Informatica Teorica

Anno Accademico 2022/2023

Fabio Zanasi
https://www.unibo.it/sitoweb/fabio.zanasi

Quarta lezione

Nelle puntate precedenti…

- La tesi di Church-Turing
- A supporto della tesi, abbiamo dimostrato che variazioni che apparentemente ‘migliorano' la macchina di Turing sono in realtà equivalenti.

In questa lezione

- Ad ulteriore supporto della tesi di Church-Turing, introduciamo due modelli di calcolo alternativi, e dimostriamo la loro equivalenza con le macchine di Turing:
  - macchine a registri
  - programmi scritti nel linguaggio WHILE.

Macchine a registri

Perché questo modello?

Un'importante differenza tra macchine di Turing e architetture hardware a noi più familiari é che le macchine di Turing possono accedere ai dati della computazione solamente in maniera **sequenziale**, scorrendo un nastro.

Introduciamo ora un modello astratto di calcolo che non ha tale limitazione: le **macchine a registri** (*unlimited register machines*, URM).

Nel mostrare che le TM e le URM sono equivalenti, raggiungiamo due scopi:

- dimostriamo che la sequenzialità non é una limitazione intrinseca delle macchine di Turing;
- diamo ulteriore supporto alla tesi di Church-Turing.

Macchine a registri

Le URM sono state introdotte nel 1963 come modello alternativo alle macchine di Turing.

Computability of Recursive Functions*

J. C. SHEPHERDSON

University of Bristol, England†

AND

H. E. STURGIS

University of California, Berkeley, USA

1. *Introduction*

As a result of the work of Turing, Post, Kleene and Church [1, 2, 3, 9, 10, 11, 12, 17, 18] it is now widely accepted[1] that the concept of "computable" as ap-

Registri, input e ouput

Una macchina a registri ha (infiniti) registri R_1, R_2, …, R_n, …

Ogni registro R_n ospita un numero naturale, indicato con r_n.

Le URM calcolano funzioni di tipo $\mathbb{N}^k \to \mathbb{N}$.

L'**input** $x \in \mathbb{N}^k$ di una URM é suddiviso tra i primi k registri R_1, R_2, …, R_k nel modo ovvio. Tutti gli altri registri sono inizializzati col valore *0*.

Se una computazione ha termine (come al solito, non é necessario che questo avvenga) allora l'**output** sarà il valore r_1 del primo registro.

Programmi e istruzioni

Una computazione procede seguendo un **programma** P. I programmi per URM assomigliano al codice assembler.

P é una sequenza di **istruzioni** I_1, I_2, …, I_j. Ci sono quattro tipi di istruzione.

1. **Zero**: Z(n) imposta il valore di R_n uguale a 0.
2. **Successor**: S(n) aggiunge uno al valore di R_n.
3. **Move**: M(n,m) imposta il valore di R_m come uguale al valore r_n di R_n.
4. **Jump**: J(n,m,p) fa ‘saltare’ il programma all'istruzione I_p se $r_n = r_m$, e continuare regolarmente in caso contrario.

Computazioni di URM

- Una computazione coinvolge un programma $P = I_1, I_2, \ldots, I_j$ e un input contenuto nei registri.
- La computazione comincia leggendo la prima istruzione I_1 in P.
- Ad ogni passo,
  - se abbiamo eseguito I_i e non é un'istruzione Jump, allora proseguiamo con I_{i+1}.
  - Se I_i é un'istruzione Jump J(n,m,p) e $r_n = r_m$, allora proseguiamo dall'istruzione I_p. Altrimenti continuiamo con l'istruzione I_{i+1}.
- Una computazione termina se proseguiamo oltre l'ultima istruzione I_j.

Esempio 1

$$
\begin{aligned}
&I_1 : J(2,3,6)\\
&I_2 : S(1)\\
&I_3 : S(3)\\
&I_4 : J(2,3,6)\\
&I_5 : J(1,1,2)
\end{aligned}
$$

Esercizio: prova sull'input (3,2).
Che cosa calcola questa URM?

Esempio 2: moltiplicazione

$$I_1 : J(1, 3, 16)$$
$$I_2 : J(2, 3, 16)$$
$$I_3 : S(3)$$
$$I_4 : J(2, 3, 17)$$
$$I_5 : M(1, 3)$$
$$I_6 : S(5)$$
$$I_7 : Z(4)$$
$$I_8 : S(1)$$

$$I_9 : S(4)$$
$$I_{10} : J(3, 4, 12)$$
$$I_{11} : J(1, 1, 8)$$
$$I_{12} : Z(4)$$
$$I_{13} : S(5)$$
$$I_{14} : J(2, 5, 17)$$
$$I_{15} : J(1, 1, 8)$$
$$I_{16} : Z(1)$$

Equivalenza con le TM

Una funzione **parziale** é una il cui valore non é necessariamente definito su tutti gli argomenti.

