Argomenti trattati durante la lezione:

- Il metodo di diagonalizzazione
- Linguaggi non decidibili, non Turing-riconoscibili e riduzione mediante funzione

There is a specific problem that is algorithmically unsolvable. Computers appear to be so powerful that you may believe that all problems will eventually yield to them.

given input string. We call it A_{TM} by analogy with A_{DFA} and A_{CFG}. But, whereas A_{DFA} and A_{CFG} were decidable, A_{TM} is not. Let

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

THEOREM 4.11

A_{TM} is undecidable.

Before we get to the proof, let's first observe that A_{TM} is Turing-recognizable. Thus, this theorem shows that recognizers *are* more powerful than deciders. Requiring a TM to halt on all inputs restricts the kinds of languages that it can recognize. The following Turing machine U recognizes A_{TM}.

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

1. Simulate M on input w.
2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *reject*."

Note that this machine loops on input $\langle M, w\rangle$ if M loops on w, which is why this machine does not decide A_{TM}. If the algorithm had some way to determine that M was not halting on w, it could *reject* in this case. However, an algorithm has no way to make this determination, as we shall see.

The Turing machine U is interesting in its own right. It is an example of the *universal Turing machine* first proposed by Alan Turing in 1936. This machine is called universal because it is capable of simulating any other Turing machine from the description of that machine. The universal Turing machine played an important early role in the development of stored-program computers.

The proof of the undecidability of A TM uses a technique called diagonalization, discovered by mathematician Georg Cantor in 1873.

A set A is countable if either it is finite or it has the same size as N .

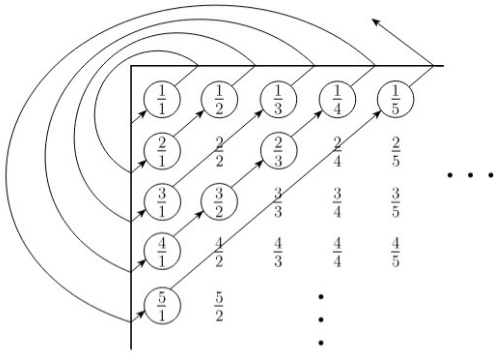


FIGURE 4.16
A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

After seeing the correspondence of $\mathcal{N}$ and $\mathcal{Q}$, you might think that any two infinite sets can be shown to have the same size. After all, you need only demonstrate a correspondence, and this example shows that surprising correspondences do exist. However, for some infinite sets, no correspondence with $\mathcal{N}$ exists. These sets are simply too big. Such sets are called ***uncountable***.

- L'insieme di tutte le macchine di Turing è numerabile
- L'insieme di tutti i linguaggi è non numerabile
- Devono esistere linguaggi non riconoscibili da una macchina di Turing

Teorema: A_{TM} è indecidibile

- Definiamo una TM D che usa H come subroutine
- D = "Su input $\langle M\rangle$, dove M è una TM:
  1. Esegue H su input $\langle M, \langle M\rangle\rangle$
  2. Dà in output l'opposto dell'output di H. Se H accetta, rifiuta; se H rifiuta, accetta."
- Cosa fa D con input $\langle D\rangle$?

$$D(\langle D\rangle) = \begin{cases} \text{accetta} & \text{se } D \text{ non accetta } \langle D\rangle \\ \text{rifiuta} & \text{se } D \text{ accetta } \langle D\rangle \end{cases}$$

- Contraddizione!

1. H accetta $\langle M, w\rangle$ esattamente quando M accetta w
   a. Banale: abbiamo assunto che H esista e decida A_{TM}
   b. M rappresenta qualsiasi TM e w è una qualsiasi stringa
2. D rifiuta $\langle M\rangle$ esattamente quando M accetta $\langle M\rangle$
   a. Cosa è successo a w?
   b. w è solo una stringa, come $\langle M\rangle$. Tutto ciò che stiamo facendo è definire quale stringa dare in input alla macchina.
3. D rifiuta $\langle D\rangle$ esattamente quando D accetta $\langle D\rangle$
   a. Questa è la contraddizione.
4. Dove si usa la diagonalizzazione?

