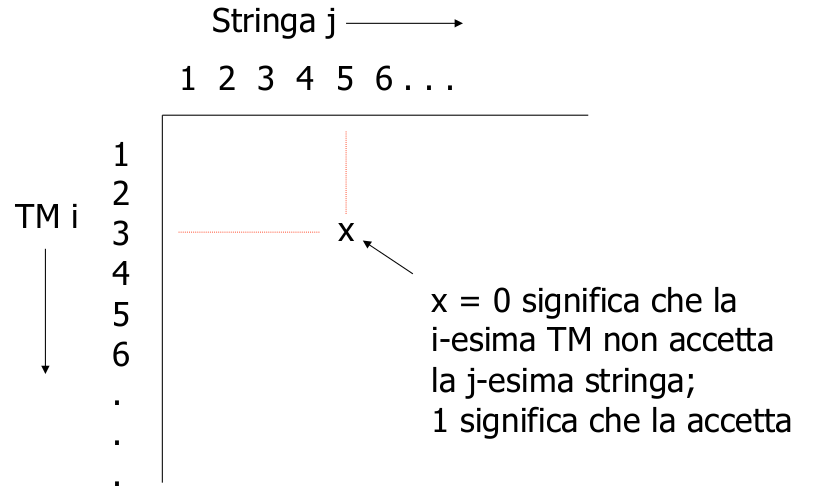
Ha senso parlare di enumerazione delle stringhe e di Macchine di Turing.



**Riduzione:**

P1 riduce P2 se è possibile trovare una soluzione per P2 che è anche soluzione per P1.

L'istanza di P2, con un certo input, ti risolve anche un'istanza di P1.

Basta trovare una sola istanza per fare la riduzione.

Es:

Ld riduce Lu complementato

Lu riduce Lne

Lu riduce Lhalt

….

Per dimostrare che L non è ricorsivo, faccio una riduzione da Lu a L.

Se ipotzzo che L è ricorsivo, mi torna utile solo per dimostrare che L complementato non è ricorsivo, e quindi non è nemmeno R.E.

Alla fine si arriva ad una contraddizione per riduzione.

Si può avere anche una contraddizione con un ciclo di implicazioni, in cui la prima nega l'ultima (es. stampa Hello World).

**Dato un programma, è possibile prevedere che stampi Hello World?**

P è il programma, I è l'input del programma

Problema originale:

H(P,I) = “yes” se il programma stampa Hello World con input I

“no” altrimenti

costruisco un'altra istanza:

H1(P,I) = “yes” se il programma stampa Hello World con input I

“Hello World” altrimenti

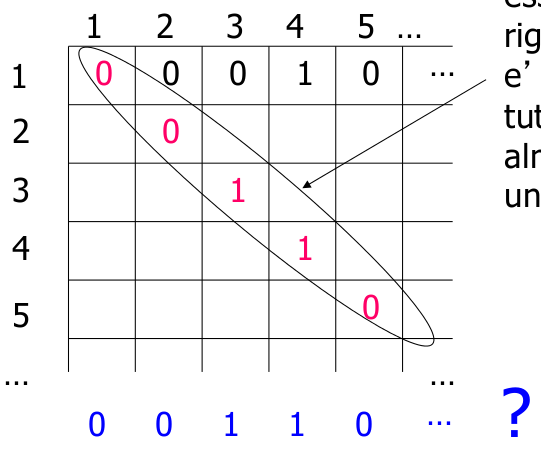
H2 = H1(P,P), cioè l'input del programma è il suo stesso codice sorgente.

Si può costruire una TM che risolve questo problema?

Se H2(P) = “yes” => H1(P,P) = “yes” => H(P,P) = “yes” => H2

**Diagonalizzazione:**

Quando si riempie la matrice sopra, si trova una stringa w che non è accettata da nessuna TM:



Il complemento della diagonale non è una riga lecita perché p diversa da tutte le altre righe.

Se fosse una riga lecita, allora esisterebbe una TM tale che ha la stessa riga, però nella casella TMk, wk il valore cambia per forza, a causa del complemento.

Quindi Ld (linguaggio di diagonalizzazione) rappresenta un linguaggio che nessuna TM riconosce.

Definizione di Ld:

**Ld = {w | w e’ la stringa i-esima, e la i-esima TM non riconosce w}**

**Linguaggio Universale:**

Il linguaggio universale, Lu, viene riconosciuto dalla TM universale, ed è R.E.

La UTM ha come input l’indice i di una certa TM M e l’indice j di una stringa w, e accetta se e solo se Mi accetta wj.

