[A]**4.10** Let $INFINITE_{\text{DFA}} = \{\langle A \rangle | \ A \text{ is a DFA and } L(A) \text{ is an infinite language}\}$. Show that $INFINITE_{\text{DFA}}$ is decidable.

4.10 The following TM I decides $INFINITE_{\text{DFA}}$.

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

1. Let k be the number of states of A.
2. Construct a DFA D that accepts all strings of length k or more.
3. Construct a DFA M such that $L(M) = L(A) \cap L(D)$.
4. Test $L(M) = \emptyset$ using the E_{DFA} decider T from Theorem 4.4.
5. If T accepts, *reject*; if T rejects, *accept*."

This algorithm works because a DFA that accepts infinitely many strings must accept arbitrarily long strings. Therefore, this algorithm accepts such DFAs. Conversely, if the algorithm accepts a DFA, the DFA accepts some string of length k or more, where k is the number of states of the DFA. This string may be pumped in the manner of the pumping lemma for regular languages to obtain infinitely many accepted strings.

$$\begin{aligned} \text{Let } ALL_{\epsilon CFG} &= \{\langle G \rangle \,|\text{G is a CFG that generates } \epsilon\} \\ &= \{\langle G \rangle \,|\text{G is a CFG and } \epsilon \in \text{L(G)}\} \end{aligned}$$

Want to show that $ALL_{\epsilon CFG}$ is decidable.
$\Rightarrow$ Construct a halting TM M that decides $\text{ALL}_{\epsilon CFG}$

Note: Given the CFG G, we can convert it into an equivalent CFG $G' = (V, \Sigma, R, S)$ in Chomsky Normal Form. By definition since G' is in Chomsky Normal Form, we know that the only possible ϵ rule in G' will be $S \to \epsilon$ where S is the start variable.

If $S \to \epsilon$ is a rule in R $\Rightarrow \epsilon \in L(G')$
Else $\epsilon \notin L(G')$

The following TM M decides $ALL_{\epsilon CFG}$:

M = "On input $\langle G \rangle$ where G is a CFG:

1. Convert G into an equivalent CFG G' in Chomsky Normal Form.
2. If G' includes the rule S $\to \epsilon$, accept. Else reject."

$\Rightarrow \text{ALL}_{\epsilon CFG}$ is decidable

[A]**4.12** Let $A = \{\langle M \rangle | \ M$ is a DFA that doesn't accept any string containing an odd number of 1s$\}$. Show that A is decidable.

4.12 The following TM decides A.

"On input $\langle M \rangle$:

1. Construct a DFA O that accepts every string containing an odd number of 1s.
2. Construct a DFA B such that $L(B) = L(M) \cap L(O)$.
3. Test whether $L(B) = \emptyset$ using the E_{DFA} decider T from Theorem 4.4.
4. If T accepts, *accept*; if T rejects, *reject*."

Indecidibilità

The proof of the undecidability of the halting problem uses a technique called *diagonalization*, discovered by mathematician Georg Cantor in 1873. Cantor was concerned with the problem of measuring the sizes of infinite sets. If we have two infinite sets, how can we tell whether one is larger than the other or whether they are of the same size? For finite sets, of course, answering these questions is easy. We simply count the elements in a finite set, and the resulting number is its size. But, if we try to count the elements of an infinite set, we will never finish! So we can't use the counting method to determine the relative sizes of infinite sets.

n	$f(n)$
1	0 . 3 1 4 1 5 9 2 6 5 3 ...
2	0 . 3 7 3 7 3 7 3 7 3 7 ...
3	0 . 1 4 2 8 5 7 1 4 2 8 ...
4	0 . 7 0 7 1 0 6 7 8 1 1 ...
5	0 . 3 7 5 0 0 0 0 0 0 0 ...
⋮	⋮

Of course, only part of the table can be shown on a piece of paper — it goes on forever down and to the right.

Questo dimostra che:

As long as we highlight at least one digit in each row and at most one digit in each column, we can change each the digits to get another number not in the table. Here, if we add 1 to all the highlighted digits, we end up with $0.42981\ldots$ — there's a real number that does not equal $f(n)$ for any positive integer n.

What is the point of all this? Precisely that the function f can't possibly be onto — there will always be (infinitely many!) missing values. Therefore, there does not exist a bijection between $\mathbb{N}$ and $[0, 1]$.

The preceding theorem has an important application to the theory of computation. It shows that some languages are not decidable or even Turing-recognizable, for the reason that there are uncountably many languages yet only countably many Turing machines. Because each Turing machine can recognize a single language and there are more languages than Turing machines, some languages are not recognized by any Turing machine. Such languages are not

"Esiste un problema specifico che è algoritmicamente irrisolvibile" → esiste un programma che si ferma sempre per qualsiasi sua computazione?'

