The Post Correspondence Problem is to determine whether a collection of dominos has a match. This problem is unsolvable by algorithms.

and a collection of dominos looks like

$$\left\{\left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right]\right\}.$$

The task is to make a list of these dominos (repetitions permitted) so that the string we get by reading off the symbols on the top is the same as the string of symbols on the bottom. This list is called a ***match***. For example, the following list is a match for this puzzle.

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right]$$

Definizione

Un problema è trattabile (facile) se esiste un algoritmo efficiente per risolverlo.

- Gli algoritmi efficienti sono algoritmi con complessità polinomiale:
  - il loro tempo di esecuzione è $O(n^k)$ per qualche costante k.
- Avere complessità polinomiale è una condizione minima per considerare un algoritmo efficiente
- Un algoritmo con complessità più che polinomiale (p.es. esponenziale) è un algoritmo non efficiente perché non è scalabile.

$$PCP = \{\langle P\rangle \mid P \text{ is an instance of the Post Correspondence Problem with a match}\}.$$

THEOREM 5.15

PCP is undecidable.

The proof in book shows that the Post Correspondence Problem (PCP) is undecidable by reducing the acceptance problem for Turing machines (ATM) to PCP. The key steps are:

1. From any Turing machine M and input w, construct an instance P of PCP such that P has a match if and only if M accepts w. This establishes that if PCP were decidable, ATM would also be decidable.

2. The constructed PCP instance P simulates the computation of M on w:

- Each domino in P corresponds to a step in the computation

- A match of the dominos represents an accepting computation history of M on w

- The match forces the Turing machine simulation to occur by linking positions in one configuration to the corresponding ones in the next configuration

3. First construct an instance P' of a Modified PCP (MPCP) where the match must start with the first domino. This is done by carefully designing dominos corresponding to the initial config, transitions, copying tape symbols, halting, etc.

4. Then convert the MPCP instance P' to a standard PCP instance P by adding separator symbols (*) around the original symbols. This forces the match to start with the first domino without explicitly requiring it.

5. If the resulting PCP instance P has a match, it corresponds to an accepting computation history of M on w. Conversely, if M accepts w, the constructed P will have a match mimicking that accepting computation.

Therefore, since ATM is known to be undecidable, and we reduced it to PCP, PCP must also be undecidable. The key idea is designing the PCP dominos to force a simulation of the Turing machine's computation.

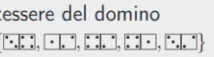
- Usiamo una riduzione mediante funzione per trovare l'algoritmo polinomiale
- Riduciamo D_1 ad un problema su grafi ...
- ... per il quale sappiamo che esiste un algoritmo polinomiale

Definition (Grafo)

Un grafo (non orientato) G è una coppia (V, E) dove:

- $V = \{v_1, v_2, \ldots, v_n\}$ è un insieme finito e non vuoto di vertici;
- $E \subseteq \{\{u, v\} \mid u, v \in V\}$ è un insieme di coppie non ordinate, ognuna delle quali corrisponde ad un arco del grafo.

Grafo del domino

- Vertici: i numeri che si trovano sulle tessere
  - $V = \{$  $\}$
- Archi: le tessere del domino
  - $E = \{$  $\}$

- **Cammino Euleriano**: percorso in un grafo che attraversa tutti gli archi una sola volta

Il problema del Cammino Euleriano

$EULER = \{\langle G \rangle \mid G$ è un grafo che possiede un cammino Euleriano$\}$

- $EULER$ è un problema classico di teoria dei grafi
- Esistono algoritmi polinomiali per risolverlo

The proof describes a reduction from the Domino problem (D1) to the Eulerian Path problem (EULER) in graph theory. The key steps are:

1. Define the Domino problem (D1):

   - Input: A set B of domino tiles

   - Question: Is there an alignment that uses all the tiles?