Definizione di LU:

**Lu = {stringa k | k-esima coppia <i,j> t.c. M i accetta stringa j }**

(un linguaggio è un insieme di stringhe, quindi le coppie vengono enumerate e rappresentate con una stringa. È un insieme di stringhe che identifica un insieme di coppie)

**Prova che Lu è RE ma non ricorsivo:**

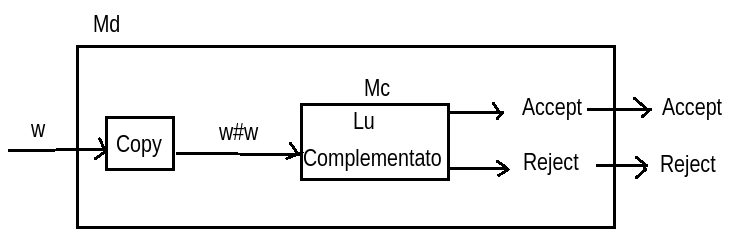
**(Lu complementato non R.E.)**

Se Lu fosse ricorsivo, allora anche Lu complementato sarebbe ricorsivo, per la proprietà di chiusura dei linguaggi ricorsivi.

Quindi è possibile costruire una TM che accetta il linguaggio Lu complementato e accetta o si ferma sempre.

È possibile ridurre un'istanza di questa TM (dando il giusto input) a un'istanza del linguaggio di diagonalizzazione Ld.

Si potrebbe costruire questa TM che accetta Ld:



Md è l'ipotetica TM che accetta Ld.

w indica una TM in una certa enumerazione.

Md fa una copia dell'input, e passa w#w a Mc, la TM ipotetica che riconosce Lu complementato.

Nel caso in cui w#w indica la TM Mi e la stringa wi, si ha la coppia della diagonale <Mi,wi>.

Con questo input, Lu complementato (Mc) accetta solamente se Mi non accetta wi, quindi sarebbe una TM che riconosce il linguaggio Ld.

Perchè è stata fatta una riduzione da Ld a Lu complementato, l'istanza di Lu complementato risolve anche Ld (basta cambiare input e si trovano tutte le macchine di Turing della diagonale).

Dato che Ld sappiamo essere NON R.E, allora anche Lu complementato **non è R.E.**

**NB Ld complementato:**

Ld complementato è R.E., perché il suo linguaggio è l'insieme delle stringhe tali che Mi accetta wi.

Si può ridurre la UTM a Ld complementato, con la stessa costruzione sopra, solo che Mc non indica Lu complementato ma Lu, quindi accetta se M accetta w.

**Le e Lne per le Macchine di Turing:**

w è la stringa che indica una TM secondo una certa enumerazione.

* Le = empty language
* Lne = not empty language

Se Mi non accetta alcun input => L(Mi) = {}, e w € Le.

Quindi Le è l'insieme di tutte le stringhe che codificano una TM, il cui linguaggio è vuoto.

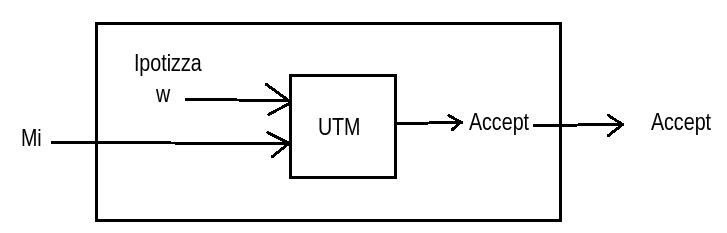
Lne è l'insime di tutte le stringhe che indicano una TM che accetta almeno un input.

Le = {M | L(M) = {} }

Lne = {M | L(M) != {} }

**Lne è R.E.**

Per dimostrare che un linguaggio è R.E., costruisco una TM che risolve il programma.



Si costruisce una TM che usa al suo interno la UTM. Usando il non determinisco,crea tante copie quante sono le stringhe generate, e poi la UTM prova a dire se Mi accetta quella stringa.

Se almeno una istanza accetta, allora il codice di Mi € a Lne.

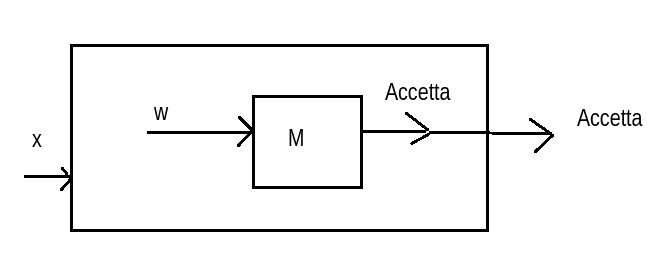
**Lne non è ricorsivo:**

Per dimostrare che Lne non è ricorsivo, faccio una riduzione da Lu a Lne.