Where is the diagonalization in the proof of Theorem 4.11? It becomes apparent when you examine tables of behavior for TMs H and D. In these tables we list all TMs down the rows, $M_1, M_2, \ldots$, and all their descriptions across the columns, $\langle M_1 \rangle, \langle M_2 \rangle, \ldots$. The entries tell whether the machine in a given row accepts the input in a given column. The entry is *accept* if the machine accepts the input but is blank if it rejects or loops on that input. We made up the entries in the following figure to illustrate the idea.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$			$\vdots$		

FIGURE 4.19
Entry i, j is *accept* if M_i accepts $\langle M_j \rangle$

In the following figure, we added D to Figure 4.20. By our assumption, H is a TM and so is D. Therefore, it must occur on the list $M_1, M_2, \ldots$ of all TMs. Note that D computes the opposite of the diagonal entries. The contradiction occurs at the point of the question mark where the entry must be the opposite of itself.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$			$\vdots$				$\ddots$

FIGURE 4.21
If D is in the figure, a contradiction occurs at "?"

- Abbiamo visto che A_{TM} è Turing-riconoscibile
- Sappiamo che l'insieme di tutte le TM è numerabile
- Sappiamo che l'insieme di tutti i linguaggi è non numerabile
- Di conseguenza deve esistere un linguaggio non Turing-riconiscibile

- C'è ancora una cosa che dobbiamo fare prima di poter mostrare un linguaggio non Turing-riconoscibile.
- Mostreremo che se un linguaggio e il suo complementare sono Turing-riconoscibili, allora il linguaggio è decidibile.
- Un linguaggio è co-Turing riconoscibile se è il complementare di un linguaggio Turing-riconoscibile

PROOF We have two directions to prove. First, if A is decidable, we can easily see that both A and its complement $\overline{A}$ are Turing-recognizable. Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

For the other direction, if both A and $\overline{A}$ are Turing-recognizable, we let M_1 be the recognizer for A and M_2 be the recognizer for $\overline{A}$. The following Turing machine M is a decider for A.

M = "On input w:

1. Run both M_1 and M_2 on input w in parallel.
2. If M_1 accepts, *accept*; if M_2 accepts, *reject*."

Running the two machines in parallel means that M has two tapes, one for simulating M_1 and the other for simulating M_2. In this case, M takes turns simulating one step of each machine, which continues until one of them accepts.

Now we show that M decides A. Every string w is either in A or $\overline{A}$. Therefore, either M_1 or M_2 must accept w. Because M halts whenever M_1 or M_2 accepts, M always halts and so it is a decider. Furthermore, it accepts all strings in A and rejects all strings not in A. So M is a decider for A, and thus A is decidable.

COROLLARY 4.23

$\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable.

PROOF We know that A_{TM} is Turing-recognizable. If $\overline{A_{\mathsf{TM}}}$ also were Turing-recognizable, A_{TM} would be decidable. Theorem 4.11 tells us that A_{TM} is not decidable, so $\overline{A_{\mathsf{TM}}}$ must not be Turing-recognizable.

(Prima di passare al prossimo set di slide) → Esercizi

A5.10 Consider the problem of determining whether a two-tape Turing machine ever writes a nonblank symbol on its second tape when it is run on input w. Formulate this problem as a language and show that it is undecidable.

5.10 Let $B = \{\langle M, w\rangle | \ M$ is a two-tape TM that writes a nonblank symbol on its second tape when it is run on $w\}$. Show that A_{TM} reduces to B. Assume for the sake of contradiction that TM R decides B. Then construct a TM S that uses R to decide A_{TM}.

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

1. Use M to construct the following two-tape TM T.
   T = "On input x:
   1. Simulate M on x using the first tape.
   2. If the simulation shows that M accepts, write a nonblank symbol on the second tape."
2. Run R on $\langle T, w\rangle$ to determine whether T on input w writes a nonblank symbol on its second tape.
3. If R accepts, M accepts w, so *accept*. Otherwise, *reject*."