2. Reduce D1 to a problem on graphs (EULER) for which a polynomial-time algorithm is known to exist.

3. Define a graph:

   - An undirected graph G is a pair (V, E), where:

     - V = {v1, v2, ..., vn} is a finite, non-empty set of vertices

     - E ⊆ {{u, v} | u, v ∈ V} is a set of unordered pairs, each corresponding to an edge of the graph

4. Construct the domino graph:

   - Vertices: The numbers on the domino tiles, V = {①, ②, ③, ④}

   - Edges: The domino tiles themselves, E = {①|②, ②|③, ③|④, ②|④, ④|①}

5. Define the Eulerian Path problem (EULER):

   - Input: A graph G

   - Question: Does G contain an Eulerian path (a path that traverses each edge exactly once)?

6. Note that EULER is a classic graph theory problem with known polynomial-time algorithms, such as Fleury's algorithm.

Fleury's Algorithm:

- Input: An undirected graph G

- Steps:

  1. Choose a vertex with odd degree (or any vertex if all degrees are even)

  2. Choose an edge such that its removal does not disconnect the graph. If no such edge exists, REJECT.

  3. Move to the vertex at the other end of the chosen edge

  4. Remove the edge from the graph

5. Repeat from step 2 until all edges are removed

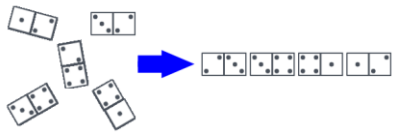
6. If all edges have been removed, ACCEPT

- Complexity: On a graph with n edges, Fleury's algorithm runs in O(n^2) time

Therefore, the reduction shows that D1 ≤m EULER, and since EULER can be solved in polynomial time (e.g., by Fleury's algorithm), D1 can also be solved in polynomial time using this reduction.

Disponete in fila le tessere del domino che vi sono state consegnate:

2 in modo che ogni numero compaia esattamente due volte (potete usare meno tessere di quelle che avete).



$D_2 = \{\langle B \rangle \mid B$ insieme di tessere del domino, ed esiste allineamento dove ogni numero compare due volte$\}$

- **Circuito Hamiltoniano**: ciclo nel grafo che attraversa tutti i vertici una sola volta

Il problema del Circuito Hamiltoniano

$HAMILTON = \{\langle G \rangle \mid G$ è un grafo con un circuito Hamiltoniano$\}$

- Come facciamo a dimostrare che $HAMILTON \leq_m D_2$?

$HAMPATH = \{\langle G, s, t \rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$.

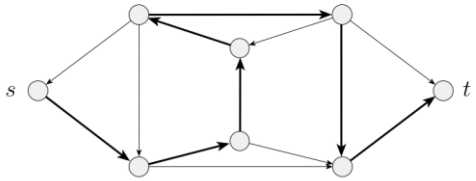


FIGURE 7.17
A Hamiltonian path goes through every node exactly once

Il discorso è:

- il problema non è risolvibile
- è invece facile *verificare* se la soluzione è corretta

I problemi per i quali esiste un algoritmo polinomiale vengono considerati *trattabili*, mentre quelli che richiedono un algoritmo più che polinomiale sono detti *intrattabili*

Stabilire con precisione qual è il confine tra problemi trattabili ed intrattabili è piuttosto difficile

DEFINITION 7.18

A ***verifier*** for a language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w, so a ***polynomial time verifier*** runs in polynomial time in the length of w. A language A is ***polynomially verifiable*** if it has a polynomial time verifier.

A verifier uses additional information to verify that a string w is a member of A. This information is called a *certificate*, or proof, of membership in A.

