

TEORIA DELLA COMPUTAZIONE 2

A.A. 2009/2010

Caputo Luca – Cimmino Giovanni – Costante Luca – Davino Cristiano – Di Giacomo Ivan – Ferri Vincenzo – Fierro Nunzio – Palo Umberto – Pepe Valeriano – Vitale Pasquale

Sommario

LEZIONE 1 – AUTOMI A PILA (PARTE 1)	3
CAPITOLO 6 – AUTOMI A PILA	3
Definizione di automa a pila	3
Descrizione istantanea di un PDA	5
LEZIONE 2 – AUTOMI A PILA (PARTE 2)	7
Linguaggi di un PDA	7
Determinismo sui PDA	7
Da stack vuoto a stato finale	8
LEZIONE 3 – AUTOMI A PILA (PARTE 3) E MACCHINE DI TURING (PARTE 1)	10
Da stato finale a stack vuoto	10
Equivalenza di PDA e CFG (grammatiche CF)	11
Dalle grammatiche agli automi a pila	11
CAPITOLO 8 - MACCHINE DI TURING	13
Notazione	13
Descrizioni istantanee delle macchine di Turing	14
Diagrammi di transizione	15
Convenzioni notazionali per le macchine di Turing	16
Linguaggio di una macchina di Turing	16
LEZIONE 4 –MACCHINE DI TURING (PARTE 2)	18
Macchine di Turing multinastro	18
Equivalenza di macchine di Turing mononastro e multinastro	19
Tempo di esecuzione	20
LEZIONE 5 – MACCHINE DI TURING (3 PARTE) – LINGUAGGI RICORSIVI (1 PARTE)	23
Macchine di Turing non deterministiche	23
CAPITOLO 9 – INDECIDIBILITÀ	26
Un linguaggio non ricorsivamente enumerabile	26
LEZIONE 6 - LINGUAGGI RICORSIVI (2 PARTE)	27
Codici per le macchine di Turing: l'importanza degli insiemi Numerabili	27
Argomento Diagonale di Cantor	27
Enumerazione delle stringhe binarie	28
Codifica della Macchina di Turing	28
Il linguaggio di diagonalizzazione	29
Dimostrazione che L_d non è ricorsivamente enumerabile	30
LEZIONE 7- LINGUAGGI RICORSIVI (3 PARTE)	31

I linguaggi ricorsivi	31
Complementi di linguaggi ricorsivi e RE	31
Il linguaggio universale	33
Indecidibilità di L_U	34
Riduzioni	35
LEZIONE 8- LINGUAGGI RICORSIVI (4 PARTE)	36
Il problema dell'arresto	36
Macchine di Turing che accettano il linguaggio vuoto	36
Il teorema di Rice e le proprietà dei linguaggi RE	38
Teorema di Rice	38
LEZIONE 9 – FUNZIONI RICORSIVE (1 PARTE)	40
CAPITOLO 12	40
Funzioni primitive ricorsive	40
Lista funzioni ricorsive primitive	41
LEZIONE 10 – FUNZIONI RICORSIVE (2 PARTE)	43
Derivazione	43
La funzione di Ackermann	43
LEZIONE 11 – FUNZIONI RICORSIVE (3 PARTE)	46
Diagonalizzazione	46
Funzione parziale ricorsiva	46
Equivalenza tra MdT e funzioni parziali ricorsive.	46
LEZIONE 12 – FUNZIONI RICORSIVE (4 PARTE)	49

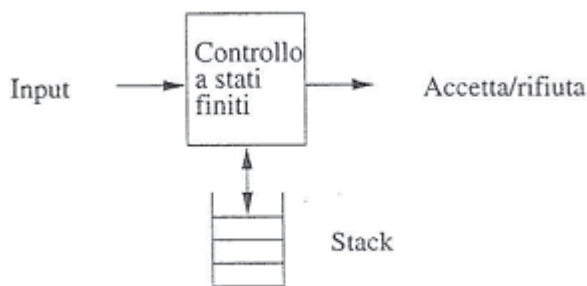
LEZIONE 1 – AUTOMI A PILA (PARTE 1)

CAPITOLO 6 – AUTOMI A PILA

Definizione di automa a pila

I linguaggi liberi dal contesto sono definiti da automi del tipo detto "a pila" (push down automaton PDA), un'estensione degli automi a stati finiti non deterministici con ε -transizioni, che è uno dei modi di definire i linguaggi regolari. In questo capitolo definiamo due versioni diverse di automa a pila: la prima accetta entrando in uno stato accettante, come fanno gli automi a stati finiti; la seconda accetta vuotando lo stack, a prescindere dallo stato in cui si trova.

Un automa a pila è un ε -NFA con una capacità aggiuntiva: uno stack in cui si può memorizzare una stringa di simboli. A differenza dell'automa a stati finiti, grazie allo stack l'automa a pila può "ricordare" una quantità illimitata di informazioni. Tuttavia, diversamente da un computer generico, che a sua volta può ricordare quantità di informazioni arbitrariamente grandi, l'automa a pila può accedere alle informazioni nello stack solo in modo "last-in-first-out" (Accesso LIFO).



Possiamo immaginare informalmente un automa a pila come il dispositivo illustrato nella Figura. Un "controllo a stati finiti" legge l'input, un simbolo per volta. L'automa a pila può osservare il simbolo alla sommità dello stack e può basare la transizione sullo stato corrente, sul simbolo di input e sul simbolo alla sommità dello stack.

In una transizione l'automa compie tre, operazioni.

1. Consuma dall'input il simbolo che usa nella transizione. Nel caso speciale di ε non si consuma alcun simbolo di input.
2. Passa a un nuovo stato, che può essere o no uguale al precedente.
3. Sostituisce il simbolo in cima allo stack con una stringa. La stringa può essere ε , che corrisponde a togliere dallo stack. Può anche essere lo stesso simbolo che c'era prima alla sommità dello stack; in questo caso lo stack non subisce nessun cambiamento effettivo. Oppure può essere un altro simbolo, con l'effetto di trasformare la sommità dello stack senza inserire o togliere. Infine il simbolo in cima allo stack può essere sostituito da due o più simboli, con l'effetto di cambiare (eventualmente) il simbolo alla sommità, e di inserire poi uno o più simboli nuovi.

Definizione formale di un automa a pila (PDA)

Un PDA include sette componenti :

Q: Insieme finito e non vuoto degli stati dell'automa (es p,q).

Σ : insieme di simboli di input all'automa (lettere minuscoli iniziali es a,b,c)

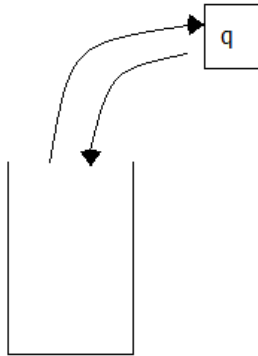
Γ : Alfabeto dello stack (lettere maiuscole finali, es: X, Y, Z)

δ : Funzione di transizione con 3 parametri

q_0 : stato iniziale dell'automa

Z_0 : Simbolo iniziale dello stack

F: Insieme degli stati finali dell'automa



Dato che nei PDA entra in gioco lo stack, la funzione di transizione sarà :

$$\delta(q, a, X) = (p, \gamma)$$

Input = q: stato corrente dell'automa; a: input della stringa, X: valore nello stack

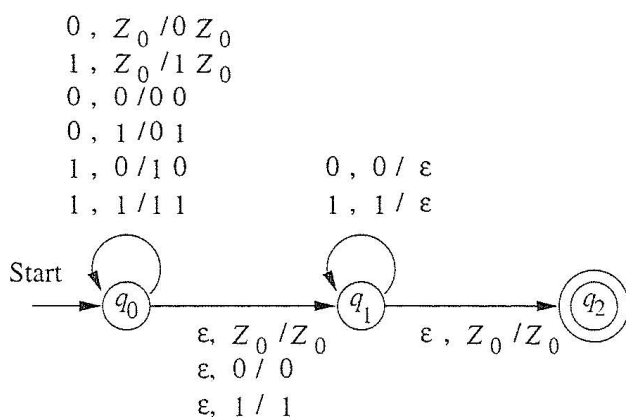
Output = p: Stato, γ : Nuovo simbolo nello stack

ESEMPIO:

$L = \{ww^R : w \in \{0,1\}^*\} \rightarrow 011110$. Non regolare.

Proseguiamo nel seguente modo

1. Partiamo da uno stato q_0 che rappresenta la "congettura" di non avere ancora esaurito la stringa w , supponiamo cioè di non aver visto la fine della stringa che deve essere seguita dal suo inverso. Finché ci troviamo nello stato q_0 leggiamo i simboli e li accumuliamo nello stack uno per volta.
2. In qualunque momento possiamo "scommettere" di aver visto la prima metà, cioè la fine di w . A questo punto w si troverà nello stack, con il suo estremo destro alla sommità e il suo estremo sinistro in fondo. Indichiamo questa scelta portandoci spontaneamente nello stato q_1 . Dato che si tratta di un automa non deterministico, facciamo in effetti due congetture: supponiamo di aver visto la fine di w , ma rimaniamo anche nello stato q_0 e continuiamo a leggere i simboli di input e a memorizzarli nello stack.
3. Una volta che ci troviamo nello stato confrontiamo il successivo simbolo di input con il simbolo alla sommità dello stack. Se corrispondono, consumiamo il simbolo di input, eliminiamo l'elemento in cima allo stack e procediamo. Se invece non corrispondono, abbiamo formulato una congettura erronea, dato che w non è seguita da w^R , come avevamo ipotizzato. Questo ramo quindi muore, sebbene altri rami dell'automa non deterministico possano sopravvivere e alla fine condurre all'accettazione.
4. Se alla fine lo stack è vuoto, abbiamo effettivamente letto w seguito da w^R . Accettiamo dunque l'input letto fino a questo punto.



Valgono le clausole seguenti.

1. I nodi corrispondono agli stati del PDA.
2. Una freccia etichettata Start indica lo stato iniziale; come per gli automi a stati finiti, gli stati contrassegnati da un doppio cerchio sono accettanti.
3. Gli archi corrispondono alle transizioni del PDA. In particolare un arco etichettato $a, X/\alpha$ dallo stato q allo stato p significa che $\delta(q, a, X)$ contiene la coppia (p, α) ed eventualmente altre coppie. In altre parole l'etichetta dell'arco indica quale input viene usato, oltre alla sommità dello stack, prima e dopo la transizione.

Descrizione istantanea di un PDA

Finora abbiamo solo una nozione informale di come un PDA "computa". Intuitivamente il PDA passa da una configurazione all'altra reagendo ai simboli di input (a o ε), ma, diversamente dagli automi a stati finiti, dove lo stato è l'unico elemento significativo, la configurazione di un PDA comprende sia lo stato sia il contenuto dello stack.

La descrizione istantanea (I.D.) è una tripla (q, w, γ) dove:

q : Stato corrente dell'automa

w : Input residuo da leggere

γ : Contenuto dello stack

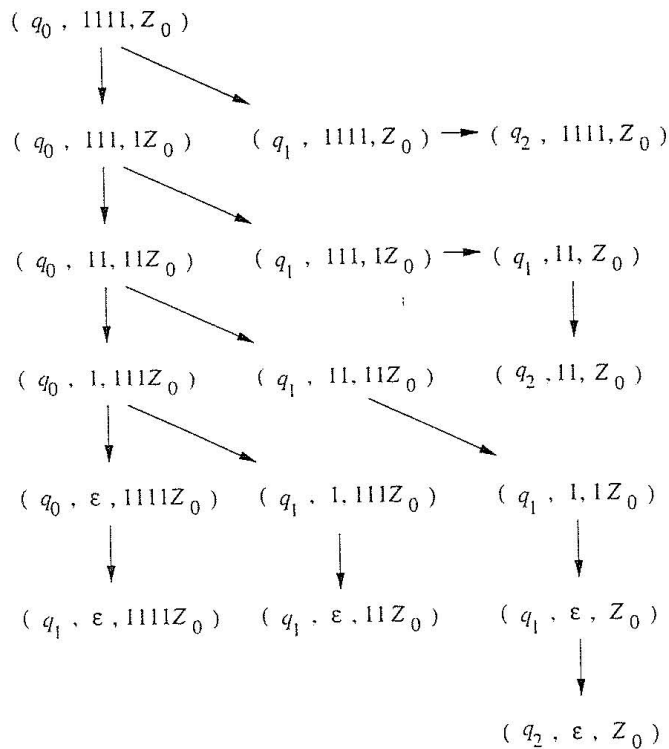
Per gli automi a stati finiti la notazione $\hat{\delta}$ è sufficiente a rappresentare sequenze di descrizioni istantanee attraverso le quali un automa si muove, perché la ID coincide con lo stato. Per i PDA, invece, occorre una notazione che descriva le trasformazioni di stato, l'input e lo stack. Definiamo con \vdash oppure \vdash_P una descrizione istantanea.

Supponiamo che $\delta(q, a, X)$ contenga (p, α) . Allora $\forall w \in \Sigma^*$ e β in Γ^* si ha che $(q, aw, X\beta) \vdash (p, w, \alpha\beta)$.

Questa mossa riflette l'idea che, consumando a (che può essere ε) dall'input e sostituendo X alla sommità dello stack con α , si può andare dallo stato q allo stato p . Notiamo che quanto rimane in input, w , e quanto sta sotto la sommità dello stack, β , non, influenza l'azione del PDA, ma viene semplicemente "conservato" e ha la possibilità di influenzare il seguito.

ESEMPIO

Consideriamo l'azione del PDA dell'Esempio precedente su input 1111. Poiché q_0 è lo stato iniziale e Z_0 è il simbolo iniziale, la ID iniziale è $(q_0, 1111, Z_0)$. Su questo input il PDA ha la possibilità di fare diverse congetture sbagliate. L'intera sequenza di ID che il PDA può raggiungere dalla ID iniziale $(q_0, 1111, Z_0)$ è rappresentata nella Figura sottostante.



Possiamo, \forall input al PDA, scegliere tutte le strade e ramificare l'automa. Arriveremo così sia a stati accettanti che a zone morte dell'automa stesso.

$$L(P) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash^* (q_f, \varepsilon \alpha)\}$$

Per accettare il linguaggio dobbiamo trovarci in uno stato finale senza avere più nulla da leggere.

Nello studio dei PDA hanno particolare importanza tre principi relativi alle ID e alle transizioni.

- ✓ Se una sequenza di ID (una computazione) è lecita per un PDA P , allora è lecita anche la computazione formata accodando una stringa (sempre la stessa) all'input (secondo componente) in ogni ID.
- ✓ Se una computazione è lecita per un PDA P , allora è lecita anche la computazione formata aggiungendo gli stessi simboli sotto quelli nello stack in ogni ID.
- ✓ Se una computazione è lecita per un PDA P , e resta una coda di input non consumata, possiamo rimuovere il residuo dall'input in ogni ID e ottenere una computazione lecita.

LEZIONE 2 – AUTOMI A PILA (PARTE 2)

Linguaggi di un PDA

Abbiamo stabilito che un PDA accetta una stringa consumandola ed entrando in uno stato accettante. Chiamiamo questa soluzione "accettazione per stato finale". Esiste una seconda via per definire il linguaggio di un PDA, che ha importanti applicazioni. Per un PDA possiamo definire il linguaggio "accettato per stack vuoto", cioè l'insieme delle stringhe che portano il PDA a vuotare lo stack, a partire dalla ID iniziale. I due metodi sono equivalenti: un linguaggio L ha un PDA che lo accetta per stato finale se e solo se L ha un PDA che lo accetta per stack vuoto. Tuttavia, per un dato PDA P , di solito i linguaggi che P accetta per stato finale e per stack vuoto sono diversi.

Un PDA ha 2 modalità di accettazione di un linguaggio:

1) Accettazione per stato finale:

Sia P un PDA. $L(P)$, il linguaggio accettato da P per stato finale è il seguente:

$$L(P) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \alpha)\} \text{ con } q_f \in F$$

-Accettiamo w se e solo se siamo in uno stato finale e non abbiamo più nulla da leggere. Il contenuto dello stack è irrilevante.

2) Accettazione per stack vuoto

Sia P un PDA. $N(P)$ il linguaggio accettato da P per stack vuoto è il seguente:

$$N(P) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$$

-Accettiamo w se e solo se abbiamo lo stack vuoto e non c'è più nulla da leggere. Lo stato q in cui ci troviamo è irrilevante.

Determinismo sui PDA

Per definizione i PDA possono essere non deterministici; il sottocaso di automa deterministico è però importante. In particolare i parser si comportano generalmente come PDA deterministici; la classe dei linguaggi accettati da questi automi è quindi interessante perché ci aiuta a capire quali costrutti sono adatti ai linguaggi di programmazione.

In modo intuitivo possiamo dire che un PDA è deterministico se in ogni situazione non c'è scelta fra mosse alternative. Le scelte sono di due tipi. Se $\delta(q, a, X)$ contiene più di una coppia, allora il PDA è sicuramente non deterministico perché si può scegliere tra queste coppie nel decidere la mossa successiva. D'altro canto, anche se $\delta(q, a, X)$ è sempre un singoletto, potremmo comunque avere la scelta tra l'uso di un vero simbolo di input oppure una mossa su ε . Definiamo quindi un DPDA (PDA Deterministici) se e solo se vengono soddisfatte le seguenti condizioni:

1) $\forall q \in Q, a \in \Sigma \cup \{\varepsilon\} \text{ e } \forall X \in \Gamma^* \text{ si ha che } |\delta(q, a, X)| \leq 1$ (in uno stato, il DPDA può effettuare solo 1 mossa)

2) Se $|\delta(q, a, X)| \geq 1$ (è non vuoto per a in Σ) $\rightarrow \delta(q, \varepsilon, X)$ dev'essere vuota, cioè se da uno stato posso effettuare una transizione su un qualsiasi simbolo dell'alfabeto, non deve esistere la transizione da quello stato con lo stesso simbolo dello stack su ε .

Es. si può dimostrare che il linguaggio L_{ww^R} è un CF per cui non esiste alcun DPDA. Collocando un segnale di mezzo tipo una c , il linguaggio diventa riconoscibile da un DPDA. Il altre parole il linguaggio $L = \{wcw^R : w \in \{0,1\}^*\}$ può essere riconosciuto da DPDA.

I DPDA accettano una classe di linguaggi che si pone tra i linguaggi regolari e i CFL. **Se L è un linguaggio regolare, allora $L = L(P)$ per un DPDA P .**

$L_N(PDA) = L_L(PDA) \cap \text{Prefisso}$.

Diciamo che un linguaggio L ha la proprietà di prefisso se \nexists due stringhe $u, v \in L$ diverse tale che u è prefisso di v .

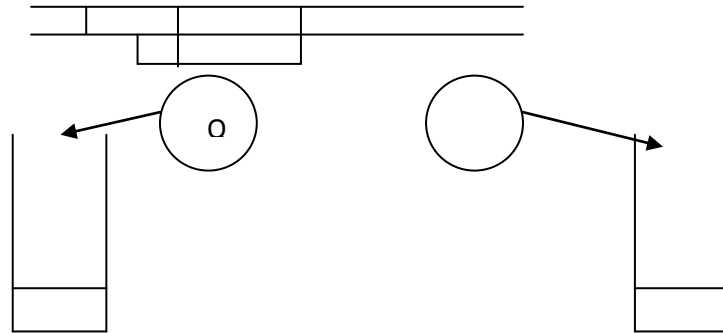
L_c è prefisso 011c110 con $c \neq 0,1$.

