La macchina di Turing

Introduzione

Note filosofiche (non impo) Bisogna in primo momento cercare di definire **cosa è la computazione** e cosa è un computer. Aristotele faceva la distinzione fra proprietà **essenziali** e **accidentali**. Quelle essenziali sono proprie dell'oggetto. > Una sedia può essere fatta di legno o di metallo, ma questa proprietà è accidentale, ovvero, essa rimane una sedia indipendentemente dal materiale di cui è fatta.

Solitamente in matematica si prova ad astrarre (vedi Astrazione sul controllo per nota generale sull'astrazione). Però in questo campo si sono trovati molte concezioni equivalenti. Fino ad arrivare a concepire la tesi di Church-Turing. Il prof. nota che questo è strano, perché in altre discipline si converge in unico modello, mentre qui molte cose sono indifferenti. Questo è importante per capire come la concezione di Computer Science si è evoluta (Denning 2010).

Nascita della calcolabilità: Turing (curiosità) Al tempo si voleva in matematica trovare un formalismo, un fondamenta logico alla matematica che poteva permettere di dimostrare tutto dalle fondamenta. Il contributo principale, e secondo il prof il lavoro più importante in tutta l'informatica era (Turing 1937), che definisce cosa significa calcolare un problema. Questo porta al significato di calcolare un problema. È interessante notare come Turing arriva al suo formalismo. Osserva 1. L'esecuzione di certe regole 2. Un foglio di carta in cui sono lette e scritte dei simboli 3. L'azione eseguita dipende dal simbolo precedente

Da queste osservazioni iniziali, hanno *astratto i dati* (ora simboli binari) e *le azioni*, un set molti semplice. Osservazione: **computazione è locale**. Questo sembra simile quanto dichiarato in (Dehaene 2014) quando si parla che l'essere umano è capace di porre davanti l'attenzione della coscienza solamente un solo simbolo alla volta.

La macchina di Turing

Vedere precedente per capire come è stata ideata questa macchina di Turing.

Definizione matematica È interessante confrontare questa definizione con Fondamenti teorica#La macchina di turing in cui usiamo un formato leggermente diverso, il formato preciso è quello precedente fatto a linguaggi. In sto corso usiamo un formalismo più semplice). Nel nostro caso è una 5-tupla di - Σ, un **alfabeto** di simboli finiti, con simbolo speciale per cella vuota - Q Un insieme di stati - $q_0 \in Q$ lo stato iniziale - $H \subseteq Q$ l'insieme degli stati finali - δ la funzione di transizione che soddisfa questo:

$$\delta : (Q - H) \times \Sigma \to Q \times \Sigma \times \{\to, \leftarrow\}$$

La differenza è che in questo caso l'alfabeto dell'input è uguale all'alfabeto del nastro, ma alla fine cambia poco, basta TODO (capire) secondo me ha sbagliato il prof. tempo fa, perché l'alfabeto di input non è mai preso in considerazione nella funzione di transizione boh.

Questa sintassi è più comprensibile della precedente quindi nice.

Come per tutti i precedenti automi, anche questi hanno una rappresentazione possibile a diagramma:

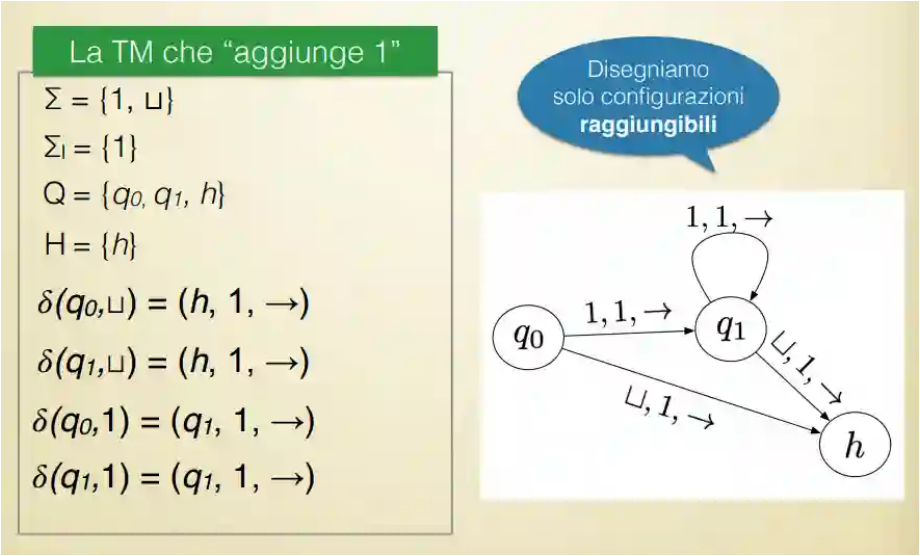

Figure 1: La macchina di Turing-20240221114813908

A lezione abbiamo anche visto esempi di macchine che computano moltiplicazione binaria o addizione binaria.

Problemi di decisione

Definizione problemi di decisione Molti problemi si possono codificare attraverso un problema di decisione, ossia un problema in cui abbiamo solamente

bisogno di una risposta sì o no. Se riusciamo a codificarlo con il linguaggio delle macchine di Turing, allora forse si può far verificare alla macchina. Si vedrà che molti problemi non vanno.

Schema di codifica Ossia un dato α sarà descritto con $code(\alpha) \in \Sigma^*$. Una nota interessante è che con Kolmogorov complexity abbiamo un dato di lunghezza minima, qui vediamo bene il link molto diretto. Questo ha delle proprietà: 1. È **Iniettiva** 2. Vorremmo capire in Σ^* quali siano dei codici possibili per un qualche α nel nostro dominio. 3. Sarebbe carino poter ritrovare α a partire dal suo codice.

Definizione decidibilità Dato un certo linguaggio, supponiamo di avere una macchina di Turing come definita di sopra #La macchina di Turing tale per cui abbia due stati finali $\{H, N\}$, allora diciamo che $\mathcal{M}$ **decide** L se vale che - Quando $x \in L$ allora $\mathcal{M}$ accetta x, ossia finisce su stato H - Quando $x \notin L$ allora $\mathcal{M}$ rigetta x, ossia finisce su stato N

Diciamo che un linguaggio L è decidibile se una macchina di Turing lo decide. Ora abbiamo formalizzato il significato di decidibilità. Un altro modo per dire che una macchina è decidibile è se **si ferma per ogni input**, questo significa che o finisce in stato accettante o in stato rigettante.

Definizione riconoscibilità/semidecibilità Uguale al precedente, con la differenza che quando la stringa non appartiene al linguaggio **diverge**.

Diciamo un linguaggio L è riconoscibile se una macchina di Turing lo riconosce. Th: **Decidibilità -> Riconoscibilità**, è facile costruire una tale macchina di Turing partendo da una che la decide, possiamo estendere il caso negativo in questo modo: se raggiungo lo stato negativo allora vado in loop infinito a caso.

Definizione non riconoscibilità Significa che non può dire in tempo finito né sì né no. Quindi è una cosa ancora più forte rispetto la semidecibilità.

Gerarchia di Chomsky + Vedere Linguaggi liberi e PDA#Classificazione dei linguaggi alla sezione schema generale delle grammatiche. La cosa da ricordare è che TM è il modello più generale fra tutti i precedenti modelli di macchine di Turing e automi.

Tesi di Church-Turing

Enunciato della tesi

> Se la soluzione di un dato problema può essere calcolata attraverso una procedura algoritmica, allora può essere calcolata da una macchina di Turing. (*Alonzo Church*)

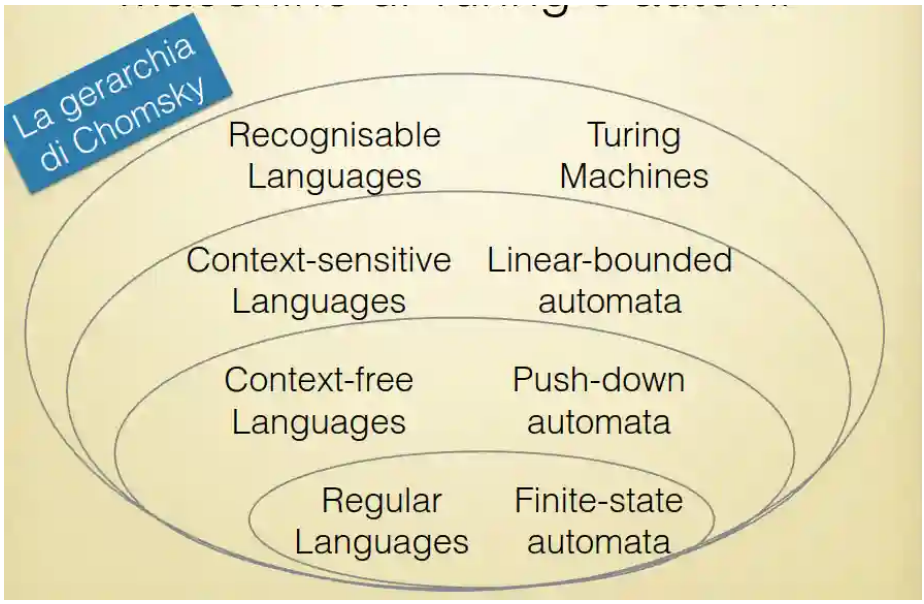

Figure 2: La macchina di Turing-20240512120403415

Alonzo proponeva una altra teoria di calcolo, è stato proponente di *lambda calcolo*. Quelli che piacevano a Asperti.

Storicamente sembra vero, perché sempre prendendo qualcosa che sembra più espressibile, resta alla fine equivalente a turing.

Esempi di conseguenze: - Macchine di Turing 'migliorate' (nondeterministiche, probabilistiche, più nastri…) - Macchine a registri - Linguaggi di programmazione di alto livello come Python, Java, C, … - (Codice macchina di) computer classici • Computer quantistici

Espressibilità vs semplicità e efficienza Probabilmente lo abbiamo citato in Fondamenti teorica, il fatto che questo è solamente una *congettura* perché non è possibile codificare tutti i formalismi possibili. Parla di **espressibilità** ossia cosa può essere calcolato, e se questo vale permette di dire che è una macchina universale. Infatti **non dice nulla su semplicità o efficienza** dell'algoritmo (in questa astrazione non ci interesssa).

Giustificazione valenza di Turing (non impo) La cosa interessante di queste macchine comunque è che La macchina di Turing ci permette di formalizzare! 1. Rigorosità di algoritmo. (anche se non mi sembra buono per esprimere certe forme di calcolo (Denning 2010)). 2. Teoremi di calcolabilità possono essere estese a qualunque altro formalismo, se vale.

Versione rafforzata Questa versione è una estensione della tesi di Church-Turing in modo che comprenda la parte in Time and Space Complexity.

> Ogni modello di calcolo deterministico fisicamente realizzabile può essere simulato da una TM (deterministica, su nastro singolo) con overhead al **più polinomiale**.

Se ci pensiamo questa versione rafforzata è simile a quanto dimostrato in Kolmogorov complexity riguardo la complessità sulla lunghezza della minima stringa che lo descrive, perché lì abbiamo un overhead al massimo costante per tradurre da una macchina all'altra.

Chiusura del linguaggio di TM

Chiusura sulla decidibilità **Complemento:** è molto semplice decidere sul complemento perché basta scambiare gli stati finali **Unione:** basta eseguirlo sul multinastro e comparare l'input, vedi Estensioni di Turing e altre macchine. **Intersezione**: Usi de morgan con i precedenti. **Concatenazione**: stessa dimostrazione per Grammatiche Regolari, concateni le macchine. **Star**: sì

Un esercizio è dettagliare la definizione di queste macchina, in modo simile a quanto facevamo al corso di linguaggi.

NOTA: ricorda di seguire la struttura della dimostrazione. Se no te la conterà come sbagliata all'esame.

Per la concatenazione è un po' più difficile del previsto, perché **non so bene dove devo andare a tagliare** per la seconda stringa, quindi vado in modo non deterministico sul taglio, così in pratica faccio tutte le cose possibili. #### Chiusura sulla riconoscibilità **Complemento**: No **Unione** sì, stessa cosa precedente. **Intersezione**: credo basti concatenarli con reset dell'input e dire che si accetta se arriva in fondo **Concatenazione**: sì **Star**: dovrebbe sì.

La non chiusura del complemento la trovi in Halting Theorem and Reducibility. Perché si dimostra che $HALT$ e il suo complementare non sono chiusi, e questo basta per il complemento. ### La macchina di Turing universale

Esempi di macchine universali Sono dei programmi in grado di eseguire altri programmi. È una cosa molto particolare dell'informatica questa cosa, permetterebbe per esempio di eseguire un programma su se stesso, una cosa ricorsiva (oroboro quasi). Esempi di macchine universali possono essere 1. Sistemi operativi 2. Stack di hardware che abbiamo, ognuna universale che esegue una sopra l'altra.

Anche questo è stato pensato in (Turing 1937). Una cosa interessante è che prima di esso, la macchina era pensata per una unica cosa, dopo Turing si può usare la stessa macchina per tutti gli algoritmi possibili. Ha introdotto la nozione di programmabilità! Utilizzare il dato (l'algoritmo) come input di sé

stesso è stato usato da Gödel nella sua dimostrazione famosa. Ha codificato teoremi come numeri, permettendo l'uso dell'aritmetica stessa. #### Descrizione UTM

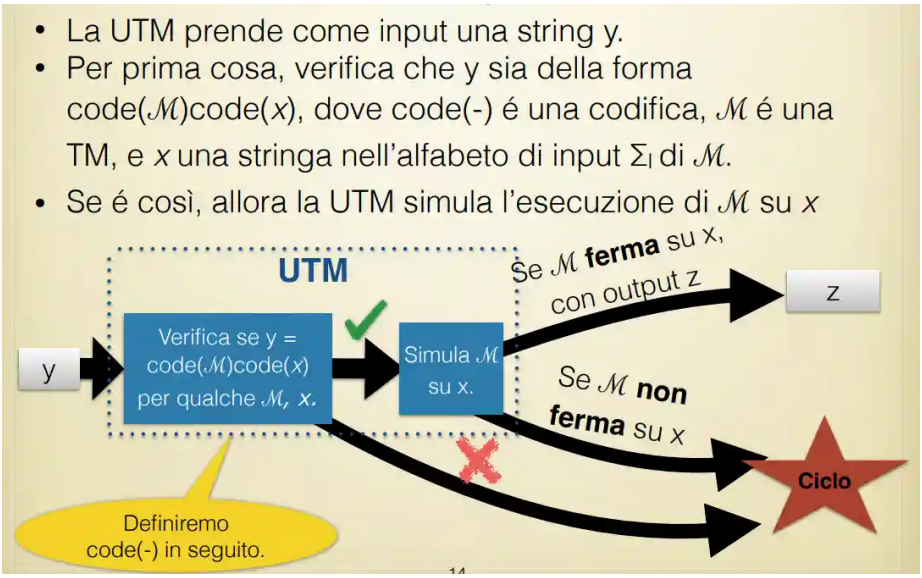

Figure 3: La macchina di Turing-20240229124153090

La cosa importante è che: > La macchina universale deve avere lo stesso comportamento di $\mathcal{M}$.

Se si ferma, si ferma con stesso output, altrimenti non si ferma.

Costruzione di UTM Codifichiamo simboli che possono apparire nella definizione di δ, per esempio $\sigma_0 = \cup, \sigma_1 = \leftarrow, \sigma_2 = \rightarrow$ e poi in modo simile per ogni simbolo dell'alfabeto finito. Poi possiamo codificare in modo unario, per esempio $code(q_i) = 111..11$ per $i+1$ volte in totale, in modo simile per σ_i. Poi uso gli 0 per separare i codici. Quindi posso avere una singola transizione, che è una tupla di simbolo in input, stato input, stato output, simbolo output, e movimento. Quindi

$$code(t) = code(q_i)0code(\sigma_n)0code(q_j)0code(\sigma(m))0code(\sigma_o)0$$

Poi possiamo separarli con uno 0 in più, o contare quanti 0 hai visto. Si può encodare anche l'input, in modo simile

$$code(\sigma_{1i}...\sigma_{in}) = 00code(\sigma_1)0code(\sigma_{12})0...0code(\sigma_{in}).$$

È interessante osservare che questo formato $code(\mathcal{M})code(i)$ è una cosa decidibile, perché è un formato che si conosce.

1. Lettura simboli macchina specifica
2. Interpretazione di essa
3. Poi si può continuare ad utilizzare il nastro per simulare la macchina stessa.

Per cominciare una simulazione della UTM: 1. Passo di preparazione: verifica che y = code( )code(x) per qualche TM e input x. Se no, cicla. Se si, allora nastro 1 contiene code( )code(x). Vai al passo 2. 2. Sposta code( ) dal nastro 1 al nastro 2. Ora nastro 1 mostra il contenuto del nastro di su input x alla configurazione iniziale, in forma codificata. 3. Scrivi code(q0) su nastro 3. 4. Posiziona testina 1 sul primo simbolo di code(x), testina 2 sul primo simbolo di code( ), and testina tre sul primo simbolo di code(q0).

Quindi poi: Cerco la funzione transizione corretta sul nastro 2 che contiene la codifica, poi aggiorno i nastri 1 e 3 a seconda di cosa scrivo, della testina e dello stato corrente in nastro 3.

References

[1] Denning "Ubiquity Symposium 'What Is Computation?': Opening Statement" Ubiquity Vol. 2010, pp. 1880066.1880067 2010

[2] Turing "On Computable Numbers, with an Application to the Entscheidungsproblem" Proceedings of the London Mathematical Society Vol. s2-42(1), pp. 230–265 1937

[3] Dehaene "Consciousness and the Brain" Singapore Books 2014