Funzioni ricorsive parziali calcolabili dalle TM che si fermano per alcuni input, ricorsive totali le TM si fermano sempre.
Tutti i linguaggi finiti sono riconosciuti da FA ma anche qualche linguaggio infinito.
FA=>Linguaggi regolari
PDA=>CF, regolari
TM=>CF, regolari, ricorsivi e alcuni ricorsivamente enumerabili
RegEXP:
Base: 1. 3 e 0 sono regexp; 2.a in Sigma =>a->L(a)={a} regexp;3.L variabile che rapp.linguaggio
Induz: E+F=>L(E+F)=L(E)UL(F) …
Formula chiusura transitiva: R^(k)_ij=R^(k-1)_ij+ R^(k-1)_ik(R^(k-1)_kk)* R^(k-1)_kj con R^(k-1)_ij regexp percorsi da i a j che passano per k stati.

SUBSET CONSTRUCTION

Qd è l'insieme potenza di Qn, gli stati accettanti di n sono quelli che contengono almeno uno stato accettante di d, rinomino grafo ed elimino isole.
Se e-NFA devo inserire anche l' E-CLOSE dello stato esaminato e lo stato iniziale è E-CLOSE(q0)
Attenzione a mettere le {} sul DFA nuovo

TABLE FILLING

Scrivo la tabella righe BCDEF…Z e colonne ABCDE…V, segno le caselle degli stati accettanti e inizio a percorrerla per colonne da sx a dx dall'alto in basso controllando se dati i due stati della casella vanno in stati distinguibili per un input, se sì segno la casella, altrimenti proseguo.
Una volta finito rieseguo da capo ricontrollando solo le caselle vuote.
Mi fermo quando ho loop tra caselle vuote oppure eseguo una scansione senza modificare nulla.

PMM

Conto I controllo seguendo il grafo, quando fallisce conto due volte.

CYK

Per controllare che stringa a sia in L(G)
15
14 25
13 24 35
12 23 34 45
11 22 33 44 55
a1 a2 a3 a4 a5
Tabella di sottostringhe che iniziano dal carattere x e finiscono in y.
Inizio dalla riga più bassa e scrivo le produzioni relative ai terminali delle sottostringhe unitarie.
Passo alla riga sopra 12 che indica la sottostringa dei primi due caratteri che può essere composta solo da 11 e 12, scrivo quindi le produzioni che producono le combinazioni delle produzioni nelle prime due caselle 11 e 22. Poi 13 può essere formata sia da 11 e 23 che da 12 e 33. Attenzione all'ordine. Mi fermo se sull'ultima casella trovo S.

CNF

1)Eliminare e-prod:
Si elimina epsilon e si aggiunge la possibilità che la produzione con epsilon non ci sia su tutte le produzioni.
2)Eliminare prod-unit:
Scrivere le coppie unitarie e quando le produzioni sono diverse (B,A) scrivere su B quello prodotto da A ed eliminare la produzione unitaria B->A
3)Eliminare stati inutili:
3a)Eliminare i non generatori: newv={Produzioni che producono terminali} T'={Terminali prodotti} P'={Produzioni usate}; oldv=newv; newv={Produzioni che producono terminali o che producono produzioni presenti in oldv} T''={T'u eventuali nuovi terminali}; P''={P''u Produzioni usate}; ecc.. Finisce quando non posso aggiungere altro
3b)Eliminare i non raggiungibili: seguire le produzioni dalla S a tutti i terminali
4)Forma Normale di Chomsky:
Ogni produzione ha 2 variabili o 1 terminale. Se ho A->a con |a|>2 per ogni terminale creo una variabile e una produzione A1->a1 A2->a2. Per ogni variabile A->B1B2…Bk introduco k-2 nuove variabili C1, C2, Ck-2 e scompongo come A->B1C1, C1->B2C2 …Ck-2->Bk-1Bk

ID PDA

Input,stack/newstack con stato precedente esempio 0, Z_0/0Z_0 oppure cancello 0,0/e o non faccio nulla e,Z_0/Z_0
(stato,input,stack)-> (stato,input,stack) (q,aw,Xb)|-(p,w,cb) consumo a e sostituisco c a X sulla cima dello stack.

