Azzolini Riccardo 2020-12-21

Il linguaggio universale

1 Problema di appartenenza al linguaggio di una MdT

Sulle macchine di Turing può essere definito il seguente problema di decisione LM:

> *Parametri*: Una macchina di Turing $M = \langle Q, \{0, 1\}, \Gamma, \delta, q_1, B, F\rangle$ e un suo input $w \in \{0, 1\}^*$ (per semplicità, si considerano solo le MdT con alfabeto di input binario, che sono equivalenti alle MdT con arbitrari alfabeti di input).
>
> *Domanda*: M accetta w?

Per studiare lo status computazionale di questo problema bisogna, come al solito, associare a esso un linguaggio e determinare se esista una MdT che lo riconosca. Si osservi che ciò introduce una sorta di "cortocircuito": si vogliono usare le macchine di Turing per risolvere un problema che riguarda le macchine di Turing stesse.

Nel seguito, si dimostrerà che il problema LM è ricorsivamente enumerabile ma indecidibile. Per fare ciò, bisognerà prima definire con precisione il linguaggio associato,

$$L(\text{LM}) = \{x \in \Sigma^* \mid x \text{ è la codifica di una coppia } (M, w) \text{ per cui } M \text{ accetta } w\}$$

che necessita di due codifiche:

- una codifica su $\Sigma = \{0, 1\}$ delle MdT, $\#_{\text{MdT}}(M)$;
- una codifica delle coppie di stringhe $(\#_{\text{MdT}}(M), w)$.

2 Codifica delle macchine di Turing

Per semplicità, nel definire una codifica per le MdT $M = \langle Q, \{0, 1\}, \Gamma, \delta, q_1, B, F\rangle$ si considerano solo le M fatte in questo modo:

- l'insieme degli stati è $Q = \{q_1, q_2, \dots, q_r\}$ (per un qualche r);
- lo stato iniziale è q_1;
- c'è un unico stato finale, $F = \{q_2\}$;
- l'insieme dei simboli di nastro è $\Gamma = \{X_1, X_2, X_3, \dots, X_s\}$ (per un qualche s), dove $X_1 = 0$, $X_2 = 1$ e $X_3 = B$.

Si potrebbe dimostrare che tali restrizioni, compreso in particolare il fatto di avere un singolo stato finale, non sono limitative, cioè che queste MdT sono equivalenti a quelle generali.

Una volta fissata la forma della macchina M, si assegnano degli interi positivi alle sue varie componenti:

- ogni stato q_i è indicato dall'intero i;
- ciascun simbolo di nastro X_k è indicato dall'intero k;
- 1 indica un movimento della testina a sinistra, $D_i = \texttt{L}$, e 2 indica un movimento a destra, $D_2 = \texttt{R}$.

Così, una regola di transizione

$$\delta(q_i, X_j) = (q_h, X_k, D_m)$$

può essere codificata su $\{0, 1\}$ rappresentando in unario gli indici i, j, h, k, m, come sequenze di 0 separate da simboli 1:

$$\underbrace{0^i}_{q_i} 1 \underbrace{0^j}_{X_j} 1 \underbrace{0^h}_{q_h} 1 \underbrace{0^k}_{X_k} 1 \underbrace{0^m}_{D_m}$$

(si osservi che, siccome ciascuno degli indici ha sempre valore ≥ 1, in una stringa così costruita non compariranno mai due 1 consecutivi).

Infine, mettendo insieme le codifiche $C_1, C_2 \dots, C_n$ di tutte le regole di transizione di M, separate da 11 (un separatore distinto dal separatore 1 usato all'interno delle singole codifiche), si ottiene la codifica dell'intera macchina M:

$$\#_{\text{MdT}}(M) = C_1\, 11\, C_2\, 11\, \dots\, 11\, C_n$$

Complessivamente, questa procedura definisce una funzione di codifica $\#_{\text{MdT}} : \text{MdT} \to \{0, 1\}^*$, che permette di rappresentare qualunque MdT (con le restrizioni elencate prima) come una stringa sull'alfabeto $\{0, 1\}$.