Esistono linguaggi molto semplici che non hanno la proprietà di prefisso. Consideriamo $\{0\}^*$, esistono coppie di stringhe delle quali una è prefisso dell'altra e dunque il linguaggio non gode della proprietà di prefisso. Non è vero che ogni linguaggio regolare è prefisso. **Un linguaggio L è $N(P)$ (accettazione per stack vuoto) per un DPDA P se e solo se L gode della proprietà di prefisso ed L è $L(P')$ per un DPDA P' .**

DIMOSTRAZIONI

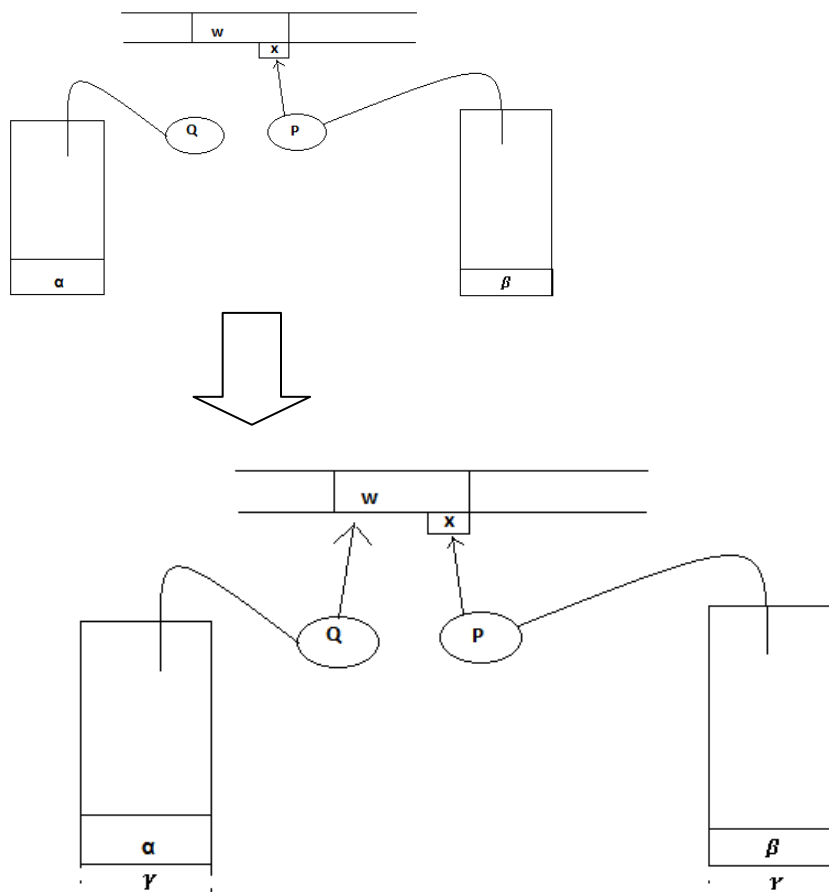
1) Se $(q, w, \alpha) \vdash^* (p, x, \beta) \rightarrow (q, wy, \alpha) \vdash^* (p, xy, \beta)$

NB: Di Questa dimostrazione è vera anche la parte "Viceversa"



2) Se $(q, w, \alpha) \vdash^* (p, x, \beta) \rightarrow (q, w, \alpha\gamma) \vdash^* (p, x, \gamma\beta)$

NB: Di questa dimostrazione non vale il "Viceversa"



Da stack vuoto a stato finale

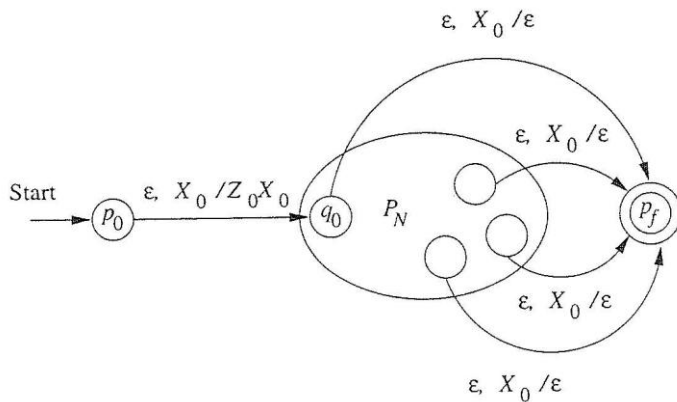
Dimostriamo che la classe dei linguaggi che sono $L(P)$ per un PDA P è uguale alla classe linguaggi che sono $N(P)$ per un PDA P . Si tratta proprio della classe dei linguaggi liberi dal contesto. La prima costruzione spiega come costruire un PDA P_F che accetta un linguaggio L per stato finale a partire da un PDA P_N che accetta L per stack vuoto.

Sia $L = N(P_N) \rightarrow \exists$ un PDA P_F tale che $L = L(P_F)$

Indico con Z_0 il simbolo di fondo del nostro P_N .

Indico con X_0 il simbolo di fondo per P_F .

Aggiungo uno stato iniziale p_0 per il nostro P_F . Aggiungo uno stato finale p_{FINALE}



Per ogni stato di P_N aggiungo una transazione che, ricevendo in input $\varepsilon, X_0 / \varepsilon$ lo porta nello stato finale del nostro automa P_F

$P_F = \{Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, q_0, X_0, \delta_F, \{p_f\}\}$

NB: Svuotando lo stack completamente eliminiamo anche Z_0 .

DIMOSTRIAMO CHE $N(P_N) = L(P_F)$

L'idea su cui poggia la dimostrazione è illustrata nella Figura della pagina precedente. Ci serviamo di un nuovo simbolo X_0 , che non deve appartenere a Γ ; X_0 è sia il simbolo iniziale di P_F sia un segnale che indica quando P_N ha svuotato lo stack. Quando P_F vede X_0 alla sommità del proprio stack, sa che P_N vuoterebbe lo stack sullo stesso input. Abbiamo inoltre bisogno di un nuovo stato iniziale, p_0 , la cui unica funzione è inserire Z_0 , il simbolo iniziale di P_N , in cima allo stack ed entrare nello stato q_0 , lo stato iniziale di " P_N ". A questo punto P_F simula P_N finché lo stack di P_N è vuoto. P_F se ne accorge perché vede X_0 alla sommità dello stack. Infine occorre un altro nuovo stato p_f che è lo stato accettante di P_F . Questo PDA entra nello stato p_f quando, scopre che P_N avrebbe svuotato il proprio stack.

$w \in N(P_N) \rightarrow w \in L(P_F)$ equivale a dire che $(q_0, w, Z_0) \vdash_{P_N}^* (q, \varepsilon, \varepsilon) \cong (p_0, w, X_0) \vdash_{P_F}^* (p_f, \varepsilon, \varepsilon)$

Partendo da $(q_0, w, Z_0) \vdash_{P_N}^* (q, \varepsilon, X_0) \vdash_{P_F}^* (p_f, \varepsilon, \varepsilon)$

(VICEVERSA)

$(p_0, w, X_0) \vdash_{P_F}^* (p_f, \varepsilon, \varepsilon)$

Partendo da $(p_0, w, X_0) \vdash_{P_F}^* (q_0, w, Z_0 X_0) \vdash_{P_N}^* (q, \varepsilon, X_0) \vdash_{P_F}^* (p_f, \varepsilon, \varepsilon)$

NB: Cancello X_0 dato che non appartiene a $P_N \rightarrow (q_0, w, Z_0) \vdash_{P_N}^* (q, \varepsilon, \varepsilon) \rightarrow$ Abbiamo infine ottenuto che $L_N(PDA) \leq L_L(PDA)$

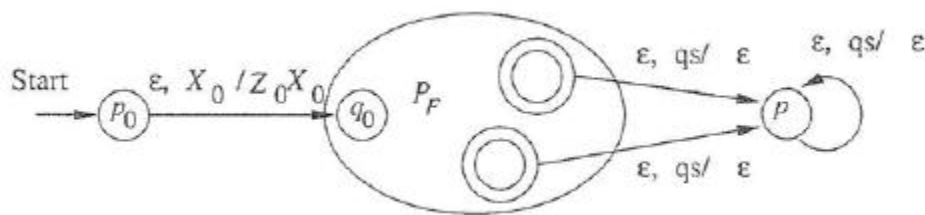
LEZIONE 3 – AUTOMI A PILA (PARTE 3) E MACCHINE DI TURING (PARTE 1)

Da stato finale a stack vuoto

Prendiamo un PDA P_F che accetta un linguaggio L , per stato finale e ne costruiamo un altro P_N che accetta L per stack vuoto. Si aggiunge una ε transizione a un nuovo stato p da ogni stato accettante di P_F . Quando si trova nello stato p , P_N svuota lo stack senza consumare input. Di conseguenza, se P_F entra in uno stato accettante dopo aver consumato l'input, P_N svuota lo stack dopo aver consumato w . Per evitare il caso in cui P_F svuota lo stack per una stringa che non va accettata, dotiamo P_N di un indicatore di fondo dello stack, X_0 . L'indicatore fa da simbolo iniziale per P_N .

Teorema

sia $L=L(P_F)$ per un PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. allora esiste un PDA P_N tale ke $L=N(P_N)$



Dimostrazione:

Sia $P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$

1. $\delta_N(p_0, \varepsilon, X_0) = \{q_0, Z_0X_0\}$: Cominciamo inserendo il simbolo iniziale di PF nello stack e andando nello stato iniziale di PF.
2. Per ogni stato q in Q , ogni simbolo di input α in Σ (o $\alpha = \varepsilon$) e ogni Y in Γ , $\delta_N(q, \alpha, Y)$ contiene tutte le coppie presenti in $\delta_F(q, \alpha, Y)$. In altre parole PN simula PF.
3. Per tutti gli stati accettanti q in F e i simboli di stack Y in Γ (o $Y = X_0$), $\delta_N(q, \varepsilon, Y)$ contiene (p, ε) . Per questa regola, ogni volta che PF accetta, PN può cominciare a vuotare il suo stack senza consumare ulteriore input.
4. Per tutti i simboli di stack Y in Γ (o $Y = X_0$), $\delta_N(q, \varepsilon, Y) = \{(p, \varepsilon)\}$. Giunto nello stato p , il che è possibile solo quando ha accettato, PN elimina ogni simbolo nel suo stack fino a vuotarlo. Non si consuma altro input.

Dobbiamo dimostrare che w è in $N(P_N)$ se e solo se w è in $L(P_F)$

(SE) supponiamo $(q_0, w, Z_0) \vdash_{P_F}^* (q, \varepsilon, \alpha)$ per uno stato accettante q e una stringa di stack α . Avvalendoci del fatto che ogni transizione di P_F è una mossa di P_N ed invocando i teorema che mantiene X_0 sotto i simboli di Γ nello stack, sappiamo che $(q_0, w, Z_0X_0) \vdash_{P_N}^* (q, \varepsilon, \alpha X_0)$ allora in P_N valgono le relazioni :

$$(p_0, w, X_0) \vdash_{P_N}^* (q_0, w, Z_0X_0) \vdash_{P_N}^* (q, \varepsilon, \alpha X_0) \vdash_{P_N}^* (p, \varepsilon, \varepsilon)$$

La prima mossa deriva dalla regola (1) della costruzione di PN, mentre l'ultima, sequenza di mosse deriva dalle regole (3) e (4). Di conseguenza w è accettata da PN per stack vuoto.

(Solo Se) PN può vuotare il proprio stack solo entrando nello stato p perché in fondo allo stack c'è X_0 e P_F non ha mosse per X_0 . P_N può entrare nello stato p solo se P_F entra in uno stato accettante. La prima mossa

di P_N e senz'altro quella derivata dalla regola (1). Ecco dunque come si configura ogni computazione accettante di P_N :

$$P_N(p_0, w, X_0) \vdash_{P_N}^* (q_0, w, Z_0 X_0) \vdash_{P_N}^* (q, \varepsilon, \alpha X_0) \vdash_{P_N}^* (p, \varepsilon, \varepsilon)$$

Dove q è stato accettante di P_F .

Oltre a ciò, tra le ID $(q_0, w, Z_0 X_0)$ e $(q, \varepsilon, \alpha X_0)$, tutte le mosse sono mosse di PF. In particolare X_0 non è mai in cima allo stack prima di raggiungere la ID $(q, \varepsilon, \alpha X_0)$. Concludiamo quindi che la stessa computazione

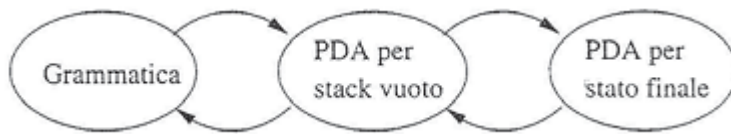
può avvenire in PF, ovviamente senza X_0 nello stack; in altre parole $(q_0, w, Z_0) \vdash_{P_F}^* (q, \varepsilon, \alpha)$. Cioè PF accetta

w , per stato finale, e quindi w è in $L(PF)$.

Equivalenza di PDA e CFG (grammatiche CF)

Dimostriamo ora che i linguaggi definiti dai PDA sono proprio i linguaggi liberi dal contesto. La via da seguire è indicata nella Figura sotto. Lo scopo è quello di provare che le tre classi di linguaggi che seguono coincidono.

1. I linguaggi liberi dal contesto, cioè quelli definiti dalle CFG.
2. I linguaggi accettati come stato finale da un PDA.
3. I linguaggi accettati per stack vuoto da un PDA



Dalle grammatiche agli automi a pila

Data una CFG G , costruiamo un PDA che ne simula le derivazioni a sinistra. Ogni forma sentenziale sinistra che non sia una stringa terminale si può scrivere nella forma xAa , dove A è la variabile più a sinistra, x è la stringa di terminali alla sua sinistra, e a è la stringa di terminali e variabili alla destra di A . Diremo che Aa è la coda di questa forma. La coda di una forma sentenziale sinistra fatta di soli terminali è ε .

La costruzione di un PDA da una grammatica poggia sull'idea di far simulare al PDA la serie di forme sentenziali sinistre usate dalla grammatica per generare una stringa terminale data w . La coda di ogni forma xAa compare sullo stack con A in cima. In quel momento x è "rappresentata" dall'aver consumato x in input, lasciando ciò che in w segue x . Quindi, se $w = xy$, in input rimane y .

Sia $G = (V, T, Q, S)$ una CFG. Costruiamo il PDA P che accetta $L(G)$ per stack vuoto:

$$P = (\{q\}, T, V \cup T, \delta, q, S)$$

La funzione di transizione δ è definita come segue.

1. Per ogni variabile A , $\delta(q, \varepsilon, A) = \{(q, \beta) \mid A \rightarrow \beta \text{ è una produzione di } G\}$
2. Per ogni terminale a , $\delta(q, a, a) = \{(q, \varepsilon)\}$.

Esempio 6.12 Convertiamo in PDA la seguente grammatica:

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

$$E \rightarrow I \mid E * E \mid E + E \mid (E)$$

Sia quindi $G = (V = \{E, I\}, T = \{a, b, 0, 1, +, *, (,)\}, Q, S = E)$

L'insieme dei terminali del PDA è $\{a, b, 0, 1, (,), +, *\}$. Questi otto simboli, con I ed E, formano l'alfabeto di stack. Definiamo la funzione di transizione.

- a) $\delta(q, \varepsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$.
- b) $\delta(q, \varepsilon, E) = \{(q, I), (q, E + E), (q, E * E), (q, (E))\}$.
- c) $\delta(q, a, a) = \{(q, \varepsilon)\}; \delta(q, b, b) = \{(q, \varepsilon)\}; \delta(q, 0, 0) = \{(q, \varepsilon)\}; \delta(q, 1, 1) = \{(q, \varepsilon)\}; \delta(q, (, () = \{(q, \varepsilon)\}; \delta(q,),) = \{(q, \varepsilon)\}; \delta(q, +, +) = \{(q, \varepsilon)\}; \delta(q, *, *) = \{(q, \varepsilon)\}$

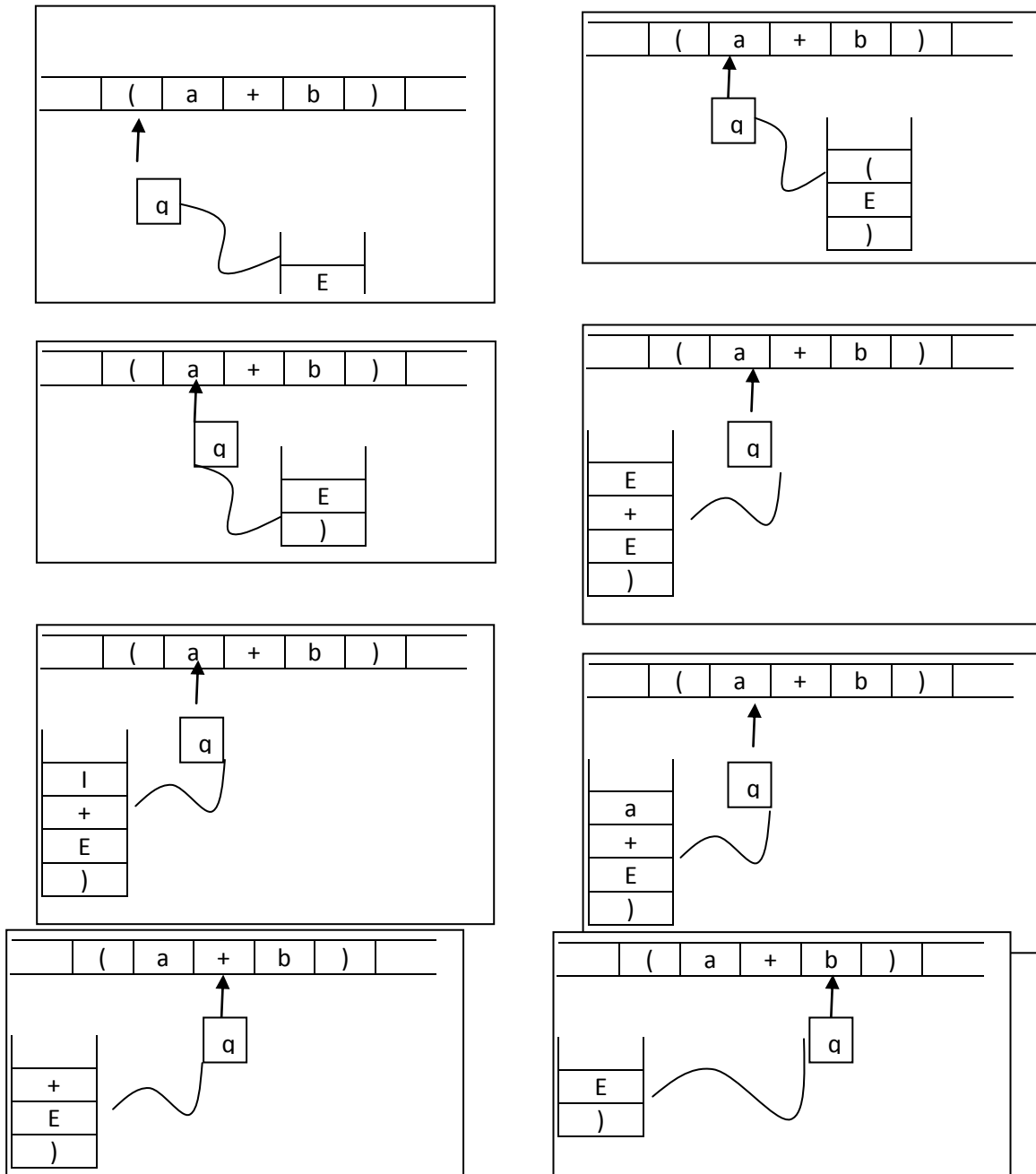
Notiamo che (a) e (b) derivano dalla regola (1), le otto transizioni di (c) dalla regola (2). Inoltre δ è l'insieme vuoto, tranne nei casi da (a) a (c).