A5.11 Consider the problem of determining whether a two-tape Turing machine ever writes a nonblank symbol on its second tape during the course of its computation on any input string. Formulate this problem as a language and show that it is undecidable.

5.11 Let $C = \{\langle M\rangle | \ M$ is a two-tape TM that writes a nonblank symbol on its second tape when it is run on some input$\}$. Show that A_{TM} reduces to C. Assume for the sake of contradiction that TM R decides C. Construct a TM S that uses R to decide A_{TM}.

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

1. Use M and w to construct the following two-tape TM T_w.
   T_w = "On any input:
   1. Simulate M on w using the first tape.
   2. If the simulation shows that M accepts, write a nonblank symbol on the second tape."
2. Run R on $\langle T_w\rangle$ to determine whether T_w ever writes a nonblank symbol on its second tape.
3. If R accepts, M accepts w, so *accept*. Otherwise, *reject*."

(Ora continuiamo)

We have already established the undecidability of A_{TM}, the problem of determining whether a Turing machine accepts a given input. Let's consider a related problem, $HALT_{\mathsf{TM}}$, the problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input. This problem is widely known as the ***halting problem***. We use the undecidability of A_{TM} to prove the undecidability of the halting problem by reducing A_{TM} to $HALT_{\mathsf{TM}}$. Let

$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle | \ M \text{ is a TM and } M \text{ halts on input } w\}.$$

THEOREM 5.1

$HALT_{\mathsf{TM}}$ is undecidable.

Riducibilità

- Una riduzione è un modo per trasformare un problema in un altro problema
- Una soluzione al secondo problema può essere usata per risolvere il primo problema
- Se A è riducibile a B, e B è decidibile, allora A è decidibile
- Se A è riducibile a B, e A è indecidibile, allora A è indecidibile

Sono usate per dimostrare che un problema è indecidibile:

1. Assumi che B sia decidibile
2. Riduci A al problema B
   - costruisci una TM che usa B per risolvere A
3. Se A è indecidibile, allora questa è una contraddizione
4. L'assunzione è sbagliata e B è indecidibile

Instead, use the assumption that you have TM R that decides $HALT_{\mathsf{TM}}$. With R, you can test whether M halts on w. If R indicates that M doesn't halt on w, reject because $\langle M, w\rangle$ isn't in A_{TM}. However, if R indicates that M does halt on w, you can do the simulation without any danger of looping.

Thus, if TM R exists, we can decide A_{TM}, but we know that A_{TM} is undecidable. By virtue of this contradiction, we can conclude that R does not exist. Therefore, $HALT_{\mathsf{TM}}$ is undecidable.

PROOF Let's assume for the purpose of obtaining a contradiction that TM R decides $HALT_{\mathsf{TM}}$. We construct TM S to decide A_{TM}, with S operating as follows.

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

Clearly, if R decides $HALT_{\mathsf{TM}}$, then S decides A_{TM}. Because A_{TM} is undecidable, $HALT_{\mathsf{TM}}$ also must be undecidable.

We now present several other theorems and their proofs as further examples of the reducibility method for proving undecidability. Let

$$E_{\mathsf{TM}} = \{\langle M\rangle | \ M \text{ is a TM and } L(M) = \emptyset\}.$$

Instead of running R on $\langle M \rangle$, we run R on a modification of $\langle M \rangle$. We modify $\langle M \rangle$ to guarantee that M rejects all strings except w, but on input w it works as usual. Then we use R to determine whether the modified machine recognizes the empty language. The only string the machine can now accept is w, so its language will be nonempty iff it accepts w. If R accepts when it is fed a description of the modified machine, we know that the modified machine doesn't accept anything and that M doesn't accept w.

PROOF Let's write the modified machine described in the proof idea using our standard notation. We call it M_1.

M_1 = "On input x:

1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

This machine has the string w as part of its description. It conducts the test of whether $x = w$ in the obvious way, by scanning the input and comparing it character by character with w to determine whether they are the same.

Putting all this together, we assume that TM R decides E_{TM} and construct TM S that decides A_{TM} as follows.