Trovo quindi un'istanza di Lne che mi risolve Lu.

Per la riduzione, un'istanza di Lu ha soluzione **se e solo se** l'istanza di L,ne ha soluzione.

Quindi, data una coppia <M,w>, M accetta w (cioè la stringa che enumera la coppia <M,w>) appartiene a Lu **se e solo se** L(M') non è vuoto, dove M' è la TM che risolve Lne.



M' ignora x.

Accetta x se e solo se la macchina di Turing M accetta w

(Non si usa la UTM, perché la UTM ha in input una coppia TM#w, io uso la coppia di

riferimento <M,w>, e trovo se questa coppia appartiene a Lu se e solo se M' non è

vuoto, cioè accetta x, cioè risolve Lne)

Dato che è stato ridotto Lu a Lne, allora Lne **non è ricorsivo.**

**NB:**

Se M' accetta x, allora accetta tutto l'alfabeto (alfabeto\*, stella di Kleene)

**Le (non R.E.):**

il linguaggio vuoto è non R.E, perché se fosse R.E allora, per la proprietà di chiusura del complemento, sia Lne che Le sarebbero ricorsivi.

Ma abbiamo dimostrato prima che Lne è R.E e non ricorsivo, quindi Le è **non R.E.**

**Proprietà dei linguaggi:**

Ogni proprietà indica un insieme di linguaggi, ad es:

- proprietà di infinitezza: indica i linguaggi infiniti

-

Sia Lp l’insieme delle stringhe corrispondenti a TM M tali che L(M) ha la proprieta’ P.

Ci sono due proprieta’ banali per cui LP e’ ricorsivo:

La proprieta’ sempre falsa, che non contiene alcun linguaggio RE

La proprieta’ sempre vera, che contiene tutti i linguaggi RE

**Teorema di Rice**:

per tutte le altre proprieta’ non banali P, Lp e’ indecidibile

Per dimostrarlo, riduciamo Lu a Lp:

**Riduzione:**

La riduzione prende in input la stringa wk (che rappresenta k-esima coppia <i,j>),

e produce la stringa ws corrispondente ad una TM M s tale che...

L(M s ) ha la proprieta’ P se e solo se Mi accetta wj

Non e` restrittivo assumere che ∅ non abbia la proprieta’ P.

Se ce l’ha, consideriamo il complemento di P

Costruzione di M s

M s ha 2 nastri:

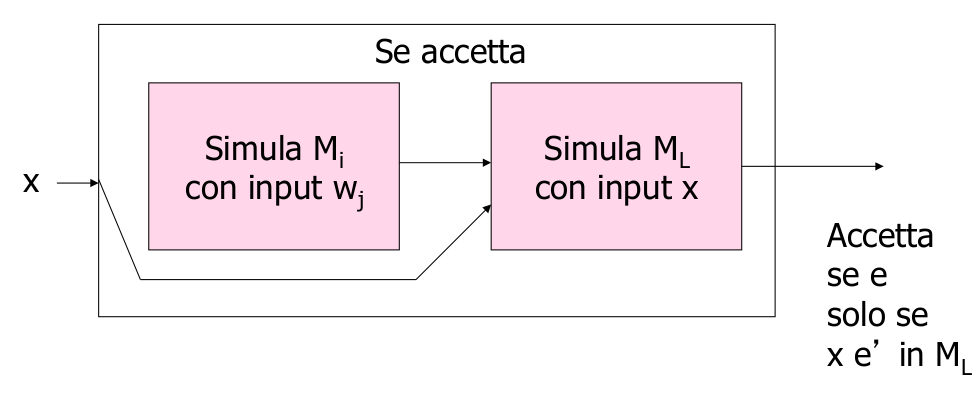
1. sul secondo nastro simula M i con input w j

2. se M i accetta w j , simula M L sull’input x (che e’ inizialmente sul primo nastro)

3. se M i non accetta w j , non si accetta x

• CONCLUSIONE: M s accetta l’input x **se e solo se** M i accetta w j e M L accetta x

Ms



**Se M i accetta w j**

L(M s ) = L, e quindi L(M s ) ha la proprieta’ P

Quindi la stringa w s che rappresenta M s e’ in L P

**Se M i non accetta w j**

L(M s ) = ∅, e quindi L(M s ) non ha la proprieta’ P

Quindi la stringa w s che rappresenta M s non e’ in L P

**Conclusioni:**

Quindi, l’algoritmo che converte la stringa w k (rappresentante la coppia <i,j> coincidenti alla TM M i e la stringa w j nella stringa w s (corrispondente alla TM M s ) e’ una riduzione di L u in L P.

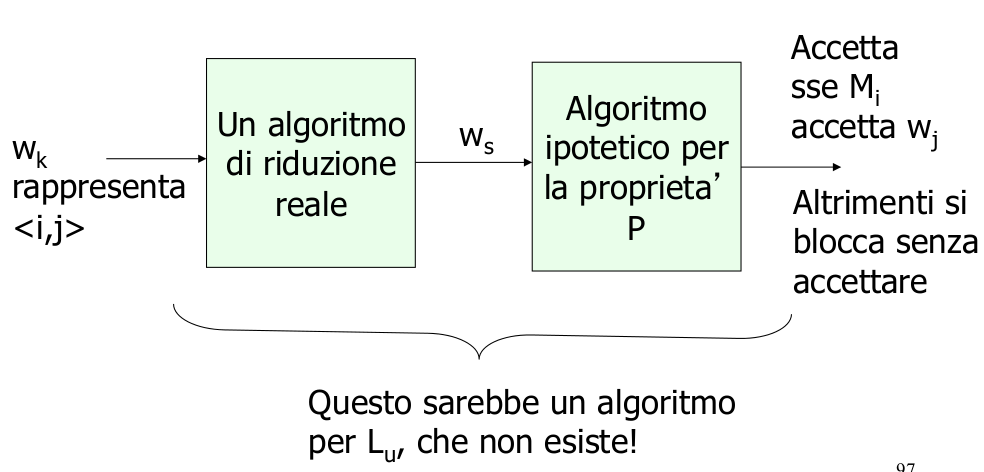
Quindi, Lp e’ indecidibile!

**N.B.**

NON tutte le domande riguardo a una TM sono indecidibili, ad es:

- una TM ha almeno 5 stati? È decidibile

- una TM fa almeno 5 mosse? È decidibile.



**Applicazioni del teorema di Rice:**

Tutti i problemi interessanti su linguaggi RE (e quindi TM) sono indecidibili:

* E’ L(M) un linguaggio regolare?
* E’ L(M) un CFL?
* L(M) include le stringhe palindrome?
* E’ L(M) vuoto?
* L(M) contiene piu’ di 1000 stringhe?

**PCP**

Lu <=> MPCP <=> PCP

Una istanza di PCP e’ una lista di coppie di stringhe non vuote su un alfabeto Σ

Ad esempio (w1 , x1), (w2 , x2),..., (wn , xn)

La risposta a tale istanza e’ “si’” se e solo se esiste una sequenza di indici i 1 ,...,i k tali che

wi1 ...wik = xi1 ...xik

**MPCP riduce PCP:**

c'è una coppia speciale che è la coppia di partenza.

Data una istanza di MPCP consideriamo:

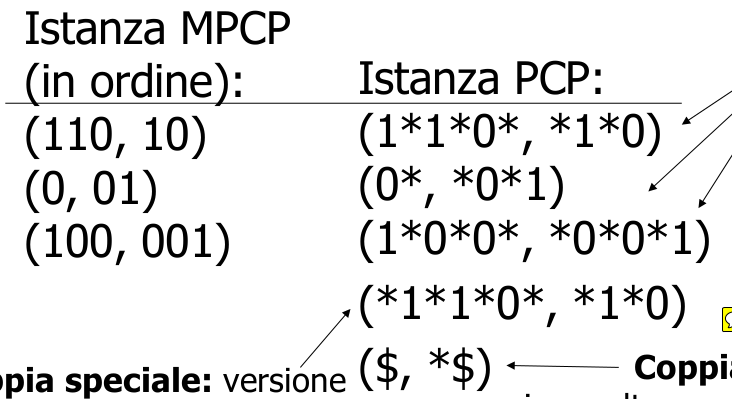
per ogni (wi , xi) del MPCP nell’istanza:

* (wi ’, xi’) è del PCP con
* wi ’ come w i ma con \* dopo ogni carattere
* xi ’ come x i ma con \* prima di ogni carattere

per la prima coppia (coppia speciale) (w1 , x1) nell’istanza:

* (w1”, x1’) con
* w1” come w1 ma con \* prima e dopo ogni carattere
* x1’ come x1 ma con \* prima di ogni carattere

una coppia finale ($, \*$).