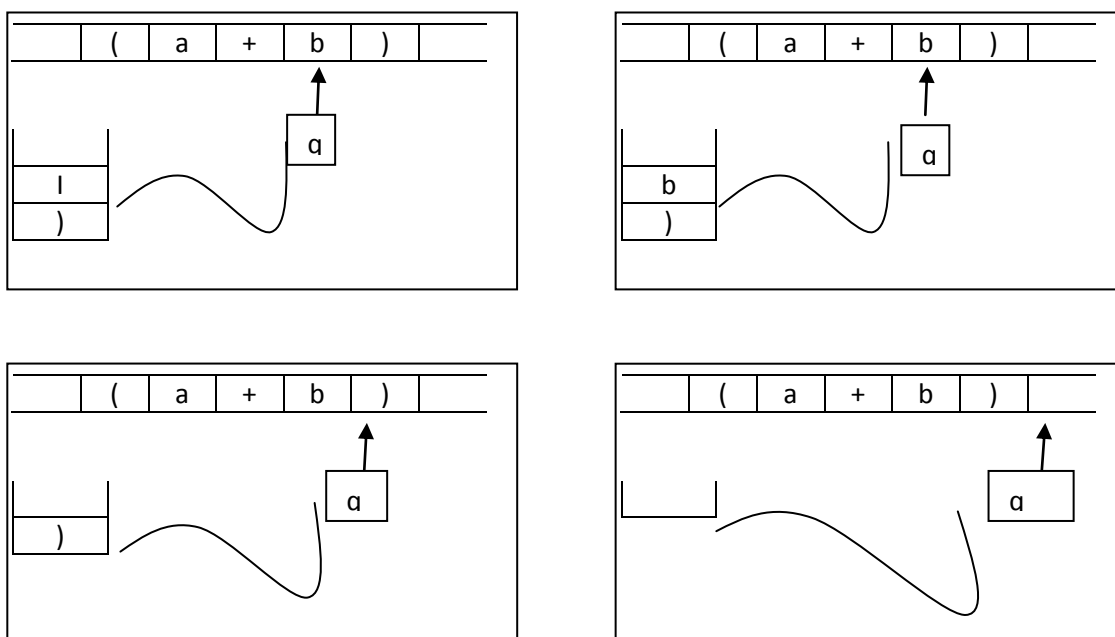
Esempio:

(a+b)

$E \rightarrow (E) \rightarrow (E + E) \rightarrow (I + E) \rightarrow (a + E) \rightarrow (a + I) \rightarrow (a + b)$

Teorema Se P è il PDA costruito dalla CFG G nel modo descritto sopra, allora $N(P)=L(G)$.



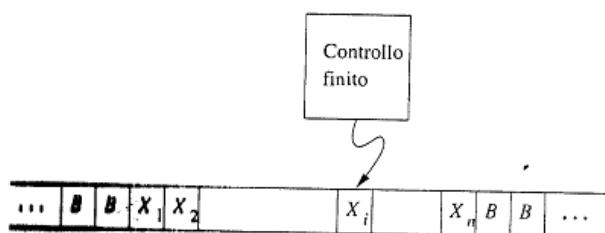


CAPITOLO 8 - MACCHINE DI TURING

Agli inizi del '900 il matematico D. Hilbert si chiese se fosse possibile trovare un algoritmo per stabilire la verità o la falsità di qualsiasi proposizione matematica. In particolare si domandò se esistesse un modo per determinare la veridicità di una formula del calcolo dei predicati del primo ordine applicata agli interi. Nel 1936 A.M. Turing propose la macchina di Turing come modello di computazione universale. Si tratta di un modello più simile a un computer che a un programma, anche se i calcolatori elettronici, o anche elettromeccanici, sarebbero arrivati alcuni anni dopo (e Turing stesso si occupò della costruzione di una macchina di questo tipo durante la seconda guerra mondiale).

Notazione

Possiamo rappresentare la macchina di Turing come nella Figura sotto. La macchina consiste di un controllo finito, che può trovarsi in uno stato, scelto in un insieme finito. C'è un nastro diviso in celle; ogni cella può contenere un simbolo scelto in un insieme finito.



L'input, una stringa di lunghezza finita formata da simboli scelti dall'alfabeto di input, viene inizialmente posto sul nastro. Tutte le altre celle, che si estendono senza limiti a sinistra e a destra, contengono all'inizio un simbolo speciale, detto blank. **Il blank è un simbolo di nastro, ma non di input**; oltre ai simboli di input e al blank, ci possono essere anche altri simboli di nastro. Una testina mobile è collocata su una delle celle del nastro.

Una mossa della macchina di Turing è una funzione dello stato del controllo e del simbolo di nastro guardato dalla testina. In una mossa la macchina di Turing compie tre azioni.

1. Cambia stato. Lo stato successivo può coincidere con lo stato corrente.
2. Scrive un simbolo di nastro nella cella che guarda. Questo simbolo sostituisce quello che si trovava in precedenza nella cella. Il nuovo simbolo può essere lo stesso che si trova già nella cella.
3. Muove la testina verso sinistra oppure verso destra. Nel nostro formalismo richiediamo un movimento: la testina non può rimanere ferma. Si tratta di una limitazione che non incide sulla capacità computazionale della macchina, dato che qualsiasi sequenza di mosse con testina fissa si può riassumere, insieme con la mossa successiva della testina, in un cambio di stato, un nuovo simbolo di nastro e un movimento a sinistra o a destra.

La notazione formale che useremo per una macchina di Turing (TM, Turing Machine) è simile a quella usata per gli automi a stati finiti o PDA. Descriviamo una TM come la settupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

i cui componenti hanno i seguenti significati.

- ✓ Q : l'insieme finito degli stati del controllo.
- ✓ Σ : l'insieme finito dei simboli di input.
- ✓ Γ : l'insieme completo dei simboli di nastro; Σ è sempre un sottoinsieme di Γ .
- ✓ δ : la funzione di transizione. Gli argomenti di δ (q, X) sono uno stato q e un simbolo di nastro X . Il valore di δ (q, X), se definito, è una tripla (p, Y, D), dove:
 - p , elemento di Q , è lo stato successivo
 - Y è il simbolo di Γ , scritto nella cella guardata, al posto di qualunque simbolo vi fosse
 - D è una direzione, L o R — rispettivamente per "left" (sinistra) e "right" (destra) — e segnala la direzione in cui si muove la testina.
- ✓ q_0 : lo stato iniziale del controllo; è un elemento di Q .
- ✓ B : il simbolo detto blank. Si trova in Γ ma non in Σ , cioè non è un simbolo di input. Inizialmente il blank compare in tutte le celle tranne quelle (finite) che contengono simboli di input.
- ✓ F : l'insieme degli stati finali o accettanti, un sottoinsieme di Q .

Descrizioni istantanee delle macchine di Turing

Per descrivere formalmente che cosa fa una macchina di Turing dobbiamo sviluppare una notazione per le configurazioni o descrizioni istantanee (ID, Instantaneous Descriptions), come quella sviluppata per i PDA. Poiché in linea di principio una TM ha un nastro infinitamente lungo, potremmo pensare che sia impossibile descrivere in termini concisi le sue configurazioni. In realtà, dopo un numero finito di mosse, la TM può aver visitato solo un numero finito di celle, anche se il numero di celle visitate può crescere oltre qualunque limite finito. Perciò in ogni ID esistono un prefisso e un suffisso infiniti di celle che non sono mai state visitate. Tutte queste celle devono contenere un blank oppure uno dei simboli di input, che sono in numero finito. **In una ID mostriamo dunque solo le celle tra il simbolo diverso dal blank più a sinistra e quello più a destra.** In casi particolari, quando la testina guarda uno dei blank in testa o in coda, anche un numero finito di blank, a sinistra o a destra della porzione non bianca del nastro dev'essere incluso nella ID.

Oltre al nastro dobbiamo rappresentare anche il controllo e la posizione della testina. A tal fine incorporiamo lo stato nel nastro e lo poniamo immediatamente a sinistra della cella guardata. Per eliminare ogni ambiguità nella stringa composta da nastro e stato, dobbiamo assicurarci di non usare come stato un simbolo che sia anche un simbolo di nastro. In ogni caso, dal momento che l'operazione della TM non dipende dai nomi degli stati, è facile cambiarli in modo che non abbiano nulla in comune con i simboli di nastro. Useremo dunque la stringa $X_1X_2...X_{i-1}X_iX_{i+1}...X_n$ (la porzione del nastro tra il simbolo diverso dal blank più a sinistra e quello più a destra

Supponiamo che $\delta(q, X_i) = (p, Y, L)$, cioè la prossima mossa è verso sinistra. Allora

$$X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash X_1 X_2 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$$

La notazione riflette il passaggio allo stato p e il fatto che ora la testina si trova sulla cella $i - 1$. Ci sono due eccezioni importanti.

1. Se $i = 1$, allora M si muove verso il blank a sinistra di X_1 . In tal caso,

$$q X_1 X_2 \dots X_n \vdash p B Y X_2 \dots X_n$$

2. Se $i = n$ e $Y = B$, allora il simbolo B scritto su X_n si unisce alla sequenza di blank in coda e non compare nella ID seguente. Perciò

$$X_1 X_2 \dots X_{n-1} q X_n \vdash X_1 X_2 \dots X_{n-2} p X_{n-1}$$

Supponiamo ora che $\delta(q, X_i) = (p, Y, R)$, cioè la prossima mossa è verso destra. Allora

$$X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash X_1 X_2 \dots X_{i-1} Y p X_{i+1} \dots X_n$$

La notazione riflette il fatto che la testina si è mossa verso la cella $i+1$. Ancora una volta ci sono 2 eccezioni importanti:

1. Se $i=n$, allora la cella $i+1$ -esima contiene un blank e non faceva parte della ID precedente. Abbiamo perciò

$$X_1 X_2 \dots X_{n-1} q X_n \vdash X_1 X_2 \dots X_{n-1} Y p B$$

2. Se $i = 1$ e $Y = B$, allora il simbolo B scritto su X_1 si unisce alla sequenza infinita di blank in testa e non compare nella ID seguente. Perciò,

$$q X_1 X_2 \dots X_n \vdash p X_2 \dots X_n$$

Esempio Definiamo La TM ch e costruiamo accetta il linguaggio $\{0^n 1^n | n \geq 1\}$

Partendo dall'estremità sinistra dell'input, cambia uno 0 in X e si muove verso destra su tutti gli 0 e Y che vede finché arriva a un 1. Cambia l' 1 in Y e si muove verso sinistra, sulle Y e gli 0, finché trova una X. A questo punto cerca uno 0 immediatamente a destra; se ne trova uno, lo cambia in X e ripete il processo trasformando un 1 corrispondente in Y. Se l' input costituito dai caratteri diversi dal blank non si trova in 0^*1^* , la TM prima o poi non ha una mossa successiva e termina senza accettare.

Stato	Simbolo				
	0	1	X	Y	B
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
q_4	—	—	—	—	—

Figura 8.9 Una macchina di Turing che accetta $\{0^n 1^n | n \geq 1\}$.

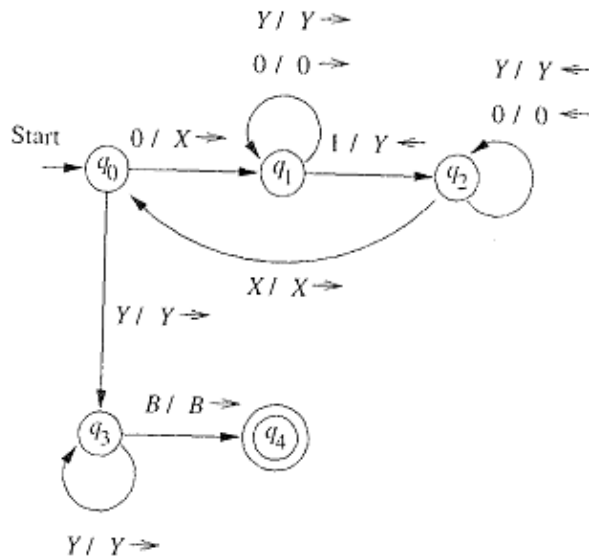
Mentre M svolge la sua computazione, la porzione del nastro su cui è passata la testina è sempre una sequenza di simboli descritta dall'espressione regolare $X^*0^*Y^*1^*$.

Diagrammi di transizione

Possiamo rappresentare le transizioni di una macchina di Turing graficamente come abbiamo fatto per i PDA. Un diagramma di transizione consiste in un insieme di nodi corrispondenti agli stati della TM. Un arco dallo stato q allo stato p è etichettato da uno o più oggetti della forma X/YD , dove X e Y sono simboli di nastro e D è una direzione, L o R . In altre parole, ogni volta che $\delta(q, X) = (p, Y, D)$, troviamo l' etichetta

X/YD sull'arco da q a p . Nei diagrammi la direzione D è rappresentata graficamente da \leftarrow per "sinistra" e \rightarrow per "destra".

Rappresentiamo lo stato iniziale con la parola "Start" e una freccia che entra in tale stato. Gli stati accettanti sono segnalati dal doppio cerchio. Dunque l'unica informazione sulla TM che non si ricava direttamente dal diagramma è il simbolo usato per il blank. Supponiamo che il simbolo sia B , salvo indicazione contraria.



Convenzioni notazionali per le macchine di Turing

I simboli usati normalmente per le macchine di Turing sono simili a quelli già visti per gli altri tipi di automa.

1. Le prime lettere minuscole dell'alfabeto indicano i simboli di input.
2. Le lettere maiuscole, di solito in prossimità della fine dell'alfabeto, sono usate per i simboli di nastro che possono essere o no simboli di input.
3. Le ultime lettere minuscole dell'alfabeto indicano stringhe di simboli di input.
4. Le lettere greche indicano stringhe di simboli di nastro.
5. Le lettere q , p e quelle circostanti indicano gli stati.

Linguaggio di una macchina di Turing

Abbiamo descritto in termini intuitivi il modo in cui una macchina di Turing accetta un linguaggio. La stringa di input viene posta sul nastro e la testina parte dal simbolo di input più a sinistra. Se la TM entra in uno stato accettante, allora l'input è accettato, altrimenti no.

In termini formali, sia $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ una macchina di Turing. Allora $L(M)$ è l'insieme di stringhe w in Σ^* tale che $q_0 w \vdash^* \alpha p \beta$ per uno stato p in F e qualunque stringa di nastro α e β . I linguaggi che possiamo accettare usando una macchina di Turing sono spesso denominati linguaggi ricorsivamente enumerabili o linguaggi RE (Recursively Enumerable). Il termine "ricorsivamente enumerabile" deriva da formalismi computazionali che precedono cronologicamente la macchina di Turing, ma che definiscono la stessa classe di linguaggi o funzioni aritmetiche.

Le macchine di Turing e l'arresto

Per le macchine di Turing si usa comunemente un'altra nozione di "accettazione": l'accettazione per arresto. Diremo che una TM si arresta se entra in uno stato q guardando un simbolo di nastro X e non ci sono mosse in questa situazione; cioè $\delta(q, X)$ è indefinito.

Caputo L. – Cimmino G. – Costante L. – Davino C. – Di Giacomo I. – Ferri V. – Fierro N. – Palo U. – Pepe V. – Vitale P.

Possiamo sempre assumere che una TM si arresti se accetta. In altre parole, senza cambiare il linguaggio accettato, possiamo rendere $\delta(q, X)$ indefinito per ogni q accettante. In generale, senza specificarlo esplicitamente, **assumiamo che una TM si arresti sempre quando si trova in uno stato accettante.**

Purtroppo non è sempre possibile richiedere che una TM si arresti, anche se non accetta. I linguaggi per cui esiste una macchina di Turing che prima o poi si arresta, indipendentemente dall'accettare o no, sono detti ricorsivi.

LEZIONE 4 –MACCHINE DI TURING (PARTE 2)

Macchine di Turing multinastro

A volte è utile considerare il nastro di una macchina di Turing come se avesse più "tracce". Ogni traccia può ospitare un simbolo, e quindi l'alfabeto di nastro della TM consiste in ennuple, con una componente per ogni traccia. Un impiego comune delle tracce multiple è di destinarne una ai dati e una seconda a opportuni segnali. Possiamo "spuntare" i simboli a mano a mano che li "usiamo" o marcare certe posizioni con un segnale. È importante perché rispetto al modello mononastro, facilita la simulazione di un vero calcolatore. Una TM multinastro si presenta come suggerito dalla Figura seguente.

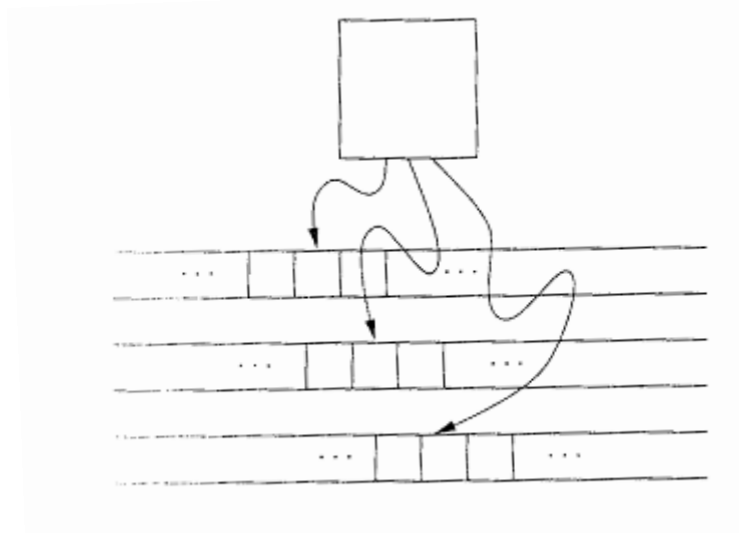


Figura 8.16 Una macchina di Turing multinastro.

Ha un controllo finito e un numero finito di nastri. Ogni nastro è diviso in celle, e ogni cella può contenere un simbolo dell'alfabeto, finito, di nastro. Come per la TM mononastro, l'insieme dei simboli di nastro comprende un blank e ha un sottoinsieme, i simboli detti di input, che non contiene il blank. L'insieme degli stati comprende uno stato iniziale e un insieme di stati accettanti. Inizialmente valgono queste condizioni:

1. L' input, una sequenza finita di simboli di input, si trova sul primo nastro.
2. Tutte le altre celle di ogni nastro contengono un blank.
3. Il controllo si trova nello stato iniziale.
4. La testina del primo nastro è all'estremo sinistro dell'input.
5. Le altre testine si trovano su celle arbitrarie. Poiché i nastri, eccetto il primo, sono bianchi, la posizione iniziale delle loro testine è irrilevante: tutte le celle sono equivalenti.

Una mossa di una TM multinastro dipende dallo stato e dai simboli letti da ciascuna testina. In una mossa la macchina compie tre operazioni.

1. Il controllo entra in un nuovo stato, che può coincidere con quello corrente.
2. Su ogni cella guardata da una testina viene scritto un nuovo simbolo di nastro, che può essere quello già contenuto nella cella.
3. Ogni testina si muove a sinistra o a destra, oppure sta ferma. I movimenti sono indipendenti: testine diverse si possono muovere in direzioni diverse o star ferme.

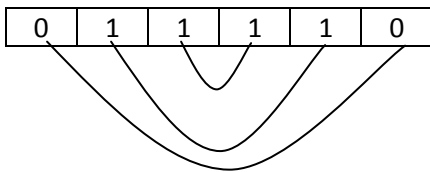
Non daremo la definizione formale delle regole di transizione, che sono una generalizzazione immediata di quelle per le TM mononastro, tranne che per la direzione, che qui è indicata dai simboli L (sinistra), R (destra) ed S (ferma). Nella macchina mononastro che abbiamo definito la testina non può stare ferma:

manca quindi il simbolo S. Le macchine di Turing multinastro accettano entrando in uno stato accettante come quelle mononastro.

Esempio.

