Informatica Teorica
Appunti

Giovanni "Qua' Qua' dancer" Palma
e
Alex "Morbidelli WhatsApp" Basta

Contents

Chapter 1

Introduzione a problemi e indecidibilita

Studieremo due arie:

- Calcolabilita: ci chiediamo se per quel problema esistera mai un algoritmo in grado di risolverlo (decidibilita' del problema). Non li incontriamo spesso irl dato che siamo piu' spinti a fare robe che sappiamo gestire. Bisogna studiare la struttura del problema per decidere se esiste o meno un algoritmo che lo risolve.
- Complessita: vogliamo determinare se un problema decidibile e' "facile" o "difficile", ovvero qual'e' la sua complessita' (diverso dalla complessita' degli algoritmi)

Quando diciamo "facile" intendiamo risolvibile in tempo polinomiale.

Definition 1.0.1: Problema

Relazione fra stringhe

Example 1.0.1 (Somma)

Dati x e y calcola $x + y \to z$. E' una relazione binaria fra stringhe:

- $< (2, 3), 5 >$
- $< (4, 3), 7 >$ $\vdots$

Dove la prima stringa e' l'input e la seconda e' l'output. Dato che elencare tutte le tuple e' impossibile, le descriviamo a parole ma stando attenti ad essere precisi. Ci servono tre elementi:

- What is input?
- What is output?
- Che relazione c'e' fra out e in?

1.1 Halting Problem

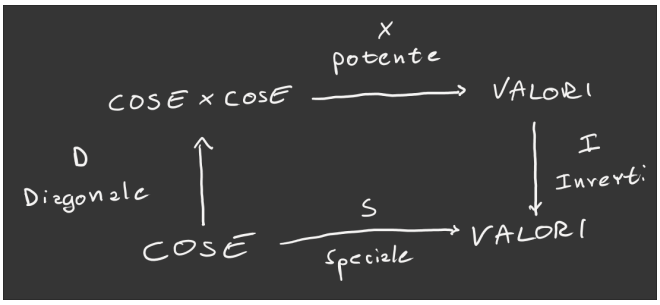
Definiamo prima per bene il problema:

- Input: Una stringa P che rapresenta il codice sorgente di un programma
- Output: Un booleano
- Relazione: Se P termina ritorna T, altrimenti F

Sembra un problema semplice, ma vedremo che in realta' e' indecidibile. Per confermare cio' dobbiamo formulare una dimostrazione formale, usando quel che viene chiamato "metodo diagonale".
Il prof non va cosi' in dettaglio per il metodo usato nella dimostrazione (dato che e' solo un esempio introduttivo), ma secondo me e' interessante e fornisce un bell'esempio per l'utilita' della Teoria delle Categorie.

1.1.1 Il Metodo Diagonale

Il *metodo diagonale* e' uno schema utile per la dimostrazione di certi teoremi. E' stato ideato per la prima volta da Cantor per dimostrare, per esempio, che $\mathbb{R}$ non e' numerabile, dove viene usata la *diagonale* di una tabella per dimostrare un assurdo. Questo metodo e' stato successivamente usato per dimostrare altre proposizioni molto importanti, come il teorema di Cantor (quello sugli insiemi potenza), il paradosso di Russel e i teoremi dell'incompletezza di Godel.
La struttura della dimostrazione generale e' la seguente:



Lo schema e' molto astratto, e' formato solo da *punti* (i vertici) e *frecce* che possono essere composte per formare altre frecce. I primi possono essere insiemi e le frecce funzioni, come vedremo. Le quattro frecce hanno delle funzioni particolari:

- La funzione "potente" X: nella Teoria delle Categorie viene definita *morfismo punto-suriettivo*(?). E' la freccia che assumiamo esista per poi dimostrare l'assurdo.
- La funzione diagonale D: dato un elemento x ritorna la coppia (x, x). Raccoglie l'essenza del metodo diagonale in quanto e' la componente che causa l'*auto-referenzialita'* che sta al cuore della dimostrazione.
- La funzione che inverte I: dato l'output di X lo *inverte* in modo da causare l'assurdita'.
- La funzione "speciale" S: composizione delle tre frecce sopra, insieme all'assunzione iniziale permette di dimostrare l'assurdo.

Note:

Lowkey non l'ho spiegato benissimo dato che non ho studiato abbastanza le sue applicazioni e so solo le basi della Teoria delle Categorie quindi non e' rigoroso per niente. Forse nel futuro ci ritornero', boh.

