tcs riassunto generale

Theoretical computer science (Politecnico di Milano)

Sottostringa x di s --> s = yxz, se y=Ɛ allora x è prefizzo; se z= Ɛ allora x è suffisso

Stella di Kleene * --> A alfabeto A* è è l'insieme di tutte le stringhe su simboli di A, inclusa la stringa

vuota es. A={a,b,c} --> A*={Ɛ, a,b,c,aa,ab,ac,ba,bb,bc,ca,..}

Un **linguaggio L** su un alfabeto A è un sottoinsieme di A*

$L^0 = \varepsilon$

Chiusura di Kleene:

- $L^* = \bigcup_{n=0}^{\infty} L^n$

{ε} ≠∅

{ε}·L=L

∅·L=∅

- $L^+ = \bigcup_{n=1}^{\infty} L^n$

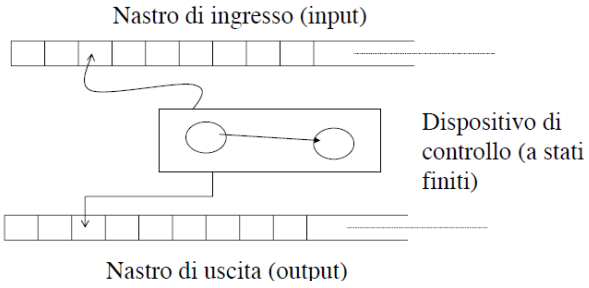
Automi a Stati Finiti **FSA**

ha in insieme finito di stati, sistema con limitato numero di configurazioni.
Un FSA è definito su un albero, quando si riceve un ingresso, il sistema cambia stato--> transazioni
FSA è una tupla <Q,A, δ , q_o ,F>

Q: insieme finito di stati
A: alfabeto di ingresso
δ: funzione di transizione QxA-->Q, può essere parziale
q_0 : stato iniziale
F: insieme di stati finali

TRANSDUTTORI A STATI FINITI **FST** **es_ s/a-> ingresso/uscita**

Automi come traduttori di linguaggi, è un FSA che lavora su due nastri



è una tupla <Q,I,δ, q_0 ,f,O,η>
O: alfabeto di uscita
η: QxI-->O*

PUMPING LEMMA: Se si attraversa un ciclo una volta, lo si può attaversare n volte

Condizione necessaria ma non sufficiente affinchè un linguaggio sia regolare.
Si utilizza per provare che un linguaggio è NON regolare.
è vero per linguaggi finiti.

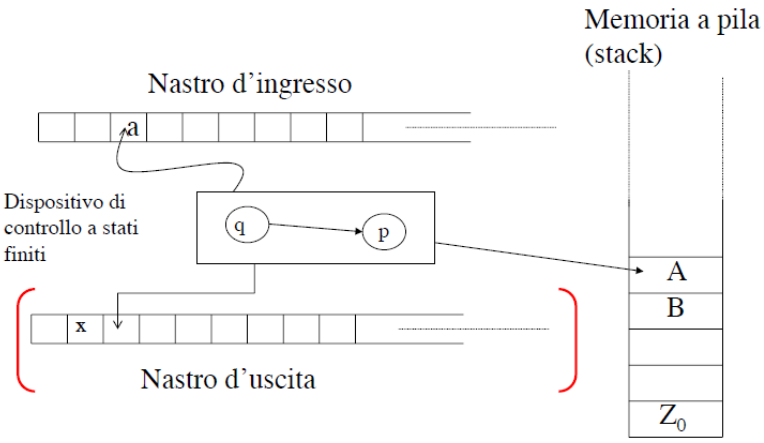
Un **insieme S** è **chiuso rispetto ad una** operazione **OP** se, quando OP è applicata agli elementi di S, il risultato è ancora un elemento di S

Leggi di DE MORGAN:

$A \cup B = \neg(\neg A \cap \neg B)$ --> ¬ è il completamento

Pushdown Automata PDA

aumentare la potenza degli FSA, incrementando la loro memoria.



i nuovi simboli sono inseriti in cima, la pila viene letta dalla cim , un simbolo letto viene estratto dalla cima

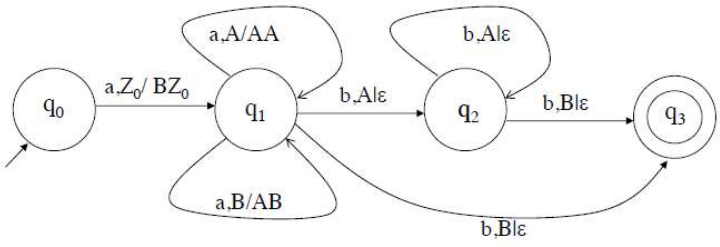
- LIFO

Automi a pila
possono usare la cima della pila per decidere quale transizione effettuare
possono manipolare la pila durante una transizione