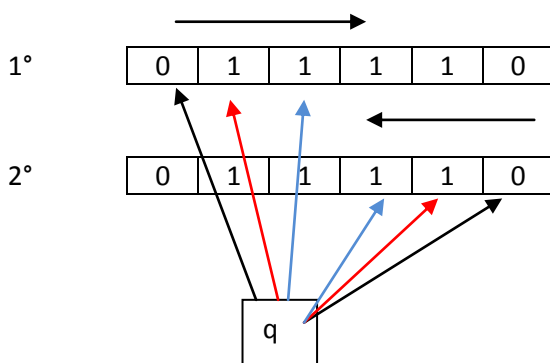
$$L = \{ww^R | w \in \{0,1\}^*\}$$

MDT tradizionale:



$$T(n) = O(n^2)$$

MDT Multinastro



$$T(n) = O(n)$$

In questo modo abbiamo guadagnato sia in semplicità di realizzazione sia in termini di complessità (tempo di esecuzione).

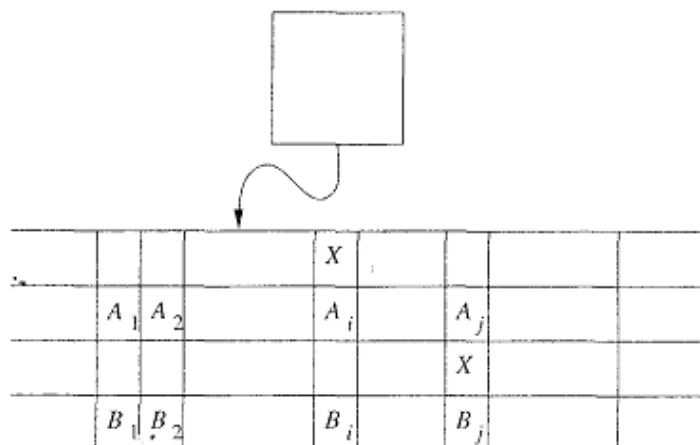
Equivalenza di macchine di Turing mononastro e multinastro

Ricordiamo che i linguaggi ricorsivamente enumerabili sono, per definizione, quelli accettati da una TM mononastro. Ovviamente le TM multinastro accettano tutti i linguaggi ricorsivamente enumerabili, perché una TM mononastro è un caso particolare di TM multinastro. Possiamo chiederci allora se esistono linguaggi non ricorsivamente enumerabili, ma accettati da una TM multinastro. La risposta è "no", e lo proveremo spiegando come simulare una TM multinastro per mezzo di una TM mononastro.

Teorema Ogni linguaggio accettato da una TM multinastro è ricorsivamente enumerabile.

DIMOSTRAZIONE La dimostrazione è illustrata nella Figura della pagina seguente.

Sia L un linguaggio accettato da una TM M con k nastri. Simuliamo M con una TM mononastro N , il cui nastro è diviso in $2k$ tracce. Metà delle tracce replicano i nastri di M ; ognuna delle restanti ospita un marcatore che indica la posizione corrente della testina del corrispondente nastro di M . Nella Figura ipotizziamo $k = 2$. La seconda e la quarta traccia replicano il contenuto del primo e del secondo nastro di M ; la traccia 1 reca la posizione della testina del nastro 1; la traccia 3 la posizione della testina del secondo nastro.



Per simulare una mossa di M , la testina di N deve visitare i k marcatori. Per non smarrirsi, N deve tener conto di quanti marcatori stanno alla sua sinistra in ogni istante; il contatore è conservato come componente del controllo di N . Dopo aver visitato ogni marcatore e memorizzato, in una componente del controllo, il simbolo nella cella corrispondente, N sa quali simboli di nastro sono guardati dalle testine di M . Inoltre N conosce lo stato di M memorizzato nel suo controllo. Perciò N può stabilire la mossa successiva di M . Ora N rivisita ogni marcatore, modifica il simbolo sulla traccia che rappresenta il nastro corrispondente di M e sposta, se richiesto, ciascun marcatore a destra o a sinistra. Infine cambia lo stato di M registrato nel suo controllo. A questo punto N ha simulato una mossa di M . Gli stati accettanti di N sono quelli in cui è registrato uno stato accettante di M . In questo modo N accetta quando, e solo quando, M accetta.

Tempo di esecuzione

Introduciamo un concetto che diventerà importante in seguito: la "complessità in tempo", o "tempo di esecuzione", di una macchina di Turing. Diciamo che il tempo di esecuzione della TM M sull'input w è il numero di passi che M compie prima di arrestarsi. Se M non si arresta su w , diciamo che il tempo di esecuzione è infinito. La complessità in tempo della TM M è la funzione $T(n)$, definita come il massimo, su tutti gli input w di lunghezza n , del tempo di esecuzione di M su w . Per una macchina di Turing che non si arresta su tutti gli input, $T(n)$ può essere infinito per alcuni n , o anche per tutti. La TM mononastro può avere tempi di esecuzione molto più lunghi di quella multinastro. D'altra parte i tempi di esecuzione delle due macchine sono commensurabili in senso debole: quello della TM mononastro non supera il quadrato del tempo dell'altra.

Dopo n mosse di M i marcatori delle testine non possono essere distanti fra loro più di $2n$ celle. Perciò, partendo dal marcatore più a sinistra, per trovarli tutti N non deve spostarsi di più di $2n$ celle a destra. A quel punto può tornare verso sinistra modificando il contenuto dei nastri simulati di M e spostando i marcatori a sinistra o a destra. Questa operazione non richiede più di $2n$ movimenti a sinistra, oltre a non più di $2k$ mosse nell'altra direzione per scrivere un marcatore X nella cella a destra (se una testina di M si sposta a destra).

Quindi il numero di mosse che N deve fare per simulare una delle prime n mosse non supera $4n + 2k$. Poiché k è una costante, indipendente dal numero di mosse simulate, questo numero è $O(n)$. La simulazione di n mosse richiede non più di n volte quella quantità, cioè $O(n^2)$.

Una macchina di turing è utilizzata per

a) Accettare linguaggi

	w	
--	---	--

 è vero se arrivo a $q \in F$ stato finale

b) Calcolare funzioni

	X_1	$*$	X_2	$*$	\dots	$*$	X_n	
--	-------	-----	-------	-----	---------	-----	-------	--

Si arriva a

*	$X_1 \dots X_n$	
---	-----------------	--

Con * separatore

c) Enumerare elementi di un linguaggio

$$\underline{W_1 * W_2 * W_3 \dots * W_n}$$

Con * separatore


Es. macchina di Turing per la somma di 2 interi codificati in unario (si inserisce il simbolo * come separatore).

B	*	1	...	1	*	1	1	...	1
		$\underbrace{\hspace{10em}}$				$\underbrace{\hspace{10em}}$			
		n				m			

n

m

B	*	1	1	1	1	B
---	---	---	---	---	-----	-----	---	---



 $m+n$

 $m+n$

Es 3+2

B	*	1	1	1	*	1	1	B
---	---	---	---	---	---	---	---	---

*	1	1	1	1	1	*
---	---	---	---	---	---	---

Es. 8.24

In questo esercizio esploriamo l'equivalenza tra la computazione di funzioni e il riconoscimento di linguaggi per le macchine di Turing. Per semplicità consideriamo solo funzioni da interi non negativi a interi non negativi, ma le idee sottiacenti a questo problema si applicano a qualsiasi funzione computabile. Ecco le due definizioni cruciali.

- ✓ Il grafo di una funzione f sia l'insieme di tutte le stringhe della forma $[x, f(x)]$, dove, x è un intero non negativo in binario e $f(x)$ è il valore della funzione f con argomento x , scritto in binario.
- ✓ Si dice che una macchina di Turing computa la funzione f se, partendo da un intero non negativo x sul nastro, in binario, si arresta (in qualunque stato) con $f(x)$, in binario, sul nastro.

$$L_f = (1, 2)(3, 4) \dots (i, i+1)$$

Rispondete ai seguenti quesiti con costruzioni informali, ma chiare.

- Mostrate come, data una TM che computa f , si può costruire una TM che accetta il grafo di f come linguaggio.
- Mostrate come, data una TM che accetta il grafo di f , si può costruire una TM che computa f .
- Si dice che una funzione è parziale se può essere indefinita su certi argomenti. Se estendiamo le idee di questo esercizio alle funzioni parziali, non richiediamo che la TM che computa f si arresti qualora il suo input x sia uno degli interi per cui il valore $f(x)$ non è definito. Le vostre costruzioni per le parti (a) e (b) sono valide se la funzione f è parziale? In caso negativo spiegate come modificare la costruzione in modo che funzioni.

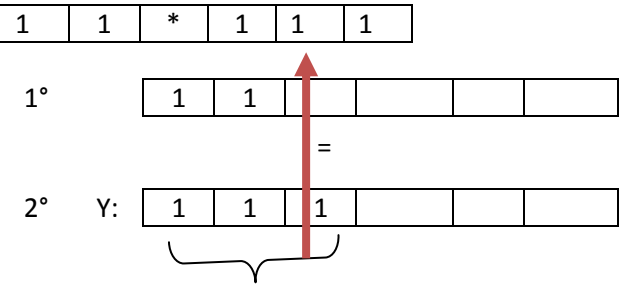
Dimostrazione

M_2	$\langle x \rangle$	$*$	$\langle f(x) \rangle$
-------	---------------------	-----	------------------------

Nel 1 nastro ho il valore di x . Nel 2 lo incremento di 1 e lo verifico con il valore di $f(x)$.

Es: Input:

Caputo L. – Cimmino G. – Costante L. – Davino C. – Di Giacomo I. – Ferri V. – Fierro N. – Palo U. – Pepe V. – Vitale P.



Se m accetta allora $y=f(x)$

LEZIONE 5 – MACCHINE DI TURING (3 PARTE) – LINGUAGGI RICORSIVI (1 PARTE)

Macchine di Turing non deterministiche

Una macchina di Turing non deterministica (NTM, Non deterministic Turing Machine) si distingue da quella deterministica nella funzione di transizione δ , che associa a ogni stato q e a ogni simbolo di nastro X un insieme di triple

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \dots, (q_k, Y_k, D_k)\}$$

dove k è un intero finito. A ogni passo una NTM sceglie una delle triple come mossa (non può scegliere però lo stato da una tripla, il simbolo nastro da un'altra e la direzione da una terza).

Il linguaggio accettato da una NTM M è definito, in modo analogo ad altri modelli non deterministici già trattati, come gli NFA e i PDA: M accetta un input w se c'è una sequenza di scelte che conduce dalla ID iniziale con w come input a una ID con stato accettante.

$$w \in L(M) \text{ se } q_0 w \vdash^* \alpha q_f \beta \quad q_f \in F$$

Come per gli NFA e i PDA, l'esistenza di scelte alternative che non portano a uno stato accettante è irrilevante. Le NTM non accettano linguaggi che non siano accettati anche da TM deterministi (DTM). La dimostrazione consiste nel costruire, per ogni NTM M_N una DTM M_D che esamina le ID raggiungibili da M_N per tutte le scelte possibili. Se M_D ne trova una con stato accettante, entra in uno dei propri stati accettanti. M_D deve procedere sistematicamente, collocando le nuove ID in una coda anziché in uno stack, in modo da simulare in un tempo finito tutte le sequenze di k mosse di M_N , per $k = 1, 2, \dots$

Teorema: Se M_N è una macchina di Turing non deterministica, esiste una macchina di Turing deterministica M_D tale che $L(M_N) = L(M_D)$.

DIMOSTRAZIONE Costruiamo M_D come TM multinastro, secondo lo schema della Figura sotto.

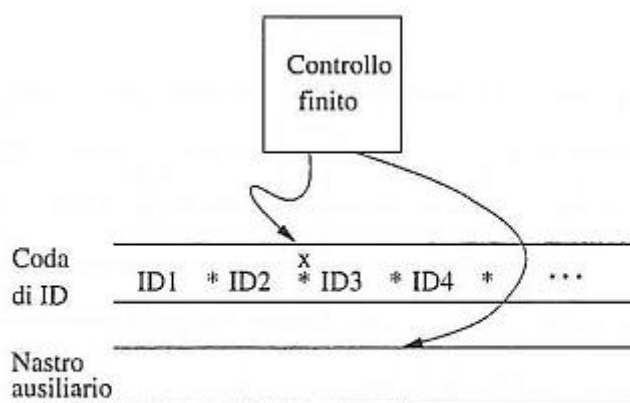


Figura 8.18 Simulazione di una NTM da parte di una DTM.

Il primo nastro di M_D contiene una sequenza di ID di M_N che comprendono lo stato di M_N . Una ID di M_N è contrassegnata come "corrente", nel senso che le sue ID successive sono in corso di elaborazione. Nella Figura la terza ID è marcata da un x e dal separatore di ID, $*$. Tutte le ID a sinistra della ID corrente sono già state elaborate e si possono ignorare.

Per elaborare la ID corrente, M_D compie quattro operazioni:

1. Esamina lo stato e il simbolo guardato della ID corrente. Nel controllo di M_D sono incorporate le scelte di mossa di M_N per ogni stato e simbolo. Se lo stato nella ID corrente è accettante, M_D accetta e smette di simulare M_N .
2. Se lo stato non è accettante, e la combinazione stato-simbolo permette k mosse, M_D copia la ID sul secondo nastro e fa k copie della ID in coda alla sequenza di ID sul nastro 1.
3. M_D modifica ognuna delle k ID secondo una delle k scelte di mossa che M_N può fare dalla ID corrente.
4. M_D torna alla ID corrente contrassegnata in precedenza, cancella il contrassegno e lo sposta alla successiva ID a destra. Il ciclo ricomincia dal passo (1).

La fedeltà della simulazione dovrebbe essere evidente: M_D accetta solo se scopre che M_N può raggiungere una ID accettante. Dobbiamo però essere certi che se M_N entra in una ID accettante dopo una sequenza di n mosse, quella ID diventa prima o poi la ID corrente di M_D , che quindi accetta.

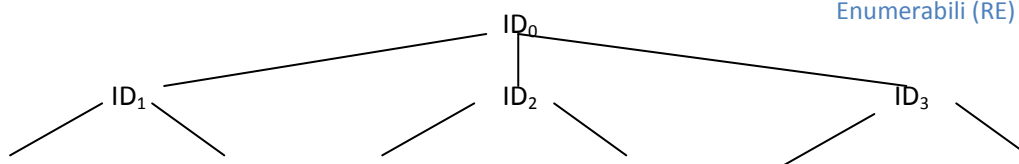
Visita in ampiezza

1

$\leq k$

...

$\leq k^n$



Linguaggi
Ricorsivamente
Enumerabili (RE)

Sia m il massimo numero di scelte di M_N nelle sue configurazioni. Allora in M_N ci sono una ID iniziale, non più di m ID raggiungibili in una mossa, non più di m^2 ID raggiungibili in 2 mosse, e così via. Dopo n mosse, M_N può aver raggiunto al massimo $1+m+m^2+\dots+m^n$ ID. questo numero non supera nm^n . M_D esplora le ID di M_N nell'ordine detto "in ampiezza": prima tutte le ID raggiungibili in 0 mosse (la ID iniziale), poi quelle raggiungibili in una mossa, poi quelle in due mosse, e così via. In particolare M_D fa diventare corrente, esaminandone le successive, ogni ID raggiungibile in non più di n mosse prima di elaborarne una raggiungibile in più di n mosse. Di conseguenza la ID accettante di M_N viene considerata da M_D fra le prime nm^n . A noi importa solo che M_D elabori questa ID dopo un tempo finito; il limite che abbiamo stabilito garantisce che prima o poi quella ID sarà elaborata. Perciò se M_N accetta, accetta anche M_D . Poiché abbiamo già notato che M_D accetta solo se M_N accetta, concludiamo che $L(M_N)=L(M_D)$.

Si osservi che la TM deterministica così costruita può impiegare un tempo esponenzialmente più alto della TM non deterministica. Non sappiamo se questo rallentamento esponenziale sia inevitabile.

Una macchina di turing può:

1. Accettare linguaggi

	w	
--	---	--

 è vero se arrivo a $q \in F$ stato finale

2. Calcolare funzioni

$\langle x \rangle$		$\langle f(x) \rangle$
---------------------	--	------------------------

3. Enumerare elementi di un linguaggio
(elencare tutte le stringhe di un linguaggio)

w_1	*	w_2	*	...	*	w_n
-------	---	-------	---	-----	---	-------

 Con * separatore

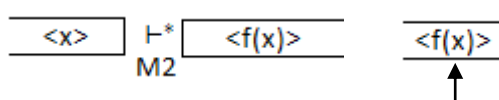
Equivalenze: $1 \approx 2$ e $1 \approx 3$

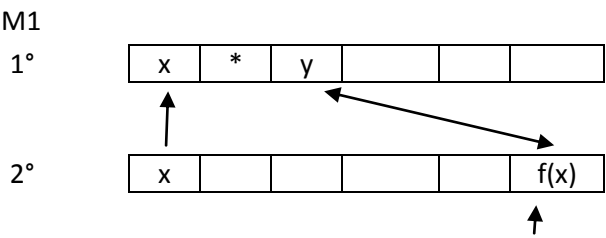
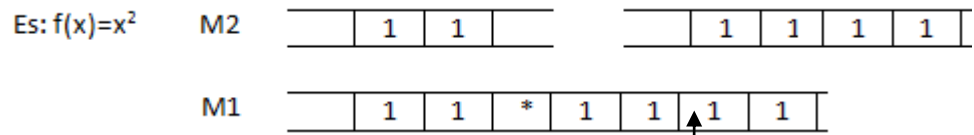
Vediamo $1 \approx 2$:

M_2 che calcola $f(x) \leftrightarrow M_1$ che accetta $L=\{(x, f(x))\}$

Se f è parziale possono verificarsi cicli per cui la MT non si ferma per certi input.

$\Rightarrow M_2$





Simula il comportamento di M2 ed accetta se $y=f(x)$. Questa simulazione funziona anche con funzioni parziali poiché se $f(x)$ non è definita allora ogni coppia $x*y \notin L(M1)$

$\leq M1$ accetta $L \rightarrow \exists M2$ calcola f

M1

x	*	y
---	---	---

 accetta se $y=f(x)$

M2

$\langle x \rangle$

 \vdash^*

$\langle f(x) \rangle$

M2

M2

x

x	*	i
---	---	---

Simula M1, se M1 accetta $(x,i) \rightarrow i=f(x)$

Questo funziona se f è definita sempre. Se invece f è parziale $x*y$ e $y \neq f(x)$ la macchina cicla all'infinito.

Per evitare che la macchina non si fermi definiamo la seguente tabella:

	1	2	3	...	n	...		mosse
1								
2								
...								
i								
...								
ID								

La simulazione viene eseguita in modo obliquo per evitare di ciclare all'infinito, infatti se visitassimo la tabella per righe in caso di cicli (rami infiniti) la macchina di Turing non si arresterebbe mai.

CAPITOLO 9 – INDECIDIBILITÀ

Distinguiamo dunque i problemi che possono essere risolti da una macchina di Turing in due classi: quelli per cui c'è un algoritmo (cioè una macchina di Turing che si arresta, a prescindere dal fatto di accettare o no il suo input) e quelli che vengono risolti solo da macchine di Turing che possono girare per sempre su input non accettati. La seconda forma di accettazione è problematica. Per quanto a lungo la TM giri, non è infatti dato sapere se l' input è accettato o no. Ci concentriamo dunque sulle tecniche per dimostrare che un problema è "indecidibile", ossia che non ha un algoritmo, indipendentemente dal fatto che sia accettato o no da una macchina di Turing che non si arresta su un certo input.

Un linguaggio non ricorsivamente enumerabile

Ricordiamo che un linguaggio L è ricorsivamente enumerabile (RE) se $L = L(M)$ per una TM M . Il nostro obiettivo finale è dimostrare l'indecidibilità del linguaggio formato dalle coppie (M, w) tali che:

1. M è una macchina di Turing (adeguatamente codificata in binario) con alfabeto di input $\{0, 1\}$
2. w è una stringa di 0 e di 1
3. M accetta l'input w .

