Aniello Murano

Problemi non decidibili e riducibilità

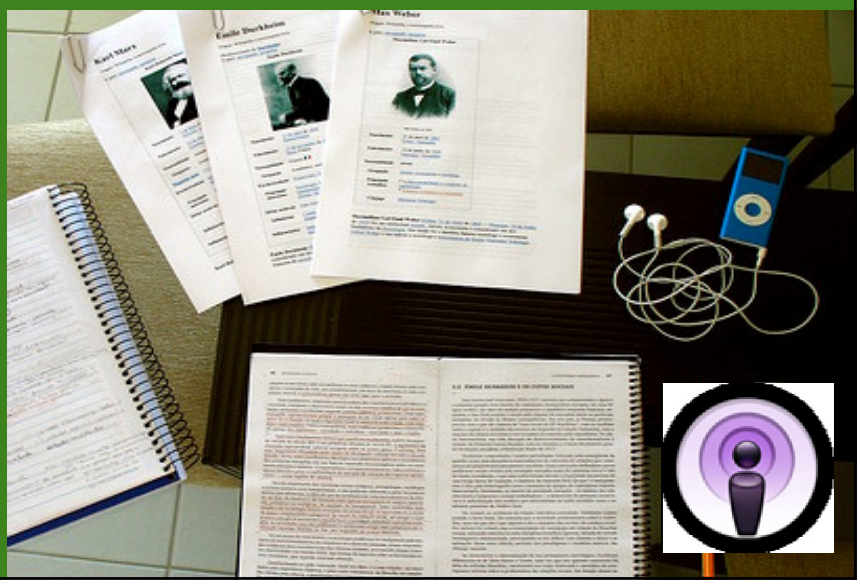
Lezione n.9
Parole chiave:
LBA e PCP

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



LBA – Linear bounded automaton

DEFINIZIONE:

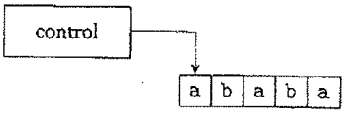
- Le **linear bounded automaton** (LBA) sono Macchine di Turing con una sola limitazione:

la testina sul nastro non può muoversi
oltre la porzione di nastro contenente l'input.

- Se un LBA prova a muovere la sua testina oltre l'input, la testina rimane ferma in quel punto come accade nella Macchina di Turing classica quando la testina tenta di andare più a sinistra del primo simbolo in input (o il simbolo di start se utilizzato).

- Un **LBA** è una Macchina di Turing con memoria limitata, com'è mostrato nella figura sottostante. Può risolvere solo problemi il cui input non superi la capienza del nastro.

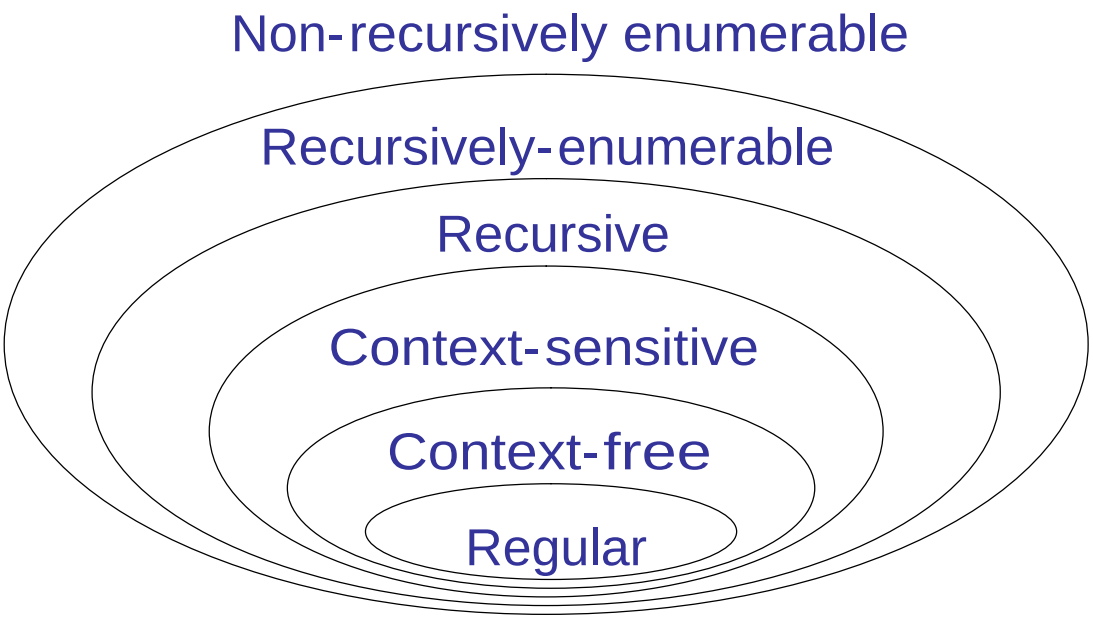
- Usando un alfabeto del nastro più ampio dell'alfabeto di input si ha che la memoria disponibile viene incrementata di un fattore costante. Perciò per un input di lunghezza n, la quantità di memoria disponibile è lineare in n. Da questo deriva il nome di questo modello.