In base a
- simbolo letto dall'ingresso (ma si può anche non leggere nulla)
- simbolo letto dalla cima della pila
- stato del dispositivo di controllo

il PDA
- cambia il proprio stato
- sposta in avanti la testina di lettura
- sostituisce il simbolo letto dalla pila con una stringa a (eventualmente vuota)

$L=\{a^n b^n \mid n>0\}$



questo linguaggio non può essere rappresentato con un FSA.
Ogni linguaggio regolare può essere riconosciuto da un PDA.
quando lo stack è vuoto l'automata deve andare allo stato finale

A differenza degli FSA, i PDA potrebbero non fermarsi dopo un numero finito di mosse, ma in questo caso non aggiungerebbero potere espressivo alla classe dei PDA. Ci sono PDA Ciclici e Aciclici (con carattere terminatore) +
L'eliminazione dei cicli è essenziale affinchè si raggiunga la fine di una stringa

RIPASSO LOGICA

FOL è la logica del primo ordine o nota come logica dei predicativi.
l'alfabero di un linguaggio FOL è composto da:

- un insieme infinito numerabile di variabili X,Y,Z...

- un insieme di simboli di funzione f,g,..
- un insieme di simboli di predicati (o relazioni) p,q,r..
- i connettivi ¬, ∧ , ∨ , → , ↔
- quantificatori ∀,∃
- simboli di punteggiatura ( ) e le virgole

MACCHINA DI TURING **TM**

la pila non può contare il numero di "a", perchè nella pila una volta letto il simbolo, questo viene distrutto, per cui può essere usata per controllare che il numero di "b" sia uguale al numero di "a" , come si può ricordare n che controllare il numero di "a"?

è necessario usare **una memoria persistente --> nastri di memoria**, questi non sono distruttivi e possono essere letti molte volte, sono come sequenze infinite di celle con un simbolo speciale detto "Blank" ("_" ,"-")

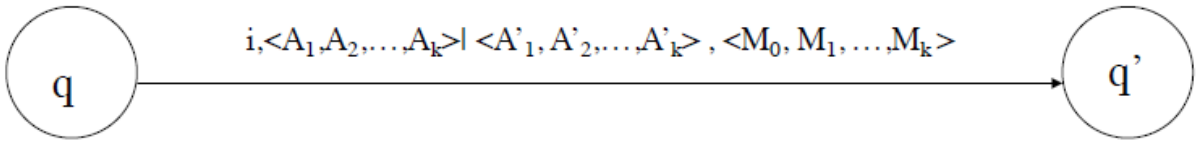
Azioni

- Cambio di stato
- Scrittura di un simbolo che rimpiazza quello letto dal nastro di memoria
- Spostamento delle K+1 testine

le testine della memoria e dell'ingresso possono essere spostate in 3 direzioni

- A destra di una posizione (R)
- A sinistra di una posizione (L)
- Ferme (S)

La direzione deve essere specificata esplicitamente



- 'i' è il simbolo di ingresso
- A_j è il simbolo letto dal j-esimo nastro di memoria
- A_j' è il simbolo che rimpiazza A_j
- M_0 è la direzione della testina del nastro d'ingresso
- M_j $(1 \le j \le k)$ è la direzione della testina del j-esimo nastro

TM e PDA

- Sappiamo che $a^n b^n c^n$ o $a^n b^n \cup a^n b^{2n}$ non possono essere riconosciuti da alcun PDA, ma possono essere riconosciuti da una TM
- Ogni linguaggio riconoscibile da un PDA può essere riconosciuto da una TM
  - Si può sempre costruire una TM che usa un nastro di memoria come se fosse una pila
- I linguaggi accettati dalle TM sono detti RICORSIVAMENTE ENUMERABILI

Le TM possono simulare le macchine di Von Neumann, la differenza riguarda l'accesso alla memoria, perchè le TM hanno accesso sequenziale, le VNM hanno accesso diretto

MODELLI OPERAZIONALI **NON DETERMINISTICI ND**

è un modello di computazione, è un'astrazione utile per descrivere problemi e algoritmi di ricerca. Sono diversi dai modelli stocastici.