Se questo problema, con input limitati all'alfabeto binario, è indecidibile, allora senz'altro è indecidibile anche il problema più generale, in cui le TM possono avere qualsiasi alfabeto.

Il linguaggio L_d è il "linguaggio di diagonalizzazione", che consiste di tutte le stringhe w tali che la TM rappresentata da M non accetta l'input w . Dimostreremo che non c'è alcuna TM che accetti L_d . Ricordiamo che la non esistenza di una macchina di Turing per un linguaggio è una proprietà più forte dell'indecidibilità del linguaggio (cioè della non esistenza di un algoritmo o di una TM che si arresta sempre).

27

Caputo L. – Cimmino G. – Costante L. – Davino C. – Di Giacomo I. – Ferri V. – Fierro N. – Palo U. – Pepe V. – Vitale P.

A questo punto emerge una contraddizione ed è nata dall'aver posto $\wp(N)$ insieme numerabile, quindi $\wp(N)$ è un insieme non numerabile.

Enumerazione delle stringhe binarie

Assegneremo numeri interi a tutte le stringhe binarie in modo che ciascuna stringa corrisponda a un unico intero e ciascun intero corrisponda a un' unica stringa. Se w è una stringa binaria; trattiamo $1w$ come un intero binario i . Chiameremo allora w la i -esima stringa. In altre parole ε è la prima stringa, 0 la seconda, 1 la terza ,00 la quarta, e così via.

Esempio:

dove $0 < 1$

ε	=	w_1	
0	=	w_2	$f:w \rightarrow \langle 1w \rangle_2$
1	=	w_3	$10 \rightarrow \langle 110 \rangle_2 = 6$
00	=	w_4	
01	=	w_5	
10	=	w_6	
11	=	...	

Con questo criterio le stringhe risultano ordinate per lunghezza, con le stringhe di lunghezza uguale ordinate lessicograficamente. D'ora in avanti ci riferiremo all' i -esima stringa come w_i .

Codifica della Macchina di Turing

Il prossimo obiettivo è ideare un codice binario per le macchine di Turing così che ogni TM con alfabeto di input $\{0,1\}$ possa essere considerata come una stringa binaria. Per rappresentare una TM $M = \{Q, \{0,1\}, \Gamma, \delta, q_1, B, F\}$ come una stringa binaria, è necessario assegnare interi agli stati, ai simboli di nastro e alle direzioni L ed R.

- ✓ Supporremo che gli stati siano q_1, q_2, \dots, q_r per un certo r . Lo stato iniziale sarà sempre q_1 , e q_2 sarà l'unico stato accettante. Avendo ipotizzato che la TM si arresti quando è in uno stato accettante, non c'è mai bisogno di più di uno stato accettante.
- ✓ Supporremo che i simboli di nastro siano X_1, X_2, \dots, X_s per un certo s . X_1 sarà sempre il simbolo 0, X_2 sarà 1 e X_3 sarà B, il blank. Gli altri simboli di nastro possono essere assegnati arbitrariamente agli altri interi
- ✓ Ci riferiremo alla direzione L come D1 e alla direzione R come D2.

Dato che per ogni TM M possiamo assegnare interi ai suoi stati e ai simboli di nastro, in molti modi diversi, per una TM ci sarà più di una codifica. Si tratta comunque di un elemento irrilevante in quanto segue, perché mostreremo che nessuna codifica può rappresentare una TM M t.c. $L(M)=L_d$.

Una volta scelti gli interi che rappresentano stati, simboli e direzioni, possiamo codificare la funzione di transazione δ . Supponiamo che una regola di transazione sia $\delta(q_i, X_j) = (q_k, X_l, D_m)$, per certi valori interi i, j, k, l ed m . Codifichiamo questa regola per mezzo della stringa $0^i 10^j 10^k 10^l 10^m$. Poiché i, j, k, l ed m valgono

Caputo L. – Cimmino G. – Costante L. – Davino C. – Di Giacomo I. – Ferri V. – Fierro N. – Palo U. – Pepe V. – Vitale P.

ciascuno almeno 1, nel codice di una transazione non compaiono mai due o più 1 consecutivi. Un codice per l'intera TM M consiste di tutti i codici delle transazioni in un ordine fissato, separati da coppie di 1:

$$C_1 11 C_2 11 \dots C_{n-1} 11 C_n$$

dove ognuna delle C è il codice di una transazione di M.

Esempio Consideriamo la TM

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$$

dove δ consiste nelle regole:

$$\delta(q_1, 1) = (q_3, 0, R)$$

$$\delta(q_3, 0) = (q_1, 1, R)$$

$$\delta(q_3, 1) = (q_2, 0, R)$$

$$\delta(q_3, B) = (q_3, 1, L)$$

I codici per le regole sono, rispettivamente :

0100100010100

0001010100100

0001001001010

000100010010010

Per esempio la prima regola può essere scritta come $\delta(q_1, X_2) = (q_3, X_1, D_2)$ in quanto $L = X_2$, $0 = X_1$ e $R = D_2$. Di conseguenza il suo codice è $0^1 10^2 10^3 10^1 10^2$.

010010001010011000101010010011000100100101011000100010010010 questa è la stringa associata alla MT, siamo riusciti a codificare una MT in una stringa binaria, essendo l'insieme delle stringhe binarie un insieme numerabile ne consegue che anche l'insieme delle MT è un insieme numerabile. Poiché nessun codice valido di una TM contiene tre 1 in fila, possiamo essere certi che la prima occorrenza di 111 separa il codice di M da w.

Il linguaggio di diagonalizzazione

Codificando le macchine di Turing, disponiamo quindi di una nozione concreta di M_i , la "i-esima macchina di Turing": la TM M il cui codice è w_i , l' i-esima stringa binaria. Molti interi non corrispondono a nessuna TM. Per esempio 11001 non comincia per 0 e 0010111010010100 ha tre 1 consecutivi. Se w_i non è un codice di TM valido, consideriamo M_i come la TM con un unico stato e nessuna transazione. In altre parole, per questi valori di i, M_i è la macchina di Turing che si arresta immediatamente su qualunque input. Di conseguenza $L(M_i)$ è l'insieme vuoto se w_i non è un codice valido.

A questo punto possiamo dare una definizione fondamentale.

Il linguaggio L_d , detto linguaggio di diagonalizzazione, è l'insieme delle stringhe w_i tali che w_i non è in $L(M_i)$.

In altre parole L_d , consiste di tutte le stringhe w tali che la TM M con codice w non accetta quando riceve in input la stringa. La ragione per cui L_d , è detto linguaggio di "diagonalizzazione" è chiara dalla figura. La tabella indica per ogni i e j se la TM M_i accetta la stringa di input w_j ; 1 significa "si accetta" e 0 significa "no non accetta".

	1	2	3	4	...
1	0	1	1	0	...
2	1	1	0	0	...
3	0	0	1	1	...
4	0	1	0	1	...
...

Figura 9.1 La tabella che rappresenta l'accettazione di stringhe da parte delle macchine di Turing.

Possiamo interpretare l' i -esima riga come il vettore caratteristico del linguaggio $L(M_i)$; in altre parole gli 1 in questa riga indicano le stringhe che appartengono al linguaggio. I valori sulla diagonale segnalano M_i accetta w_i . Per costituire L_d , completiamo la diagonale. Perciò L_d contenebbe $w_1 = \varepsilon$, e non le stringhe da w_4 che sono 0100, e via di seguito.

Il metodo di complementare la diagonale per costruire il vettore caratteristico di un linguaggio che non compare in nessuna riga è detto diagonalizzazione. Esso funziona perché il complemento della diagonale è di per sé un vettore caratteristico che descrive l'appartenenza ad un determinato linguaggio, e cioè L_d .

Quindi il complemento della diagonale non può essere il vettore caratteristico di una macchina di Turing. (In effetti non stiamo applicando lo stesso metodo che c'è alla base della dimostrazione se un insieme è numerabile o meno)

Dimostrazione che L_d non è ricorsivamente enumerabile

$$L_d = \{ w_i | w_i \notin L(M_i) \}$$

$$\overline{L_d} = \{ w_i | w_i \in L(M_i) \}$$

L_d non è un linguaggio ricorsivamente enumerabile. In altre parole non esiste alcuna macchina di Turing che accetta L_d . Nessuna macchina di Turing accetta L_d .

$$L_d = \{ w_i | w_i \notin L(M_i) \}$$

oppure

$$L_d = \{ \langle M \rangle | \langle M \rangle \notin L(M) \}$$

Dim.

Supponiamo $L_d = L(M)$ per una TM M . Poiché L_d è un linguaggio sull'alfabeto $\{0,1\}$ M rientra nell'elenco di macchine di Turing che abbiamo costruito, dato che questo elenco include tutte le TM con alfabeto di input $\{0,1\}$. Di conseguenza esiste almeno un codice per M , poniamo i ; ossia $M = M_i$.

Chiediamoci ora se w_i è in L_d . Se w_i è in L_d allora M_i accetta w_i . Per tanto per la definizione di L_d , w_i non è in L_d poiché L_d contiene solo le stringhe w_j tali che M_j non accetta w_j . Analogamente, se w_i non è in L_d allora M_i non accetta w_i . Di conseguenza per la definizione di L_d , w_i è in L_d .

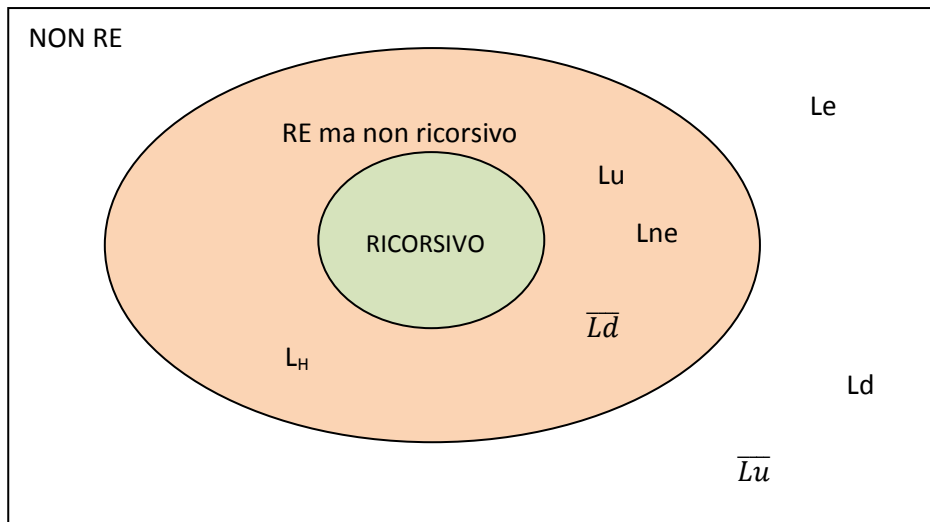
Poiché w_i non può essere né non essere in L_d , abbiamo una contraddizione con l'ipotesi che M esista. In altre parole L_d non è un linguaggio ricorsivamente enumerabile.

LEZIONE 7- LINGUAGGI RICORSIVI (3 PARTE)

I linguaggi ricorsivi

Diremo ricorsivo un linguaggio L se $L = L(M)$ per una macchina di Turing M tale che:

1. se w è in L , allora M accetta (e dunque si arresta)
2. se w non è in L , allora M si arresta pur non entrando in uno stato accettante.



La Figura illustra la relazione fra tre classi di linguaggi:

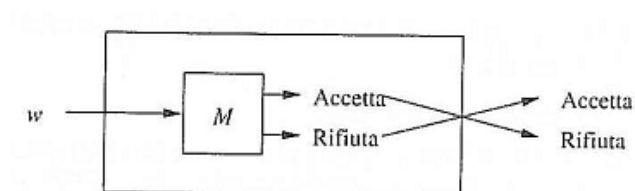
- 1) linguaggi ricorsivi
- 2) linguaggi ricorsivamente enumerabili ma non ricorsivi
- 3) linguaggi non ricorsivamente enumerabili (non RE).

Complementi di linguaggi ricorsivi e RE

Mostreremo che i linguaggi ricorsivi sono chiusi rispetto all'operazione di complementazione. Perciò se un linguaggio L è RE ma il suo complemento \bar{L} non lo è, L non può essere ricorsivo. Infatti se L fosse ricorsivo, anche \bar{L} sarebbe ricorsivo e dunque RE.

Teorema 9.3 Se L è un linguaggio ricorsivo, lo è anche \bar{L} .

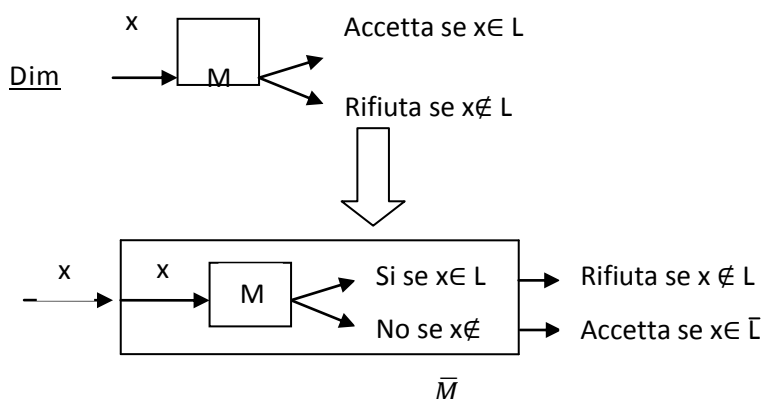
DIMOSTRAZIONE Sia $L = L(M)$ per una TM M che si arresta sempre. Costruiamo una TM \bar{M} tale che $\bar{L} = L(\bar{M})$ secondo lo schema illustrato nella figura seguente.



Come si vede, \bar{M} si comporta esattamente come M . Per formare \bar{M} modifichiamo M nel modo seguente:

1. Gli stati accettanti di M diventano stati non accettanti di \bar{M} senza transizioni; in questi stati M si arresta senza accettare.
2. \bar{M} ha un nuovo stato accettante r ; non esiste alcuna transizione uscente da r .
3. Per ogni combinazione di uno stato non accettante e di un simbolo di nastro di M tale che M non abbia alcuna transizione (cioè M si arresta senza accettare), aggiungiamo una transizione verso lo stato accettante r .

Poiché è garantito che M si arresta, lo stesso vale per \bar{M} . Inoltre \bar{M} accetta proprio le stringhe che M non accetta. Di conseguenza \bar{M} accetta \bar{L} .



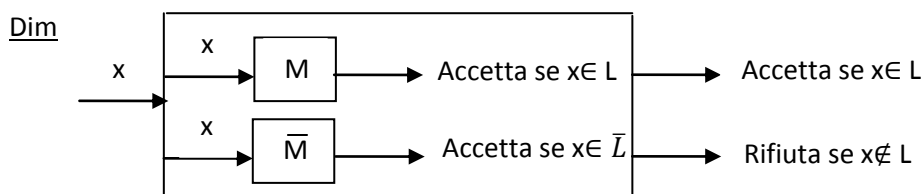
M si ferma se esiste $\bar{\delta}_M(q, x)$ non definita. In questi casi bisogna fare in modo che M accetti. Quindi definiamo $\bar{\delta}_M(q, x) = (r, x, R)$

M accetta quando arriva in uno stato finale e si ferma

Teorema 9.4 Se un Linguaggio L e il suo complemento sono RE, allora L è ricorsivo. Osserviamo che in questo caso, per il teorema 9.3, anche \bar{L} è ricorsivo.

DIMOSTRAZIONE La dimostrazione è illustrata nella Figura sotto.

Siano $L = L(M_1)$ e $\bar{L} = L(M_2)$. M_1 e M_2 sono simulate in parallelo da una TM M . Possiamo trasformare M in una TM a due nastri e poi convertirla in una TM a nastro singolo per facilitare la simulazione. Un nastro di M simula quello di M_1 e M_2 , l'altro simula il nastro di M_2 . Gli stati di M_1 ed M_2 sono componenti dello stato di M . Se l'input w di M è in L , M_1 accetta in un tempo finito, quindi M accetta e si arresta. Se w non è in L , allora è in \bar{L} , dunque M_2 prima o poi accetta. A quel punto M si arresta senza accettare. Di conseguenza M si arresta su tutti gli input, ed $L(M)$ è esattamente L . Poiché M si arresta sempre e $L(M) = L$, concludiamo che L è ricorsivo.



Questa macchina si fermerà sicuramente poiché una delle 2 interne si fermerà prima o poi.

E' una macchina a 3 nastri: input, simulazione M , simulazione \bar{M} . Possiamo riassumere i Teoremi 9.3 e 9.4 come segue. Delle nove possibilità di collocare un linguaggio L e il suo complemento nel diagramma, solo quattro sono consentite.

1. Sia L sia \bar{L} sono ricorsivi, cioè si trovano entrambi nel cerchio interno.
2. Né L né \bar{L} sono RE, cioè sono entrambi nel cerchio esterno.
3. L è RE ma non ricorsivo, ed \bar{L} non è RE; uno si trova nel cerchio intermedio, l'altro nel cerchio esterno
4. \bar{L} è RE ma non ricorsivo, ed L non è RE; il caso è analogo al punto (3), ma L ed \bar{L} sono invertiti.

Il linguaggio universale

Definiamo il linguaggio universale L_u

$$L_u = \{(M, w) \mid w \in L(M)\}$$

ovvero come l'insieme delle stringhe binarie che codificano una coppia (M, w) , dove M è una TM con alfabeto di input binario, e w è una stringa in $\{0,1\}^*$ tale che w sia in $L(M)$. In altre parole L_u è l'insieme delle stringhe che rappresentano una TM e un input da essa accettato. Mostriamo che esiste una TM U capace di simulare il comportamento di ogni altra macchina di Turing su qualsiasi input.

Descriviamo U come una macchina multinastro. Le transizioni di M sono memorizzate inizialmente sul primo nastro, insieme alla codifica di w , un secondo nastro sarà utilizzato per simulare il comportamento di M , il terzo nastro conterrà lo stato di M , infine il quarto nastro sarà ausiliario.

Vediamo come opera U :

1. Esamina l'input per assicurarsi che il codice di M sia un codice legittimo per una TM. Se non è così, U si arresta senza accettare. Poiché si presuppone che i codici non validi rappresentino la TM senza mosse, che non accetta alcun input, si tratta di un'azione corretta.
2. Prepara il secondo nastro con l'input w nella sua forma codificata. Sul secondo nastro scrive 10 per ogni 0 e 100 per ogni 1 di w . I blank, sul nastro simulato di M , che sono rappresentati da 1000, non compariranno sul nastro; tutte le celle oltre quelle usate per w conterranno il blank di U . U sa comunque che, se dovesse cercare un simbolo simulato di M e trovare il proprio blank, deve sostituire quest'ultimo con la sequenza 1000 per simulare il blank di M .
3. Scrive 0, lo stato iniziale di M , sul terzo nastro, e muove la testina del secondo nastro verso la prima cella simulata.
4. Per simulare una mossa di M , U percorre il primo nastro alla ricerca di una transizione $0^i 10^j 10^k 10^l 10^m$ tale che 0^i sia lo stato sul nastro 3 e 0^j sia il simbolo di nastro di M collocato sul nastro 2 a partire dalla posizione guardata da U . Questa è la transizione che M farebbe come passo successivo. U deve fare tre cose:
 - a. Trasformare il contenuto del nastro 3 in 0^k , ossia simulare il cambiamento di stato di M . A tal fine U trasforma prima in blank tutti gli 0 sul nastro 3, e poi copia 0^k dal nastro 1 al nastro 3.
 - b. Sostituire 0^j sul nastro 2 con 0^l , cioè trasformare il simbolo di nastro di M . Se c'è bisogno di maggior/minor spazio, ovvero $i \neq l$, U si serve del nastro ausiliario.
 - c. Muove la testina del nastro 2 verso la posizione del successivo 1 a sinistra o a destra. A seconda delle due opzioni 0^m : $m = 1$ (mossa verso sinistra) o $m = 2$ (mossa verso destra). Di conseguenza U simula il movimento di M verso sinistra o destra.
5. Se M non ha transizioni per lo stato simulato e il simbolo di nastro, non ci sarà alcuna transizione in (4). M si arresta nella configurazione simulata e U deve fare altrettanto.
6. Se M entra nel suo stato accettante, allora U accetta.