La TM normale riconosce il linguaggio, ma non lo accetta:

$A_{TM} = \{\langle M, w\rangle \mid M \text{ è una TM che accetta la stringa } w\}$

- Chiarimento: A_{TM} è Turing-riconoscibile
- Conseguenza: i riconoscitori sono più potenti dei decisori
- U = "Su input $\langle M, w\rangle$, dove M è una TM e w una stringa:
  1. Simula M su input w
  2. Se la simulazione raggiunge lo stato di accettazione, accetta; se raggiunge lo stato di rifiuto, rifiuta."
- U è un riconoscitore. Perché non è un decisore?

Usiamo il concetto di Macchina universale di Turing, capace di computare qualsiasi sequenza, dato uno specifico programma o sottoprogramma, in un numero di passi finiti.

Dimostrazione:

- per contraddizione. Assumiamo A_{TM} decidibile per poi trovare una contraddizione
- Supponiamo H decisore per A_{TM}
- Cosa fa H con input $\langle M, w\rangle$?

$$H(\langle M, w\rangle) = \begin{cases} \text{accetta} & \text{se } M \text{ accetta } w \\ \text{rifiuta} & \text{se } M \text{ non accetta } w \end{cases}$$

- 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?

La diagonalizzazione conferma questa cosa; non esiste definitamente un modo di "contare sempre" dove la macchina si fermi, ad un certo punto otterremo l'elemento opposto, non essendo sempre finitamente numerabile.

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 ATM 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-riconoscibile
- Inoltre, un linguaggio è co-Turing riconoscibile se è il complementare di un linguaggio Turing-riconoscibile

A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.

In other words, a language is decidable exactly when both it and its complement are Turing-recognizable.

[A]**5.10** 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*."

1. **(12 punti)** Una *R2-L3 Turing Machine* è una macchina di Turing deterministica a nastro semi-infinito che può effettuare solo due mosse: spostarsi a destra di due celle (R2), oppure spostarsi a sinistra di tre celle (L3). Se in uno spostamento a sinistra la macchina tenta di spostare la testina a sinistra dell'inizio del nastro, allora lo spostamento termina con la testina nella prima cella del nastro.

   (a) Dai una definizione formale della funzione di transizione di una R2-L3 Turing Machine.

   (b) Dimostra che le R2-L3 Turing Machine riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

Soluzione.

(a) $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L3, R2\}$

(b) Per dimostrare che le R2-L3 Turing Machine riconoscono la classe dei linguaggi Turing-riconoscibili dobbiamo dimostrare due cose: che ogni linguaggio Turing-riconoscibile è riconosciuto da una R2-L3 Turing Machine, e che ogni linguaggio riconosciuto da una R2-L3 Turing Machine è Turing-riconoscibile.

Per la prima dimostrazione, mostriamo come convertire una macchina di Turing deterministica a nastro semi-infinito M in una R2-L3 Turing Machine S equivalente.

$S =$ "Su input w:

1. Per simulare una mossa del tipo $\delta(q, a) = (r, b, L)$, S scrive b sul nastro e muove la testina di due celle a destra, e subito dopo di tre celle a sinistra. Se lo spostamento a sinistra porta la testina oltre l'inizio del nastro, allora vuol dire che la simulazione era partita dalla prima cella del nastro. In questo caso la simulazione riprende con la testina nella prima cella del nastro, come per le TM standard. Negli altri casi, la simulazione continua con la testina in corrispondenza della cella immediatamente a sinistra di quella di partenza.
2. Per simulare una mossa del tipo $\delta(q, a) = (r, b, R)$, S scrive b sul nastro e muove la testina di due celle a destra, di nuovo di due celle a destra, e subito dopo di tre celle a sinistra. La simulazione continua con la testina in corrispondenza della cella immediatamente a destra di quella di partenza.
3. 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."

Per dimostrare che ogni linguaggio riconosciuto da una R2-L3 Turing Machine è Turing-riconoscibile, mostriamo come convertire una R2-L3 Turing Machine S in una TM deterministica a nastro semi-infinito M equivalente.

M = "Su input w:

1. Per simulare una mossa del tipo $\delta(q, a) = (r, b, L3)$, M scrive b sul nastro e muove la testina di tre celle a sinistra. Se lo spostamento a sinistra porta la testina oltre l'inizio del nastro, allora lo sposta meno a sinistra si ferma con la testina nella prima cella del nastro, come per le R2-L3 Turing Machine.
2. Per simulare una mossa del tipo $\delta(q, a) = (r, b, R2)$, M scrive b sul nastro e muove la testina di due celle a destra.
3. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di S, allora M termina con accettazione. Se in qualsiasi momento la simulazione raggiunge lo stato di rifiuto di S, allora M termina con rifiuto. Negli altri casi continua la simulazione dal punto 2."