Osservazione: La codifica appena descritta rappresenta concretamente solo la tabella di transizione di M, ma da essa si possono ricavare anche tutte le altre informazioni significative:

- l'insieme degli stati è dato da tutti gli stati che compaiono nelle transizioni (ed eventuali stati che sono presenti nella macchina M originale, ma non compaiono nelle transizioni, non sono significativi per le computazioni);
- allo stesso modo, i simboli di nastro significativi sono quelli che compaiono nelle transizioni;
- l'alfabeto di input $\{0, 1\}$, lo stato iniziale q_1, l'insieme di stati finali $F = \{q_2\}$ e il simbolo di blank $B = X_3$ sono fissati dalla forma delle MdT considerate in questa codifica.

2.1 Esempio

Si consideri la MdT

$$M = \langle \{q_1, q_2, q_3\}, \{0, 1\}, \{\overbrace{0}^{X_1}, \overbrace{1}^{X_2}, \overbrace{B}^{X_3}\}, \delta, q_1, B, \{q_2\}\rangle$$

con funzione di transizione

$$\delta(q_1, 1) = (q_3, 0, \mathtt{R}) \quad \delta(q_3, 0) = (q_1, 1, \mathtt{R}) \quad \delta(q_3, 1) = (q_2, 0, \mathtt{R}) \quad \delta(q_3, B) = (q_3, 1, \mathtt{L})$$

(e indefinita su tutte le altre coppie stato-simbolo). Le regole di transizione di M vengono codificate come

$$\begin{aligned}
\delta(q_1, 1) = (q_3, 0, \mathtt{R}) \quad &\longrightarrow \quad C_1 = \underbrace{0}_{q_1} 1 \underbrace{00}_{X_2=1} 1 \underbrace{000}_{q_3} 1 \underbrace{0}_{X_1=0} 1 \underbrace{00}_{D_2=\mathtt{R}} \\
\delta(q_3, 0) = (q_1, 1, \mathtt{R}) \quad &\longrightarrow \quad C_2 = \underbrace{000}_{q_3} 1 \underbrace{0}_{X_1=0} 1 \underbrace{0}_{q_1} 1 \underbrace{00}_{X_2=1} 1 \underbrace{00}_{D_2=\mathtt{R}} \\
\delta(q_3, 1) = (q_2, 0, \mathtt{R}) \quad &\longrightarrow \quad C_3 = \underbrace{000}_{q_3} 1 \underbrace{00}_{X_2=1} 1 \underbrace{00}_{q_2} 1 \underbrace{0}_{X_1=0} 1 \underbrace{00}_{D_2=\mathtt{R}} \\
\delta(q_3, B) = (q_3, 1, \mathtt{L}) \quad &\longrightarrow \quad C_4 = \underbrace{000}_{q_3} 1 \underbrace{000}_{X_3=B} 1 \underbrace{000}_{q_3} 1 \underbrace{00}_{X_2=1} 1 \underbrace{0}_{D_1=\mathtt{L}}
\end{aligned}$$

quindi la codifica della macchina è:

$$\#_{\text{MdT}}(M) = \underbrace{0100100010100}_{C_1} 11 \underbrace{0001010100100}_{C_2} 11 \underbrace{00010010010100}_{C_3} 11 \underbrace{0001000100010010}_{C_4}$$

2.2 Decodifica

Perché $\#_{\text{MdT}} : \text{MdT} \to \{0, 1\}^*$ soddisfi le proprietà richieste per una funzione di codifica, è necessario che essa sia invertibile, ovvero che esista una corrispondente funzione di decodifica $\#^{-1}_{\text{MdT}} : \{0, 1\}^* \to \text{MdT}$ tale che $\#^{-1}_{\text{MdT}}(\#_{\text{MdT}}(M)) = M$ per ogni M.

Intuitivamente, data una codifica della forma $C_1\, 11\, C_2\, 11\, \ldots\, 11\, C_n$, dove ogni C_i è una codifica corretta di una regola di transizione, si può ricostruire la MdT corrispondente. Tuttavia, una funzione di decodifica "ben fatta" dovrebbe agire su *ogni* stringa in $\{0, 1\}^*$. Allora, bisogna specificare come essa si comporti sulle stringhe che non sono codifiche valide (ad esempio, la stringa 11 non è valida perché il separatore 11 può comparire solo tra le codifiche di due transizioni, che qui non sono presenti). Per convenzione, si sceglie di associare a ogni stringa che non ha la struttura corretta l'MdT "banale"

$$M_0 = \langle \{q_1, q_2\}, \{0, 1\}, \{0, 1, B\}, \varnothing, q_1, B, \{q_2\}\rangle$$

che ha:

- solo due stati e tre simboli di alfabeto (il minimo possibile per una MdT della forma considerata dalla codifica $\#_{\text{MdT}}$);
- una funzione di transizione "vuota", indefinita per tutte le coppie, $\delta = \varnothing$.