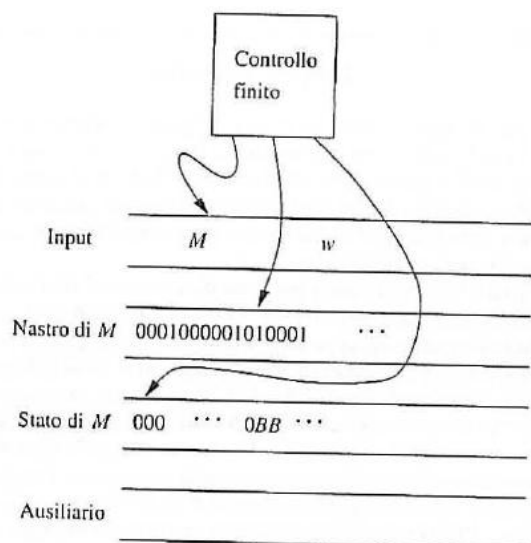


Figura 9.5 Struttura di una macchina di Turing universale.

Indecidibilità di L_U

Abbiamo individuato un problema che è RE ma non ricorsivo: il linguaggio L_U . Per molti aspetti sapere che L_U è indecidibile (non è un linguaggio ricorsivo) vale più della precedente scoperta che L_d non è RE. Il motivo è che possiamo ridurre L_U a un altro problema P per dimostrare che non esiste alcun algoritmo in grado di risolvere P , a prescindere dal fatto che P sia o no RE. La riduzione di L_d a P è però possibile solo se P non è RE, quindi non possiamo usare L_d per mostrare l'indecidibilità di quei problemi che sono RE ma non ricorsivi. D'altro canto, se vogliamo dimostrare che un problema non è RE, possiamo usare solo L_d , L_U non serve, dato che è RE.

Teorema L_U è RE ma non ricorsivo.

Dim Supponiamo che L_U sia ricorsivo. Per il Teorema 9.3 anche $\overline{L_U}$, il complemento di L_U , è ricorsivo. Se abbiamo una TM che accetta $\overline{L_U}$ possiamo costruire una TM che accetta L_d (grazie al metodo illustrato sopra). Poiché sappiamo già che L_d non è RE, ne ricaviamo una contraddizione con l'ipotesi che L_U sia ricorsivo.

Supponiamo $L(M) = \overline{L_U}$. Come Possiamo modificare la TM M in una TM M' che accetta L_d .

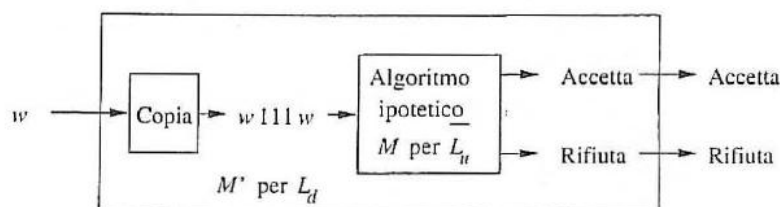


Figura 9.6 Riduzione di L_d a $\overline{L_U}$.

1. Data una stringa w in input, M' la trasforma in $w111w$. Il ragionamento a sostegno della fattibilità del programma consiste nell'uso di un secondo nastro per copiare w e nella conversione della TM a due nastri in una a nastro singolo.
2. M' simula M sul nuovo input, se nella nostra numerazione w è w_i , allora M' determina se M_i accetta w_i . Dato che M accetta $\overline{L_U}$, accetterà se e solo se M_i non accetterà w_i , quindi w_i è in L_d .

Di conseguenza M' accetta w se e solo se w è in L_d . Poiché sappiamo che M' non può esistere (Teorema illustrato in precedenza), concludiamo che L_U non è ricorsivo.

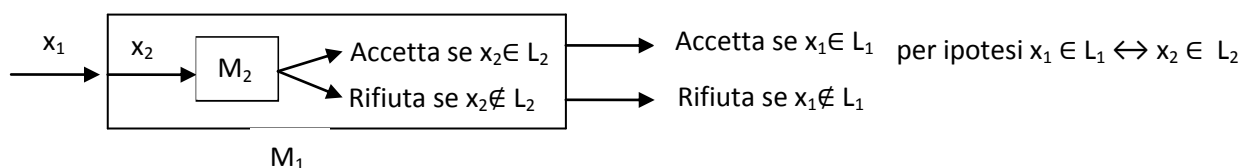
Riduzioni

Diciamo che un linguaggio L_1 può essere ridotto ad uno L_2 , ovvero $L_1 \leq L_2$, se esiste una funzione f tale che $f: x_1 \rightarrow x_2 = f(x_1)$ con $x_1 \in L_1$ ed $x_2 \in L_2$

Possiamo definire 2 proprietà:

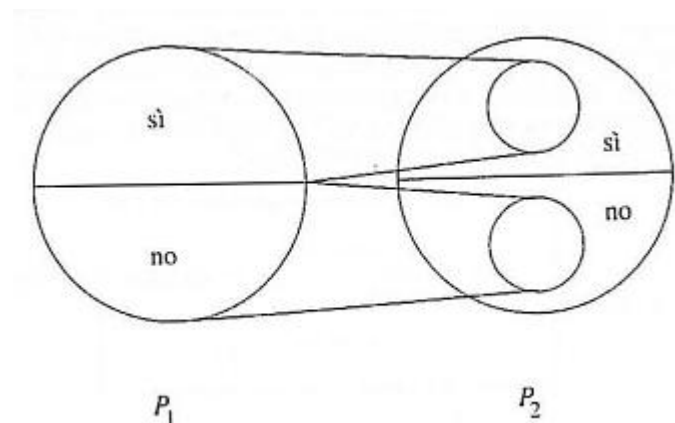
- Se $L_1 \leq L_2$ e $L_2 \in RE$, allora $L_1 \in RE$
- Se $L_1 \leq L_2$ e L_2 è Ricorsivo, allora L_1 è ricorsivo

Dim a)



Dim b)

Se abbiamo un algoritmo per convertire le istanze di un problema P_1 in istanze di un problema P_2 che hanno la stessa risposta, allora diciamo che P_1 si riduce a P_2 . Possiamo avvalerci di questa dimostrazione per provare che P_2 è "difficile" almeno quanto P_1 . Di conseguenza, se P_1 non è ricorsivo, allora P_2 non può essere ricorsivo. Se P_1 è non RE, allora P_2 non può essere RE.



Una riduzione deve trasformare qualsiasi istanza di P_1 che ha una risposta affermativa ("sì") in un'istanza di P_2 con una risposta affermativa ("sì"), e ogni istanza di P_1 con una risposta negativa ("no") in un'istanza di P_2 con una risposta negativa ("no"). Non è essenziale che ogni istanza di P_2 sia l'immagine di una o più istanze di P_1 : di fatto è normale che solo una piccola frazione di P_2 sia l'immagine della riduzione. In termini formali una riduzione da P_1 a P_2 è una macchina di Turing che riceve un'istanza di P_1 scritta sul nastro e si arresta con un'istanza di P_2 sul nastro. Nella pratica descriveremo generalmente le riduzioni come se fossero programmi per computer che ricevono in input un'istanza di P_1 e producono come output un'istanza di P_2 .

Teorema 9.7 Se esiste una riduzione da P_1 a P_2 , allora:

- se P_1 è indecidibile, lo è anche P_2 (entrambi non ricorsivi)
- se P_1 è non RE, lo è anche P_2 .

LEZIONE 8- LINGUAGGI RICORSIVI (4 PARTE)

Il problema dell'arresto

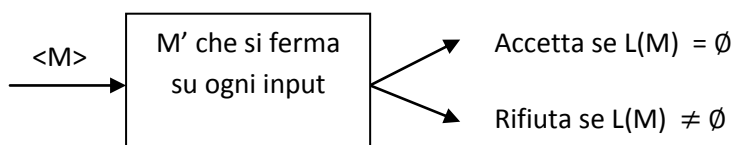
A volte si parla del problema dell'arresto per le macchine di Turing come di un problema simile a Lu, cioè RE ma non ricorsivo. In effetti la macchina di Turing originale di Turing accetta per terminazione, non per stato finale. Per una TM M possiamo definire $H(M)$ come l'insieme dei w tali che M si arresta quando riceve w in input, senza tener conto se M lo accetta o no. In quel caso il problema dell'arresto è l'insieme delle coppie (M, w) tali che w è in $H(M)$. Questo è un altro esempio di problema/linguaggio RE ma non ricorsivo.

Macchine di Turing che accettano il linguaggio vuoto

Come esempio di riduzioni che coinvolgono le macchine di Turing, studiamo due linguaggi, che chiameremo L_e ed L_{ne} , composti da stringhe binarie. Se w è una stringa binaria, allora rappresenta una MT M_i nell'enumerazione vista alcune lezioni precedenti.

Se $L(M_i) = \emptyset$, ossia M_i non accetta alcun input, allora w è in L_e . Quindi L_e è il linguaggio formato da tutte le MT codificate il cui linguaggio è vuoto. D'altro canto, se $L(M_i)$ non è il linguaggio vuoto, allora w è in L_{ne} . Dunque L_{ne} è il linguaggio di tutte le macchine di Turing che accettano almeno una stringa in input. Nel seguito sarà opportuno considerare le stringhe come le macchine di Turing che rappresentano. Possiamo così definire i due linguaggi appena citati:

- $L_e = \{ M \mid L(M) = \emptyset \}$
- $L_{ne} = \{ M \mid L(M) \neq \emptyset \}$



Osserviamo che L_e e L_{ne} sono entrambi i linguaggi sull'alfabeto binario $\{0,1\}$ e che sono l'uno il complemento dell'altro. Vedremo che L_{ne} è il "più facile" dei due linguaggi; è RE ma non ricorsivo. Da parte sua L_e è non RE.

Teorema. L_{ne} è ricorsivamente enumerabile.

DIMOSTRAZIONE:

Dobbiamo solo esibire una MT che accetta L_{ne} . Il modo più semplice consiste nel descrivere una MT non deterministica M , il cui schema è rappresentato nella Figura 9.8. M può essere convertita in una MT deterministica. Sfruttiamo la MT per il linguaggio universale.

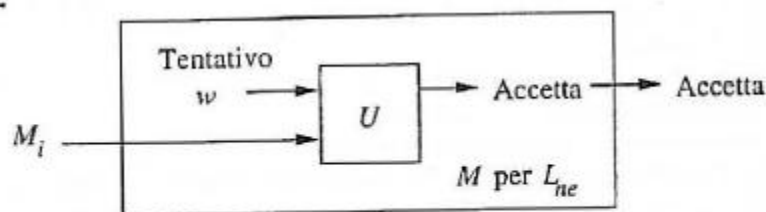


Figura 9.8 Costruzione di una NTM che accetta L_{ne} .

Descriviamo le operazioni di M .

1. M riceve in input un codice di TM M_i .

2. Impiegando la sua capacità non deterministica M “tenta” un input w che M_i potrebbe accettare.
3. M verifica se M_i accetta w . Per questa parte M può simulare la MT universale U che accetta L_u .
4. Se M_i accetta w , allora M accetta il proprio input, ossia M_i .

In questo modo, se M_i accetta anche solo una stringa, M la tenderà (tra tutte le altre, ovviamente) e accetterà M_i . Se però $L(M_i) = \emptyset$, nessun tentativo w porta all'accettazione da parte di M_i , ed M non accetta M_i . Di conseguenza $L(M) = L_{ne}$.

Il passo successivo consiste nel dimostrare che L_{ne} non è ricorsivo. A tal fine riduciamo L_u a L_{ne} . In altre parole descriviamo un algoritmo che trasforma in input (M, w) in un output M' , il codice di un'altra macchina di Turing, tale che w si trova in $L(M)$ se e solo se $L(M')$ non è vuoto. Questo significa che M accetta w se e solo se M' accetta almeno una stringa. Il trucco è far sì che M' ignori il suo input e simuli invece M su input w . Se M accetta, allora M' accetta il proprio input; di conseguenza l'accettazione di w da parte di M equivale a $L(M')$ non vuoto. Se L_{ne} fosse ricorsivo, allora avremmo un algoritmo che indica se M accetta o no w ; costruiamo M' e verifichiamo se $L(M') = \emptyset$.

Teorema. L_{ne} non è ricorsivo.

DIMOSTRAZIONE:

Dobbiamo definire un algoritmo che converte in input formato da una coppia codificata in binario (M, w) in una MT M' tale che $L(M') \neq \emptyset$ se e solo se M accetta l'input w . La costruzione di M' è delineata nella Figura 9.9. Come vedremo, se M non accetta w , allora M' non accetta nessuno dei suoi input; cioè $L(M') = \emptyset$. Se però M accetta w , allora M' accetta ogni input, e di conseguenza $L(M')$ sicuramente non è \emptyset .

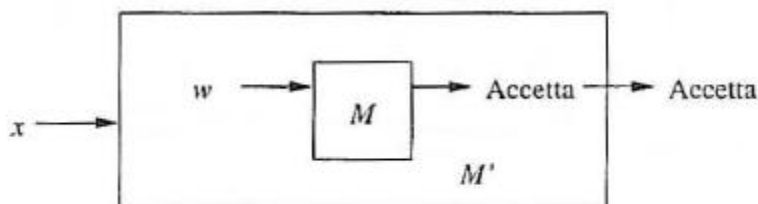


Figura 9.9 Schema della TM M' costruita da (M, w) nel Teorema 9.9. M' accetta un input arbitrario se e solo se M accetta w .

Definiamo M' affinché compia le seguenti operazioni.

1. M' ignora il proprio input x , o meglio, lo sostituisce con la stringa che rappresenta la MT M e la stringa di input w . Poiché M' è concepita per una specifica coppia (M, w) , di lunghezza n , possiamo costruire M' in modo che abbia una sequenza di stati q_0, q_1, \dots, q_n , dove q_0 è lo stato iniziale.
 - a) Nello stato q_i , per $i=0, 1, \dots, n-1$, M' scrive l' $(i+1)$ -esimo bit del codice per (M, w) , va nello stato q_{i+1} e si muove verso destra.
 - b) Nello stato q_n , se necessario M' si muove verso destra rimpiazzando eventuali simboli diversi dal blank (che formano la parte finale di x se tale input di M' è più lungo di n) con blank.
2. Quando M' raggiunge un blank nello stato q_n , usa un'analoga serie di stati per ricollocare la testina all'estremità sinistra del nastro.
3. Ricorrendo a stati aggiuntivi M' simula una MT universale U sul suo nastro corrente.
4. Se U accetta, allora accetta anche M' . Se U non accetta mai, allora neanche M' accetta mai.

Esiste quindi un algoritmo per effettuare la riduzione di L_u a L_{ne} . Vediamo inoltre che se M accetta w , allora M' accetta qualsiasi input x presente sul nastro all'inizio. Di conseguenza, se M accetta w , allora il codice di M' si trova in L_{ne} .

Viceversa, se M non accetta w , allora M' non accetta mai, indipendentemente dalla natura del suo input. Perciò in questo caso il codice di M' non si trova in L_{ne} . Siamo riusciti a ridurre L_u a L_{ne} per mezzo dell'algoritmo che costruisce M' da M e w ; possiamo concludere che, poiché L_u non è ricorsivo, non lo è neanche L_{ne} . L'esistenza della riduzione è sufficiente a completare la dimostrazione.

A questo punto possiamo conoscere lo stato di L_e . Se L_e fosse RE, allora per il teorema 9.4 (se un linguaggio L e il suo complemento sono RE, allora L è ricorsivo) sia L_e sia L_{ne} sarebbero ricorsivi. Dato che per il teorema 9.9 L_{ne} non è ricorsivo concludiamo che: L_e non è RE.

Il teorema di Rice e le proprietà dei linguaggi RE

Il fatto che linguaggi come L_e ed L_{ne} siano indecidibili è un caso speciale di un teorema molto più generale: tutte le proprietà non banali dei linguaggi RE sono indecidibili, nel senso che è impossibile riconoscere per mezzo di una macchina di Turing le stringhe binarie che rappresentano codici di una MT il cui linguaggio soddisfa la proprietà. Un esempio di proprietà dei linguaggi RE è il “il linguaggio è libero dal contesto”. Come caso speciale del principio generale che tutte le proprietà non banali dei linguaggi RE sono indecidibili, il problema se una data MT accetti un linguaggio libero dal contesto è indecidibile.

Una **proprietà** dei linguaggi RE è semplicemente un insieme dei linguaggi. Di conseguenza la proprietà di “essere libero dal contesto” è in termini formali l’insieme di tutti i CFL. La proprietà di essere vuoto è l’insieme $\{\emptyset\}$, che consiste del solo linguaggio vuoto.

Una proprietà è **banale** se è vuota (ossia se non viene soddisfatta da nessun linguaggio) o comprende tutti i linguaggi RE.

- ✓ $P=RE$ è una proprietà banale (tutti i linguaggi la verificano)
- ✓ $P=\emptyset$ è una proprietà banale (nessun linguaggio la soddisfa)

Altrimenti è **non banale**:

- Osserviamo che la proprietà vuota, \emptyset , è diversa dalla proprietà di essere un linguaggio vuoto $\{\emptyset\}$.
- $P=\{\emptyset\}$ è non banale

Non possiamo riconoscere un insieme di linguaggi come i linguaggi stessi. La ragione è che il tipico linguaggio, essendo infinito, non può essere espresso come una stringa di lunghezza finita che possa essere l’input di una MT. Dobbiamo piuttosto riconoscere le macchine di Turing che accettano quei linguaggi; il codice della MT è finito anche se il linguaggio che accetta è infinito. Di conseguenza, se P è una proprietà dei linguaggi RE, il linguaggio L_P è l’insieme delle macchine di Turing M_i tali che $L(M_i)$ è un linguaggio in P . Quando parliamo di decidibilità di una proprietà P , intendiamo la decidibilità del linguaggio L_P .

Teorema di Rice

TH. Ogni proprietà non banale dei linguaggi RE è indecidibile.

DIMOSTRAZIONE:

Sia P una proprietà non banale dei linguaggi RE. Per cominciare, supponiamo che \emptyset , il linguaggio vuoto, non sia in P ; ci occuperemo più tardi del caso opposto. Dato che P è non banale, deve esistere un linguaggio non vuoto L che sia in P . Sia M_L un MT che accetta L .

Ridurremo L_u a L_P , dimostrando in questo modo che L_P è indecidibile, dal momento che L_u è indecidibile. L’algoritmo per la riduzione riceve in input una coppia (M, w) e produce una MT M' . Lo schema di M' è illustrato nella figura 9.10; $L(M')$ è \emptyset se M non accetta w , e $L(M') = L$ se M accetta w .

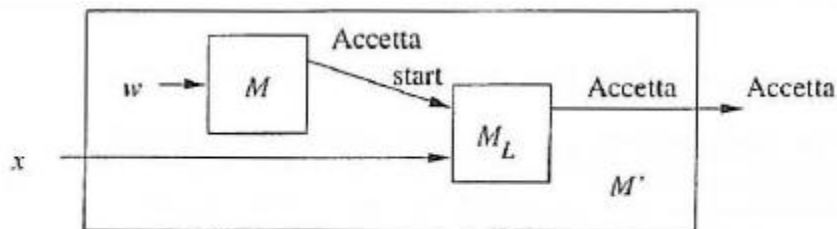


Figura 9.10 Costruzione di M' per la dimostrazione del teorema di Rice.