ID TM

Legge e scrive quello che c'è a dx dello stato e si sposta nella direzione scritta.
(q0,a)=(q0,b,R)
(q0,b)=(q1,B,L)
q0abb|-bq0bb|-q1bb
Se a sx non c'è nulla scrivere B

DERIVAZIONI

Inferenza ricorsiva
Derivazione

CODIFICA TM

Inizia e finisce con 111, tra una f.transizione e l'altra 11, le transizioni (qi,Xj)=(qk,Xl,Dm) diventano 0i10j10k10l10m con L=D1 e R=D2

REGEXP<->DFA

Rij(k)=Rij(k-1)+Rik(k-1)[Rkk(k-1)]*Rkj(k-1)
R+S parallelo, RS serie, R* loop antiorario1-e->R<-e->2 1-e->2
Semplificazioni (e+R)*=R* (e+R)R*=R* 0*=e R0=0R=0 R+0=R
Per eliminazioni di stati: per ogni stato accettante q applico il processo di riduzione per produrre un automa con le etichette sugli archi regexp eliminando tutti gli stati intermedi fino a q0.
Se lo stato iniziale è anche accettante vanno eliminati tutti gli stati tranne q0. La regexp è la somma di tutte le regexp trovate per ogni stato accettante.

CONVERSIONI:

NFA->DFA O(n^3 2^n);DFA->NFA O(n);NFA->DFA->REGEXP O(n^3 4^(n^3 2^n));
REGEXP->e-NFA O(n);e-NFA->NFA O(n^3)

DIMOSTRAZIONI:

L è accettato da e-NFA sse L è accettato da DFA.
(Se)Trasformo D in e-NFA aggiungendo la transizione d(q,e)=0 per ogni stato q di D e aggiungendo le {} su ogni d(q,a)={p}
(Solo Se) dim che dE(q0,w)=dD(qD,w) su lunghezza di w
Base: se |w|=0 allora w=e dE(qo,e)=ECLOSE(q0) e qD=ECLOSE(q0) per definizione e per DFA d(p,e)=p per ogni p in particolare per qD => dD(qD,e)=ECLOSE(q0)
Induzione: Suppongo w=xa e dE(q0,x)=dD(qD,x)={p1,p2,..,pk}

1. UdE(pi,a)={r1,r2,…,rm} su i fino a k
2. dE(q0,w)=U ECLOSE(rj) su j fino a m

Per costruzione per sottoinsiemi dD({p1,p2,…,pk},a) si ottiene dagli stessi passi 1 e 2 quindi dD(qD,w)=dD({p1,p2,…,pk},a}=dE(q0,w) CVD

REGEXP: se L=L(A) per DFA A allora esiste regexp R tale che L=L(R) con R^(k)_ij regexp il cui linguaggio è l'insieme delle stringhe w tali che w sia l'etichetta del cammino da i a j in A e che nessun nodo intermedio abbia numero maggiore di k.
Base:k=0 cammino senza stati intermedi 1. un arco da i a j 2. un cammino di lunghezza 0 che contiene sono il nodo i. Se i<>j è possibile solo la 1. Esamino il DFA per trovare i simboli di input a tali che esista una transizione dallo stato i a j. A)se non esiste a allora R^(0)_ij=0 b) se esiste un simbolo a R^(0)=a c) se esistono simboli a1,a2,…,ak che etichettano archi dallo stato i allo stato j allora R^(0)=a1+a2+…+ak
Se invece i=j i cammini leciti sono il cammino di lunghezza 0 e tutti i cicli da i a se stesso. Il cammino di lunghezza 0 è 3 e va aggiunto alle varie espressioni dei punti a, b e c.
Induzione: esiste un cammino da i a j che non attraversa nessuno stato superiore a k. 1. il cammino non passa per k quindi l'etichetta appartiene al linguaggio R^(k-1)_ij 2. il cammino passa per k almeno una volta e posso scomporlo in segmenti R^(k-1)_ik(R^(k-1)_kk)* R^(k-1)_kj e unendo l'espressione regolare del punto 1 ho R(k) che dipende solo dalle espressioni con apice inferiore. Quindi la regexp per il linguaggio è la somma (unione) di tutte le regexp R(n)_ij tali che lo stato j sia accettante. CVD

Ogni linguaggio definito da una regexp è definito anche da un FA.