Potenza espressiva degli LBA

- Nonostante questi vincoli sulla memoria, gli LBA sono piuttosto potenti in termini di accettori di linguaggi. Ad esempio, i decisori per A_{DFA} e E_{DFA} sono tutti LBA.
- Gli LBA sono strettamente più potenti dei PDA. Infatti, si può provare che ogni CFL può essere deciso da un LBA. Inoltre esistono linguaggi accettati da LBA che non sono context-free come ad esempio $L=\{a^n b^n c^n\}$ e $L=\{a^{n!}\}$.
- Gli LBA sono anche meno potenti delle Macchine di Turing (si pensi a qualsiasi linguaggio che ha bisogno di memoria "non constante" in aggiunta all'input per poterlo processare)
- I linguaggi accettati dagli LBA sono generati da grammatiche denominate ***"context-sensitive".***
- ***Problema aperto:*** La versione deterministica degli LBA è meno potente?

Gerarchia di Chomsky estesa



A_{LBA}

- Come accennato, gli LBA sono piuttosto potenti in termini di accettori di linguaggi. Vediamone un esempio:

- A_{LBA} è il problema di determinare se un LBA accetta il suo input.
- Formalmente,

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

- Ricordiamo che il problema analogo per le MT, ovvero A_{TM}, è indecidibile

- A_{LBA} è decidibile.
- La dimostrazione della decidibilità di A_{LBA} è semplice qualora si possa asserire che il numero di configurazioni possibili è limitato, come mostrato dal Lemma che presentiamo nella prossima diapositiva.

PARTICOLARITA' DI UN LBA

LEMMA:

- Preso un LBA M con q stati e g simboli nell'alfabeto del nastro. Ci sono esattamente qng^n configurazioni distinte di M per un nastro di lunghezza n.

DIMOSTRAZIONE:

- Ricordando che una configurazione di M è come uno "snaphshot" nel mezzo di una computazione.
- Una configurazione consiste dello stato di controllo, posizione della testina e il contenuto del nastro.
- Per quanto riguarda le "grandezze" degli oggetti coinvolti nelle configurazioni, notiamo che:
  - M ha q stati.
  - la lunghezza del nastro è n, cosi la testina può essere al più in n posizioni differenti,
  - g^n sono le possibili stringhe di simboli del nastro che si possono trovare sul nastro.
- Il prodotto di queste tre quantità è il numero totale di configurazioni differenti di M con un nastro di lunghezza n, ovvero qng^n.

DECIDIBILITA' DI A_{LBA}

TEOREMA

A_{LBA} è decidibile

IDEA PER LA DIMOSTRAZIONE:

- Prima di tutto per decidere se un LBA M accetta l'input w, simuliamo M su w con una MdT M'
- Durante il corso della simulazione, se M si ferma e accetta o rigetta, allora la prova termina perché M' si ferma. La difficoltà nasce se M gira all'infinito su w.
- Questo si può ovviare individuando quando la macchina va in loop, cosi possiamo fermarci e rifiutare.
- L'idea per capire quando M va in loop è questa:
  - Quando M computa su w, M passa da una configurazione ad un'altra. Se M ripete più volte una stessa configurazione significa che è entrata in un loop.
  - Poiché M è un LBA, sappiamo che la capienza del nastro è limitata. Attraverso il lemma visto nella precedente slide, sappiamo che M può avere solo un numero limitato di configurazioni in base alla capienza del suo nastro.
  - Quindi M ha un limite di volte prima di entrare in qualche configurazione che ha già fatto precedentemente. Come conseguenza del lemma, possiamo dire che se M non si ferma dopo un certo numero di passi significa che è in loop. Questo M' è in grado di rilevarlo.

