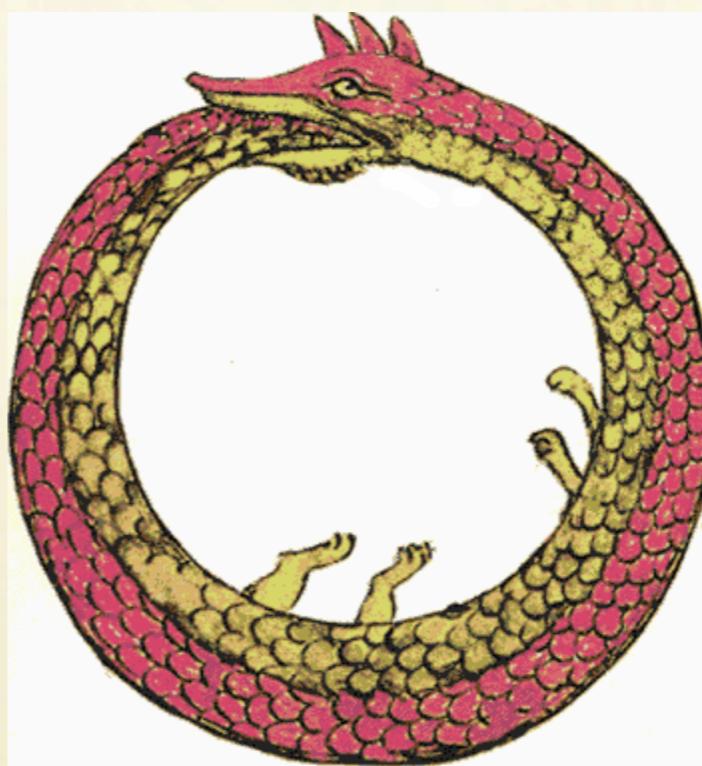


Informatica Teorica



Anno Accademico 2022/2023

Fabio Zanasi

<https://www.unibo.it/sitoweb/fabio.zanasi>

Quinta lezione

Nelle puntate precedenti

- Macchine di Turing, problemi decidibili e riconoscibili
- La tesi di Church-Turing e l'equivalenza delle macchine di Turing con altri modelli computazionali.

In questa lezione

Introdurremo un concetto cardine dell'informatica:
la possibilità di una macchina di Turing **universale**.

Facciamo un esperimento

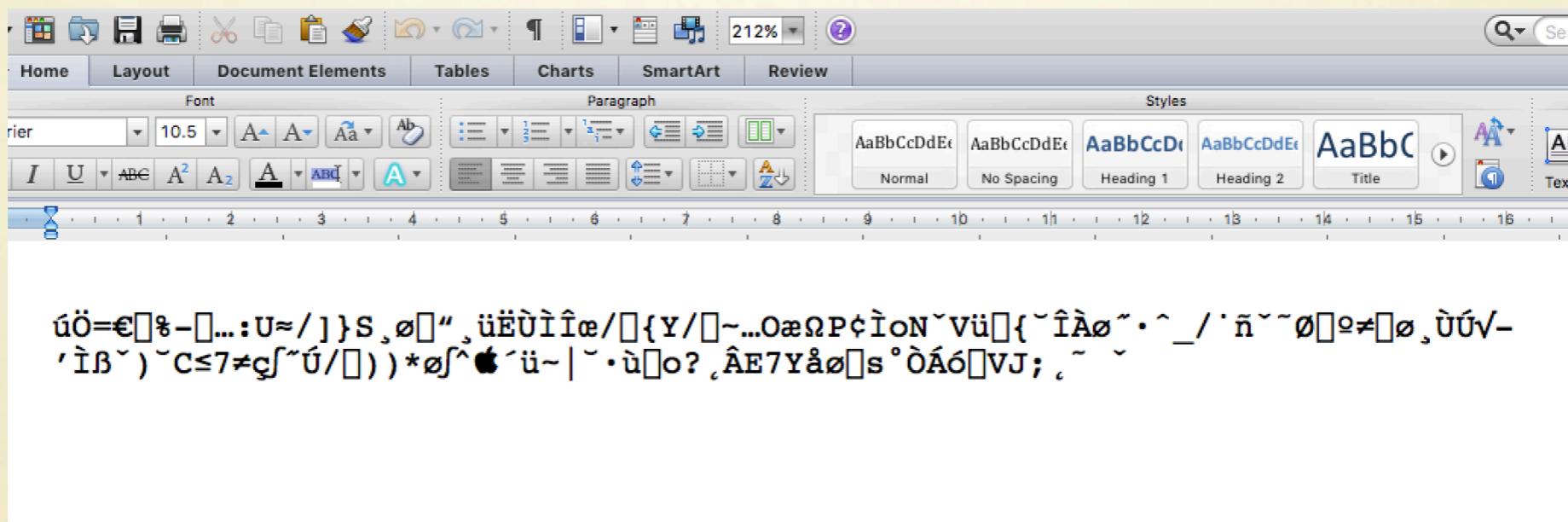
Consideriamo un programma di modifica di tesi, ad esempio Microsoft Word.