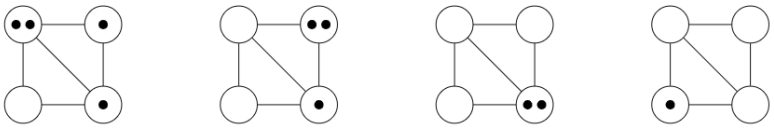
4. *Fornisci un verificatore polinomiale per il seguente problema:*

$$\text{DOUBLEHAMCIRCUIT} = \{\langle G \rangle \mid G \text{ è un grafo non orientato che contiene un ciclo che visita ogni vertice } \textit{esattamente due volte} \text{ e attraversa ogni arco } \textit{esattamente una volta}\}$$

$V =$ "su input $\langle G, C \rangle$, dove G è un grafo ed il certificato C è una sequenza di vertici $(v_1, \ldots, v_k)$:

1. Controlla che ogni elemento di C sia un vertice del grafo;
2. controlla che C sia un ciclo ($v_1 = v_k$);
3. controlla che ogni vertice del grafo compaia 2 volte in C, tranne v_1 che deve comparire 3 volte;
4. controlla che (v_i, v_{i+1}) sia un arco del grafo per ogni $i = 1, \ldots, k-1$;
5. controlla che ogni arco compaia in C esattamente una volta;
6. se tutti i test sono superati accetta, altrimenti rifiuta."

4. Pebbling è un solitario giocato su un grafo non orientato G, in cui ogni vertice ha zero o più ciottoli. Una mossa del gioco consiste nel rimuovere due ciottoli da un vertice v e aggiungere un ciottolo ad un vertice u adiacente a v (il vertice v deve avere almeno due ciottoli all'inizio della mossa). Il problema PEBBLEDESTRUCTION chiede, dato un grafo $G = (V, E)$ ed un numero di ciottoli $p(v)$ per ogni vertice v, di determinare se esiste una sequenza di mosse che rimuove tutti i sassolini tranne uno.



Una soluzione in 3 mosse di PEBBLEDESTRUCTION.

Fornisci un verificatore per PEBBLEDESTRUCTION.

Per fornire un verificatore per PebbleDestruction, procediamo come segue:

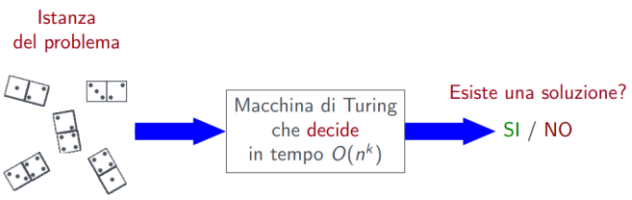
1. L'input è una tripla (G, p, S) dove:
   - G = (V, E) è un grafo non orientato
   - p è una funzione che assegna un numero di ciottoli a ogni vertice
   - S è una sequenza di mosse
2. Per ogni mossa (v, u) in S:
   - a. Verifica che v abbia almeno 2 ciottoli (altrimenti rifiuta)
   - b. Rimuovi 2 ciottoli da v
   - c. Aggiungi 1 ciottolo a u (se u != v)
3. Dopo aver processato tutte le mosse, verifica che ogni vertice abbia 0 o 1 ciottoli
4. Se tutte le verifiche hanno successo, accetta. Altrimenti, rifiuta.

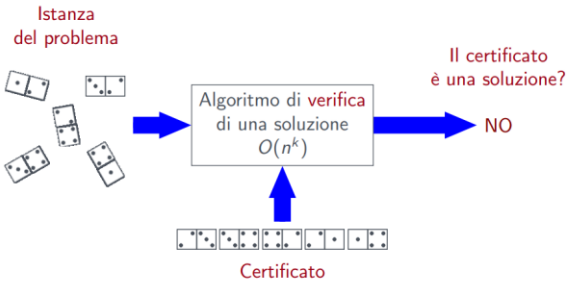
DEFINITION 7.19

NP is the class of languages that have polynomial time verifiers.

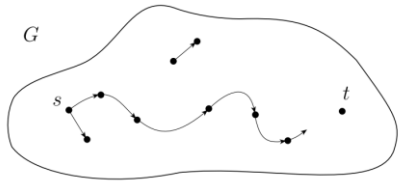
- **P** è la classe dei linguaggi che possono essere decisi da una macchina di Turing deterministica che impiega tempo polinomiale.
- **NP** è la classe dei linguaggi che ammettono un verificatore che impiega tempo polinomiale.
- **Equivalente:** è la classe dei linguaggi che possono essere decisi da una macchina di Turing non deterministica che impiega tempo polinomiale.