3 Codifica delle coppie (M, w)

Per definire il linguaggio associato al problema LM bisogna codificare le coppie (M, w) di una macchina di Turing M (con alfabeto di input $\{0, 1\}$) e un input $w \in \{0, 1\}^*$ su cui eseguirla.

w è di fatto già codificata, in quanto stringa su $\{0, 1\}$, mentre per M si sfrutta la codifica $\#_{\text{MdT}}$ appena definita, e come separatore si usa 111, che per definizione non compare mai in $\#_{\text{MdT}}(M)$. Complessivamente, la coppia (M, w) viene dunque codificata come $\#_{\text{MdT}}(M)\,111\,w$.

3.1 Decodifica

Una stringa $\alpha \in \{0, 1\}^*$ può essere decodificata tramite una funzione $\texttt{estraiCoppia}(\alpha)$, che restituisce:

- la coppia (u, w), con $u, w \in \{0, 1\}^*$, se α è una codifica ben fatta, ovvero è appunto la codifica di una coppia (u, w), in cui $u = \#_{\text{MdT}}(M)$ è a sua volta la codifica di una MdT;
- `null` se α non è una codifica ben fatta.

Intuitivamente, si può scrivere un programma che calcoli la funzione `estraiCoppia` e termini per ogni input.

4 Simulazione delle MdT

Adesso che si hanno tutti gli elementi per definire il linguaggio associato a LM, rimane il problema di come determinare l'appartenenza di una stringa a tale linguaggio, ovvero di costruire una macchina di Turing che, data la codifica di una coppia (M, w), *simuli* la computazione di M su input w, e osservi se essa accetta o meno. Siccome questa MdT in grado di simulare altre MdT sarebbe molto complessa, per semplificare la trattazione mostrerà invece un algoritmo intuitivo (espresso in linguaggio "pseudo-Java") che esegua la simulazione: per la tesi di Church-Turing, l'esistenza di tale algoritmo implica l'esistenza di una MdT equivalente, che faccia questa stessa simulazione, una sorta di *interprete universale per le macchine di Turing.*

Il metodo principale dell'algoritmo di simulazione è $\texttt{simula}(C, w)$, che riceve come argomenti la codifica C di una MdT e una stringa w, e simula il comportamento della macchina $M = \#_{\text{MdT}}^{-1}(C)$ sull'input w:

$$\texttt{simula}(C, w) = \begin{cases} \texttt{ACCETTA} & \text{se } M \text{ su input } w \text{ si arresta in uno stato finale} \\ \texttt{RIFIUTA} & \text{se } M \text{ su input } w \text{ si arresta in uno stato non finale} \\ \text{non termina} & \text{se } M \text{ su input } w \text{ non si arresta} \end{cases}$$

4.1 Strutture dati

Per poter implementare il metodo `simula`, bisogna determinare come rappresentare in Java i vari elementi della MdT. Ad esempio:

- gli stati possono essere rappresentati tramite un'opportuna classe `Stato`;
- i simboli di Γ possono essere rappresentati semplicemente come caratteri (`char` / `Character`);
- le mosse della testina possono essere rappresentate tramite un tipo enumerativo `Mossa` con due valori, `L` e `R`;
- la funzione di transizione può essere rappresentata da una tabella che associa a ogni coppia `(stato, simbolo)` (formata da uno `Stato` e un `char`) una tripla `(stato, simbolo, mossa_testina)` (formata da uno `Stato`, un `char` e una `Mossa`, e rappresentata come oggetto di un'opportuna classe `Tripla`);
- la macchina può essere completamente modellata da una classe `MdT`, che memorizza le informazioni che la caratterizzano.

Per convenienza, la classe `MdT` può fornire un metodo

```
Tripla delta(Stato stato, char simbolo)
```

che implementa la funzione di transizione δ, consultando la tabella descritta prima. Esso restituisce la tripla `(stato, simbolo, mossa_testina)` corrispondente alla coppia `(stato, simbolo)` passata come argomento, se δ è definita su tale coppia, o `null` se invece δ non è definita.

Infine, è necessario un qualche metodo (ad esempio un metodo statico o costruttore di `MdT`) che implementi la funzione di decodifica $\#^{-1}_{\mathrm{MdT}}(C)$, estraendo dalla codifica $C = \#_{\mathrm{MdT}}(M)$ le componenti della macchina M e costruendo la corrispondente istanza della classe `MdT`.

