Aniello Murano

Macchina di Turing universale e problema della fermata

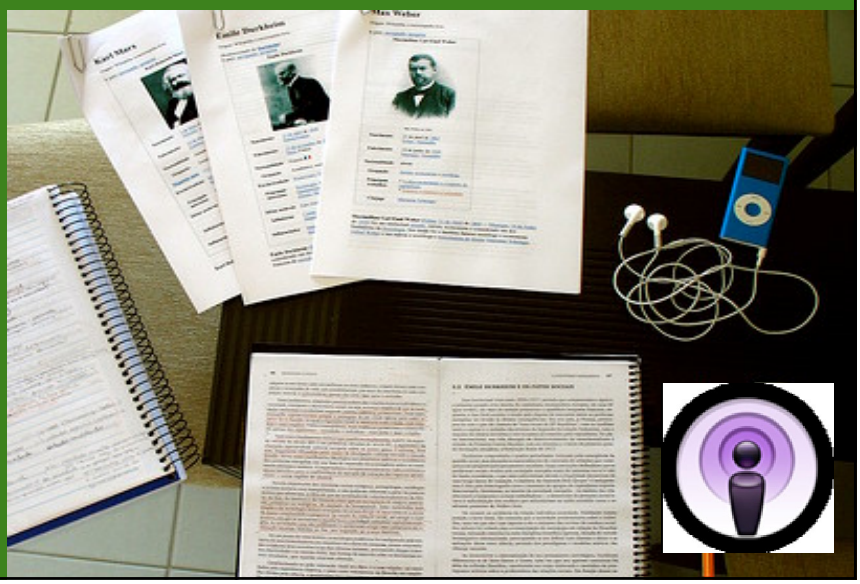
Lezione n.6
Parole chiave:
Universal Turing machine

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Riassunto delle lezioni precedenti

- Nelle lezioni precedenti abbiamo introdotto il concetto di macchina di Turing deterministica e non deterministica e abbiamo mostrato alcuni semplici linguaggi accettati.
- Abbiamo visto che le due versioni sono equivalenti ma passare da una macchina non deterministica a una deterministica può richiedere un salto esponenziale
- Con le macchine di Turing deterministiche è stato possibile caratterizzare la classe di complessità TIME(f(n)) di cui fa parte la classe P, e la classe SPACE(f(n)) di cui va parte la classe L (logarithmic)
- Con le macchine di Turing nondeterministiche è stato possibile caratterizzare la classe di complessità NTIME(f(n)) di cui fa parte la classe NP, e la classe NSPACE(f(n)) di cui va parte la classe NL.

L'interesse nella macchina di Turing

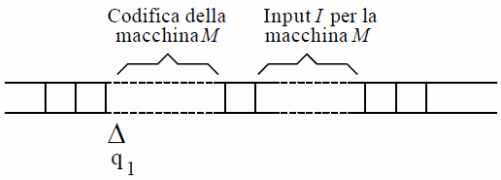
- L'interesse nella macchina di Turing (TM) per gli informatici risiede soprattutto nel fatto che essa rappresenta un **modello di calcolo algoritmico**, di un tipo di calcolo cioè che è automatizzabile in quanto eseguibile da un dispositivo meccanico.
- Ogni TM è il modello astratto di un calcolatore - astratto in quanto prescinde da alcuni vincoli di limitatezza cui i calcolatori reali devono sottostare;
- Per esempio, la memoria di una TM (vale a dire il suo nastro) è potenzialmente estendibile all'infinito (anche se, in ogni fase del calcolo, una TM può sempre utilizzarne solo una porzione finita), mentre un calcolatore reale ha sempre limiti ben definiti di memoria.
- Dunque, una TM M che accetta un linguaggio è analoga a un programma che implementa un algoritmo.
- In questa lezione, mostreremo come una TM si può in realtà vedere come equivalente a un moderno calcolatore con un linguaggio di programmazione e una riserva illimitata di memoria.

La macchina di Turing universale e il modello di von Neumann

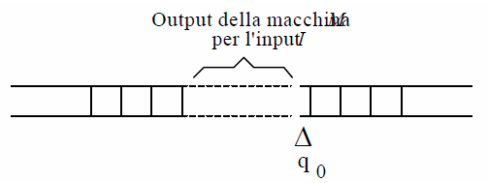
- Sino ad ora abbiamo considerato TM in grado di effettuare un solo tipo di calcolo, sono cioè dotate di un insieme di oggetti che consente loro di calcolare una singola funzione (ad esempio la somma, il prodotto, ecc.).
- Tuttavia, è possibile definire una TM, detta **Macchina di Turing Universale** (MTU), in grado di simulare il comportamento di ogni altra TM.
- Questo è reso possibile dal fatto che gli oggetti che definiscono ogni TM possono essere rappresentati in modo tale da essere scritte sul nastro di una TM.
- In particolare, è possibile sviluppare un metodo per codificare mediante numeri naturali la tavola di transizione di una qualsiasi TM.
- In questo modo, il codice di una TM può essere scritto sul nastro e dato in input a un'altra TM.
- La codifica può essere definita in maniera tale che, dato un codice, si possa ottenere la tavola di transizione corrispondente e viceversa mediante un procedimento algoritmico (una codifica che goda di questa proprietà è detta una **codifica effettiva**).

L'interesse nella macchina di Turing

- Si può dimostrare che esiste un TM (la MTU appunto) che, preso in input un opportuno codice effettivo delle componenti di un'altra macchina, ne simula il comportamento.
- Più formalmente, la MTU U è una macchina il cui input è composto dalla concatenazione di due elementi (si veda la figura in alto a lato):
  1. la codifica della tavola di transizioni di una TM M;
  2. un input I per M.
- Per ogni M e per ogni I, la MTU "decodifica" le tuple che definiscono M, e le applica ad I, ottenendo lo stesso output che M avrebbe ottenuto a partire da I (come mostrato nella figura a lato in basso).
- Formalmente si dice che U(M;I)=M(I)



Inizio della simulazione di M



Fine della simulazione di M

Dettagli sulla simulazione (1)

- Siccome U deve poter simulare qualsiasi TM M, non può essere considerato un limite superiore "a priori" per il numero di stati e di simboli di M che U deve considerare.
- Per questo motivo si assume che stati e simboli di M sono numeri interi.
- In particolare, si assume che
  - l'alfabeto Σ di M sia {1, 2, … , |Σ|},
  - l'insieme di stati K sia {|Σ|+1, |Σ|+2, … , |Σ|+|K|},
  - lo stato iniziale s=|Σ|+1,
  - Gli stati "accept", "reject" sono codificati con |Σ|+2 e |Σ|+3
  - I numeri |Σ|+|K|+1 e |Σ|+|K|+2 codificano gli spostamenti "left" e "right"
  - La funzione di transizione è ottenuta in modo ovvio, rappresentano le regole come tuple di numeri.
  - Tutti i numeri saranno codificati come numeri binari di lunghezza $\lceil \log(|K| + |\Sigma|) \rceil$

Dettagli sulla simulazione (2)

- La codifica della TM M in input per U comincerà con il numero |K| e poi |Σ| entrambi in binario e separati da virgole.
- Segue poi una descrizione di δ in termini di quintuple ((q,a),(p,b,*d*)), con *d* in {left, right}.
- Poi segue un ";" che ha il compito di segnalare la fine della descrizione di M.
- Ancora, si inserisce la codifica in binario della parola input $x = x_1, \ldots, x_k$, con la virgola usata come separatore degli interi binari che codificano i singoli simboli.
- Gli oggetti aggiuntivi (parentesi, virgola, punto e virgola, ecc.) possono anche essere codificati con altri interi successivi a quelli utilizzati.
- **Nota:** Ogni codifica "algoritmica" effettiva va bene.
- **Nota:** La rappresentazione di M e la rappresentazione del suo input possono anche essere messi su due nastri differenti, vista l'equivalenza (polinomiale) tra una macchina di Turing a più nastri e una ad un solo nastro.

Dettagli sulla simulazione (3)

- La MTU sull'input $<M,x_1 \ldots x_k>$ ha due nastri:
- il primo contiene l'input $<M,x_1 \ldots x_k>$
- il secondo contiene la (codifica della) configurazione corrente di M, nella forma (q,u,v) dove uv è il contenuto del nastro di M ad un certo punto della sua computazione, q è lo stato in cui si trova M e il simbolo in lettura è il primo di v.
- I primi passi della MTU servono per scrivere sul secondo nastro la codifica della configurazione iniziale, $(s,x_1 \ldots x_k)$
- Per simulare un passo di M:
  - U esamina il secondo nastro fino a trovare la codifica binaria dello stato corrente q (ricordiamo che è un numero tra $\{|\Sigma|+1, |\Sigma|+2, \ldots, |\Sigma|+|K|\}$)
  - cerca sul primo nastro una regola per q
  - poi muove la testina del secondo nastro per individuare il simbolo in lettura per M e controlla se la regola in lettura sul primo nastro coinvolge lo stesso simbolo input; se sì la regola viene implementata (cambiando la configurazione sul secondo nastro in corrispondenza) altrimenti si controlla la regola successiva
- Chiaramente, quando M si ferma, anche U si ferma.

Macchine di Turing programmabili

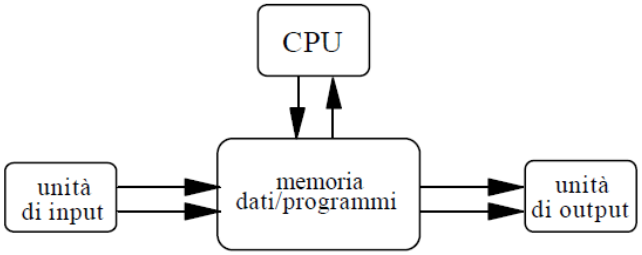
- Poiché la MTU è in grado di simulare il comportamento di qualsiasi TM, allora essa, in virtù della Tesi di Church, è in grado di calcolare qualsiasi funzione che sia calcolabile mediante un algoritmo.
- Ciò che caratterizza la MTU rispetto alle TM usuali è costituito dal fatto di essere una macchina calcolatrice **programmabile**.
- Mentre infatti le normali macchine di Turing eseguono un solo programma, che è "incorporato" nella tavola di transizione, la MTU assume in input il programma che deve eseguire.
- In pratica la MTU riceve in input la codifica di una TM M che deve simulare, dove la codifica di M ha proprio la funzione di consentire alla MTU di interpretare e di eseguire il "programma" rappresentato dalla TM M.

Macchine di Turing e l'architettura di Von Neumann (1)

- Un'altra caratteristica fondamentale della MTU è data dal tipo di trattamento riservato ai programmi.
- La MTU tratta i programmi (cioè la codifica delle tuple della TM da simulare) e i dati (l'input della TM da simulare) in maniera sostanzialmente analoga: essi vengono memorizzati sullo stesso supporto (il nastro), rappresentati utilizzando lo stesso alfabeto di simboli ed elaborati in modo simile.
- Queste caratteristiche sono condivise dagli attuali calcolatori, che presentano la struttura nota come *architettura di von Neumann* (dal nome dello scienziato di origine ungherese John von Neumann che la ideò).

Macchine di Turing e l'architettura di Von Neumann (2)

- La struttura di un calcolatore di von Neumann è raffigurata, molto schematicamente, nella figura.
- Un dispositivo di input e un dispositivo di output permettono di accedere dall'esterno alla memoria del calcolatore, consentendo, rispettivamente, di inserirvi e di estrarne dei dati.
- Le informazioni contenute in memoria vengono elaborate da una singola unità di calcolo (detta CPU - *Central Processing Unit*), che opera sequenzialmente su di essi.



Macchine di Turing e l'architettura di Von Neumann (3)

- La caratteristica più importante della macchina di von Neumann è costituita dal fatto che sia dati che programmi vengono trattati in modo sostanzialmente omogeneo, ed immagazzinati nella stessa unità di memoria.
- Questo consente una grande flessibilità al sistema.
- Ad esempio, poiché dati e programmi sono oggetti di natura omogenea, è possibile costruire programmi che prendano in input altri programmi e li elaborino, e che producano programmi in output.
- Queste possibilità sono ampiamente sfruttate negli attuali calcolatori digitali, e da esse deriva gran parte della loro potenza e della loro facilità d'uso (ad esempio, un compilatore o un sistema operativo sono essenzialmente programmi che operano su altri programmi).
- In questo senso limitato, un calcolatore di von Neumann costituisce una realizzazione concreta della MTU (e la memoria dati/programmi può essere considerata l'equivalente del nastro della MTU).

Macchine di Turing e l'architettura di Von Neumann (4)

- Anche la potenza computazionale tra il calcolatore di von Neumann e la macchina di Turing è la stessa:
- se si suppone che il calcolatore di von Neumann è dotato di memoria e tempi di calcolo illimitati, esso è in grado di calcolare tutte le funzioni computabili secondo la Tesi di Church (per questo si dice che una macchina di von Neumann è un *calcolatore universale*).
- La MTU costituisce quindi un modello astratto degli attuali calcolatori digitali (elaborato prima della loro realizzazione fisica).
- Si noti che anche i vari linguaggi di programmazione sviluppati in informatica consentono di definire tutte e sole le funzioni ricorsive (purché, ovviamente, si supponga che tali linguaggi "girino" su calcolatori ideali con memoria e tempi di calcolo illimitati).
- Questo vale sia per i linguaggi di programmazione di alto livello (come PASCAL, FORTRAN, BASIC, VISUAL BASIC, C, C++, JAVA, LISP, PROLOG, eccetera), sia per i vari tipi di *codice assembler*.