Apriamo ora un file *.doc con Microsoft Word.

Apriamo ora un file *.exe con Microsoft Word.

Cosa osserviamo?

Al contrario dei file *.doc, i file *.exe non producono un output sensato quando aperti con Microsoft Word.



Ma l'osservazione importante è che, in linea di principio, Microsoft Word **può eseguire** file *.exe.

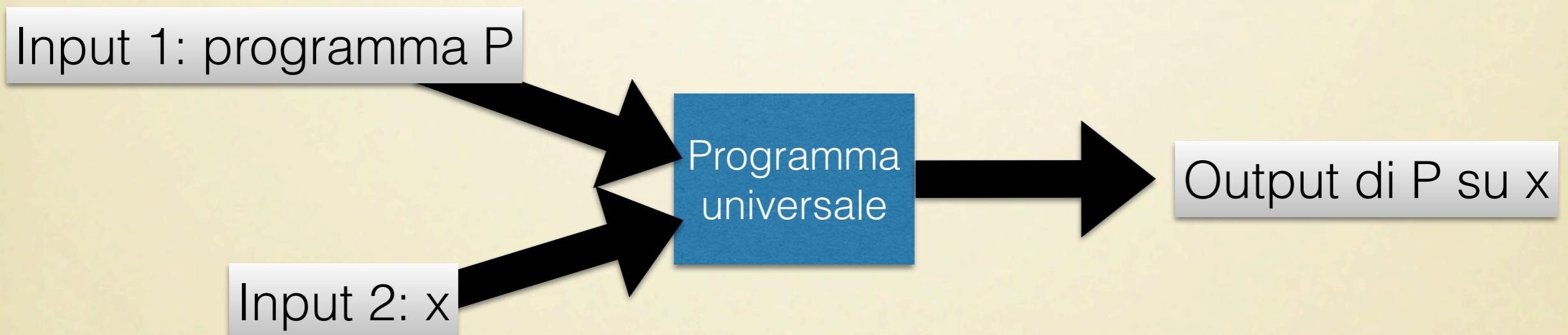
Può anche eseguire **se stesso** (WINWORD.exe).

Ricapitoliamo

1. Qualsiasi programma può eseguire qualsiasi file come input.
(sebbene l'output corrispondente non sarà interessante a meno che l'input è di un formato pensato per essere eseguito da quel programma.)
2. I programmi sono file come gli altri. Quindi un programma può ricevere altri programmi come input.
3. Un programma può addirittura ricevere il suo stesso codice come input.