M' è una MT a due nastri. Un nastro è usato per simulare M su w . Ricordiamo che l’algoritmo che effettua la riduzione riceve come input M e w , e li usa nella definizione delle transizioni di M' . Di conseguenza la simulazione di M su w è incorporata in M' ; la seconda MT non deve leggere le transizioni di M sui suoi nastri.

Se necessario, l'altro nastro di M' viene usato per simulare M_L sull'input x di M' . Anche qui le transizioni di M_L sono note all'algoritmo di riduzione e possono essere incorporate nelle transizioni di M' . La MT M' è costruita per operare nel modo seguente.

1. Simula M su input w . Osserviamo che w non è l'input di M' ; M' scrive invece M e w su uno dei due nastri e simula la MT universale U su tale coppia.
2. Se M non accetta w , allora M' non fa nient'altro. M' non accetta mai il proprio input x , per cui $L(M') = \emptyset$. Poiché assumiamo che \emptyset non sia nella proprietà P , ciò significa che il codice di M' non è in L_P .
3. Se M accetta w , allora M' comincia a simulare M_L sul proprio input x . Perciò M' accetterà esattamente il linguaggio L . poiché L è in P il codice di M' è in L_P .

Si può notare che la costruzione di M' da M e w può essere realizzata da un algoritmo. Dato che tale algoritmo trasforma (M,w) in una M' che è in L_P se e solo se (M,w) è in L_U , esso è una riduzione di L_U a L_P , e dimostra che la proprietà P è indecidibile.

Ci resta da considerare il caso in cui \emptyset è in P . Esaminiamo allora la proprietà complemento \bar{P} , l'insieme dei linguaggi RE che non hanno la proprietà P . Per quanto abbiamo visto sopra, \bar{P} è indecidibile. Ma poiché ogni MT accetta un linguaggio RE, $\overline{L_P}$ l'insieme di (codici per) macchine di Turing che non accettano un linguaggio in P è uguale a $L_{\bar{P}}$, l'insieme di MT che accettano un linguaggio in complemento di \bar{P} . Supponiamo che L_P sia decidibile. Allora lo sarebbe anche $L_{\bar{P}}$, perché il complemento di un linguaggio ricorsivo è ricorsivo. L_P quindi non è ricorsivo.

LEZIONE 9 – FUNZIONI RICORSIVE (1 PARTE)

CAPITOLO 12

La definizione intuitiva del concetto di funzione effettivamente calcolabile o funzione calcolabile mediante un algoritmo, ovvero mediante una sequenza discreta di passi elementari di calcolo di una MdT può sembrare limitativa rispetto a ciò che intuitivamente ci sembra effettivamente calcolabile.

Funzioni primitive ricorsive

L'insieme dei numeri naturali è facilmente caratterizzabile mediante induzione ovvero affermando che ogni numero naturale n è esprimibile come una iterazione della operazione elementare di successore applicata al valore costante 0: $S^n(0)$. Pertanto

$$N = \{0, S(0), S(S(0)), S(S(S(0))), \dots\}$$

Nel seguito indicheremo spesso $S(x)$ con $x+1$. Questo procedimento induttivo per definire i numeri naturali si basa sull'idea che ogni numero naturale è definibile a partire da un numero più semplice applicando l'operazione di successore. **Una definizione ricorsiva di funzione è una definizione dove il valore della funzione per un dato argomento è direttamente correlato al valore della medesima funzione su argomenti più semplici o al valore di funzioni più semplici.**

Trattando i naturali, consideriamo dunque la costante 0 come una funzione costante elementare. Supponiamo inoltre di avere a disposizione l'operazione di successore, che ci permette di definire qualsiasi numero naturale, e la funzione identità. Queste assunzioni ci portano a definire il concetto di funzione ricorsiva di base.

Si dicono funzioni ricorsive di base le seguenti funzioni:

- ✓ la funzione costante 0: 0 oppure $0: x \rightarrow 0$
- ✓ La funzioni, successore $S: \lambda x. x + 1$ oppure $S: x \rightarrow x + 1$
- ✓ La funzione identità, o i -esima proiezione,
 $\pi_i: \lambda x_1 \dots x_n. x_i$ con $1 \leq i \leq n$; oppure $\pi_i^k(x_1 \dots x_n) \rightarrow x_i$ con $1 \leq i \leq n$

I costrutti sono:

1. Composizione:

da $g_1, \dots, g_k: \mathbb{N}^n \rightarrow \mathbb{N}$ e $h: \mathbb{N}^k \rightarrow \mathbb{N}$ se $f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$.

Esempio:

$$f: x \rightarrow 1$$

$$f(x): S(0(x)) \text{ dove } n = 1, h = S, k = 1, g_1 = 0$$

2. Definita per ricorsione primitiva:

da $g: \mathbb{N}^{n-1} \rightarrow \mathbb{N}$ e $h: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ se $\begin{cases} f(x_1, \dots, x_{n-1}, 0) = g(x_1, \dots, x_{n-1}) \\ f(x_1, \dots, x_{n-1}, y+1) = h(x_1, \dots, x_{n-1}, y, f(x_1, \dots, x_{n-1}, y)) \end{cases}$

Esempio:

$$f = +; n = 2; y = \pi_1^1(x); \text{dove } h(a, b, c) = S(\pi_3(a, b, c))$$

$$\begin{cases} +(x, 0) = \pi_1(x) \\ \end{cases}$$

$$\begin{cases} +(x, y+1) = h(x, y, +(x, y)) = S(+(x, y)) \end{cases}$$

Tutte sole le funzioni definibili a partire dalle precedenti funzioni ricorsive di base mediante composizione e ricorsione primitiva definiscono l'insieme delle funzioni primitive ricorsive. L'idea è quella di costruire via

via funzioni effettivamente calcolabili a partire dalle di base, ovvero 0, S ed identità. La classe P delle funzioni primitive ricorsive è la più piccola classe (ovvero l'intersezione di tutte le classi) di funzioni contenenti le funzioni ricorsive di base e chiuse per composizione e ricorsione primitiva.

Per ogni funzione $f, f \in P$ sse esiste una sequenza finita di funzioni f_1, \dots, f_n , tale che: $f_n = f$ e per ogni funzione f_j , con $j \leq n$, o f_j è una funzione ricorsiva di base, oppure è ottenuta mediante composizione o ricorsione primitiva a partire da funzioni f_{i_1}, \dots, f_{i_k} con $i_1, \dots, i_k < j$ e per le quali vale la stessa proprietà.

Lista funzioni ricorsive primitive

Le seguenti funzioni sono ricorsive primitive:

SOMMA

$$\begin{cases} +(x, 0) = x \\ +(x, y + 1) = S(+ (x, y)) \end{cases}$$

Esempio:

$$\begin{aligned} &+ \left(S(S(0)), S \left(S(S(0)) \right) \right) = \\ &S(+ (S(S(0)), S(S(0)))) = S(S(+ (S(S(0)), S(0)))) = S \left(S \left(S \left(+ (S(S(0)), 0) \right) \right) \right) = S \left(S \left(S(S(S(0))) \right) \right) \end{aligned}$$

MOLTIPLICAZIONE

$$\begin{cases} \cdot (x, 0) = 0 \\ \cdot (x, y + 1) = +(\cdot (x, y), y) \end{cases}$$

POTENZA

$$\begin{cases} x^0 = S(0) \\ x^{y+1} = \cdot (x^y, x) \end{cases}$$

IPERPOTENZA

$$\begin{cases} iper(x, 0) = \pi_1(x) \\ iper(x, y + 1) = iper(x, y)^x \end{cases} \text{ si osservi che } iper(x, y) = x^{x^{\dots^x}} = x^{(x^y)}$$

PREDECESSORE

$$\begin{cases} pred(0) = 0 \\ pred(y + 1) = \pi_1(y) \end{cases}$$

DIFFERENZA

$$\begin{cases} -(x, 0) = \pi_1(x) \\ -(x, y + 1) = pred(-(x, y)) \end{cases}$$

FATTORIALE

$$\begin{cases} 0! = S(0) \\ (y + 1)! = \cdot (S(y), y!) \end{cases}$$

SEGNO

$$\begin{cases} sg(0) = 0 \\ sg(y + 1) = S(0) \end{cases} \text{ definiamo inoltre } \overline{sg}(x) = -(1, sg(x))$$

VALORE ASSOLUTO DELLA DIFFERENZA

$$|-(x, y)| = +(-(x, y), -(y, x))$$

MINIMO E MASSIMO

$$\begin{cases} \min(x, y) = -(x, -(x, y)) \\ \max(x, y) = +(x, -(y, x)) \end{cases}$$

DIVISIONE INTERA: assumiamo che $\text{div}(x, 0) = \text{mod}(x, 0) = 0$

$$\begin{cases} \text{mod}(x, 0) = 0 \\ \text{mod}(y + 1, x) = S(\text{mod}(y, x)) \cdot sg(|-(x, S(\text{mod}(y, x)))|) \end{cases}$$

$$\begin{cases} \text{div}(x, 0) = 0 \\ \text{div}(y + 1, x) = \text{div}(y, x) + \overline{sg}(|-(x, S(\text{mod}(y, x)))|) \end{cases}$$

OPERATORI RELAZIONALI

$$geq(x, y) \begin{cases} 1 \text{ se } x \geq y \\ 0 \text{ se } x < y \end{cases}$$

$$gt(x, y) \begin{cases} 1 \text{ se } x > y \\ 0 \text{ se } x \leq y \end{cases}$$

$$leq(x, y) \begin{cases} 1 \text{ se } x \leq y \\ 0 \text{ se } x > y \end{cases}$$

$$lt(x, y) \begin{cases} 1 \text{ se } x < y \\ 0 \text{ se } x \geq y \end{cases}$$

$$eq(x, y) \begin{cases} 1 \text{ se } x = y \\ 0 \text{ se } x \neq y \end{cases}$$

$$neq(x, y) = \begin{cases} 1 \text{ se } x \neq y \\ 0 \text{ se } x = y \end{cases}$$

Differenza intera

$$\div(x, y) = \begin{cases} x - y & \text{se } x > y \\ 0 & \text{se } x \leq y \end{cases}$$

$$\begin{cases} \div(x, 0) = x \\ \div(x, y + 1) = \text{pred}(\div(x, y)) = x - (y + 1) = (x - y) + 1 \end{cases}$$

Valore intero della radice quadrata

$$sqrt(x) = \lfloor \sqrt{x} \rfloor$$

$$sqrt(x) = \text{pred}(\mu z. (x < z^2)) = \text{pred}\left(\mu z \left(geq\left(\pi_1^2(x, z), \left(\pi_2^2(x, z), \pi_2^2(x, z)\right) = 0\right)\right)\right)$$

$$\text{Es. } sqrt(16) = \mu z(z^2 > 16) - 1 = 5 - 1 = 4$$

$$sqrt(15) = \mu z(z^2 > 15) - 1 = 4 - 1 = 3$$

LEZIONE 10 – FUNZIONI RICORSIVE (2 PARTE)

Esempio di ricorsione primitiva:

$$\text{predecessore} \rightarrow \text{pred}(x) = \begin{cases} x - 1 & \text{se } x > 0 \\ 0 & \text{altrimenti} \end{cases}$$

$$\begin{cases} \text{pred}(0) \\ \text{pred}(y + 1) = y \end{cases} \quad \text{pred}(y+1) = h(y, \text{pred}(y)) = \pi_1^2(y, \text{pred}(y))$$

Derivazione

+per R.P.(ricorsione primitiva) $g = \pi_1^1$ e h la composizione di S e π_3^3

$+=\{\pi_1^1, (S, \pi_3^3)\} \leftarrow$ associo una stringa binaria quindi... le funzioni R.P. sono numerabili.

$$\forall f \in P \rightarrow w_h \in \{0,1\}^*$$

$$+=\{\pi_1^1 (S, \pi_3^3)\}$$

$$+=f_5$$

f_5 è la somma $f = \pi_1^1, f_3 = h(S, \pi_3^3), f_2 = \pi_3^3, f_1 = S$

f_5, f_4, f_3, f_2, f_1 ogni freccia indica una derivazione.

La funzione di Ackermann

Le funzioni P non sono tutte le f "calcolabili".

1) le funzioni di P sono numerabili + diagonalizzazione

Teorema: Le funzioni primitive ricorsive di P sono totali

Per induzione strutturale sulla complessità (nel senso di numero di operazioni di composizione e ricorsione primitiva) delle funzioni definite in P osserviamo che le funzioni ricorsive di base sono totali, e la composizione di funzioni totali è totale. Il teorema afferma che le funzioni primitive ricorsive sono totali e, come vedremo, questa è una limitazione, ovvero una funzione calcolabile può essere indefinita su alcuni (o tutti) i valori di input. Le funzioni primitive ricorsive hanno inoltre un'ulteriore limitazione anche all'interno delle funzioni totali sui naturali. * Resta da dimostrare che la composizione mediante ricorsione primitiva di funzioni totali è totale.

(PASSO INDUTTIVO) Per " f " ottenute da P , quindi è composizione di $h(g_1, \dots, g_k)$

- ✓ f totali \leftarrow totali
- ✓ f per ricorsione primitiva da g, h
- ✓ f totale $\leftarrow g$ e h totali per induzione su y

* Per dimostrare questo fatto, è sufficiente dare un testimone, ovvero una funzione evidentemente calcolabile e totale, ma non appartenente a P . una tale funzione è la funzione di Ackermann. La funzione di Ackermann è una funzione il 3 argomenti, $ack: \mathbb{N}^3 \rightarrow \mathbb{N}$, definita come segue

1. $ack(0, 0, y) = y$
2. $ack(0, x + 1, y) = ack(0, x, y) + 1$
3. $ack(1, 0, y) = 0$
4. $ack(z + 2, 0, y) = 1$
5. $ack(z + 1, x + 1, y) = ack(z, ack(z + 1, x, y), y)$

Esempio:

$$f(x) = \begin{cases} 2 & \text{se } x = 3 \\ 5 & \text{se } x \neq 3 \end{cases}$$

if(x=3) then return 2
else return 5

Equivalenza

$$\text{eq}(x,y) = \begin{cases} 1 & \text{se } x = y \\ 0 & \text{se } x \neq y \end{cases}$$

$$\text{neq}(x,y) = \begin{cases} 1 & \text{se } x \neq y \\ 0 & \text{se } x = y \end{cases}$$

Esempio

$$\text{quindi } f(x) = \begin{cases} 2 & \text{se } x = 3, \text{eq.}(x, 3) = 1 \\ 5 & \text{se } x \neq 3, \text{neq.}(x, 3) = 1 \end{cases}$$

$$\text{SOL} \rightarrow f(x) = 2 * \text{eq.}(x, 3) + 5 * \text{neq}(x, 3)$$

$$\text{se } x=y \rightarrow f(x) = 2 * 1 + 5 * 0 = 2$$

$$\text{se } x \neq y \rightarrow f(x) = 2 * 0 + 5 * 1 = 5 \quad \text{VA BENE}$$

quindi tornando a

$$2 * \text{eq.}(x, 3) + 5 * \text{neq}(x, 3) = + \underset{h}{(*)} \underset{g_1(x)}{(2(x), \text{eq.}(x, 3))}, \underset{g_2(x)}{*(5(x), \text{neq}(x, 3))}$$

N.B.=queste funzioni per casi si possono realizzare in questo modo:

1) tradurlo in codice, 2) poi facendo le funzioni opposta, 3) comporlo in f composizioni di funzioni primitive ricorsive.

LEZIONE 11 – FUNZIONI RICORSIVE (3 PARTE)

Diagonalizzazione

In questa lezione studieremo uno degli strumenti fondamentali della teoria delle funzioni calcolabili: la diagonalizzazione. La diagonalizzazione fu per la prima volta utilizzata da Cantor per dimostrare uno dei risultati fondamentali nella teoria classica degli insiemi, ovvero la non numerabilità dei numeri reali.

L'idea della diagonalizzazione è la seguente: Dato un insieme S numerabile, costruiamo una matrice nel modo seguente: in ogni riga i è presente una possibile enumerazione di S : $s_{i,0}, s_{i,1}, s_{i,2}, \dots$ che identifica univocamente una funzione totale $f: \mathbb{N} \rightarrow S$. Supponiamo che l'insieme di tali enumerazioni sia enumerabile e tale enumerazione la leggiamo scandendo la matrice riga per riga.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$...
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$...
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$...
...

Sia $d: S \rightarrow S$ una funzione totale che non sia mai l'identità su S , ovvero tale che $\forall s \in S, d(s) \neq s$. Da questa costruzione consideriamo la diagonale $s_{0,0}, s_{1,1}, s_{2,2}, \dots$ che viene trasformata da d nella sequenza: $d(s_{0,0}), d(s_{1,1}), d(s_{2,2}), \dots$

La caratteristica essenziale di questa sequenza è che essa differisce da ogni altra riga della matrice. Pertanto esiste una enumerazione di S non presente nella matrice. Ciò contraddice con l'ipotesi che l'insieme delle enumerazioni n (e dunque delle funzioni totali da \mathbb{N} a S) è numerabile.

Funzione parziale ricorsiva

Sia f una funzione totale $n+1$ aria, allora definiamo la funzione

$$\varphi(x_1, \dots, x_n) = \mu z. (f(x_1, \dots, x_n, z) = 0) = \begin{cases} \text{il più piccolo } z \text{ t.c. } f(x_1, \dots, x_n, z) = 0 & \text{se } z \text{ esiste} \\ \uparrow \text{ indefinita} & \text{altrimenti} \end{cases}$$

Tale operatore tra funzioni è detto μ – *operatore* ed una funzione così definita è detta definita per minimizzazione o μ – *ricorsione* da f .

Equivalenza tra MdT e funzioni parziali ricorsive.

TEOREMA (Aritmetizzazione di Godel):

$f: \mathbb{N} \rightarrow \mathbb{N}$ è *parziale ricorsiva* PR se e solo se è *TURING CALCOLABILE*.

(MOSTRIAMO L'IMPLICAZIONE INVERSA \leftarrow) Teorema: f : Se f è Turing-calcolabile allora f è parziale ricorsiva.

Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione calcolabile da una MdT Z . L'obiettivo è quello di codificare la MdT Z come una funzione sui naturali che sia parziale ricorsiva.

Tale MdT Z è definita con 2 simboli, che corrispondono ai numeri 0 e 1 rispettivamente:

$$\Sigma = \{0, 1\}$$

$$Q = \{q_0, \dots, q_k\}$$

Con queste ipotesi possiamo rappresentare una generica ID come una tupla di numeri, associando un numero nell'intervallo $[0, k]$ ad ogni simbolo di stato in Q .

Definiamo $\alpha: ID \rightarrow \mathbb{N}^4$ tale che, per ogni $\alpha \in ID$ della forma:

$$\alpha = \dots b_2 b_1 b_0 s q_h c_0 c_1 c_2 \dots$$

con $s \in \{0, 1\}$ il simbolo che stiamo puntando; $h \in [0, k]$ l'indice dello stato, e $i \in \mathbb{N}$ $\{b_i, c_i\} \subseteq \Sigma$,

dove b_i è la codifica della stringa a sinistra e c_i è la codifica della stringa a destra;

allora $ar(\alpha) = (h, m, s, n)$ dove

$$\begin{aligned} m &= \sum_{i=0}^{\infty} b_i 2^i = \sum_{i=0}^{k_m} b_i 2^i \\ n &= \sum_{i=0}^{\infty} c_i 2^i = \sum_{i=0}^{k_n} c_i 2^i \end{aligned}$$

dove k_m (k_n) è il massimo intero i per cui $b_i \neq 0$ (rispettivamente $c_i \neq 0$).