Sia URM che TM calcolano funzioni parziali di tipo $\mathbb{N}^k \to \mathbb{N}$, in quanto su alcuni input potrebbero non terminare.

Teorema Una funzione (parziale) é calcolabile da una URM se e solo se é calcolabile da una TM.

Idea della dimostrazione Il punto cruciale é comprendere come passare da un formato di memoria all'altro (nastri vs registri).

TM simula URM: la costruzione

Per dimostrare una direzione dell'equivalenza, costruiamo una macchina di Turing con sei nastri.

- Il nastro 1 mantiene il "contatore del programma", cioé l'indice dell'istruzione che stiamo eseguendo al momento dell'URM.
- Il nastro 2 mantiene il codice del programma.
- Il nastro 3 mantiene i valori nei registri, scritti in notazione unaria e separati da ⊔.
- I nastri 4-6 saranno "fogli di brutta" (cache).

contatore del programma
codice del programma
contenuto dei registri

TM simula URM: la costruzione

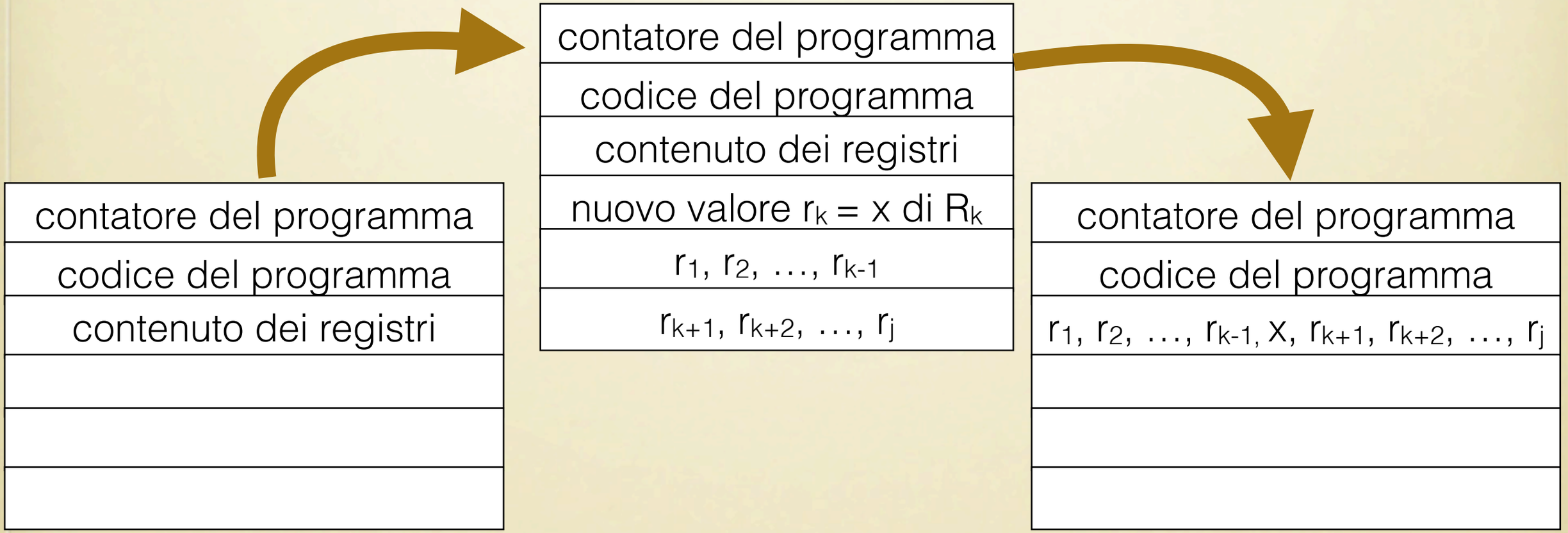
La macchina di Turing simula la computazione dell'URM come segue.