Programmi universali

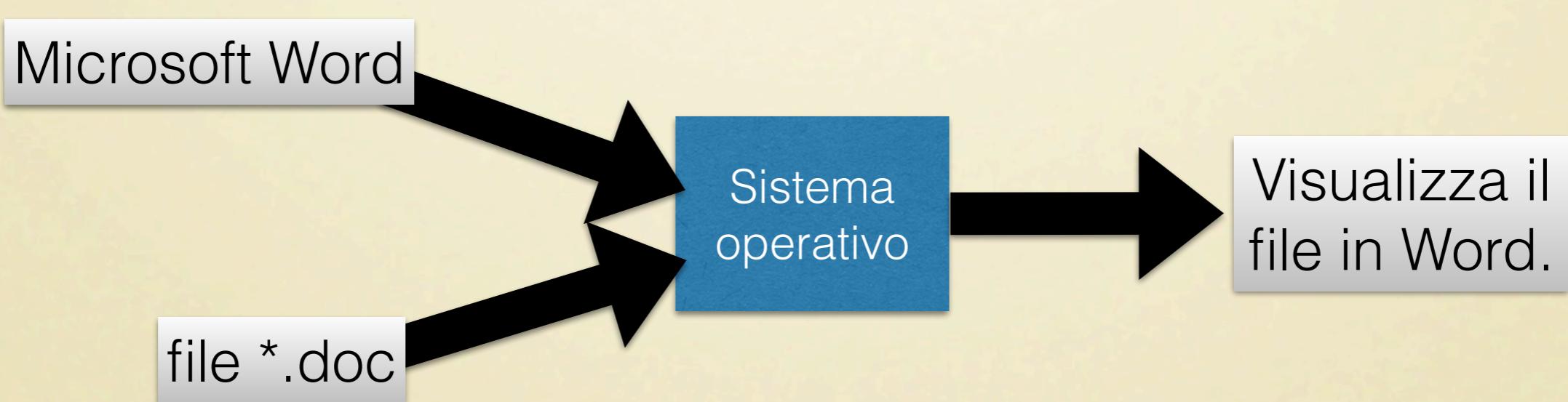
Un programma universale é un programma pensato per ricevere altri programmi come input ed eseguirli.



Programmi universali

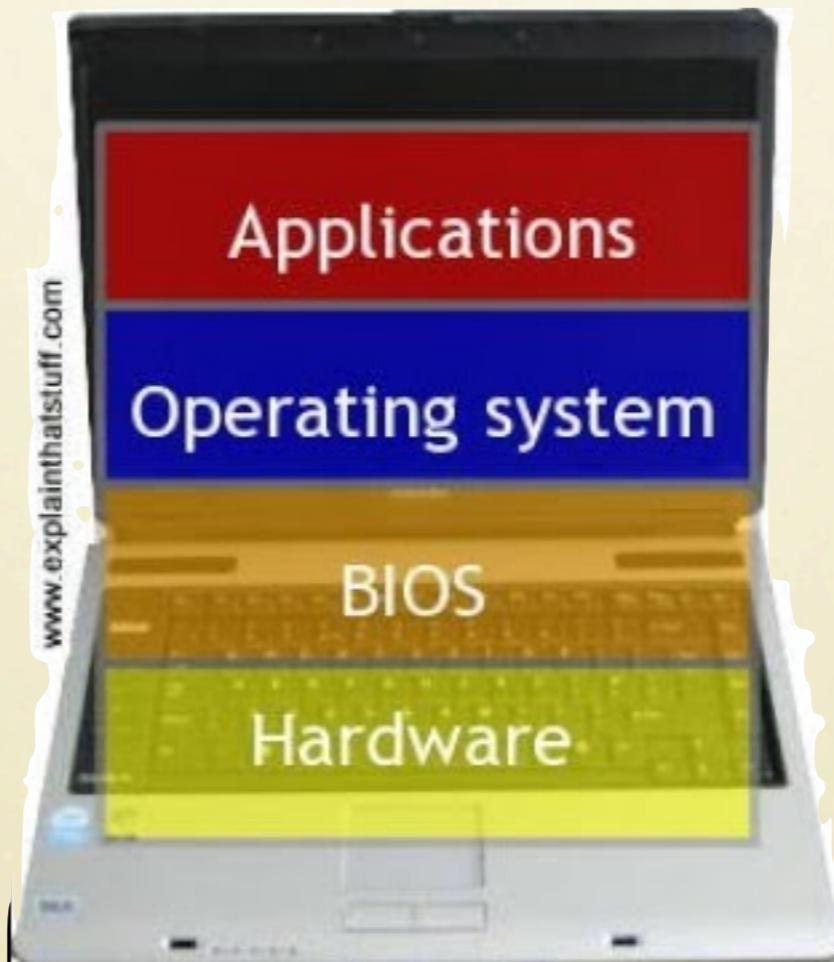
I sistemi operativi (Windows, OS-X, Linux, ...) sono esempi di programmi universali.

