $L(D) = L(R)$, e di conseguenza $L(D) = L(D_R)$ perché D_R è un DFA equivalente ad R. Quindi $\langle D, D_R \rangle \in EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con accettazione. N ritorna lo stesso risultato di M, quindi accetta.
- Viceversa, se $\langle D, R \rangle \notin EQ_{\text{DFA,REX}}$ allora $L(D) \neq L(R)$, e di conseguenza $L(D) \neq L(D_R)$ perché D_R è un DFA equivalente ad R. Quindi $\langle D, D_R \rangle \notin EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con rifiuto. N ritorna lo stesso risultato di M, quindi rifiuta.

2. (12 punti) Dati due DFA, considera il problema di determinare se esiste una stringa accettata da entrambi.

(a) Formula questo problema come un linguaggio $AGREE_{\text{DFA}}$.

(b) Dimostra che $AGREE_{\text{DFA}}$ è decidibile.

Soluzione.

(a) $AGREE_{\text{DFA}} = \{\langle A, B\rangle \mid A, B \text{ sono DFA, ed esiste una parola } w \text{ tale che } w \in L(A) \text{ e } w \in L(B)\}$

(b) La seguente macchina N usa la Turing machine M che decide E_{DFA} per decidere $AGREE_{\text{DFA}}$:

N = "su input $\langle A, B\rangle$, dove A, B sono DFA:

1. Costruisci il DFA C che accetta l'intersezione dei linguaggi di A e B
2. Esegui M su input $\langle C\rangle$. Se M accetta, rifiuta, se M rifiuta, accetta."

Mostriamo che N è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Sappiamo che esiste un algoritmo per costruire l'intersezione di due DFA. Il primo step di N si implementa eseguendo questo algoritmo, e termina sempre perché la costruzione dell'intersezione termina. Il secondo step termina sempre perché sappiamo che E_{DFA} è un linguaggio decidibile. Quindi N termina sempre la computazione.

Vediamo ora che N dà la risposta corretta:

- Se $\langle A, B\rangle \in AGREE_{\text{DFA}}$ allora esiste una parola che viene accettata sia da A che da B, e quindi il linguaggio $L(A) \cap L(B)$ non può essere vuoto. Quindi $\langle C\rangle \notin E_{\text{DFA}}$, e l'esecuzione di M terminerà con rifiuto. N ritorna l'opposto di M, quindi accetta.
- Viceversa, se $\langle A, B\rangle \notin AGREE_{\text{DFA}}$ allora non esiste una parola che sia accettata sia da A che da B, e quindi il linguaggio $L(A) \cap L(B)$ è vuoto. Quindi $\langle C\rangle \in E_{\text{DFA}}$, e l'esecuzione di M terminerà con accettazione. N ritorna l'opposto di M, quindi rifiuta.