	P	NP
Facili da risolvere	✓	?
Facili da verificare	✓	✓
Esempi	Domino[1], Euler, Path, Relprime, ...	Domino[2], Hamilton, Sudoku, Protein folding, Crittografia, ...





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



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:

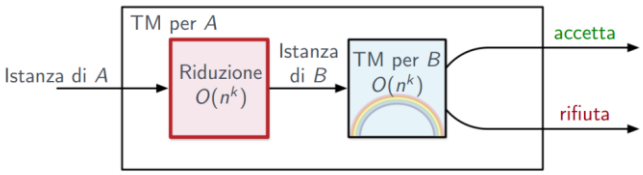
1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

Another polynomially verifiable problem is compositeness. Recall that a natural number is ***composite*** if it is the product of two integers greater than 1 (i.e., a composite number is one that is not a prime number). Let

$$COMPOSITES = \{x \mid x = pq, \text{ for integers } p, q > 1\}.$$

We can easily verify that a number is composite—all that is needed is a divisor of that number. Recently, a polynomial time algorithm for testing whether a number is prime or composite was discovered, but it is considerably more complicated than the preceding method for verifying compositeness.

Se $A \leq_P B$, e $B \in P$, allora $A \in P$:



4. (8 punti) *Supponiamo che un impianto industriale costituito da m linee di produzione identiche debba eseguire n lavori distinti. Ognuno dei lavori può essere svolto da una qualsiasi delle linee di produzione, e richiede un certo tempo per essere completato. Il* problema del bilanciamento del carico *(*LoadBalance*) chiede di trovare un* assegnamento *dei lavori alle linee di produzione che permetta di completare tutti i lavori entro un tempo limite k.*

Più precisamente, possiamo rappresentare l'input del problema con una tripla $\langle m, T, k\rangle$ dove:

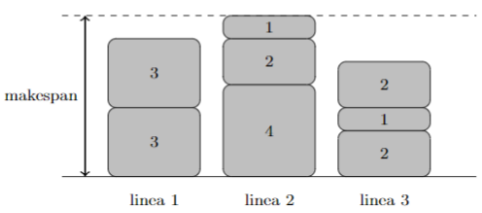
- *m è il numero di linee di produzione;*
- *$T[1 \ldots n]$ è un array di numeri interi positivi dove $T[j]$ è il tempo di esecuzione del lavoro j;*
- *k è un limite superiore al tempo di completamento di tutti i lavori.*

Per risolvere il problema vi si chiede di trovare un array $A[1 \ldots n]$ con gli assegnamenti, dove $A[j] = i$ significa che il lavoro j è assegnato alla linea di produzione i. Il tempo di completamento (o makespan*) di A è il tempo massimo di occupazione di una qualsiasi linea di produzione:*

$$\text{makespan}(A) = \max_{1 \leq i \leq m} \sum_{A[j]=i} T[j]$$

LoadBalance *è il problema di trovare un assegnamento con makespan minore o uguale al limite superiore k:*

$$\text{LoadBalance} = \{\langle m, T, k\rangle \mid \textit{esiste un assegnamento } A \textit{ degli } n \textit{ lavori su } m \textit{ linee di produzione tale che } \text{makespan}(A) \leq k\}$$



(a) LOADBALANCE è in NP. L'array A con gli assegnamenti è il certificato. Il seguente algoritmo è un verificatore per LOADBALANCE:

V = "Su input $\langle\langle m, T, k\rangle, A\rangle$:
1. Controlla che A sia un vettore di n elementi dove ogni elemento ha un valore compreso tra 1 e m. Se non lo è, rifiuta.
2. Calcola makespan(A): se è minore o uguale a k accetta, altrimenti rifiuta."

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. La prima fase è un controllo sugli n elementi del vettore A, e quindi richiede un tempo polinomiale rispetto alla dimensione dell'input. Per calcolare il makespan, la seconda fase deve calcolare il tempo di occupazione di ognuna delle m linee e poi trovare il massimo tra i tempi di occupazione, operazioni che si possono fare in tempo polinomiale rispetto alla dimensione dell'input.

23. Sia A un linguaggio. Dimostrare che A è Turing-riconoscibile *se e solo se* esiste un linguaggio decidibile B tale che $A = \{x \mid \text{esiste } y \text{ tale che } \langle x, y\rangle \in B\}$.

Per dimostrare questa affermazione, dobbiamo provare due direzioni:

1. Se A è Turing-riconoscibile, allora esiste un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}.

2. Se esiste un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}, allora A è Turing-riconoscibile.

Direzione 1: Supponiamo che A sia Turing-riconoscibile. Allora esiste una Macchina di Turing M che riconosce A. Costruiamo un linguaggio B nel seguente modo:

B = {<x, y> | M accetta x in al più |y| passi}

Chiaramente, A = {x | esiste y tale che <x, y> ∈ B}, perché se x ∈ A, allora M accetta x in un certo numero di passi, e possiamo prendere y come una codifica binaria di quel numero di passi.

Inoltre, B è decidibile. Ecco una Macchina di Turing M' che decide B:

1. Su input <x, y>, M' simula M su x per al più |y| passi.

2. Se M accetta x entro questi passi, M' accetta <x, y>. Altrimenti, M' rifiuta <x, y>.

Questa simulazione richiede un numero di passi finito (al più |y|), quindi M' decide B.

Direzione 2: Supponiamo che esista un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}. Poiché B è decidibile, esiste una Macchina di Turing M che decide B. Costruiamo una Macchina di Turing M' che riconosce A:

1. Su input x, M' genera sistematicamente tutte le possibili stringhe y (in ordine di lunghezza crescente).

2. Per ogni y generata, M' costruisce la coppia <x, y> e simula M su <x, y>.

3. Se M accetta <x, y>, allora M' accetta x. Se M rifiuta <x, y>, M' passa alla prossima y.

Se x ∈ A, allora esiste una stringa y tale che <x, y> ∈ B, quindi M accetterà <x, y> quando M' la genera, e M' alla fine accetterà x.

Se x ∉ A, allora non esiste alcuna y tale che <x, y> ∈ B, quindi M rifiuterà tutte le coppie <x, y> generate da M', e M' non accetterà mai x.

Quindi M' riconosce A, il che significa che A è Turing-riconoscibile.

Queste due direzioni insieme dimostrano l'affermazione: A è Turing-riconoscibile se e solo se esiste un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}.

In conclusione, X è indecidibile.

2. (**Eco-friendly TM**) An *eco-friendly* Turing machine (ETM) is the same as an ordinary (deterministic) one-tape Turing machine, but it can read and write on both sides of each tape square: front and back.

   At the end of each computation step, the head of the eco-friendly TM can move left (L), move right (R), or flip to the other side of the tape (F).

   (a) Give a formal definition of the syntax of the transition function of an eco-friendly TM. (Modify Part 4 of Definition 3.3 on page 168 of the textbook.)

   (b) Show that eco-friendly TMs recognize the class of Turing-recognizable languages. That is, use a simulation argument to show that they have exactly the same power as ordinary TMs.

(a) La funzione di transizione δ di una ETM è definita come:

δ: Q × (Γ × {F,B}) → Q × (Γ × {F,B}) × {L,R,F}

Dove:

- Q è l'insieme finito degli stati

- Γ è l'alfabeto del nastro che include il simbolo blank (▢)

- {F,B} indica il lato del nastro (F per fronte, B per retro)

- {L,R,F} indica il movimento della testina (L per sinistra, R per destra, F per girare sull'altro lato)

Quindi, δ(q,a,s) = (p,b,t,m) significa che se la ETM è nello stato q, legge il simbolo a sul lato s del nastro, allora passa nello stato p, scrive il simbolo b sul lato t del nastro, e muove la testina secondo m.

(b) Per mostrare che le ETM riconoscono esattamente i linguaggi Turing-riconoscibili, dobbiamo provare due direzioni:

1. Ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

2. Ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM.

Direzione 1: Data una ETM M, possiamo costruire una TM ordinaria M' che simula M. M' usa tre nastri:

- Il primo nastro simula il lato frontale del nastro di M.

- Il secondo nastro simula il lato posteriore del nastro di M.

- Il terzo nastro tiene traccia della posizione e del lato della testina di M.

Ad ogni passo, M' legge lo stato corrente di M dal suo stato, il simbolo dal nastro appropriato (primo o secondo) in base alla posizione e al lato della testina, applica la funzione di transizione di M, scrive il nuovo simbolo sul nastro appropriato, aggiorna lo stato e muove la sua testina sui tre nastri in base al movimento della testina di M.

M' accetta se e solo se M accetta. Quindi, ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

Direzione 2: Data una TM ordinaria M, possiamo costruire una ETM M' che simula M. M' usa solo il lato frontale del suo nastro per simulare il nastro di M, e non usa mai il movimento F.

Ad ogni passo, M' legge il suo stato corrente e il simbolo sul lato frontale del nastro, applica la funzione di transizione di M, scrive il nuovo simbolo sul lato frontale e muove la sua testina a sinistra o a destra in base al movimento della testina di M.

M' accetta se e solo se M accetta. Quindi, ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM.

Queste due direzioni insieme dimostrano che le ETM riconoscono esattamente la classe dei linguaggi Turing-riconoscibili.

2. (**Enthusiastic TM**) Consider the problem of determining whether a given TM ever[1] writes "332" on three adjacent squares of its tape. You may assume that the input alphabet of this TM is $\{0, 1\}$ and the tape alphabet is $\{0, 1, 2, \ldots, 9\}$.

   (a) Formulate this problem as a language $ENTHUSIASTIC_{\text{TM}}$.

   (b) Show $ENTHUSIASTIC_{\text{TM}}$ is undecidable.

   (c) Prove that $ENTHUSIASTIC_{\text{TM}}$ is Turing-recognizable.

   (d) Is $\overline{ENTHUSIASTIC_{\text{TM}}}$ Turing-recognizable? Prove or disprove.

(a) Per formulare il problema come un linguaggio, definiamo ENTHUSIASTICTM come:

ENTHUSIASTICTM = {<M> | M è una TM che su input vuoto non scrive mai "332" su tre celle adiacenti del suo nastro}

Dove <M> è la codifica della Macchina di Turing M.

(b) Per dimostrare che ENTHUSIASTICTM è indecidibile, possiamo ridurre da HALTTM, che sappiamo essere indecidibile.

Costruiamo una riduzione f da HALTTM a ENTHUSIASTICTM:

Per una data istanza <M, w> di HALTTM, definiamo f(<M, w>) = <M'>, dove M' è una TM che funziona come segue:

1. M' scrive w sul suo nastro.

2. M' simula M su w.

3. Se M si ferma, M' scrive "332" su tre celle adiacenti del nastro e poi si ferma.

Se <M, w> ∈ HALTTM, allora M si ferma su w, quindi M' scriverà "332" e <M'> ∉ ENTHUSIASTICTM.

Se <M, w> ∉ HALTTM, allora M non si ferma su w, quindi M' non scriverà mai "332" e <M'> ∈ ENTHUSIASTICTM.

Quindi HALTTM ≤m ENTHUSIASTICTM. Poiché HALTTM è indecidibile, anche ENTHUSIASTICTM deve essere indecidibile.