FSA+ND--> **NFA** è UNA TUPLA < Q, I, δ, q_0 , F > ciò che cambia è δ

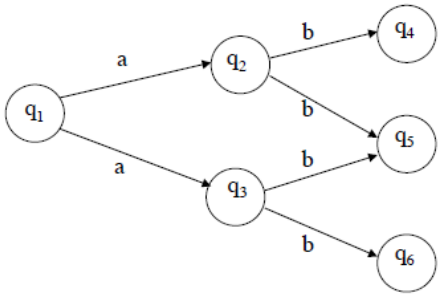
- Esempio:

  $\delta(q_1,a) = \{q_2, q_3\}$,

  $\delta(q_2,b) = \{q_4, q_5\}$,

  $\delta(q_3,b) = \{q_6, q_5\}$

→ $\delta^*(q_1,ab) = \{q_4, q_5, q_6\}$



Dato un NFA si può sintetizzare automanicamente un DFA come segue:

Se $A_{ND} = <Q, I, \delta, q_0, F>$ allora

$A_D = <Q_D, I, \delta_D, q_{0D}, F_D>$, dove

- $Q_D = \mathcal{P}(Q)$
- $\delta_D(q_D,i) = \bigcup_{q \in q_D} \delta(q,i)$
- $q_{0D} = \{q_0\}$
- $F_D = \{q_D \mid q_D \in Q_D \wedge q_D \cap F \neq \emptyset\}$

TURING $\delta: (Q-F) \times I \times \Gamma^k \rightarrow \mathcal{P}(Q \times \Gamma^k \times \{R,L,S\}^{k+1})$ MACHINE NON DETERMINISTICHE-- > **NTM**

Occorre $\eta: (Q-F) \times I \times \Gamma^k \rightarrow \mathcal{P}(O \times \{R,S\})$ cambiare la funzione di transizione e la funzione di traduzione.

la prima:

la seconda:

COMPUTABILITA'

Esiste un insieme di macchine che riescono a risolvere i porblemi in maniera automatica.
Molti problemi possono essere formalizzati come riconoscimento di linguaggi o traduzione
I nostri formalismi sono adeguati per tutti i problemi con domini numerabili
(in es. per Ogni --> F)

Tesi di Church

Non c'è nessun formalismo per modellare il calcolo meccanico che sia più potente della TM (o formalismi equivalenti)
Ogni algoritmo può essere codificato mediante una TM --> problema computabile

cardinalità delle funzioni computabili N-->N è maggiore o uguale della cardinalità delle funzioni definite da N-->{0,1} Che possiamo considerare uguale alla cardinalità del power set di N = 2^N0 che è la cardinalità nel continuo.
{Solvable Problems} cotenuto {Definable Problems}

Given (predicate) function $g(y, x) = 1$ if $f_y(x)\neq\perp$, $g(y,x) = 0$ if $f_y(x)=\perp$
Does there exist a TM that computes g?

no

Proof

- It employs a typical *diagonal technique* (adopted also in the Cantor theorem to show that $\aleph_0 < 2^{\aleph 0}$)
- Let us ***assume*** (by contradiction) that the total function :
  $g(y,x) = 1$ if $f_y(x) \neq\perp$, $g(y,x) = 0$ if $f_y(x) =\perp$
  is computable
- Then also the partial function
  $h(x) =$ 1 if $g(x,x) = 0$ (i.e., if $f_x(x) =\perp$),
  $\perp$ if $g(x,x) = 1$ (i.e., if $f_x(x) \neq\perp$)
  is computable
  NB: we went on the *diagonal* $y=x$, we changed the no answer ($g(x,x) = 0$) into a yes answer ($h(x)=1$), and we turned the yes ($g(x,x) = 1$) into a nontermination ($h(x)=\perp$), which can always be easily done
- If h is computable then $h = f_{xh}$ for some xh.
- Question: $h(xh) = 1$ or $h(xh) =\perp$?

f NON INDEFINITA
F INDEFINITA

xh indice di Goedel
non esiste una TM in grado di computare se una funzione è definita o indefinita

- Let us assume that $h(xh) = f_{xh}(xh) = 1$
- Then $g(xh,xh) = 0$, that is, $f_{xh}(xh) =\perp$:
- A contradiction

- Then let us assume the opposite: $h(xh) = f_{xh}(xh) =\perp$
- Then $g(xh,xh) = 1$, that is, $f_{xh}(xh) \neq\perp$:
- Another contradiction

then the assumption was wrong

QED

corollario:
Decidability and semidecidability
problemi con risposta binaria

1)Se S è decidibile allora è anche semidecidibile
se S è vuoto allora è semid. per definizione

supponendo che S non sia vuoto e data la sua funzione caratteristica allora deve esistere una k appartenente ad S tale che cs(k) =1, allora è possibile definire una funzione generatrice gs per cui sarà = x se cs(k)=1 se no gs=k. gs totale e computabile--> S= Igs --> S semidecidibile.

slide 65

2) S è decidibile se e solo se sia S che il suo complementare sono semidecidibili

B) equivalent to: (B.1 S recursive → both S and $S^\wedge$ RE) **and**
(B.2 both S and $S^\wedge$ RE → S recursive)