Finita la costruzione, i prossimi passi sono:

1. Codifica S con una *COSA*, k.
2. Cosa succede se passo k a S?

Note:

In verita' non sono sicurissimo se questi passi fanno parte del metodo generale o solo dell'halting

1.1.2 Dimostrazione Halting Problem

Andiamo ora a vedere come puo' essere dimostrato che l'halting problem e' indecidibile.
Per prima cosa dobbiamo definire la controparte concreta delle frecce e punti dello schema:

- I punti *COSE* sono insiemi di tutti i programmi o dati possibili. Sfruttando la **Numerazione di Godel** possiamo attribuire a ognuno di questi un numero naturale. Quindi *COSE* diventa $\mathbb{N}$.
- I punti *VALORI* sono tutti i possibili output di un programma, quindi dei dati. Quindi *VALORI* diventa $\mathbb{N}$.

- La freccia X diventa il programma *HALT*, ovvero la funzione totale che prende un *Programma* P e un *Dato* D (entrambe $\in \mathbb{N}$) tale che:

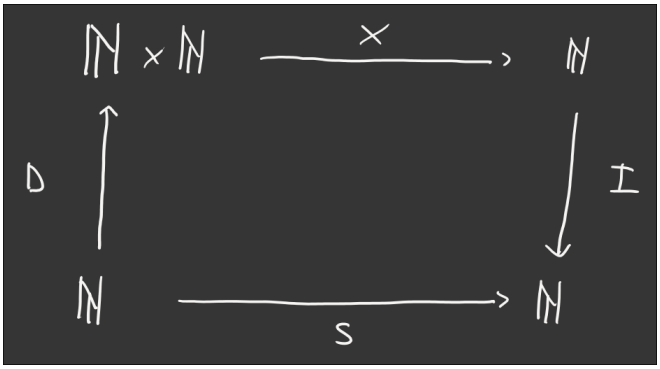
$$X(P, D) = \begin{cases} \text{"termina"} & P \text{ termina su } D \\ \text{"diverge"} & P \text{ non termina su } D \end{cases}$$

  Per ora assumiamo l'esistenza di questo programma.

- La freccia D e' il programma che implementa la funzione totale $\mathbb{N} \to \mathbb{N} \times \mathbb{N}$

$$D(x) = (x, x)$$

  che e' facile da costruire e quindi esiste.



Non e' chiaro come dobbiamo definire I in modo da dimostrare l'assurdo, quindi lavoriamo top-down.
Dato che una composizione di programmi e' un programma, anche S e' sicuramente un programma, quindi una funzione parziale che puo' terminare o divergere dato un input.
Essendo S un programma, e' possibile rappresentarlo con un numero $k \in \mathbb{N}$. Analizziamo $S(k)$:

- $S(k)$ termina: in questo caso $X(k, k) =$ "termina", quindi $S(k) = I($"termina"$)$. Per rimanere coerente I deve terminare sull'input "termina".
- $S(k)$ non termina: in questo caso $X(k, k) =$ "diverge", quindi $S(k) = I($"$diverge$"$)$. Per rimanere coerente I deve divergere sull'input "diverge".

Ma noi non vogliamo la coerenza, vogliamo costruire S in modo da dimostrare l'assurdo. Quindi definiamo I come la funzione parziale che:

- Diverge sull'input "termina"
- Termina sull'input "diverge"

Abbiamo quindi che:

- Se $X(k, k) =$ "termina", allora $S(k)$ diverge
- Se $X(k, k) =$ "$diverge$", allora $S(k)$ termina

Queste due implicazioni sono in opposizione alla definizione del programma *HALT*, che quindi non puo' esistere.

1.2 Tipi di problemi

Ci sono due categorie:

- **Ricerca**: vogliamo calcolare un risultato arbitrario, ovvero *ricercano* un output vario.
- **Decisione**: l'output e' *booleano*.

Queste due tipologie non sono completamente sganciate, esiste una relazione.
Dato un problema di ricerca, possiamo ricavare la versione "decisionale" che e' piu' facile da calcolare.

Example 1.2.1 (Somma)

Ricerca: dati a, b calcolare la somma $c = a + b$.
Decisione: dati a, b, c determinare se $c = a + b$

Per questo motivo parleremo quasi totalmente solo di problemi di decisione.

Note:

E' da qui che deriva il termine "decidibile".

1.2.1 Problemi di decisione

Theorem 1.2.1

Tutti i problemi di decisione possono essere ridotti al riconoscimento di uno specifico linguaggio.