Suppongo L=L(R) per regexp R. Dim che L=L(E) per e-NFA E con:
uno stato accettante, nessun arco entrante in q0, nessun arco uscente dall'accettante.
Base:a) >o->e->O il linguaggio è {e} b) >o O il linguaggio è 0 e c) >o->a->O accetta la sola stringa a L=L(a)
Induzione:1) R+S partendo da q0 posso andare con e o in R o in S quindi L(R)UL(S) 2) R->e->S concatenazione L(R)L(S) 3)R* (cappello) >o->e->or<-eOr->e->O ma anche >o->e->O 4) Le parentesi non cambiano il linguaggio (R)

Pumping Lemma per Linguaggi:
Sia L regolare. Allora esiste una costante n tale che, per ogni stringa w in L per la quale |w|=>n, possiamo scomporre w in tre stringhe w=xyz in modo che:
y!=e;
|xy|<=n;
per ogni k=>0 anche la stringa x (y^k) z appartiene a L
Dim: Supponiamo che L sia regolare. Allora L=L(A) per un certo DFA A con n stati. w=a1a2..am con m=> n. Per i=0,1,…,n definiamo lo stato pi come d(q0,a1a2…ai). Per il principio della piccionaia, dato che ci sono solo n stati, gli n+1 stati pi per i=0,1,…,n non possono essere tutti distinti. Per ciò ci sono due interi distinti i,j con 0<=i<j<=n tali che pi=pj e possiamo scomporre w=xyz: 1)x=a1a2…ai 2)y=ai+1…aj 3)z=aj+1…am
Se A riceve in input xy^kz per un k=>0 Se k=0 l'automa va da p0 a pi con x e poi da pj che è pi allo stato accettante quindi xz è in L. Se k>0 va fino a pi con x poi cicla k volte su pi e va all'accettante con z. Quindi xy^kz appartiene a L.
Pumping Lemma come giocatori
1)sceglie L che vuole dim non reg.
2)sceglie n
1)w in L |w|=>n
2)w=xyz con y!=e e |xy|<=n
1)vince se individua un k| xy^kz !in L

Proprietà linguaggi regolari
Unione: se L e M sono regolari lo è anche LUM. Se sono regolari sono descritti da regexp L=L(R) e M=L(S) allora LUM=L(R+S) per definizione di + nelle regexp
Complemento: se L è regolare anche !L=Sigma*-L è regolare. L=L(A) per DFA A allora !L=L(B) con B costruito uguale ad A ma con gli stati accettanti diventati non accettanti e viceversa allora w è in L(B) sse è in Q-F cioè quando non è in L(A).
Intersezione: uso de Morgan L int M= !(!LU!M) oppure costruisco un automa A che simuli Al e Am, dove gli stati sono coppie di stati dei due automi e gli stati accettanti sse erano entrambi accettanti
A=(Ql x Qm, Sigma, d, (ql,qm), Fl x Fm) e d((p,q),a)=(dl(p,a),dm(q,a)) quindi A accetta solo se w è accettato sia da Am che da Al.
Differenza: L-M=L /\ !M
Inversione: L definito da regexp E. Esiste una regexp E^R tale che L(E^R)=(L(E))^R.
Chiusura: Base: E=e oppure 0 oppure a allora E^R=E
Induzione: 1)E=E1+E2 allora E^R=E1^R+E2^R 2) E=E1E2 allora E^R=E2^RE1^R cioè (L(E1E2))^R=L(E2^RE1^R) 3) E=E1* allora E^R=(E1^R)* qualunque w in L(E) può essere scritta come w1w2..wn con wi in L(E) idem per w^R che è in L((E^R)*) con wi^R in L(E^R) e viceversa.
Omomorfismo

Verifica linguaggi:
L è vuoto? Se nell'automa non esiste un cammino dallo stato iniziale ad uno accettante. O(n^2) Oppure con regexp O(n) sapendo che R=R1+R2 è vuoto sse R1 e R2 sono vuoti, R=R1R2 se almeno uno dei due è vuoto, se R=R1* non è mai vuoto, R=(R1) sse R1 è vuoto
Appartenenza a linguaggio: se L è rappresentato da DFA si simula l'elaborazione e se accetta w è in L O(n) Se la rappresentazione è diversa si traduce in DFA in tempo O(ns^2) con NFA con s stati e |w|=n.
L è non vuoto sse FA accetta almeno una stringa di lunghezza <n [<=] L(M)<>0 perché w in L(M)
[=>]sia w in L(M) corta quanto la più corta stringa in L(M). Per PL deve essere |w|<n perché se fosse |w|>=n allora w=xyz con |xy|<n e y<>e e xy^kz in L per ogni k, se k=0 allora xz in L allora |xz|<|w|

L è infinito? FA accetta stringa di lunghezza compresa tra n e 2n [<=] w in L, per PL xy^kz in L è un insieme infinito di stringhe.[=>] L infinito=> esiste w in L| |w|=>n 1) |w|<=2n ok 2) |w|=>2n ponendo w la più corta stringa con |w|>2n Per PL si ha w=xyz e xy^kz in L, se k=0 => xz in L a) se |xz|>2n b) |xz|<2n si torna nel caso 1.