S = "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Use the description of M and w to construct the TM M_1 just described.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

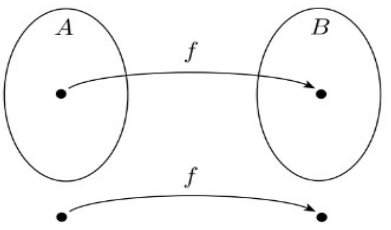
Note that S must actually be able to compute a description of M_1 from a description of M and w. It is able to do so because it only needs to add extra states to M that perform the $x = w$ test.

If R were a decider for E_{TM}, S would be a decider for A_{TM}. A decider for A_{TM} cannot exist, so we know that E_{TM} must be undecidable.

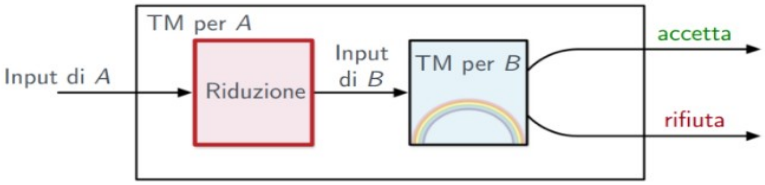
(Same for REGULAR_TM oppure per EQ_TM)

Un linguaggio A è riducibile mediante funzione al linguaggio B ($A \leq_m B$), se esiste una funzione calcolabile $f : \Sigma^* \mapsto \Sigma^*$ tale che

per ogni w : $w \in A$ se e solo se $f(w) \in B$



f è la riduzione da A a B



THEOREM 5.22

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

COROLLARY 5.23

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

(Ora passiamo a vari esercizi)

4. A *useless state* in a Turing machine is one that is never entered on any input string. Consider the problem of determining whether a state in a Turing machine is useless. Formulate this problem as a language and show it is undecidable.

Answer: We define the decision problem with the language

$$USELESS_{\mathrm{TM}} = \{\langle M, q\rangle \mid q \text{ is a useless state in TM } M\}.$$

We show that $USELESS_{\mathrm{TM}}$ is undecidable by reducing E_{TM} to $USELESS_{\mathrm{TM}}$, where $E_{\mathrm{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$. We know E_{TM} is undecidable by Theorem 5.2.

Suppose that $USELESS_{\mathrm{TM}}$ is decidable and that TM R decides it. Note that for any Turing machine M with accept state q_{accept}, q_{accept} is useless if and only if $L(M) = \emptyset$. Thus, because TM R solves $USELESS_{\mathrm{TM}}$, we can use R to check if q_{accept} is a useless state to decide E_{TM}. Specifically, below is a TM S that decides E_{TM} by using the decider R for $USELESS_{\mathrm{TM}}$ as a subroutine:

S = "On input $\langle M\rangle$, where M is a TM:

1. Run TM R on input $\langle M, q_{\text{accept}}\rangle$, where q_{accept} is the accept state of M.
2. If R accepts, *accept*. If R rejects, *reject*."

However, because we known E_{TM} is undecidable, there cannot exist a TM that decides $USELESS_{\mathrm{TM}}$.

2. Considera il linguaggio ALWAYSDIVERGE = $\{\langle M\rangle \mid M$ è una TM tale che per ogni $w \in \Sigma^*$ la computazione di M su input w non termina$\}$.

(a) Dimostra che il ALWAYSDIVERGE è indecidibile.

(b) ALWAYSDIVERGE è un linguaggio Turing-riconoscibile, coTuring-riconoscibile oppure né Turing-riconoscibile né coTuring-riconoscibile? Giustifica la tua risposta.

2. (a) Dimostriamo che ALWAYSDIVERGE è un linguaggio indecidibile mostrando che $\overline{A_{TM}}$ è riducibile ad ALWAYSDIVERGE. La funzione di riduzione f è calcolata dalla seguente macchina di Turing:

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. Se $x \neq w$, vai in loop.
   2. Se $x = w$, esegue M su input w.
   3. Se M accetta, *accetta*.
   4. Se M rifiuta, vai in loop."
2. Restituisci $\langle M'\rangle$."