Quando clicchiamo su un file, il sistema operativo seleziona il programma adatto ad eseguire quel file.



Programmi universali

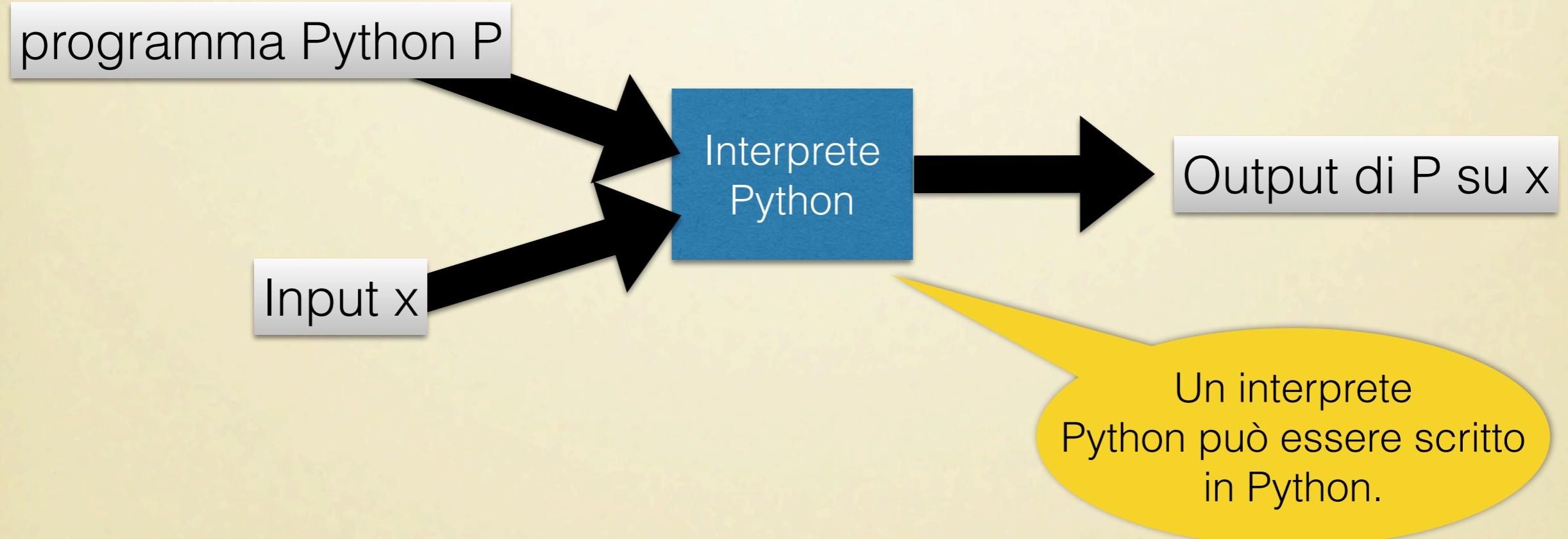
Possiamo pensare alle architetture dei calcolatori (*computer architectures*) come ad una gerarchia di macchine universali, le une eseguite dalle altre.



Programmi universali

Un **interprete** é un altro esempio di programma universale.

```
Fabios-MBP:~ Fabio$ python program.py input
```



Un po' di storia

Tutti questi concetti discendono dalla **macchina di Turing universale** (introdotta nel famoso articolo del 1936).

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO
THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The “computable” numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions

Un po' di storia

Nella storia umana le macchine per il calcolo sono state abitualmente costruite per assolvere un **singolo compito** (eseguire un determinato programma).



Le macchine multi-uso sono una invenzione moderna (di Turing, e in una certa misura anche di Charles Babbage). L'intuizione fondamentale è stata comprendere che non c'è differenza intrinseca tra un programma ed i suoi dati.



Una **macchina universale** è una che è in linea di principio capace di riprodurre qualsiasi algoritmo (è **programmabile**).

Autoreferenza

La scorsa settimana abbiamo visto come le URM possano essere codificate come input di TM.

Ora vedremo che le stesse TM possono essere codificate come input di altre macchine di Turing.

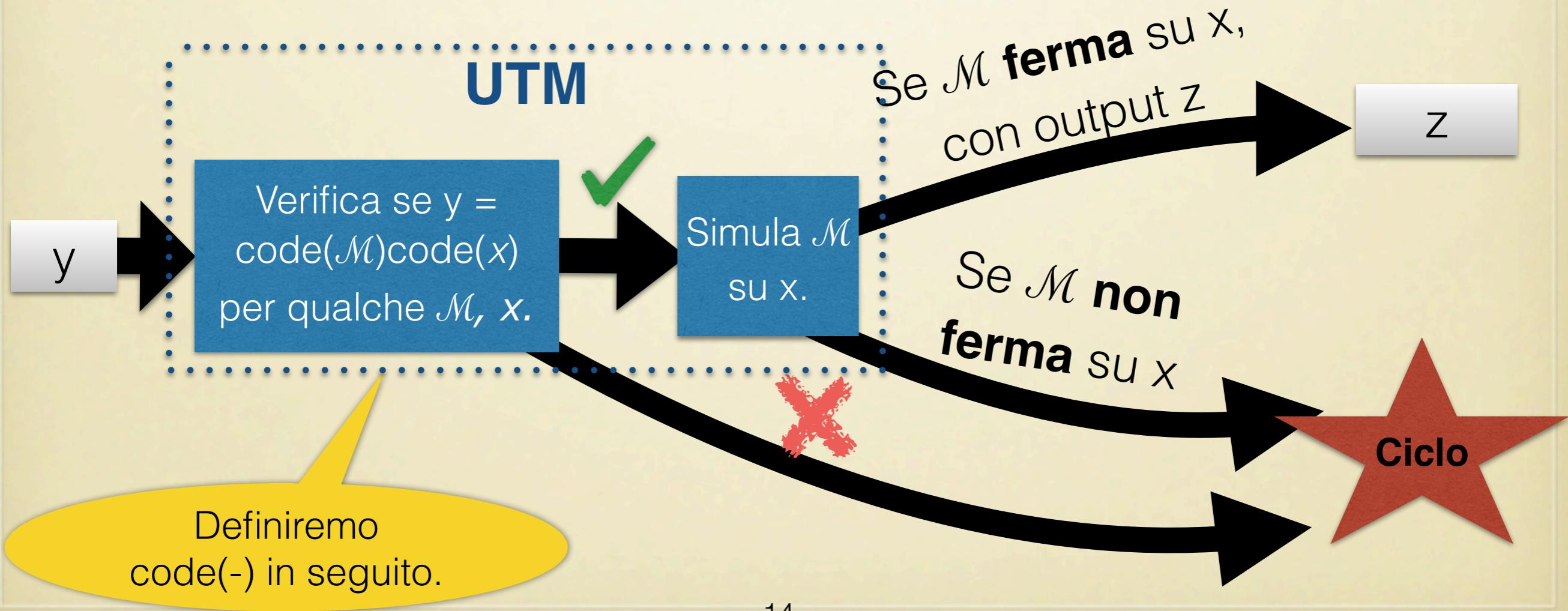
Questo tipo di codifica autoreferenziale è talvolta chiamato **Gödelizzazione**, da Kurt Gödel. Gödel l'utilizzò per codificare enunciati riguardanti l'aritmetica dentro l'aritmetica stessa (come numeri): un passo chiave nel suo *teorema di incompletezza* (1931).



Come visto, quest'idea è tutt'ora onnipresente in informatica (interpreti, funzioni d'ordine superiore, object oriented, ...)

La macchina di Turing universale (UTM)

- La UTM prende come input una stringa y .
- Per prima cosa, verifica che y sia della forma $\text{code}(\mathcal{M})\text{code}(x)$, dove $\text{code}(-)$ é una codifica, \mathcal{M} é una TM, e x una stringa nell'alfabeto di input $\Sigma_{\mathcal{I}}$ di \mathcal{M} .
- Se é cosí, allora la UTM simula l'esecuzione di \mathcal{M} su x



Cosa significa 'simulare' \mathcal{M} ?