1. Usiamo il contatore (nastro 1) per identificare qual é l'istruzione corrente da eseguire nel codice del programma (nastro 2).
2. Eseguiamo l'istruzione (una tra Zero, Move, Successor e Jump).
3. Ripeti da 1.

contatore del programma
codice del programma
contenuto dei registri

TM simula URM: la costruzione

Notiamo che ciascun tipo di istruzione modifica il valore di qualche registro (nastro 3) eccetto Jump. La modifica del valore di un registro R_k avviene utilizzando i nastri ausiliari:



URM simula TM: la costruzione

Per l'altro verso dell'equivalenza, vogliamo costruire una URM che simuli l'esecuzione di una TM:

$$\mathcal{M} = \langle \Sigma, Q, q_0, H, \delta \rangle$$

La costruzione richiede la definizione di uno **schema di codifica** che traduca la definizione ed il comportamento di $\mathcal{M}$ in numeri naturali (da immagazzinare nei registri).

URM simula TM: la costruzione

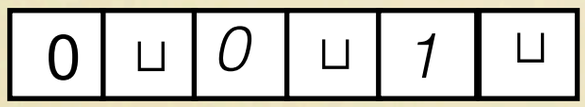
Pere cominciare, introduciamo un registro TAPE il cui contenuto codifica il contenuto corrente del nastro della TM $\mathcal{M}$.

Possiamo assumere senza perdita di generalità che $\Sigma = \{0, 1, \sqcup\}$. Introduciamo la codifica

code(⊔) = 0 code(**0**) = 1 code(1) = 2

e possiamo esprimere il contenuto del nastro in modo univoco come un numero a base-3, ad esempio:

$$801 = 1 \times 3^0 + 0 \times 3^1 + 1 \times 3^2 + 0 \times 3^3 + 2 \times 3^4$$

codifica 

⋯

URM simula TM: la costruzione

Idea della dimostrazione nel suo complesso:

1. introduciamo registri che codifichino la funzione di transizione di $\mathcal{M}$:

$Q\text{-}IN_i$ $\sigma\text{-}IN_i$ $Q\text{-}OUT_i$ $\sigma\text{-}OUT_i$

per ogni tupla $(q_{in}, \sigma_{in}, q_{out}, \sigma_{out})_i$ in δ.

2. introduciamo registri che codifichino qualsiasi configurazione di $\mathcal{M}$:

TAPE HEAD-POS HEAD-SYM HEAD-STATE

3. definiamo nuovi tipi di istruzione ausiliari Read, Write, Move-left, Move-right utilizzando ripetutamente i quattro tipi di istruzione di base.

4. scriviamo un programma di URM che usi queste istruzioni su questi registri per simulare una computazione di $\mathcal{M}$.

WHILE: un linguaggio di programmazione di alto livello

Perché questo modello?

Entrambi i modelli che abbiamo visto finora (TM and URM) sono più simili al linguaggio macchina che ad un linguaggio di alto livello di cui possiate avere esperienza quotidiana come Python, Java, C.

Nonostante le apparenze, la tesi di Church-Turing implica che anche questi linguaggi non hanno più potere espressive di modelli apparentemente più primitivi come le TM e le URM.

Dimostriamo ciò introducendo un linguaggio di programmazione ‘prototipico’, WHILE, e mostrandolo equivalente alle macchine di Turing.

Per approfondire: Kfoury, Moll, Arbib - *A programming approach to computability*.

Sintassi di WHILE, informalmente

Il linguaggio WHILE ha tutti gli ingredienti di base di un linguaggio imperativo standard.

1. Assegnazione di valore a una variabile (i valori sono numeri naturali)

   X := 3

2. Cicli while

   while X ≠ Y do PROGRAM

3. Sequenziamento di programmi

   PROGRAM1 ; PROGRAM2 ; PROGRAM3

4. Altri costrutti come if-then-else possono essere definiti a partire da questi primitivi.

Sintassi, definizione formale

PROG ::= begin end | begin SEQCMD end

SEQCMD ::= CMD | SEQCOM ; CMD

CMD ::= ASS | while TEST do CMD | PROG

ASS ::= VAR := 0 | VAR := s(VAR) | VAR := p(VAR)

TEST ::= VAR ≠ VAR

VAR ::= CHAR | VAR CHAR | VAR DIGIT

CHAR ::= A | B | … | Z

DIGIT ::= 0 | 1 | … | 9

Esempio

```
begin
  begin Z := s(X); Z := p(Z) end;
  begin U := 0;
    while U ≠ Y do
      begin Z := s(Z); U := s(U) end
  end
end
```

Esercizio: qual é il valore di Z dopo aver eseguito il programma?

Semantica

Un programma WHILE computa una funzione parziale $\mathbb{N}^k \to \mathbb{N}^k$.

Esempio

```
begin
  begin Z := s(X); Z := p(Z) end;
  begin U := 0;
    while U ≠ Y do
    begin Z := s(Z); U := s(U) end
  end
end
```

Computa (x, y, z, u) ↦ (x, y, x+y, y)

La definizione completa della semantica si può trovare in *A programming approach to computability*, Cap.2.3.