Da inferenza ad albero:
induzione su n passi per dedurre che w è in A
Base: un solo passo, esiste la produzione A->w, l'albero ha una sola foglia per ogni posizione di w e ha prodotto w e radice A, se w=e l'albero ha una solo foglia etichettata e, lecito.
Induzione: suppongo di aver dedotto che w è in A dopo n+1 passi di inferenza e che l'enunciato sia valido per tutte le stringhe x e variabili B tali che l'appartenenza di x in B si deduca in n o meno passi. Considero l'ultimo passo dell'inferenza di w in A che impega la produzione A->X1X2..Xk dove w=w1w2…wk se Xi è un terminale allora wi=Xi, se Xi è una variabile allora wi è stata già dedotta da Xi quindi esiste un albero sintattico con prodotto wi e radice Xi. Costruiamo poi un albero con radice A e prodotto w, i figli di A sono gli Xi, ciascuna Xi diventa radice di un sottoalbero con prodotto wi. Se è terminale ho albero banale con un solo nodo Xi e una sola foglia wi, altrimenti ipotesi induttiva. Quindi il prodotto totale è la concatenazione dei prodotti dei sottoalberi Xi da sinistra a destra w1w2…wk=w CVD

Da albero a derivazione: per induzione su altezza dell'albero. Base: altezza 1, radice A figli che formano w A->w deve essere una produzione quindi A=>w è derivazione di un solo passo di w da A.
Induzione: albero di altezza n>1. Radice A e figli X1X2…Xk con Xi terminale allora wi è stringa formata da Xi oppure variabile cioè radice di un sottoalbero con prodotto fatto di terminali wi di altezza <n quindi esiste una derivazione sinistra Xi=>*wi.
W=w1w2…wk. Costruisco A=>X1X2…Xk e per ogni i ho
A=>w1w2…wiXi+1Xi+2…Xk e per induzione su i: base i=0 x ipotesi A=>X1X2…Xk, per induzione vero che A=>w1w2…wi-1XiXi+1…Xk e se Xi è terminale lo sostituisco con wi e se Xi è variabile continuo con una derivazione di wi da Xi. Quando i=k il risultato è una derivazione sinistra di w da A.CVD

Da derivazione a inferenza: sia G una CFG e supponiamo esista una derivazione A=>*w con w in T*. Allora l'inferenza ricorsiva applicata a G determina che w è in A.
Per induzione su lunghezza della derivazione. Base: formata da un unico passo allora A->w è una produzione e w è soli terminali.
Induzione: valido per n o meno passi. A=>X1X2…Xk=>*w allora w=w1w2…wk.
Se Xi è un terminale allora wi=Xi Se Xi è una variabile Xi=>*wi per ipotesi induttiva wi è nel linguaggio di Xi.
Abbiamo quindi A->X1X2…Xk dove wi=Xi oppure vale l'ipotesi induttiva. Nella successiva iterazione di inferenza si scopre che w1w2…wk è in A quindi w è in A. CVD

PDA P(Q, Sigma, Tau,dn,q0,Z0) accettazione da stack vuoto a stato finale.
Se L=N(Pn) per stack vuoto allora esiste un PDA Pf | L=L(Pf) stato finale. Nuovo simbolo X0 che non appartiene all'alfabeto di stack. X0 è sia simbolo iniziale di Pf sia un segnale che indica quando Pn ha svuotato lo stack. Quando Pf vede X0 sulla sommità sa che Pm vuoterebbe lo stack sullo stesso input.
Poi nuovo stato iniziale che inserisce Z0 all'inizio.
>p0->e,X0/Z0X0->q0->...Pn…=>w,X0/e->Pf
Pf=(QU{p0,pf}, Sigma,Tau U {X0},df,p0,X0,{pf})
df(p0,e,X0)={(q0,Z0X0)}; per tutti gli stati q in Q, gli input a in Sigma o a=e, e i simboli di stack Y in Tau, df(q,a,Y) contiene tutte le coppie dn(q,a,Y); df(q,e,X0) contiene (pf,e) per ogni stato q in Q.
Dim w è in L(Pf) sse w è in L(Pn): [Se] Sappiamo che (q0,w,Z0)|-*(q,e,e) per un q. Posso inserire X0 e avere (q0,w,Z0X0)|-*(q,e,X0). Pf contiene tutte le mosse di Pn quindi vale anche in Pf. Compongo con le mosse aggiunte (p0,w,X0)|-(q0,w,Z0X0)|-*(q,e,X0)|-(pf,e,e). E Pf accetta per stato finale.
[SoloSe] Dalle regole aggiunte: l'ultima usata se lo stack di Pf contiene solo X0 e la prima può essere usata solo al primo passo. Le intermedie sono uguali a Pn e X0 è