2. (12 punti) Considera il problema di determinare se i linguaggi di due DFA sono l'uno il complemento dell'altro.

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

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

Soluzione.

(a) $COMPLEMENT_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ e } B \text{ sono DFA e } L(A) = \overline{L(B)}\}$

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

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

1. Costruisci l'automa $\overline{B}$ che riconosce il complementare del linguaggio di B
2. Esegui M su input $\langle A, \overline{B}\rangle$, 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 costruire il complementare di un DFA (basta invertire stati finali e stati non finali nella definizione dell'automa). Di conseguenza, il primo step di N termina sempre. 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 $\langle A, B\rangle \in COMPLEMENT_{\text{DFA}}$ allora $L(A) = \overline{L(B)}$, e di conseguenza $L(A) = L(\overline{B})$ perché $\overline{B}$ è il complementare di B. Quindi $\langle A, \overline{B}\rangle \in EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con accettazione. N ritorna lo stesso risultato di M, quindi accetta.
- Viceversa, se $\langle A, B\rangle \notin COMPLEMENT_{\text{DFA}}$ allora $L(A) \neq \overline{L(B)}$, e di conseguenza $L(A) \neq L(\overline{B})$ perché $\overline{B}$ è il complementare di B. Quindi $\langle A, \overline{B}\rangle \notin EQ_{\text{DFA}}$, e l'esecuzione di M terminerà con rifiuto. N ritorna lo stesso risultato di M, quindi rifiuta.

1. **(12 punti)** Una macchina di Turing salva-nastro è simile a una normale macchina di Turing deterministica a nastro singolo semi-infinito, ma può spostare la testina al centro della parte non vuota del nastro. In particolare, se le prime s celle del nastro non sono vuote, allora la testina può spostarsi nella cella numero $\lfloor s/2 \rfloor$. A ogni passo, la testina della TM salva-nastro può spostarsi a sinistra di una cella (L), a destra di una cella (R) o al centro della parte non vuota del nastro (J).

   (a) Dai una definizione formale della funzione di transizione di una TM salva-nastro.

   (b) Dimostra che le TM salva-nastro riconoscono la classe dei linguaggi Turing-riconoscibili. Usa una descrizione a livello implementativo per definire le macchine di Turing.

Soluzione.

(a) $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, J\}$

(b) Per dimostrare che TM salva-nastro riconoscono la classe dei linguaggi Turing-riconoscibili dobbiamo dimostrare due cose: che ogni linguaggio Turing-riconoscibile è riconosciuto da una TM salva-nastro, e che ogni linguaggio riconosciuto da una TM salva-nastro è Turing-riconoscibile.

La prima dimostrazione è banale: le TM deterministiche a singolo nastro sono un caso particolare di TM salva-nastro che non effettuano mai la mossa J per saltare al centro della parte non vuota del nastro. Di conseguenza, ogni linguaggio Turing-riconoscibile è riconosciuto da una TM salva-nastro.

Per dimostrare che ogni linguaggio riconosciuto da una TM salva-nastro è Turing-riconoscibile, mostriamo come convertire una macchina di Turing salva-nastro M in una TM deterministica a nastro singolo S equivalente.

$S =$ "Su input w:

1. Inizialmente S mette il suo nastro in un formato che gli consente di implementare l'operazione di salto al centro della parte non vuota del nastro, usando il simbolo speciale # per marcare l'inizio del nastro. Se w è l'input della TM, la configurazione iniziale del nastro è $\#w$.
2. 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 il # che marca l'inizio del nastro, S si muove immediatamente di una cella a destra, lasciando inalterato il #. La simulazione continua con la testina in corrispondenza del simbolo subito dopo il #.
3. 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.
4. Per simulare una mossa del tipo $\delta(q, a) = (r, b, J)$ la TM S scrive b nella cella corrente, e poi sposta la testina a sinistra fino a ritornare in corrispondenza del # che marca l'inizio del nastro. A questo punto si sposta a destra: se il simbolo dopo il # è un blank, la simulazione continua con la testina in corrispondenza del blank. Se il simbolo dopo il # è diverso dal blank, la TM lo marca con un pallino, poi si sposta a destra fino ad arrivare al primo blank. Dopodiché marca l'ultimo cella non vuota prima del blank e procede a zig-zag, marcando via via una cella all'inizio e una alla fine della porzione di nastro non vuota. Quando la prossima cella da marcare è una cella che è già stata marcata, allora la TM si sposta a sinistra, e marca la cella con un simbolo diverso, come una barra. Poi scorre il nastro per togliere tutti i pallini, e riprende la simulazione con la testina in corrispondenza della cella marcata con la barra.
5. 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."