4.2 Funzionamento di `simula`

La prima cosa che $\texttt{simula}(C, w)$ deve fare è costruire l'oggetto $\texttt{M} = \#^{-1}_{\mathrm{MdT}}(C)$, l'istanza della classe `MdT` che rappresenta la macchina codificata da C.

Poi, il metodo ha delle variabili che rappresentano la descrizione istantanea della MdT:

- `Stato stato`, lo stato corrente della macchina;
- `List<Character> nastro`, il contenuto significativo del nastro;
- `int posizione`, la posizione della testina sul nastro (relativamente all'inizio del contenuto significativo).

Queste devono essere inizializzate alla configurazione iniziale della macchina:

- lo stato deve essere quello iniziale, `stato` = q_1;
- la lista `nastro` deve memorizzare i caratteri della stringa di input w, a partire dalla posizione 0;
- la testina deve essere sul primo carattere dell'input, `posizione = 0`.

A questo punto, si può procedere con la simulazione della computazione della MdT, che avviene per mezzo del seguente algoritmo:

```
while (M.delta(stato, nastro.get(posizione)) != null) {
  sia (p, Y, D) = M.delta(stato, nastro.get(posizione));
  stato = p;
  nastro.set(posizione, Y);
  if (D == L) { ... } else { ... } // aggiorna posizione (e nastro)
}
if (stato == q2) return ACCETTA else return RIFIUTA;
```

In sostanza, finché la macchina non è bloccata:

1. si seleziona la mossa da eseguire;
2. si aggiorna la configurazione della macchina in modo opportuno (esattamente come nella definizione formale di passo di computazione di una MdT).

Se poi la computazione si blocca, il ciclo si arresta, e `simula` restituisce il valore:

- `ACCETTA` se lo stato in cui si è fermata la macchina è q_2, l'unico stato finale;
- `RIFIUTA` se invece la macchina si è fermata in uno stato non finale.

Non è però garantito che il ciclo, e quindi il metodo `simula`, si arresti, perché la computazione della MdT M su input w che viene simulata potrebbe non terminare mai.

5 Linguaggio e MdT universali

Il linguaggio associato al problema di decisione LM,

$$\begin{aligned} L(\mathrm{LM}) &= \{x \in \Sigma^* \mid x \text{ è la codifica di una coppia } (M, w) \text{ per cui } M \text{ accetta } w\} \\ &= \{\#_{\mathrm{MdT}}(M)\, 111\, w \mid M \text{ accetta } w\} \end{aligned}$$

è chiamato **linguaggio universale**, ed è indicato con L_u.

Per mostrare che L_u è ricorsivamente enumerabile, si presenta in seguito un programma intuitivo P_u che, data in input una qualunque stringa $\alpha \in \{0,1\}^*$, restituisce `ACCETTA` se e solo se $\alpha \in L_u$, cioè α è la codifica ben fatta di una coppia (M, w) tale che M accetta w:

```
C = estraiCoppia(α);
if (C == null) {
  return RIFIUTA;
} else {
  sia (u, w) = C;
  return simula(u, w);
}
```

Per la tesi di Church-Turing, l'esistenza di P_u implica l'esistenza di una MdT U che riconosce il linguaggio universale, cioè tale che $L(U) = L_u$, quindi L_u è appunto ricorsivamente enumerabile. La macchina U è detta **macchina di Turing universale**, in quanto permette di simulare il comportamento di ogni altra macchina di Turing.

P_u non dimostra invece che L_u sia decidibile (e si mostrerà più avanti che L_u è indecidibile), perché la chiamata `simula(u, w)`, e dunque complessivamente l'algoritmo, può non terminare.

6 Macchina universale per il calcolo di funzioni

La MdT universale U appena definita simula il comportamento delle MdT viste come riconoscitori di linguaggi, ma è possibile definire una macchina universale U_f anche quando le MdT vengono usate come modello per calcolare funzioni. U_f prende in input la codifica di una MdT M e di un input w, simula l'esecuzione di M su w, e, se la simulazione termina, lascia sul nastro il risultato della funzione calcolata da M su input w.

Se si interpreta una MdT M come un programma, allora U_f è un programma che può eseguire qualunque programma. Questo è il concetto alla base dei calcolatori: un processore è di fatto un'implementazione hardware di un programma equivalente a una macchina di Turing universale.