sempre in fondo allo stack. Se X0 fosse in cima la computazione finirebbe al passo successivo. Quindi (q0,w,Z0)|-*(q,e,e) cioè w è in N(Pn) CVD
Da stato finale a stack vuoto: Se L=L(Pf) allora esiste un PDA Pn | L=N(Pn).
Sia Pn=(QU{p0,p},Sigma,Tau U {X0},dn,p0,X0) con dn(p0,e,X0)={(q0,Z0X0)}; per ogni stato q in Q, ogni simbolo a in Sigma o a=e e ogni Y=Tau, dn(q,a,Y) contiene tutte le coppie presenti in df(q,a,Y) cioè Pn simula Pf.; per tutti gli stati accettanti q in F e i simboli di stack Y in Tau (o Y= X0) dn(q,e,Y) contiene (p,e); per tutti i simboli di stack Y in Tau (o Y=X0), dn(p,e,Y)={(p,e)} giunto in p Pn elimina ogni simbolo fino a vuotarlo.
>p0->e,X0/Z0X0->q0-Pf=>F=>e,qs/e->p<loop e,qs/e con qs, qualsiasi.
Dim w è in N(Pn) sse w è in L(Pf) [Se] Supponiamo (q0,w,Z0)|-*(q,e,alpha) per stato accettante q e stringa di stack alpha. Ogni transizione di Pf è una mossa di Pn, per tenere X0 sotto i simboli di stack (q0,w,Z0X0)|-*(q,e,alphaX0) allora in Pn vale (p0,w,X0)|-(q0,w,Z0X0)|-*(q,e,alphaX0)|-*(p,e,e) w è accettata da Pn per stack vuoto.
[SoloSe]Pn può vuotare lo stack solo entrando nello stato p perché in fondo allo stack c'è X0 e Pf non ha mosse per X0.Ogni configurazione accettante è (p0,w,X0)|-(q0,w,Z0X0)|-*(q,e,alphaX0)|-*(p,e,e) dove q è accettante di Pf. Poi tra le ID (q0,w,Z0X0) e (q,e,alphaX0) tutte le mosse sono di Pf quindi (q0,w,Z0)|-*(q,e,alpha). Cioè Pf accetta w per stato finale.

CFL
Riconosciuto da PDA non deterministico perchè se è ambigua i DPDA non la riconoscono
Sostituzione alfabeto
Unione: usando il teorema di sostituzione siano L1 e L2 due CFL, allora L1UL2 è il linguaggio s(L) dove L è il linguaggio {1,2} ed s è la sostituzione definita da s(1)=L1 e s(2)=L2
Concatenazione:L1L2 è il linguaggio s(l) con L={12} e s come prima
Chiusura: L è il linguaggio {1}* e s(1)=L1; allora L1*=s(L). Uguale se L^+
Omomorfismo: s la sostituzione che rimpiazza ogni simbolo a con il linguaggio formato dalla sola stringa h(a) s(a)={h(a)}. Allora h(L)=s(L)
Inversione: G^R(V,T,P^R,S) con P^R insieme degli inversi delle produzioni in P. se A->alpha diventa A->alpha^R. Con induzione su lunghezza derivazioni L(G^R)=L^R
Intersezione con linguaggio regolare: CFL rappresentato da automa a pila e L da FA. Eseguo in parallelo un DFA e un PDA (con l'AND finale) e ottengo un altro PDA P'=(Qp x Qa, Sigma,Tau,d,(qp,qa),Z0,FpxFa) con d((q,p),a) è l'insieme di coppie ((r,s),y) tali che: s=d^ a(p,a) e (r,y) in dp(q,a,X). Con induzione su mosse fatte dal PDA ho (qp,w,Z0)|-*(q,e,y) sse ((qp,qa),w,Z0)|-*((q,p),e,y) dove p=d^(pa,w). Quindi P' accetta w sse sia P sia A accettano cioè se w è in L /\ R
Differenza con linguaggio regolare L-R=L/\!R
!L non può essere CFL poiché L1/\L2=!(!L1U!L2) e siccome non sono chiusi rispetto all'intersezione ma rispetto all'unione significa che non sono chiusi rispetto alla complementazione.
L1-L2 può non essere CFL perché Sigma* è CFL. Se L1-L2 fosse CFL allora Sigma*-L è un CFL se lo è L. Ma per un'opportuna scelta di Sigma, Sigma*-L=!L contraddizione.