Equivalenza

Teorema Una funzione (parziale) é computabile da un programma WHILE se e solo se é computabile da una macchina di Turing.

Idea della dimostrazione

Ci focalizziamo sulla simulazione di un programma WHILE da parte di una macchina di Turing. Questa é forse la parte più ‘sorprendente' del teorema, dal momento che le macchine di Turing sono all’apparenza più primitive.

Equivalenza

Idea della dimostrazione

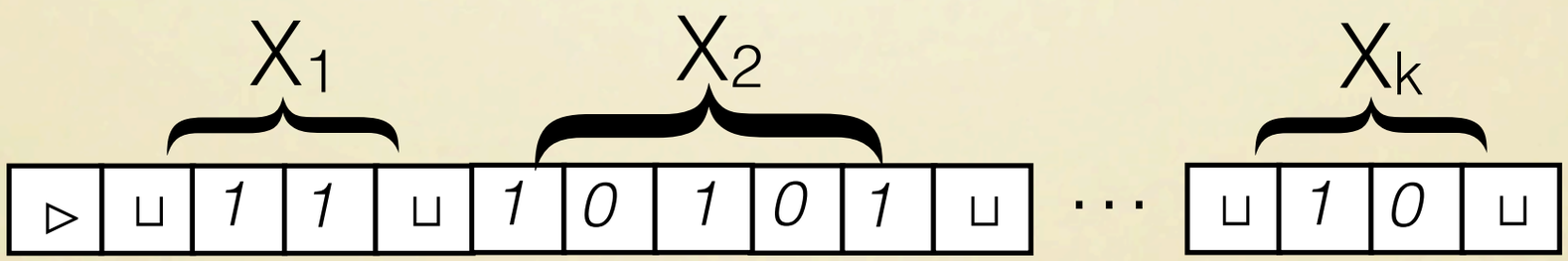
La dimostrazione é per induzione sulla struttura di un programma WHILE, supponiamolo basato su variabili $X_1, \ldots, X_k$.

Caso base

Se il programma é un'assegnazione di 0 a una variabile:

begin X_j := 0 end

la TM corrispondente é quella che su input



rimpiazza il valore di X_j con 0.

Equivalenza

Caso base

I rimanenti casi base sono il programma vuoto e le altre due forme di assegnazione (esercizio).

Caso induttivo

Ci sono due casi induttivi da considerare: sequenza e ciclo while.

Equivalenza

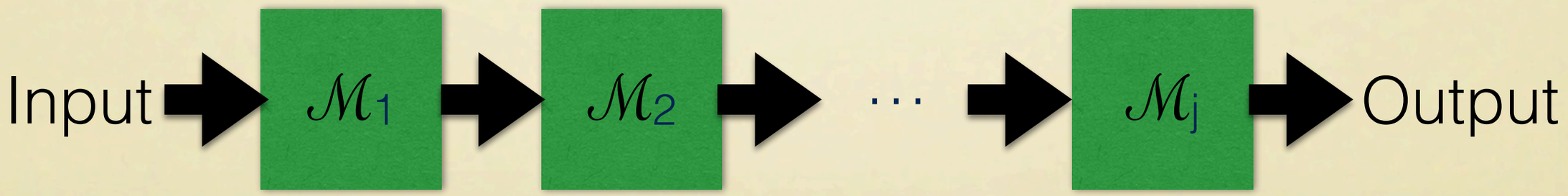
Caso induttivo: sequenza

Per assunzione, abbiamo un programma della forma

begin P_1 ; P_2 ; … ; P_j end

dove P_1, P_2, … , P_j sono programmi per i quali abbiamo, da ipotesi induttiva, TM equivalenti $\mathcal{M}_1$, $\mathcal{M}_2$, …, $\mathcal{M}_j$ rispettivamente.

La TM per begin P_1 ; P_2 ; … ; P_j end é definita nel seguente modo a partire da esse, dove l'output di $\mathcal{M}_i$ viene dato come input di $\mathcal{M}_{i+1}$.



Equivalenza

Caso induttivo: ciclo while

Assumiamo un programma della forma

begin while $X_i \neq X_j$ do P end

dove P é un programma per il quale, da ipotesi induttiva, abbiamo una TM $\mathcal{M}$ equivalente.

Possiamo costruire facilmente una TM $\mathcal{M}_{\text{test}}$ che rigetta l'input se il valore di X_i e X_j é diverso, ed accetta altrimenti.

La TM per il nostro programma é costruita come segue: