Lezione 10 – modelli di calcolo e Tesi di Church-Turing

Lezioni del 26/03/2025

Ri-facciamo il punto

- Siamo partiti cercando di capire come risolvere automaticamente i problemi
- E abbiamo studiato la soluzione proposta da Alan Turing che, partendo dalla sua analisi del processo di soluzione è arrivato a definire un modello di calcolo: la Macchina di Turing
  - che è un linguaggio per descrivere algoritmi
  - e ogni macchina di Turing è un algoritmo
- Poi, abbiamo introdotto i concetti di linguaggi decidibili e accettabili, e di funzioni calcolabili
  - che corrispondono, informalmente, ai problemi che sappiamo risolvere con la Macchina di Turing
- considerando, così, implicitamente, la possibilità che possano esistere linguaggi non decidibili – o, persino, non accettabili (uhmm…) – e funzioni non calcolabili
  - la possibilità che esistano problemi irrisolvibili – con la Macchina di Turing

A questo punto

- In effetti, poi, abbiamo dimostrato che linguaggi non decidibili (mediante macchine di Turing) esistono davvero e ne abbiamo anche visto uno (piuttosto importante: l'Halting Problem)
- A questo punto, la domanda sorge spontanea: non sarà forse possibile decidere / accettare quel linguaggio con un altro modello di calcolo?
- In fondo, cos'ha di tanto speciale questa Macchina di Turing????

- La nostra questione, ora, è dunque la seguente: esiste un modello di calcolo più potente della Macchina di di Turing? Ossia, un modello di calcolo che "sa risolvere più problemi" della Macchina di Turing?

Modelli di calcolo

- Siamo al paragrafo 3.3 della dispensa 3
- Dove si dice che sono stati definiti un sacco di modelli di calcolo
  - che, guarda caso, sono tutti basati sullo stesso concetto di "operazione elementare" utilizzato da Turing
  - perché esso corrisponde proprio all'umano modo di calcolare
- Ebbene: per tutti i modelli di calcolo definiti fino ad ora, è stato dimostrato che sanno "risolvere" tutti e soli i problemi che possono essere "risolti" mediante la Macchina di Turing
  - non uno di più, non uno di meno
- ossia, tutti i modelli di calcolo introdotti sino ad ora sono ***Turing-equivalenti***
- Viene quasi da pensare che un modello di calcolo più potente della Macchina di Turing non esista
  - posto che esso consideri "operazione elementare" una operazione che possa essere eseguita "a mente" da un umano medio!

La tesi di Church-Turing

- Questa tesi assume che non esista un modello di calcolo più potente della Macchina di Turing: dato un qualunque altro modello di calcolo $\mathcal{M}$,
  - se un linguaggio L è decidibile/accettabile nel modello $\mathcal{M}$ allora L è decidibile/accettabile nel modello Macchina di Turing
  - se una funzione f è calcolabile nel modello $\mathcal{M}$ allora f è calcolabile nel modello Macchina di Turing
  - e viceversa
- Purché $\mathcal{M}$ sia un modello" ragionevole"
  - ossia, sia basato sul concetto di *operazione elementare* del quale abbiamo parlato diffusamente
  - perché, se definiamo un modello di calcolo che disponga dell'unica operazione elementare "qualunque sia il problema, qualunque sia l'istanza del problema, trova la soluzione di quell'istanza",…
  - beh, è ovvio che questo modello è più potente della macchina di Turing!
  - Ma non è mica tanto realistico (nel senso che dalla Macchina di Turing sono nati i calcolatori, ma è difficile che nascano macchine reali che corrispondano a questo modello)

La tesi di Church-Turing

- Dunque:

  è calcolabile tutto e solo ciò che può essere calcolato dalla Macchina di Turing

- Attenzione: è una tesi, non è un teorema!
  - Non è mai stata dimostrata!
  - E sembra difficile riuscire a dimostrarla: sembra difficile riuscire a prevedere i modelli di calcolo che potrebbero essere definiti nel futuro…
  - Tuttavia, sembra poco probabile riuscire a progettare un modello di calcolo che non la soddisfi
  - e, non dimentichiamolo, tutti i modelli di calcolo esistenti la soddisfano
  - infatti, è generalmente accettata!
- Noi vedremo un paio di esempi di validità di questa tesi:
  - un linguaggio di programmazione
  - le grammatiche formali

Il modello di calcolo PascalMinimo

- È un linguaggio di programmazione – perché ogni linguaggio di programmazione è un modello di calcolo!
- Dispone di tutte le istruzioni "tipiche" dei linguaggi di programmazione
  - istruzione di assegnazione: a ← b
  - istruzione condizionale **if** … **then** ... **else**
  - istruzioni di loop **while** (…) **do** e **for** (...)
  - funzioni
  - istruzioni per l'input e l'output
  - ecc. ecc.
- E dispone di variabili semplici (intere, caratteri, ...) ma anche di variabili che corrispondono a collezioni di oggetti – insiemi e, soprattutto, **array**
  - se avete provato a risolvere un certo esercizio, questo dovrebbe dirvi qualcosa...
- E la descrizione di questo linguaggio la trovate a pag. 7 della dispensa 3

Il modello di calcolo PascalMinimo

- Nella dispensa 3, a partire da pag. 7, si accenna alla dimostrazione che il modello di calcolo PascalMinimo è equivalente alla Macchina di Turing
  - nel Teorema 3.5 si dà un'idea (grossolana) di come "trasformare" un programma in PascalMinimo in una macchina di Turing che "faccia le stesse cose"
  - nel Teorema 3.6 si dà un'idea (abbastanza precisa) di come "trasformare" una macchina di Turing in un programma in PascalMinimo che "faccia le stesse cose"

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.

- Sulla dispensa viene descritta l'idea della dimostrazione
  - in particolare, viene mostrato come "trasformare" un programma in PascalMinimo in una macchina di Turing di tipo trasduttore solo nel caso in cui il programma non utilizzi variabili strutturate
  - come, ad esempio, gli array

- Guardiamo questa idea di dimostrazione insieme
  - includendo nella nostra discussione anche gli array

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, prima di mostrare come costruire una macchina di Turing T che ''si comporta'' come $\mathcal{P}$, scriviamo $\mathcal{P}$ in una forma opportuna, ossia
  - 1) utilizzando variabili ausiliarie, eliminiamo gli array eventualmente presenti nelle condizioni
    - ad esempio,''if (A[i]=1) then ... '' diventa ''ausilA = A[i]; if (ausilA=1) then ...''
    - naturalmente, potremo utilizzare la stessa variabile di appoggio in più punti del programma
    - in questo modo, gli array compaiono solo nelle istruzioni di assegnazione
  - 2) scriviamo una sola istruzione in ciascuna riga del programma
  - 3) numeriamo le righe del programma

Il modello di calcolo PascalMinimo

- Idea della dimostrazione. Esempio: il seguente programma

Input: n valori interi positivi memorizzati nell'array A e un intero memorizzato nella variabile k

```
p ← 1; i ← 1;
 while (i ≤ n) do
       if (A[i] > p) then p ← A[i];
if (p > k) then ris ← k else ris ← p
output ris
```

- diventa:

```
1) p ← 1;
2) i ← 1;
3) while (i ≤ n) do  begin
4)     ausilA ← A[i];
5)     if (ausilA > p) then
6)             p ← A[i];
7) end
8) if (p > k) then ris ← k
9)     else ris ← p
10) output ris
```

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, costruiamo T, multinastro a testine indipendenti
  - Oltre ai nastri input e output, T utilizza:
    - un nastro per ciascuna variabile (incluse le variabili di tipo array)
    - un nastro aggiuntivo per ogni variabile di tipo array sul quale memorizzare in unario l'indice dell'array al quale si vuole accedere
    - un nastro di lavoro per la valutazione delle espressioni e delle condizioni contenute nel programma.
  - I contenuti dei nastri sono codificati in binario (o in unario) con una sola eccezione: nei nastri che corrispondono a variabili di tipo array viene utilizzato un carattere speciale, '$', come carattere separatore fra gli elementi dell'array

Il modello di calcolo PascalMinimo

- Idea della dimostrazione. Esempio: il seguente programma

Input: n valori interi positivi memorizzati nell'array A e un intero memorizzato nella variabile k

```
1) p ← 1;
2) i ← 1;
3) while (i ≤ n) do  begin
4)     ausilA ← A[i];
5)     if (ausilA > p) then
6)             p ← A[i];
7) end
8) if (p > k) then ris ← k
9)     else ris ← p
10) output ris
```

- La macchina corrispondente al programma utilizza 8 nastri:
  - 5 nastri per le variabili semplici: il nastro N_p per p, il nastro N_i per i, , il nastro N_{ausilA} per ausilA. il nastro N_k per k, il nastro N_{ris} per ris
  - un nastro N_A per la variabile di tipo array A e un nastro N_{indA} per l'indice con il quale accedere ad A
  - un nastro di lavoro

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, costruiamo T assegnando uno stato interno di T ad ogni riga del programma che contenga una istruzione,
- Vedremo ora come costruire le quintuple di T:
  - la descrizione che presenteremo è ad alto livello, rappresentando lo stato di partenza, le azioni che devono essere compiute a partire da quello stato e lo stato in cui entra T dopo che quelle azioni sono state eseguite
  - ad esempio con

    ⟨ q, **[copia N_b su N_a riavvolgendo le testine su N_b e N_a]**, q', ferme ⟩

    intendiamo che quando la macchina è nello stato q, indipendentemente da quello che viene letto sui nastri, T inizia ad eseguire una serie di quintuple che (utilizzando un certo numero di stati intermedi) copiano il contenuto del nastro N_b sul nastro N_a, poi riavvolgono le testine su quei nastri e infine, mantenendo ferme tutte le testine, portano T nello stato q'

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, costruiamo T assegnando uno stato interno di T ad ogni riga del programma che contenga una istruzione,
- Vedremo ora come costruire le quintuple di T:
  - la descrizione che presenteremo è ad alto livello, rappresentando lo stato di partenza, le azioni che devono essere compiute a partire da quello stato e lo stato in cui entra T dopo che quelle azioni sono state eseguite
  - invece con

    ⟨ q, **[sul nastro N_c è scritto xxx]**, q', ferme ⟩

    intendiamo che quando la macchina è nello stato q, se sul nastro N_c trova scritto xxx entra nello stato q' senza muovere le testine
  - xxx può essere una parola (invece che un singolo carattere): in tal caso T deve eseguire una serie di quintuple per verificare che su N_c sia scritta la parola xxx

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione di assegnazione fra variabili semplici, dopo aver eseguito l'assegnazione T entra nello stato q_{i+1}
  - ESEMPIO: **21**) a = b;
    22) if (a > 5) then
  - diventa:
    ⟨ q_{21}, **[copia N_b su N_a e posiziona le testine sui primi simboli di N_b e N_a]**, q_{22}, ferme ⟩
    - assumiamo che quando T entra in uno stato corrispondente a un'istruzione (ad esempio, in q_{21}) le testine sono sempre posizionate sui simboli più a sinistra su ciascun nastro
    - **[copia N_b su N_a e posiziona le testine sui primi simboli di N_b su N_a] corrisponde, in effetti ad un insieme di quintuple: terminata la copia e riposizionate le testine a sinistra, T entra in q_{22}**

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione di assegnazione che coinvolge una variabile A di tipo array:
    - 1) viene copiato su N_{indA} in unario il valore dell'indice dell'elemento di A cui si vuole accedere,
    - 2) ci si posiziona su quell'elemento,
    - 3) si esegue l'assegnazione dopo la quale T entra nello stato q_{i+1}

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - ESEMPIO: **15**) A[i] = b;
    16) ...
  - per semplificarci la vita, assumiamo che ogni elemento di A possa essere contenuto in un'unica cella del nastro N_A
    - in questo modo non utilizziamo il carattere separatore $ gli elementi dell'array sono scritti in celle contigue di N_A
  - allora, i tre punti descritti alla pagina precedente diventano

    $\langle q_{15}$, **[copia N_i in unario su N_{indA} e riposiziona le testine a sinistra]**, q_{15}^1, ferme $\rangle$
    $\langle q_{15}^1$, **[muovi a destra su N_{indA} e N_A sino al blank su N_{indA}]**, q_{15}^2, ferme $\rangle$
    $\langle q_{15}^2$, **[copia N_b su N_A e riposiziona le testine a sinistra]**, q_{16}, ferme $\rangle$

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione di assegnazione che coinvolge una variabile A di tipo array: 19 viene copiato su N_{indA} in unario il valore dell'indice dell'elemento di A cui si vuole accedere, 2) ci si posiziona su quell'elemento, 3) si esegue l'assegnazione dopo la quale T entra nello stato q_{i+1}
  - ESEMPIO: **15**) A[i] = b;
16) ...
  - per semplificarci la vita, abbiamo assunto che ogni elemento di A possa essere contenuto in un'unica cella del nastro N_A
  - questa assunzione può essere eliminata utilizzando il carattere separatore '$' su N_A (esercizio non proprio facile facile)

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione **if (*condizione*) then ...**, calcola *condizione* : se è vera T entra nello stato q_{i+1}, altrimenti entra nello stato corrispondente alo stato dell'istruzione da eseguire quando è falsa
  - ESEMPIO: **22**) if (a > 5) then
23) c = a;
24) ...
  - diventa:
$\langle q_{22}$, **[calcola il predicato $N_a > N_5$ scrivendo il risultato sul nastro di lavoro]**, q_{22}^1, ferme $\rangle$
$\langle q_{22}^1$, **[sul nastro di lavoro c'è scritto *vero*]**, q_{23}, ferme $\rangle$
$\langle q_{22}^1$, **[sul nastro di lavoro c'è scritto *falso*]**, q_{24}, ferme $\rangle$

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione **while (*condizione*) do...**, calcola *condizione* : se è vera T entra nello stato q_{i+1},*e poi di seguito fino all'ultima istruzione del loop che riporta T in* q_i, altrimenti entra nello stato corrispondente alo stato dell'istruzione da eseguire dopo il loop
  - ESEMPIO: **24**) while (a > 5) do begin
    25) a = a-1;
    26) . i = i+1; end
    27) ...
  - diventa:
    ⟨q_{24}, **[calcola il predicato $N_a > N_5$ scrivendo il risultato sul nastro di lavoro]**, q_{24}^1, ferme ⟩
    ⟨q_{24}^1, **[sul nastro di lavoro c'è scritto *vero*]**, q_{25}, ferme ⟩
    ⟨q_{25}, **[sottrai 1 al contenuto di N_a]**, q_{26}, ferme ⟩
    ⟨q_{26}, **[somma 1 al contenuto di N_i]**, q_{24}, ferme ⟩
    ⟨q_{24}^1, **[sul nastro di lavoro c'è scritto *falso*]**, q_{27}, ferme ⟩

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: dato $\mathcal{P}$ un programma in PascalMinimo, abbiamo costruito una macchina di Turing T
  - che utilizza tanti nastri quante sono le variabili e le costanti in $\mathcal{P}$ (più i nastri indice degli array)
  - che dispone di un insieme di stati interni ''principali'' ciascuno dei quali corrisponde a una istruzione in $\mathcal{P}$
- descrivendo ad alto livello le sue quintuple
- poiché le quintuple di T simulano, passo passo, le istruzioni di $\mathcal{P}$, intuitivamente quanto stabilito dall'asserto del teorema è vero
- ossia, abbiamo dimostrato *informalmente* il teorema
  - in effetti, abbiamo visto solo un' *idea della dimostrazione*