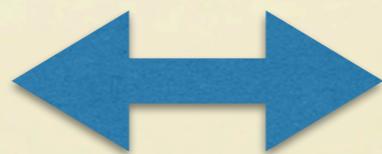
Eseguire la UTM su input $y = \text{code}(\mathcal{M})\text{code}(x)$ dovrebbe dare lo stesso risultato che eseguire \mathcal{M} su input x .

UTM non ferma su
 $\text{code}(\mathcal{M})\text{code}(x)$



\mathcal{M} non
ferma su x

UTM ferma
 $\text{code}(\mathcal{M})\text{code}(x)$
con output z .



\mathcal{M} ferma su x
con output z .

Esistenza di una UTM

Theorem Esiste una macchina di Turing universale.

Proof Ne costruiremo una esplicitamente (in una maniera simile a come fece Turing stesso).

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO
THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The “computable” numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions

Codificare macchine di
Turing

Codificare macchine di Turing

Ci concentriamo sulla definizione di code(-).

Tradurrà una TM in una stringa su alfabeto {0,1}.

Nota: ci sono molti modi del tutto corretti di definire una tale codifica.

Codificare macchine di Turing

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

Introduciamo alcune convenzioni.

- Gli stati in Q sono ordinati come q_0, q_1, q_2, \dots con q_0 iniziale.
- Ordiniamo i simboli che possono apparire nella definizione di δ

$$\sigma_0 = \sqcup$$

$$\sigma_1 = \rightarrow$$

$$\sigma_2 = \leftarrow$$

e gli altri simboli in Σ come $\sigma_4, \sigma_5, \dots$

Possiamo codificare gli stati ed i simboli come stringhe unarie:

$$\text{code}(q_i) = \underbrace{11\dots1}_{i+1 \text{ volte}}$$

$$\text{code}(\sigma_i) = \underbrace{11\dots1}_{i+1 \text{ volte}}$$

Codificare macchine di Turing

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

- Codifichiamo una tupla $t = \langle q_i, \sigma_n, q_j, \sigma_m, \sigma_o \rangle$ di δ come:

$$\text{code}(t) = \text{code}(q_i)0\text{code}(\sigma_n)0\text{code}(q_j)0\text{code}(\sigma_m)0\text{code}(\sigma_o)0$$

- E tutta la funzione di transizione $\delta = \{t_1, t_2, \dots, t_k\}$ come

$$\text{code}(\delta) = \text{code}(t_1)0\text{code}(t_2)0\dots0\text{code}(t_k)0$$

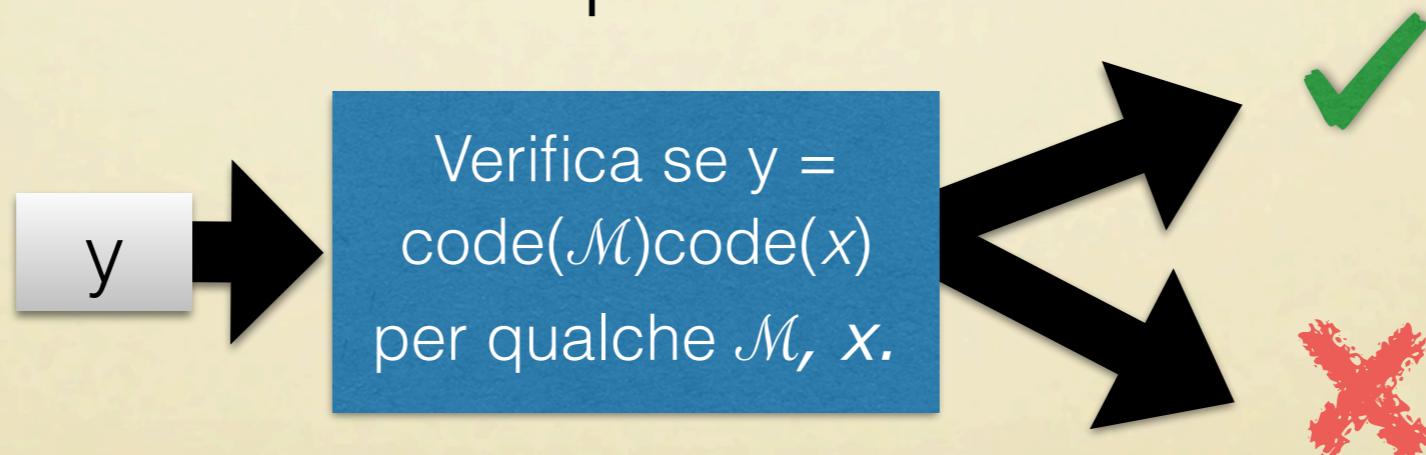
- Possiamo dedurre quali sono gli stati finali di H : sono quelli su cui δ non è definita (= non occorrono mai in terza posizione in una tupla).

Esempio

Alla lavagna.

Osservazioni sulla codifica

- È possibile che ci siano due o più macchine di Turing che computino la stessa funzione, ma codificate come stringhe differenti (intuitivamente: se esprimono un diverso algoritmo)
- Nondimeno, la codifica è *iniettiva*: due macchine differenti saranno codificate da stringhe differenti.
- Data una stringa su $\{0,1\}$, è possibile determinare se sia o meno il codice di una TM (e di quale). In particolare: si tratta di un problema **decidibile**.



Costruzione della macchina di Turing universale

Promemoria

Eseguire la UTM su input $y = \text{code}(\mathcal{M})\text{code}(x)$ dovrebbe dare lo stesso risultato che eseguire \mathcal{M} su input x .

UTM non ferma su
 $\text{code}(\mathcal{M})\text{code}(x)$



\mathcal{M} non
ferma su x

UTM ferma
 $\text{code}(\mathcal{M})\text{code}(x)$
con output z .



\mathcal{M} ferma su x
con output z .

Costruzione della UTM

La UTM é definita come una macchina di Turing con tre nastri.

Nastro 1 manterrà il nastro di \mathcal{M} in forma codificata.

Nastro 2 manterrà $\text{code}(\mathcal{M})$.

Nastro 3 manterrà lo stato corrente di \mathcal{M} in forma codificata.

Costruzione della UTM

Per cominciare una simulazione della UTM:

1. Passo di preparazione: verifica che $y = \text{code}(\mathcal{M})\text{code}(x)$ per qualche TM \mathcal{M} e input x . Se no, cicla. Se si, allora nastro 1 contiene $\text{code}(\mathcal{M})\text{code}(x)$. Vai al passo 2.
2. Sposta $\text{code}(\mathcal{M})$ dal nastro 1 al nastro 2. Ora nastro 1 mostra il contenuto del nastro di \mathcal{M} su input x alla configurazione iniziale, in forma codificata.
3. Scrivi $\text{code}(q_0)$ su nastro 3.
4. Posiziona testina 1 sul primo simbolo di $\text{code}(x)$, testina 2 sul primo simbolo di $\text{code}(\mathcal{M})$, and testina tre sul primo simbolo di $\text{code}(q_0)$.

Costruzione della UTM

Un passo della simulazione di \mathcal{M} da parte della UTM funziona nel seguente modo.

1. Cerca in $\text{code}(\mathcal{M})$ una tupla $\langle q_i, \sigma_n, q_j, \sigma_m, \sigma_o \rangle$ dove q_i concida con lo stato sul nastro 3 e σ_n coincida con il simbolo attualmente esaminato da \mathcal{M} .
2. Aggiorna nastro 1 con il nuovo simbolo σ_o e sposta la testina nella direzione σ_m .
3. Aggiorna nastro 3 con lo stato q_j . Se è finale, fermati.

Considerazioni finali

Questa costruzione mostra l'esistenza di una macchina di Turing universale, \mathcal{M}_U .

Niente impedisce a \mathcal{M}_U di ricevere la sua stessa codifica $\text{code}(\mathcal{M}_U)$ come parte dell'input!

Questa forma di autoreferenzialità sarà utilizzata nella prossima lezione per dimostrare che esiste un problema indecidibile.