Dimostrazione della decidibilità di A_{LBA}

DIMOSTRAZIONE:

- L'Algoritmo che decide A_{LBA} è il seguente.

M':="Su input <M,w>, dove M è un LBA e w è una stringa:

1. Simula M su w finché non si ferma o, alternativamente, per qng^n passi.
2. Se M si ferma, abbiamo i seguenti casi:
   I. Se M accetta allora M' accetta
   II. Se M rifiuta allora M' rifiuta
3. Se M non si ferma, allora M' rifiuta."

- **Domanda:** Perché l'algoritmo può rifiutare se la macchina non si ferma in qng^n passi?

- **Risposta:** Per il lemma precedente, M deve aver incontrato almeno una volta una configurazione precedentemente incontrata, dunque si trova in un loop.

Il linguaggio E_{LBA}

- Abbiamo visto un problema decidibile per un LBA ma che non lo è per una Macchina di Turing classica.
- Esistono tuttavia problemi indecidibili per le TM che rimangono tali anche per le LBA. Un esempio è il seguente linguaggio:

$$E_{LBA} = \{ \langle M \rangle \mid M \text{ è un LBA dove } L(M) = \varnothing \}$$

Per la dimostrazione di indecidibilità, dobbiamo prima introdurre il concetto di "computation histories"

Computation Histories

- Una **Computation Historie** (CH) di una MdT M è la sequenza finita di configurazioni che M attraversa per decidere un input.
-
- Sia *M* una MdT e *w* un ingresso per *M*. Una **CH accettante** per *M* su *w* è una sequenza di configurazioni $C_1, C_2, \ldots, C_l$ dove:
  - C_1 è la configurazione di partenza di *M* su *w*
  - C_l è una configurazione accettante
  - ogni C_i segue C_{i-1} secondo le regole di *M*
- Una **CH non accettante** è simile ad una CH accettante tranne che C_l è una configurazione non accettante.
- Nota: Le MdT deterministiche hanno una sola CH per un dato input, mentre le MdT non deterministiche ne possono avere diverse.

Prova che E_{LBA} è indecidibile

- Riduciamo A_{TM} a E_{LBA}.
- Per ogni MdT M e ingresso w, definiamo un LBA *B* tale che

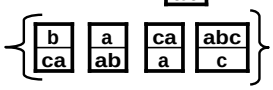
  L(B) = {CH | CH è accettante per una data MdT M e un dato ingresso w}

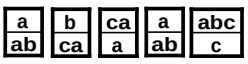
  - Se *L(B)* = Ø vuol dire che *M* non accetta *w* e dunque (M,w) non è in A_{TM}.
  - Viceversa se L(B) non è vuoto allora *M* accetta *w* e dunque (M,w) è in A_{TM}.

- Per verificare che ogni sequenza in ingresso per *B* è una CH accettante devono essere verificate le condizioni descritte in precedenza.
- Costruiamo il decisore *S* per A_{TM}, supponendo che esiste un decisore *R* per L(B)
- *S* := su ogni ingresso <*M, w*>,
  - costruisce un LBA *B* da (*M*,*w*) come definito sopra.
  - esegue *R* su ingresso *(B)*
  - se *R* accetta → S rifiuta (*L(B)* è vuoto → *M* non accetta *w*)
  - se *R* non accetta → S accetta (*L(B)* è non vuoto → *M* ha una CH accettante per *w*)
- Quindi, l'esistenza di un decisore per E_{LBA} implica un decisore per A_{TM}. Sappiamo però che A_{TM} è indecidibile, quindi non esiste un decisore per E_{LBA}.