Teorema di Myhill-Nerode: sono equivalenti 1)L contenuto in Sigma* + regolare 2) L è unione di qualche classe di equivalenza che sia invariante a destra e di indice finito 3) sia Rl definita per w,y in Sigma* xRly sse per ogn z in Sigma* "xy in L e yz in L" o "xz !in L e yz !in L" allora Rl ha indice finito.
Corollario Esiste ed è unico un DFA con #stati minimo
PvsNP
A si riduce a B
A->B A non può essere più difficile di B, B può essere più difficile di A
Linguaggi ricorsivi:
Complemento, intersezione, unione
Se L è RE e !L è RE L è ricorsivo
Se L non RE, !L è non RE
Se L è RE ma non ricorsivo, !L non RE
Se !L è RE ma non ricorsivo, L è non RE
Lu={(M,w)|M accetta w} è RE ma non ricorsivo
dove M è una TM codificata e M accetta la stringa w.
Esiste una turing machine universale con 3 nastri 1. (M,w) 2.per il codice del nastro simulato di M 3.per lo stato di M
U simula M su w e U accetta (M,w) sse M accetta w quindi Lu è RE.
Se Lu fosse anche ricorsivo lo sarebbe anche il suo complemento quindi esisterebbe una TM che accetta !L=Ld. Assurdo.
Lne è RE
Ld={wi|wi non è in L(Mi)} è non RE wj =j-esima parola in ordine canonico su (0+1)* Mi=TM avente come codifica la parola i-esima in ordine canonico. (i,j)=1 se TM acceta wj, 0 altrimenti. Ld complemento la diagonale.
Se L=L1UL2 è regolare => L1 e L2 sono regolari. FALSO: L1 non regolare, L2=Sigma*=>L regolare.
L1 non regolare,L2 non regolare => L1 /\ L2 è regolare. Dipende: L1 /\ 0 è regolare L1^Sigma* no.
PMM:
La f(i) è la dimensione del più lungo prefisso-suffisso scorrendo x con indice i
Aabaa
0 1 0 1 2
a
AA
aab
AabA
AAbAA
f(1)=0
ho f(1)…f(j) devo calcolare f(j+1)
bj+1=bi+1 -> f(j+1)=f(i)+1
bj+1!=bi+1 -> riallineamento
Pseudocodice:
f(1)=0
for j=2 to n
  i=f(j-1)
  while (bj<>bi+1 and i>0) do
    i=f(i)
if (b[j]<>P[i+1] and i=0) than \\match di j+1 caratteri
  f(j)=0
else f(j)=i+1

Una stringa è libera da quadrato se w=xyyz con y=e

DIM TM
Se un linguaggio L è riconosciuto da una TM M allora esiste una TM M' che genera tutte e sole le stringhe di L.
Dim: un linguaggio riconosciuto da una TM è RE. E non è richiesto che la TM si fermi in tempo finito.
Per evitare che la computazione di wi sia infinita metto in M' un generatore di coppie G1 (i,j) ordinate.
Usando l'output di G1 simulo wi su M per j passi riuscendo a costruire il generatore M'. Poichè i assume ogni possibile valore le stringhe wi vengono esaminate tutte e poichè j è finito se wi è in L allra E j tale che M accetta
wi in j passi ma se wi non in L, M si ferma senza accettare.
L deve essere ricorsivo

Stringe acc da M con n stati è
1) non vuoto se M accetta almeno una stringa di lunghezza <n
2)infinito se M accetta una qualche stringa di lunghezza tra n e 2n
1)(solo se) |w|<n accettata L(M)!=0
(se) w stringa + piccola, per PL |w|<n altrimenti sarebbe xy^kz e con k=0 potrei averne una + piccola.
2)(solo se) vale PL quindi L infinito
(se) L è infinito E |w|>=n a)<=2n cvd b)>2n prendo la più piccola, per PL w=xy^kz con k=0 xz in L i)se >2n non era la più corta. Ii)se <=2n falsa b)