- B.1.1) S recursive → S RE (already proved in part A)
- B.1.2) S recursive → $c_S(x)$ (= 1 *if* $x \in S$; = 0 *if* $x \notin S$) computable
  → $c_{S^\wedge}(x)$ (= 0 *if* $x \in S$; = 1 *if* $x \notin S$) computable
  → $S^\wedge$ recursive → $S^\wedge$ RE
- B.2) S RE → construct the enumeration $S = \{g_S(0), g_S(1), g_S(2), g_S(3),...\}$
  $S^\wedge$ RE → construct $S^\wedge = \{g_{S^\wedge}(0), g_{S^\wedge}(1), g_{S^\wedge}(2), g_{S^\wedge}(3),...\}$
  But $S \cup S^\wedge = \mathcal{N}$, $S \cap S^\wedge = \varnothing$
  hence $\forall x \in \mathcal{N}$, x belongs to exactly one of the two enumerations →
  If one constructs the enumeration
  $\{g_S(0), g_{S^\wedge}(0), g_S(1), g_{S^\wedge}(1), g_S(2), g_{S^\wedge}(2), g_S(3), g_{S^\wedge}(3),...\}$
  one can certainly find in it any x: if x is at an odd position, then $x \in S$,
  if it is at an even position then $x \in S^\wedge$. Hence c_S can be computed.

S è semidecidibile <--> S= Dh h funzione parziale e computabile

S is RE $\leftrightarrow S = D_h$, with h computable and partial: $S = D_h = \{x \mid h(x) \neq \perp\}$
and
S is RE $\leftrightarrow S = I_g$, with g computable and partial: $S = I_g = \{x \mid \exists y \in \mathcal{N} : x = g(y)\}$

esistono dei set semidecidibili che non sono decidibili

$K = \{x \mid f_x(x) \neq \perp\}$ is semidecidable because $K = D_h$ with $h(x) = f_x(x)$.
We know however that the characteristic function of K,
($c_K(x) = 1$ *if* $f_x(x) \neq \perp$, 0 otherwise) is not computable (see function $h(x)$, slide 178)
$\Rightarrow K$ is not decidable

Teorema di Rice

capisco se una funzione è decdibile o meno, ossia se F è un insieme vuoto, per cui non estite nessuna funzione computabile, oppure se F è l'insieme di tutte le funzioni computabile, allora è decidibile—> paradosso, non esiste una macchinache risolve qualsiasi tipo di funzione

<u>per il teorema di rice la funzione non è decidibile</u>

Se il set contiene tutte le possibili funzioni computabile, se ogni funzione ha come risposta si o no--> è decidibile

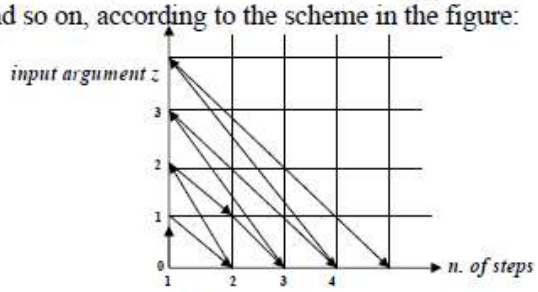
- If we find an algorithm that always terminates → decidable
- If we find an algorithm that may not terminate, but it always terminates when the answer is positive → semidecidable

Problem reduction

.....

dovetailing

- In general: Theorem (abstract formulation of the various concrete cases above):
  - The set of values x for which $\exists\, z$ such that $f_x(z) \neq \perp$ is semidecidable
    (NB: here the "input parameter" of the problem is the TM index x)
  - Sketch of the proof
    - If I compute $f_x(0)$ and find it $\neq \perp$ then there is no problem in providing an answer;
    - The problem arises if the computation of $f_x(0)$ does not terminate and $f_x(1) \neq \perp$: how can I find it?
    - Then I use the following trick, known as ***dovetailing***:
      - I simulate 1 execution step of $f_x(0)$: if it stops, then I have answered positively the question;
      - Otherwise I can simulate one computation step of $f_x(1)$;
      - Again if it does not stop I can simulate 2 steps of $f_x(0)$; next 1 step of $f_x(2)$; 2 steps of $f_x(1)$; 3 of $f_x(0)$; and so on, according to the scheme in the figure:



This way, if $\exists\, z$ s.t. $f_x(z) \neq \perp$, eventually I find it because eventually I will simulate enough computation steps of $f_x(z)$ to reach the stop

COMPLESSITA'

Posto che un dato problema sia risolvibile algoritmicamente, vogliamo sapere quanto ci costa risolverlo

Effettueremo analisi quantitative su:

Tempo di calcolo impiegato

Spazio occupato (registri, cache, RAM, disco, nastro)