PCP – POST CORRESPONDENCE PROBLEM

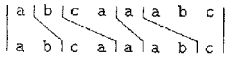
- Il fenomeno dell'indecidibilità non è un problema che riguarda solo gli automi. Per mostrare un esempio, consideriamo un problema indecidibile riguardante una "semplice" manipolazione di stringhe: il **Post correspondence problem** (PCP).
- Il PCP fu introdotto e provato indecidibile per la prima volta da E.L.Post in "A variant of Recursivley Unsolvable problem", Bull. Amer. Math. Soc.52 (1946)
- Possiamo descrivere questo problema paragonandolo ad un tipo di puzzle. Iniziamo con un insieme di domini, ognuno contenente due stringhe, ognuna su un lato.
- Un pezzo singolo del domino è rappresentato in questo modo: 

- Mentre un insieme di domini sarà di questo tipo: 

- L'obbiettivo è quello di costruire una lista di questi domini (le ripetizioni sono ammesse) in modo che la stringa dei simboli scritti nella parte superiore dei domini è uguale a quella dei simboli inferiori. Questa lista è chiamata un **match**.
- Ad esempio la seguente lista è un match per questo puzzle: 

PCP(2)

- Leggendo l'inizio della stringa otteniamo "abcaaabc", che è uguale alla stringa sotto. Possiamo anche rappresentare questo match deformando i domini in modo che creiamo la corrispondenza da sopra a sotto la linea.



- Per alcuni insiemi di domini non è sempre possibile trovare un match. Per esempio l'insieme:

$$\left\{\left[\frac{abc}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{acc}{ba}\right]\right\}$$

non può contenere un match perché ogni stringa che si trova sopra è più lunga rispetto alla stringa che si trova sotto.

- Il **Post correspondence problem** riguarda la possibilità di determinare se una collezione di domini ha un match. Questo problema non è risolvibile con algoritmi. Prima di dare la definizione formale di questo teorema con la sua definizione, formuliamo precisamente il problema e in seguito definiamo il suo linguaggio.

INDECIDIBILITA' DEL PCP

- Un instanza del PCP è un insieme P di domini: $\left[\frac{t_1}{b_a}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right]$
- Un match è una sequenza di interi $i_1, i_2, \ldots, i_l$, dove $t_{i1}, t_{i2}, \ldots t_{in} = b_{i1}, b_{i2}, \ldots, b_{in}$.
- Nota: la stessa coppia può apparire più volte, ovvero, può essere che per certi $m \neq j$, si ha che $i_j = i_m$.

- Il problema del PCP è quello di determinare se P ha un match.

- Il linguaggio corrispondente al problema è

PCP={<P>| P è un istanza del Post Correspondence Problem con un match}

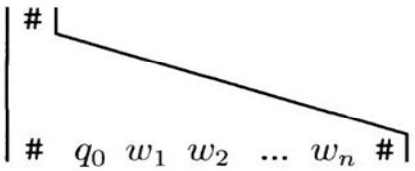
TEOREMA: **PCP è indecidibile**

PCP è indecidibile

- Per la prova riduciamo A_{TM} al PCP attraverso le Computation Histories.
- Dati ⟨M, w⟩, costruiamo un insieme P di domino nel modo seguente:
- Iniziamo con un domino che ha nella parte di sotto la configurazione iniziale di M e sopra una stringa vuota.
- Aggiungiamo a P dei domino la cui parte superiore corrisponde alla configurazione corrente e la parte inferiore è la configurazione successiva.
- Questi domino devono anche tenere conto di
  - Il movimento della testina.
  - Aumentare lo spazio del nastro con degli spazi vuoti (quando richiesto).
  - Forzare ad usare come primo domino della soluzione del match quello corrispondente alla configurazione iniziale di M.
- Per semplicità, assumiamo che ogni soluzione possibile deve sempre inizialre con il primo domino.
- Chiameremo il PCP con questa regola aggiuntiva "**modified PCP**" o semplicemente MPCP.
- Successivamente vedremo che questa non è una limitazione e che può essere rimossa.

Prova di ind. per PCP: Step 1. Inizializzazione

- Dati ⟨M, w⟩, costruiamo il primo domino dell'insieme P come corrispondente alla configurazione iniziale è q_0w nel modo seguente:



- Il simbolo # è un nuovo simbolo non appartenente all'alfabeto del nastro di M e serve come marcatore

Prova di ind. per PCP: Step2. transizione

- Ad ogni passo di computazione di M facciamo corrispondere dei domino in P in modo da poter copiare la configurazione corrente di M dalla parte di sotto dei domino (delimitata tra #) nella parte superiore e sostituiamo nella parte sottostante corrispondente la prossima configurazione