Example 1.2.2

PATH $< G, s, t >$: vogliamo stabilire se esiste un percorso da s a t.
Mostriamo di poter trasformare questo problema decisionale in un problema che chiede se una "parola" appartiene a uno specifico linguaggio.

$$L_{\text{PATH}} = \{< G, s, t > | G\text{e' un Graf} \wedge s, t\text{sono nodi del grafo} G \wedge \exists \text{ un percorso da } s \rightarrow t\}$$

Ci riduciamo quindi a decidere se una data tripla appartiene a L_{PATH} o meno.

Quindi, mentre studiamo decidibilita' dei problemi guarderemo linguaggi e i loro *automi*.

1.3 Linguaggi

1.3.1 Definizioni

Definition 1.3.1: Alfabeto

$\Sigma = \{a, b, c, ...\}$ e' l'*alfabeto*
E' un insieme di simboli.

Definition 1.3.2: Parola

w e' una parola o striga su Σ e' un concatenamento di 0 o piu' simboli di Σ.

Definition 1.3.3: Kleene

def

Definition 1.3.4: Linguaggio

Sottoinsieme delle parole che si possono costruire su un determinato alfabeto Σ:

$$L \subseteq \Sigma^*$$

Definition 1.3.5

Dato un linguaggio L **fissato** per una stringa w, decidere se:

$$w \in L$$

Il problema di appartenenza a un linguaggio L (**fissato!**) e' definito come:
dato il linguaggio L, prendere in input una parola w e decidere se $w \in L$.
input: stringa output: booleano
Perche' vogliamo usare automi? : e' un formalismo piu' basico e piu' semplice da studiare rispetto a un arbitrario linguaggio di prorammazione.

1.3.2 Automi

Vabbe' introduce gli automi, il Guerriero ci ha gia preparato adeguatamente.
Alright whatsapp.

1.3.3 Macchia di Turing

Automa piu' potente che Turing si invento per studiare i problemi importanti da chiudere, fra cui l'automatizzazione dei teoremi.
E' nato per formalizzare il calcolo automatico in un tempo in cui non esistevano i computer, era la versione "automatizzata" di lui che scriveva su carta.
Iniziamo con una spiegazione informale:
E' una macchina caratterizzata da un nastro con delle cellette
In ognuna cella c'e' un simbolo, ed e' infinito in entrambe le direzioni
Ha una testina posizionata su una cella da cui puo' leggere il simbolo
La testina puo' spostarsi a dx o sx, e a differenza degli automi visti puo' benissio tornare indietro e cambiare la decisione presa prima (backtracking)
La testina puo' anche scrivere su una cella, perdendo cio' che c'era prima (una seconda differenza)
Per noi sta roba e' un algoritmo, dato che la macchina di Turing, in quanto automa, ha vari stati di funzionamento

- Legge simbolo cella
- In base allo stato, sovrascrive qualcosa
- Sposto la testina avanti o indietro
- Cambio lo stato

Example 1.3.1

Progetta una MdT che riconosca il linguaggio

$$L = \{a^m b^m | m \geqslant 0\}$$

La stringa da decidere si trova sul nastro da sx a dx e la testina si trova sopra il primo simbolo.
Un algoritmo e' un filmato di quello che succede nella nostra testa per risolvere un problema.

- Leggi cella, se e' vuota fine (ok)
- altrimenti se e' b fine (no)
- se e' a, cancella
- spostati a dx finche' la cella e' bianca
- vai a sx di uno e leggi
- se e' a, fine (no)
- se e' b, cancella e vai a sx finche e' bianca
- muoviti di uno a dx

TODO: disegna automa

Definizioni formali

Definition 1.3.6: Macchina di Turing

E' la settupla

$$M = \{Q, q_0, F, \Sigma, \Gamma, b/, \delta\}$$

Dove:

- Q e' l'insieme di stati
- q_0 e' lo stato iniziale
- F e' l'insieme degli stati finali
- Σ e' l'insime dei simboli in input
- Γ e' l'insieme dei simboli che la macchina puo' scrivere $\Sigma \subset \Gamma$
- $b/$ e' il simbolo di cella vuota ("blank") $b/ \in \Gamma$
- δ e' la funzione

$$Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$$

Chapter 2

Macchine di Turing e Linguaggi

Chapter 3

Riduzioni

Chapter 4

Teorema di Rice

Chapter 5

Complessita Strutturale

Chapter 6

Complessita Spaziale

Chapter 7

Oracoli e Classi Funzionali