In questo modo, poiché il numero di 1 sul nastro non può essere infinito (siamo partiti con un numero finito di 1 ed abbiamo effettuato un numero finito di passi), per ogni ID in una computazione di Z , segue che ar associa in modo univoco ad ogni ID una quadrupla di naturali (aritmetizzazione di ID).

Una computazione di una MdT è definita come una sequenza (anche infinita) di ID: $\alpha_1 \vdash \alpha_2 \vdash \dots$.

Definiamo dunque una funzione che esprime un passo di transizione tra ID, ovvero una funzione che manipola quadruple di naturali. A tal proposito, definiamo le seguenti funzioni: $\delta_Q: Q \times \Sigma \rightarrow Q$, $\delta_\Sigma: Q \times \Sigma \rightarrow \Sigma$ e $\delta_x: Q \times \Sigma \rightarrow \{0, 1\}$ tali che:

- $\delta_Q(q, s)$ è lo stato che la MdT Z assume trovandosi nello stato q con simbolo in lettura s ;
- $\delta_\Sigma(q, s)$ è il simbolo che la MdT Z produce trovandosi nello stato q con simbolo in lettura s ;
- $\delta_x(q, s)$ è lo spostamento a destra (= 1) o sinistra (= 0) compiuto dalla MdT Z trovandosi nello stato q con simbolo in lettura s .

Per i valori non definiti si assegna il valore $k+1$ per renderle totali. Si noti che δ_Q , δ_Σ e δ_x sono funzioni definite dalla matrice funzionale di Z , e sono chiaramente funzioni primitive ricorsive.

Possiamo quindi definire le trasformazioni compiute eseguendo un singolo passo della MdT Z sulle quadruple rappresentanti una generica ID di Z :

$$g_Q(q, s, m, n) = \delta_Q(q, s)$$

$$g_\Sigma(q, s, m, n) = (m \bmod 2)(1 - \delta_x(q, s)) + (n \bmod 2) \delta_x(q, s)$$

$$g_M(q, s, m, n) = (2m + \delta_\Sigma(q, s))\delta_x(q, s) + (m \div 2)(1 - \delta_x(q, s))$$

$$g_N(q, s, m, n) = (2n + \delta_\Sigma(q, s))(1 - \delta_x(q, s)) + (n \div 2) \delta_x(q, s).$$

Queste funzioni sono chiaramente primitive ricorsive, essendo la composizione di funzioni primitive ricorsive. Possiamo dunque rappresentare una transizione $\alpha \vdash \beta$ nel modo seguente: Se $ar(\alpha) = (q, s, m, n)$ allora $ar(\beta) = (g_Q(q, s, m, n), g_\Sigma(q, s, m, n), g_M(q, s, m, n), g_N(q, s, m, n))$.

Per rappresentare l'effetto di t -transizioni o passi di calcolo della MdT Z , definiamo le seguenti funzioni di 5 argomenti:

- $P_Q(t, q, s, m, n)$ è lo stato ($\in [0, k]$) ottenuto dopo t -passi partendo da una ID α tale che $ar(\alpha) = (q, s, m, n)$;
- $P_\Sigma(t, q, s, m, n)$ è il simbolo ($\in \{0, 1\}$) ottenuto dopo t -passi partendo da una ID α tale che $ar(\alpha) = (q, s, m, n)$;
- $P_M(t, q, s, m, n)$ è il valore ($\in \mathbb{N}$) rappresentante il nastro a sinistra della testina ottenuto dopo t -passi partendo da una ID α tale che $ar(\alpha) = (q, s, m, n)$;
- $P_N(t, q, s, m, n)$ è il valore ($\in \mathbb{N}$) rappresentante il nastro a destra della testina ottenuto dopo t -passi partendo da una ID α tale che $ar(\alpha) = (q, s, m, n)$.

E' facile vedere che, ad esempio, P_Q può essere definita nel modo seguente:

- $P_Q(0, q, s, m, n) = q$
- $P_Q(t + 1, q, s, m, n) = P_{\delta Q}(t, g_Q(q, s, m, n), g_{\Sigma}(q, s, m, n), g_M(q, s, m, n), g_N(q, s, m, n))$.

Supponiamo che la MdT Z abbia uno stato $q_1 \in Q$ di terminazione, ovvero tale che le uniche caselle vuote della matrice funzionale di Z siano in corrispondenza della riga q_1 .

E' chiaro che ogni MdT può essere trasformata in modo tale da avere uno stato di terminazione di questo tipo. Pertanto, la MdT Z termina il suo calcolo nel primo (minimo) t tale che: $P_Q(t, q_0, s_0, m_0, n_0) = k + 1$, essendo α_0 la configurazione iniziale tale che $ar(\alpha_0) = (q_0, s_0, m_0, n_0)$. Ovvero, il numero di passi necessario per terminare, è definibile mediante μ -ricorsione come: $\mu t \rightarrow (k + 1 - P_Q(t, q_0, s_0, m_0, n_0) = 0)$.

Sia $x \in \mathbb{N}$ un numero naturale di input. Al solito, assumiamo che la MdT inizi il suo calcolo avendo il numero x sul nastro alla destra della sua testina codificato come una sequenza di $x + 1$ -uni

$$\begin{array}{ccccccc} \dots & 0 & 0 & 1 & \dots & 1 & 0 & \dots \\ & & \uparrow & \underbrace{}_{x+1} & & & & \\ & & \boxed{q_0} & & & & & \end{array}$$

Pertanto avremo che $q = 0, s = 0, m = 0, n = 2^{x+1} - 1$. Usiamo la solita assunzione che la MdT si fermerà in uno stato:

$$\begin{array}{ccccccc} \dots & 0 & 0 & 1 & \dots & 1 & 0 & \dots \\ & & \uparrow & \underbrace{}_{f(x)} & & & & \\ & & \boxed{q_0} & & & & & \end{array}$$

per cui $n = (2^{f(x)+1} - 1) + \dots$ dove in \dots ci saranno esponenti di 2 di grado superiore (ma lo 0 dopo l'output ci permette di determinare $f(x)$ da n). Sia C la funzione primitiva ricorsiva, che dato n numero del tipo sopra, permette di calcolare $f(x)$.

E' dunque chiaro che, avendo un modo algoritmico primitivo ricorsivo dato da una funzione C per rappresentare come potenza di 2 il numero alla destra della testina nella ID terminale, la funzione f calcolata da Z è esprimibile dalla seguente funzione parziale ricorsiva:

$$f(x) = C(P_N(\mu t \rightarrow (k + 1 - P_{\delta Q}(t, 0, 0, 0, 2^{x+1} - 1) = 0), 0, 0, 0, 2^{x+1} - 1)).$$

COROLLARIO: Per ogni funzione $\varphi \in PR$ (o equivalentemente Turing Calcolabile) esistono $f, g \in P$ t. c. $\varphi = \lambda x. f((\mu t. g(t, x) = 0), x)$

LEZIONE 12 – FUNZIONI RICORSIVE (4 PARTE)

PROOF. Per dimostrare il verso (\rightarrow), conviene dimostrare un teorema più forte: se φ è parzialmente ricorsiva, allora esiste una MdT che la calcola che:

- (1) Funziona anche se il nastro a sinistra dell'input e a destra della posizione iniziale della testina non è una sequenza infinita di \$.
- (2) Quando termina, termina subito a destra dell'input, il quale non viene modificato nella computazione (inoltre, per definizione, l'output è subito a destra della testina).
- (3) Inoltre, la parte del nastro che sta a sinistra della posizione iniziale della testina (in particolare l'input come già detto nel punto precedente) non viene modificata.

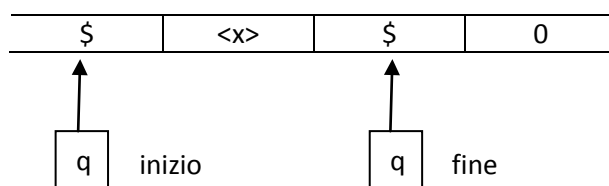
Queste 3 assunzioni ci permetteranno di applicare l'ipotesi induttiva.

Si tratta ora di descrivere le MdT che calcolano le funzioni di base soddisfacendo alle restrizioni suddette e poi mostrare che disponendo delle MdT che soddisfano le condizioni sopra e che calcolano delle funzioni, siamo in grado di definire quelle necessarie per calcolare la funzione che fa la loro composizione (le proprietà sopra saranno essenziali), ricorsione primitiva e minimizzazione. Non daremo le funzioni di transizione di tali macchine di Turing, ma solo la descrizione del loro funzionamento.

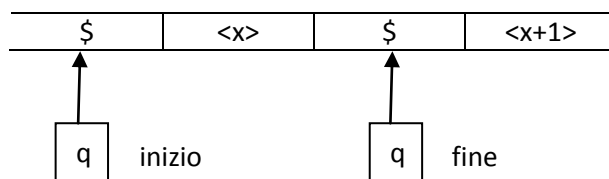
Base:

Per base, si deve tener conto delle funzioni primitive di base: 0, S (successore) e π (proiezione).

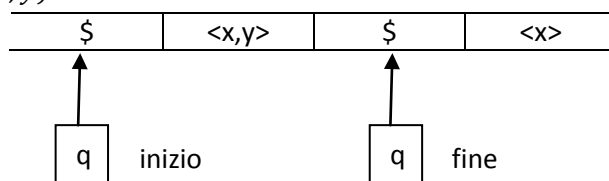
1. 0: $\exists \text{ MdT } t.c$



2. Successore: $\exists \text{ MdT } t.c$



3. Proiezione: Es. $\pi_1^2(x, y) = x$



$f \in P.R. \rightarrow \exists \text{ MdT } t.c \text{ calcola } f \text{ e } t.c. \text{ soddisfa } 1), 2), 3)$

Passo: Studiamo dapprima il caso della composizione. Per non perdere il senso della dimostrazione distratti da apici e pedici, ragioniamo nel caso di dover definire una macchina che calcola $f(g(x), h(x))$ nell'ipotesi di possedere le macchine di Turing M_f, M_g, M_h che calcolano le funzione $f, g, h: \mathbb{N} \rightarrow \mathbb{N}$. La dimostrazione nel caso generale di k funzioni non necessariamente unarie è del tutto analoga.

Nella Figura 1 è illustrata la successione di azioni da svolgere. Si tratta di richiamare le macchine di Turing definenti f, g, h inframezzando ogni chiamata da uno spostamento di una stringa (operazione che abbiamo già implementato in precedenza). Si osservi come i prerequisiti (1)-(3) sono rispettati dalla macchina risultante.

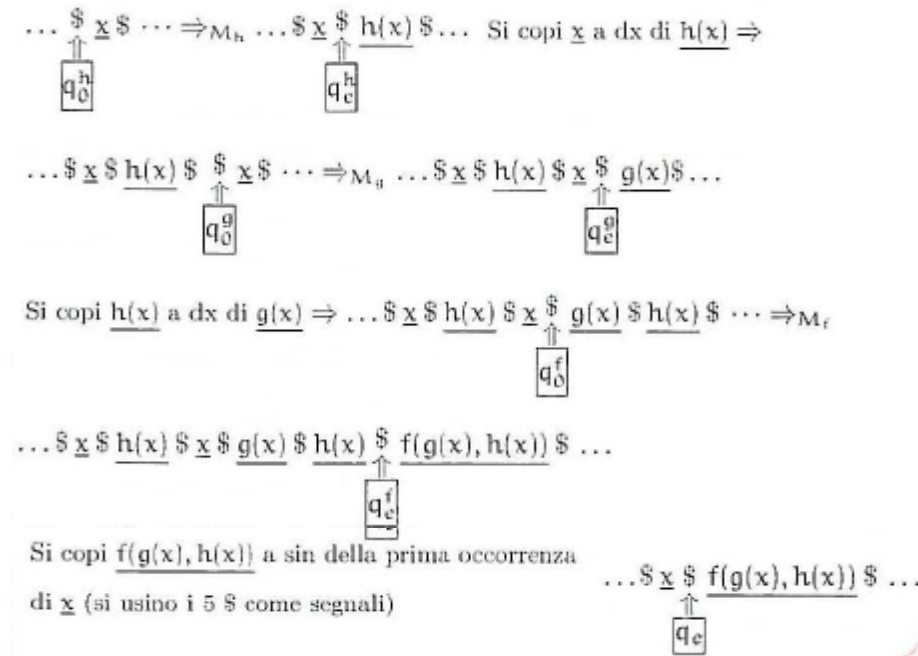


FIGURE 1. Macchina di Turing per la composizione

Analizziamo ora il caso della ricorsione primitiva. Dare le macchine di Turing M_g, M_h che definiscono le funzioni $g: \mathbb{N} \rightarrow \mathbb{N}$ e $h: \mathbb{N}^3 \rightarrow \mathbb{N}$, definiamo la MdT che calcola la funzione di $f: \mathbb{N}^2 \rightarrow \mathbb{N}$ definita per ricorsione primitiva:

$$\begin{cases} f(x, 0) = g(x) \\ f(x, y + 1) = h(x, y, f(x, y)) \end{cases}$$

(x potrebbe essere sostituito da x_1, \dots, x_{n-1} senza bisogno di particolari modifiche nel prosieguo della dimostrazione). Si osservi che la ricorsione primitiva può essere rimpiazzata da un ciclo for, ad esempio,

$$\begin{aligned} f(x, 3) &= h(x, 2, f(x, 2)) = h(x, 2, h(x, 1, f(x, 1))) = \\ &= h(x, 2, h(x, 1, h(x, 0, f(x, 0)))) = h(x, 2, h(x, 1, h(x, 0, g(x)))) \end{aligned}$$

Equivarrebbe a

```
F = g(x);
for(i=0; i < 3; i++)
    F = h(x, i, F);
```

Si osservi come i prerequisiti (1)-(3) sono rispettati dalla macchina risultante. Sul nastro partiamo con x e y .

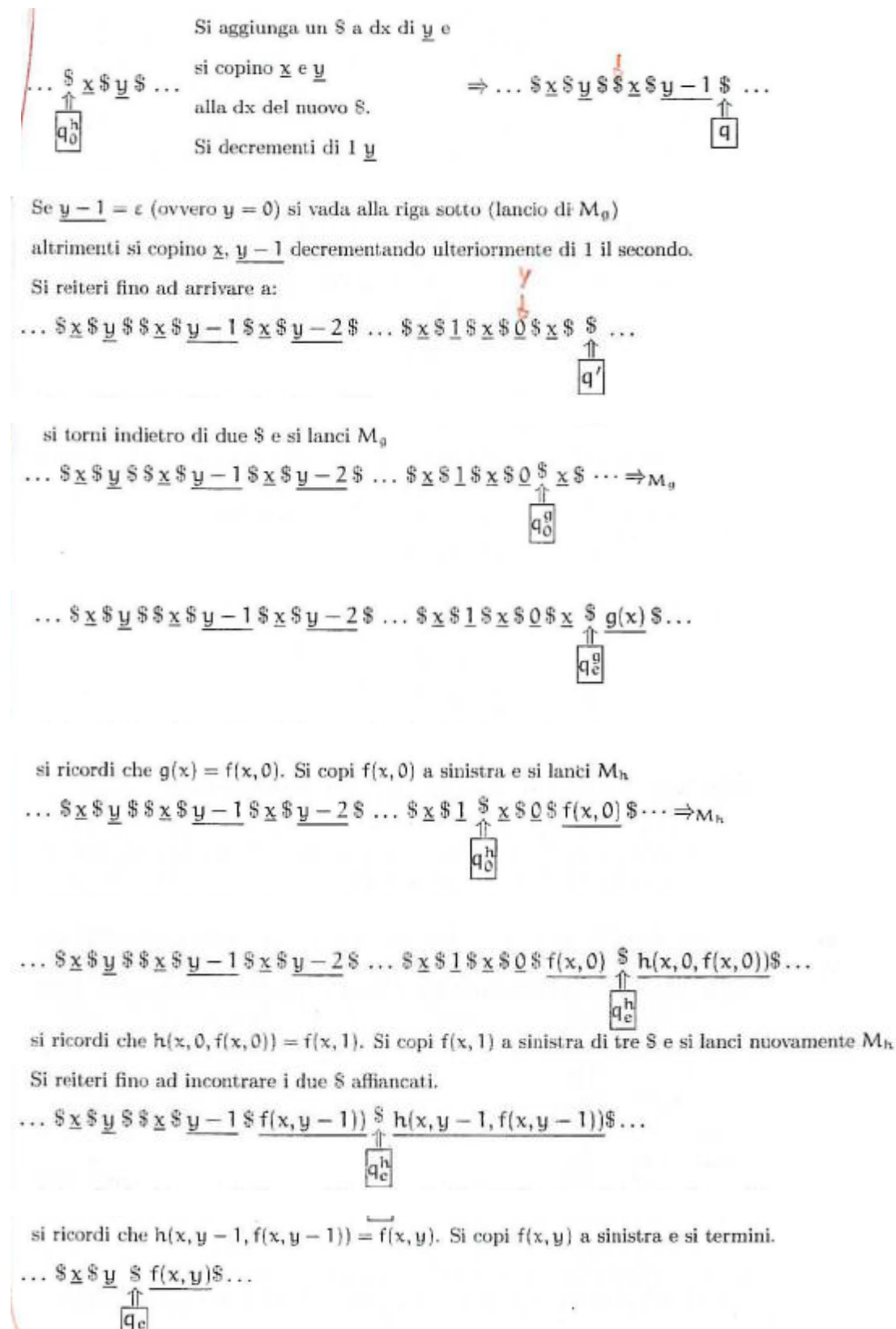
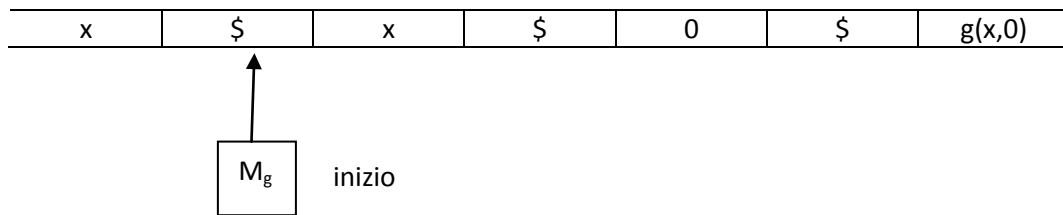
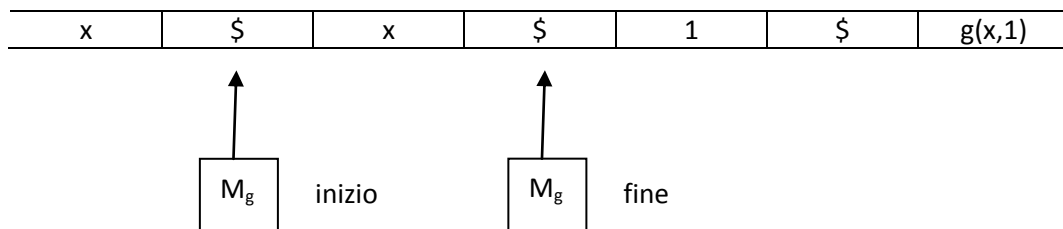


FIGURE 2. Macchina di Turing per la composizione

Rimane il caso del μ – *operatore*, ovvero, data una MdT M_g che calcola la funzione $g: \mathbb{N}^2 \rightarrow \mathbb{N}$, si deve definire la MdT M_f che calcola la funzione $f(x) = \mu y (g(x, y) = 0)$. Si tratta di lanciare M_g dapprima con $x, 0$.



Se $g(x,0)=0$ ci si ferma fermandoci sullo 0. Altrimenti si incrementa il secondo parametro e si richiama M_g iterativamente.



Prova per tutti i tentativi e si ferma quando $g(x,z)=0$ e si ferma sul valore prima di $g(x,z)$

$g(x, z) \stackrel{?}{=} 0$

si
no

si ferma
incrementa ed itera