Da MPCP a PCP si ottiene la stessa stringa, con \* prima e dopo di ogni simbolo, e alla fine il $.

Da PCP a MPCP si rimuovono tutti gli \* e il $ alla fine.

**Lu riduce MPCP:**

Data una TM Mi con input wj usiamo MPCP per simulare la sequenza di ID che Mi esegue con input wj.

Per la riduzione, MPCP ha soluzione se e solo se Mi accetta wj, e viceversa.

Se q0 w j ⊦ID1 ⊦ID2 ⊦ ... e’ la sequenza di ID di M i con input wj , allora l’eventuale

soluzione all’istanza di MPCP che definiremo iniziera’ con la stringa #q0 wj #ID1 #ID2 #...

Il secondo elemento delle coppie (la seconda lista) è sempre una ID più avanti, se alla fine si arriva in uno stato di accettazioe, anche la prima coppia torna in pari.

Prima coppia MPCP: (#, #q0 wj #)

Iniziamo con la seconda stringa che contiene l’ID iniziale

(cosi’, risulta essere avanti di una intera ID)

Per ogni stato q di M i e simbolo del nastro X, consideriamo le coppie: (4 casi, cambi stato e simbolo scritto e direzione)

(prima si cambia stato, poi la testina scrive il simbolo alla sua destra, poi cambia direzione)

(qX, Yp) se δ(q, X) = (p, Y, R)

(ZqX, pZY) se δ(q, X) = (p, Y, L) [per ogni Z]

Inoltre, se la testina e’ in fondo alla ID

(q#, Yp#) se δ(q, B) = (p, Y, R)

(Zq#, pZY#) se δ(q, B) = (p, Y, L) [per ogni Z]

L’istanza di MPCP cosi’ generata ha soluzione **se e solo se** la TM Mi riconosce l’input wj.

**Da PCP a CFG**

Sia (w i , x i ) la i-esima coppia

Consideriamo i simboli indice a 1 ,..., a k

(assumendo che non siano nell’alfabeto dell’istanza di PCP)

Il linguaggio della lista per w 1 ,...,w k e’ definito dalla CFG:

A → wi A ai , A → wi ai [per i = 1, 2,..., k]

In modo simile, dal secondo componente di ogni coppia, possiamo costruire il linguaggio della lista:

B → xi B ai , B → xi ai [per i = 1, 2,..., k]

ESEMPIO:

(a,ab), (baa,aab), (bba,ba)

Useremo 1, 2, 3 come simboli indice per le tre coppie precedenti:

A → aA1 | baaA2 | bbaA3 | a1 | baa2 | bba3

B → abB1 | aabB2 | baB3 | ab1 | aab2 | ba3

**Riduzione:**

una grammatica è ambigua **se e solo se** il PCP ha soluzione, in questo caso A e B generano la stessa stringa.

Si aggiunge alla CFG la grammatica la produzione S → A | B

se si hanno 2 derivazioni left-most differenti, allora la CFG è ambigua e si è trovata una soluzione al PCP.

La riduzione è corretta, perché se si trovano 2 derivazioni per una stringa, allora iniziano con 2 produzioni diverse..

Perchè la derivazione per A è solo una e per B è solo una.

Quindi, data una CFG , è ambigua? È indecidibile.

**Complemento di un linguaggio della lista (CFL)**

Si può costruire un PDA che accetta il complemento del linguaggio della lista.

(Il linguaggio della lista sono gli indici delle coppie, tali che si ha la stessa stringa sia sopra che sotto)

**PDA:**

Inizialmente si aspettano in input simboli del PCP, e si mettono sullo stack.

Successivamente si aspettano simboli indice, e si controlla se lo stack contiene il reverse delle corrispondenti stringhe.

In caso di input diverso dalle attese, il PDA legge il restante input qualunque esso sia.

Il PDA e’ sempre in uno stato di accettazione, ad esclusione del caso in

cui riappaia la base dello stack.

Ma se c’e’ ancora input, lo si consuma tornando in uno stato di accettazione

**Uso dei complementi:**

L= LAc ∪ LBc

L'istanza del PCP ha soluzione se e solo se L != (alfabeto)\*

perché se esiste una soluzione al PCP, allora La intersecato Lb contiene almeno una stringa, che non appartiene all'unione dei complementi.

Da questa conclusione, deriva anche l'indecidibilità di != (alfabeto)\*, perché abbiamo ridotto il PCP a questa soluzione.

Quindi anche la **verifica di uguaglianza fra CFL** è indecidibile.