Dimostriamo che f è una funzione di riduzione da $\overline{A_{TM}}$ ad ALWAYSDIVERGE.

- Se $\langle M, w\rangle \in \overline{A_{TM}}$ allora la computazione di M su w non termina oppure termina rifiutando. Di conseguenza la macchina M' costruita dalla funzione non termina mai la computazione per qualsiasi input. Quindi $f(\langle M, w\rangle) = \langle M'\rangle \in$ ALWAYSDIVERGE.
- Viceversa, se $\langle M, w\rangle \notin \overline{A_{TM}}$ allora la computazione di M su w termina con accettazione. Di conseguenza la macchina M' termina la computazione sull'input w e $f(\langle M, w\rangle) = \langle M'\rangle \notin$ ALWAYSDIVERGE.

Per concludere, siccome abbiamo dimostrato che $\overline{A_{TM}} \leq_m$ ALWAYSDIVERGE e sappiamo che $\overline{A_{TM}}$ è indecidibile, allora possiamo concludere che ALWAYSDIVERGE è indecidibile.

(b) ALWAYSDIVERGE è un linguaggio coTuring-riconoscibile. Dato un ordinamento $s_1, s_2, \ldots$ di tutte le stringhe in Σ^*, la seguente TM riconosce il complementare $\overline{\text{ALWAYSDIVERGE}}$:

D = "su input $\langle M\rangle$, dove M è una TM:

1. Ripeti per $i = 1, 2, 3, \ldots$:
2. Esegue M per i passi di computazione su ogni input $s_1, s_2, \ldots s_i$.
3. Se M termina la computazione su almeno uno, *accetta*. Altrimenti continua."

(Altri esercizi risolti → PDF)

Altri possibili:

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.

Mostriamo come simulare una TM bidimensionale B con una TM deterministica a nastro singolo S. S memorizza il contenuto della griglia bidimensionale sul nastro come una sequenza di stringhe separate da #, ognuna delle quali rappresenta una riga della griglia. Due cancelletti consecutivi ## segnano l'inzio e la fine della rappresentazione della griglia. La posizione della testina di B viene indicata marcando la cella con $\hat{}$. Nelle altre righe, un pallino $^{\bullet}$ indica che la testina si trova su quella colonna della griglia, ma in una riga diversa. La TM a nastro singolo S funziona come segue:

$S =$ "su input w:

1. Sostituisce w con la configurazione iniziale $\#\#w\#\#$ e marca con $\hat{}$ il primo simbolo di w.
2. Scorre il nastro finché non trova la cella marcata con $\hat{}$.
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 $\hat{}$ 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 $\hat{}$ 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 $\hat{}$ 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 $\hat{}$ 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."

2. *Dimostra che il seguente linguaggio è indecidibile:*

$$A_{1010} = \{\langle M \rangle \mid M \text{ è una TM tale che } 1010 \in L(M)\}.$$

Dimostriamo che A_{1010} è un linguaggio indecidibile mostrando che A_{TM} è riducibile ad A_{1010}. La funzione di riduzione f è calcolata dalla seguente macchina di Turing:

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

1. Costruisci la seguente macchina M_w:
   M_w = "su input x:
   1. Se $x \neq 1010$, rifiuta.
   2. Se $x = 1010$, esegue M su input w.
   3. Se M accetta, *accetta*.
   4. Se M rifiuta, *rifiuta*."
2. Restituisci $\langle M_w \rangle$."

Dimostriamo che f è una funzione di riduzione da A_{TM} ad A_{1010}.

- Se $\langle M, w \rangle \in A_{TM}$ allora la TM M accetta w. Di conseguenza la macchina M_w costruita dalla funzione accetta la parola 1010. Quindi $f(\langle M, w \rangle) = \langle M_w \rangle \in A_{1010}$.
- Viceversa, se $\langle M, w \rangle \notin A_{TM}$ allora la computazione di M su w non termina o termina con rifiuto. Di conseguenza la macchina M_w rifiuta 1010 e $f(\langle M, w \rangle) = \langle M_w \rangle \notin A_{1010}$.