(c) Poiché ENTHUSIASTICTM è il complemento di {<M> | M è una TM che su input vuoto scrive "332" su tre celle adiacenti del suo nastro}, e abbiamo dimostrato che ENTHUSIASTICTM è indecidibile, segue dal Teorema di Rice (per ogni proprietà non banale delle funzioni parziali calcolabili, l'insieme delle Macchine di Turing che calcolano una funzione con quella proprietà è indecidibile) che ENTHUSIASTICTM non è Turing-riconoscibile.

(d) ENTHUSIASTICTM non è Turing-riconoscibile, come dimostrato in (c).

(a) ($\text{AGREE}_{\text{DFA}}$) Consider the problem of determining, given two DFAs, whether there is a string that both of them accept.
   i. Formulate this problem as a language $\text{AGREE}_{\text{DFA}}$.
   ii. Show that $\text{AGREE}_{\text{DFA}}$ is decidable by using a decider for one of the languages from Chapter 4 in the book.
   iii. Show that $\text{AGREE}_{\text{DFA}}$ is decidable by testing the two DFAs on all strings up to a certain length. Calculate the size that works.

(b) (REV_{DFA}) Consider the problem of determining if the languages of two given DFAs are reverses of each other.
   i. Formulate this problem as a language REV_{DFA} and show that it is decidable.
   ii. Why does a similar approach fail to show that REV_{PDA} is decidable? (This language is defined analogously to REV_{DFA}, with PDAs instead of DFAs as inputs.)

Possiamo formulare il problema AGREEDFA come il linguaggio:

AGREEDFA = {<A, B, w> | A e B sono DFA che entrambi accettano la stringa w}

ii. Per mostrare che AGREEDFA è decidibile, possiamo usare il decider per EQDFA (l'equivalenza di DFA) dal Capitolo 4.

Data un'istanza <A, B, w> di AGREEDFA, costruiamo due nuovi DFA A' e B' come segue:

A' simula A su w. Se A accetta w, A' accetta tutte le stringhe. Altrimenti, A' rifiuta tutte le stringhe.

B' simula B su w. Se B accetta w, B' accetta tutte le stringhe. Altrimenti, B' rifiuta tutte le stringhe.

Ora, <A, B, w> ∈ AGREEDFA se e solo se A' e B' sono equivalenti (cioè, accettano esattamente le stesse stringhe).

Quindi, possiamo decidere AGREEDFA eseguendo il decisore per EQDFA su <A', B'>.

iii. Per mostrare che AGREEDFA è decidibile testando direttamente i DFA su stringhe fino a una certa lunghezza:

Sia n la somma degli stati in A e B. Qualsiasi stringa di lunghezza n o superiore che è accettata da A o B deve fare entrare A o B in un loop (per il pumping lemma).

Quindi, se esiste una stringa accettata sia da A che da B, ce ne deve essere una di lunghezza inferiore a n.

Possiamo quindi decidere AGREEDFA generando tutte le stringhe fino alla lunghezza n-1 e testandole su entrambi i DFA. Se troviamo una stringa accettata da entrambi, accettiamo. Altrimenti, rifiutiamo.

(b)

i. Possiamo formulare REVDFA come:

REVDFA = {<A, B> | A e B sono DFA e L(A) = L(B)^R}

dove L(A) è il linguaggio accettato da A e L(B)^R è il linguaggio formato invertendo ogni stringa in L(B).

Per mostrare che REVDFA è decidibile, data un'istanza <A,B>, costruiamo un DFA B' che inverte ogni stringa accettata da B (scambiando stati finali e iniziali e invertendo le transizioni). Quindi eseguiamo il decider per EQDFA su <A, B'>.

ii. Se provassimo a decidere REVPDA analogamente, usando un decider per l'equivalenza di PDA invece di EQDFA, falliremmo perché non esiste un tale decider. L'equivalenza di PDA è indecidibile.

Intuitivamente, invertire un PDA è molto più complicato che invertire un DFA a causa dello stack. Non possiamo semplicemente scambiare stati finali/iniziali e invertire le transizioni; dovremmo anche invertire in qualche modo le operazioni di stack, che è impossibile in generale.