$\#C_0\#C_1\#\ldots C_{k-1}\#$
$\#C_0\#C_1\#\ldots C_{k-1}\#C_k$

$\longrightarrow$

$\#C_0\#C_1\#\ldots C_{k-1}\#$
$\#C_0\#C_1\#\ldots C_{k-1}\#C_k$

- Per fare questo, abbiamo bisogno di fare più passi elementari nel modo seguente:
- Una configurazione racchiusa tra # avrà sempre la forma del tipo α**bqc**β.
- Per calcolare la prossima configurazione, occorre copiare α sia nella parte superiore che inferiore
- Copiare bqc sulla parte superiore e la prossima configurazione sulla parte inferiore
- Copiare β sia nella parte superiore che inferiore.
- Per copiare α e β, aggiungiamo il sequente domino in P, per ogni a in Γ:

a
a

- Nelle slide successive indichiamo come trattare le transizioni.

Prova di ind. per PCP: Step 3. transizione R

- Per ogni transizione del tipo δ(q,c)=(p,d,R), noi aggiungiamo a P il seguente domino

qc
dp

- Questo domino ci permetterà di muovere da

#. . . # αb
#. . . #α**bqc**β # αb

$\longrightarrow$

#. . . # αbqc
#. . . #α**bqc**β # αbdp

- Per il caso speciale in cui c può essere il carattere blank "-", aggiungiamo in P il domino

q#
dp#

- L'ultimo domino sarà utile quando M leggerà spazi blank a destra della stringa di input

Prova di ind. per PCP: Step 4. transizione L

- Per ogni transizione del tipo $\delta(q,c)=(p,d,L)$, noi aggiungiamo a P il seguente domino

bqc
pbd

- Questo domino ci permetterà di muovere da

#. . . # α
#. . . #α**bqc**β # α

→

#. . . # αbqc
#. . . #α**bqc**β # αpbd

- Per il caso speciale in cui b può essere il carattere blank "-", aggiungiamo in P il domino

#qc
#pd

- Quest'ultimo domino sarà utile quando M è all'inizio dell'input e cerca di andare a sinistra

Prova di ind. per PCP: Step 5. matching con q_{accept}

- M accetta w se possiamo raggiungere nel PCP la seguente situazione:

$\#C_0\#\ldots\#C_{n-1}\#$
$\#C_0\#\ldots\#C_{n-1}\#\alpha bq_{accept}\beta\#$

- Prima di terminare con la riduzione, dobbiamo sistemare il fatto che la stringa nella parte inferiore dei domino è più lunga di una configurazione della parte superiore. Per questo motivo, aggiungiamo a P i domino del tipo:

cq_{accept}
q_{accept}

e

$q_{accept}c$
q_{accept}

- Gli ultimi domino permetteranno di scartare uno alla volta i simboli in eccesso, fino ad arrivare alla seguente situazione:

$\#C_0\#\ldots\#q_{accept}c\#$
$\#C_0\#\ldots\#q_{accept}c\#q_{accept}$

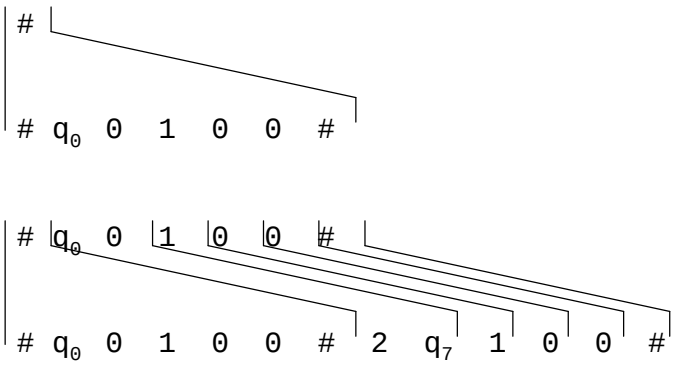
Prova di ind. per PCP: Step 6. Chiusura con q_{accept}

- Per finire dobbiamo aggiungere il seguente ulteriore domino a P:

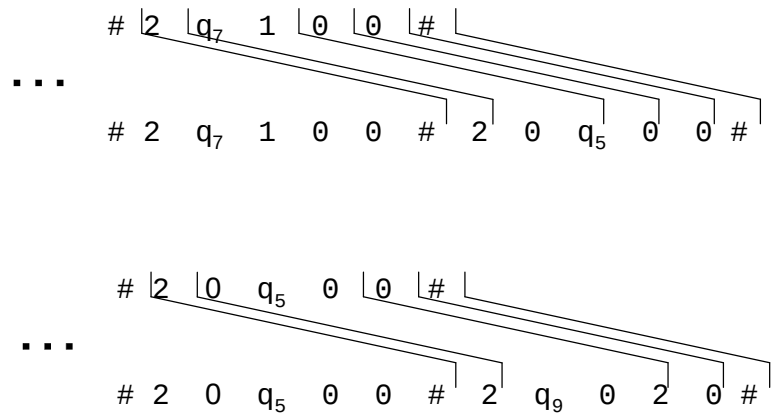
q_{accept} # #
#

- Con questa costruzione, abbiamo una istanza per MPCP che ammette soluzione se e solo se M accetta w.

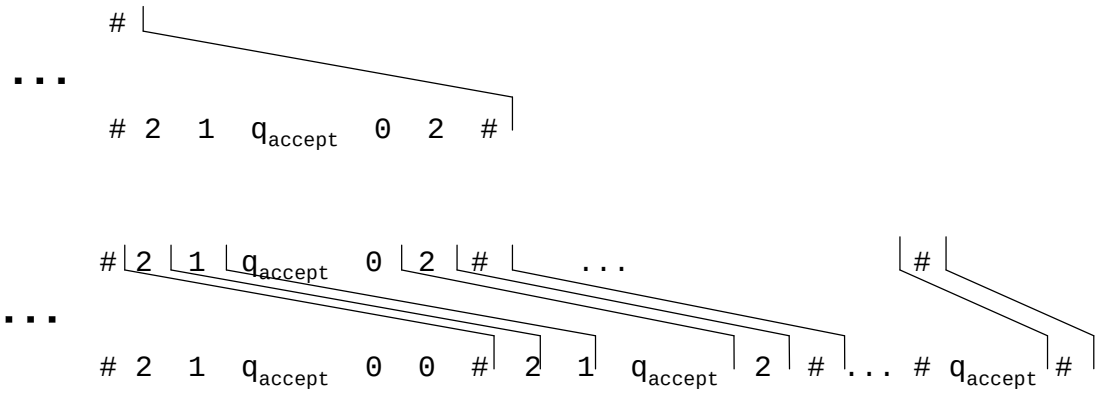
Esempio: Step 1, Step 2 + Step 3



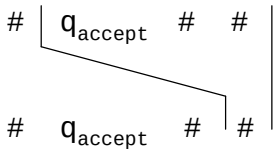
Esempio: Step 2+3, Step 2+4



Esempio: Step 2+4, Step 2+5



Esempio: Step 6



Da MPCP a PCP

- Per completare la dimostrazione, dobbiamo forzare il fatto che il primo domino deve essere il primo ad essere usato in ogni possibile soluzione del PCP.
- Sia "⋆" un nuovo simbolo (cioè non presente in Γ U {#}).
- Per ogni stringa s, sia⋆s la stringa ottenuta inserendo il simbolo "⋆"
- Prima di ogni simbolo in s. Per esempio, ⋆(abc) = ⋆a ⋆ b ⋆ c.
- Analogamente, definiamo s⋆ e ⋆s⋆ come la stringa ottenuta aggiungendo solo dopo, oppure sia prima che dopo, ogni simbolo di s il simbolo ⋆
- Dato l'insieme di domino P costruiti precedentemente, si sostituiscano i domino nel modo seguente:
  - t1/b1 = t1⋆/ ⋆ b1⋆
  - ogni altro ti/bi= ⋆ ti / b1⋆
  - il domino finale q_{accept}##/ # = ⋆q_{accept}# ⋆ #/#
- In questo modo, il primo domino deve essere assolutamente il primo altrimenti non ci sarà matching

Esercizio

- Provare che il seguente problema è indecidibile.

Problema del domino:

Dato un insieme finito di colori e un insieme di piastrelle quadrate con i quattro lati colorati di un qualche colore tra quelli a disposizione, trovare, se esiste, un modo per piastrellare una parete infinita in lunghezza e larghezza assicurando che i lati adiacenti delle piastrelle abbiano lo stesso colore.