Per concludere, siccome abbiamo dimostrato che $A_{TM} \leq_m A_{1010}$ e sappiamo che A_{TM} è indecidibile, allora possiamo concludere che A_{1010} è indecidibile.

3. (8 punti) *Una* Turing machine con alfabeto binario *è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è* $\Sigma = \{0, 1\}$ *e l'alfabeto del nastro è* $\Gamma = \{0, 1, \sqcup\}$. *Questo significa che la macchina può scrivere sul nastro solo i simboli* $0, 1$ *e blank: non può usare altri simboli né marcare i simboli sul nastro.*

Dimostra che che le Turing machine con alfabeto binario machine riconoscono tutti e soli i linguaggi Turing-riconoscibili sull'alfabeto $\{0, 1\}$.

Per risolvere l'esercizio dobbiamo dimostrare che *(a)* ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è Turing-riconoscibile e *(b)* ogni linguaggio Turing-riconoscibile sull'alfabeto $\{0, 1\}$ è riconosciuto da una Turing machine con alfabeto binario.

(a) Questo caso è semplice: una Turing machine con alfabeto binario è un caso speciale di Turing machine deterministica a nastro singolo. Quindi ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è anche Turing-riconoscibile.

(b) Per dimostrare questo caso, consideriamo un linguaggio L Turing-riconoscibile, e sia M una Turing machine deterministica a nastro singolo che lo riconosce. Questa TM potrebbe avere un alfabeto del nastro Γ che contiene altri simboli oltre a $0, 1$ e blank. Per esempio potrebbe contenere simboli marcati o separatori.
Per costruire una TM con alfabeto binario B che simula il comportamento di M dobbiamo come prima cosa stabilire una *codifica binaria* dei simboli nell'alfabeto del nastro Γ di M. Questa codifica è una funzione C che assegna ad ogni simbolo $a \in \Gamma$ una sequenza di k cifre binarie, dove k è un valore scelto in modo tale che ad ogni simbolo corrisponda una codifica diversa. Per esempio, se Γ contiene 4 simboli, allora $k = 2$, perché con 2 bit si rappresentano 4 valori diversi. Se Γ contiene 8 simboli, allora $k = 3$, e così via.

La TM con alfabeto binario B che simula M è definita in questo modo:

$B =$ "su input w:

1. Sostituisce $w = w_1 w_2 \dots w_n$ con la codifica binaria $C(w_1)C(w_2)\dots C(w_n)$, e riporta la testina sul primo simbolo di $C(w_1)$.
2. Scorre il nastro verso destra per leggere k cifre binarie: in questo modo la macchina stabilisce qual è il simbolo a presente sul nastro di M. Va a sinistra di k celle.
3. Aggiorna il nastro in accordo con la funzione di transizione di M:
   - Se $\delta(r, a) = (s, b, R)$, scrive la codifica binaria di b sul nastro.
   - Se $\delta(r, a) = (s, b, L)$, scrive la codifica binaria di b sul nastro e sposta la testina a sinistra di $2k$ celle.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

2. (12 punti) Una variabile A in una grammatica context-free G è *persistente* se compare in ogni derivazione di ogni stringa w in $L(G)$. Data una grammatica context-free G e una variabile A, considera il problema di verificare se A è persistente.

(a) Formula questo problema come un linguaggio $PERSISTENT_{\text{CFG}}$.

(b) Dimostra che $PERSISTENT_{\text{CFG}}$ è decidibile.

Soluzione.

(a) $PERSISTENT_{\text{CFG}} = \{\langle G, A\rangle \mid G \text{ è una CFG, } A \text{ è una variabile persistente}\}$

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

$N =$ "su input $\langle G, A\rangle$, dove G è una CFG e A una variabile:

1. Verifica che A appartenga alle variabili di G. In caso negativo, rifiuta.
2. Costruisci una CFG G' eliminando tutte le regole dove compare A dalla grammatica G.
3. Esegui M su input $\langle G'\rangle$, e ritorna lo stesso risultato di M."

Mostriamo che N è un decisore dimostrando che termina sempre e che ritorna il risultato corretto. Verificare che una variabile appartenga alle variabili di G è una operazione che si può implementare scorrendo la codifica di G per controllare se A compare nella codifica. Il secondo passo si può implementare copiando la codifica di G senza riportare le regole dove compare A. Di conseguenza, il primo ed il secondo step terminano sempre. Anche il terzo step termina sempre perché sappiamo che E_{CFG} è un linguaggio decidibile. Quindi N termina sempre la computazione.

Vediamo ora che N dà la risposta corretta:

- Se $\langle G, A\rangle \in PERSISTENT_{\text{CFG}}$ allora A è una variabile persistente, quindi compare in ogni derivazione di ogni stringa $w \in L(G)$. Se la eliminiamo dalla grammatica, eliminando tutte le regole dove compare A, allora otteniamo una grammatica G' dove non esistono derivazioni che permettano di derivare una stringa di soli simboli terminali, e di conseguenza G' ha linguaggio vuoto. Quindi $\langle G'\rangle \in E_{\text{CFG}}$, e l'esecuzione di M terminerà con accettazione. N ritorna lo stesso risultato di M, quindi accetta.
- Viceversa, se $\langle G, A\rangle \in PERSISTENT_{\text{CFG}}$ allora A non è una variabile persistente, quindi esiste almeno una derivazione di una parola $w \in L(G)$ dove A non compare. Se eliminiamo A dalla grammatica, eliminando tutte le regole dove compare, allora otteniamo una grammatica G' che può derivare w, e di conseguenza G' ha linguaggio vuoto. Quindi $\langle G'\rangle \notin E_{\text{CFG}}$, e l'esecuzione di M terminerà con rifiuto. N ritorna lo stesso risultato di M, quindi rifiuta.

3. (12 punti) Considera le stringhe sull'alfabeto $\Sigma = \{1, 2, \ldots, 9\}$. Una stringa w di lunghezza n su Σ si dice *ordinata* se $w = w_1 w_2 \ldots w_n$ e tutti i caratteri $w_1, w_2, \ldots, w_n \in \Sigma$ sono tali che $w_1 \leq w_2 \leq \cdots \leq w_n$. Ad esempio, la stringa 1112778 è ordinata, ma le stringhe 5531 e 44427 non lo sono (la stringa vuota viene considerata ordinata). Diciamo che una Turing machine è *ossessionata dall'ordinamento* se ogni stringa che accetta è ordinata (ma non è necessario che accetti tutte queste stringhe). Considera il problema di determinare se una TM con alfabeto $\Sigma = \{1, 2, \ldots, 9\}$ è ossessionata dall'ordinamento.

(a) Formula questo problema come un linguaggio SO_{TM}.

(b) Dimostra che il linguaggio SO_{TM} è indecidibile.

Soluzione.

(a) $SO_{\text{TM}} = \{\langle M \rangle \mid M \text{ è una TM con alfabeto } \Sigma = \{1, 2, \ldots, 9\} \text{ che accetta solo parole ordinate}\}$

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

$F =$ "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. Se $x = 111$, accetta.
   2. Se $x = 211$, esegue M su input w e ritorna lo stesso risultato di M.
   3. In tutti gli altri casi, rifiuta.

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

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

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

- Se $\langle M, w \rangle \in \overline{A_{TM}}$ allora la macchina M rifiuta o va in loop su w. In questo caso la macchina M' accetta la parola ordinata 111 e rifiuta tutte le altre, quindi è ossessionata dall'ordinamento e di conseguenza $\langle M' \rangle \in SO_{\text{TM}}$.
- Viceversa, se $\langle M, w \rangle \notin \overline{A_{TM}}$ allora la macchina M accetta w. Di conseguenza, la macchina M' accetta sia la parola ordinata 111 che la parola non ordinata 211, quindi non è ossessionata dall'ordinamento. Di conseguenza $\langle M' \rangle \notin SO_{\text{TM}}$.

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) 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.