Indice

Fin'ora abbiamo visto automi in cui la testina si muove su un nastro read only in un unico verso, e questi si chiamano automi one-way.

Possiamo pensare alcune modifiche a questo modello, prima di tutto non ha senso che la macchina scriva sul nastro, visto che non può tornare indietro. Questo diventa interessante nel caso invece di un modello two way, quindi un modello in cui si può muovere la testina in due direzioni.

Permettendo un nastro read-write con testina two way si ottiene un modello molto più potente, nel caso di nastro potenzialmente infinito otteniamo una macchina di Turing.

Le macchine di Turing hanno diverse definizioni equivalenti, ad esempio, equivalentemente si può avere un nastro read only e one way, ed un certo numero di nastri di lavoro read-write e two way.

Se invece invece che avere un nastro potenzialmente infinito teniamo un nastro limitato dalla dimensione dell'input otteniamo quelli che si chiamano automi limitati linearmente. Questi riconoscono linguaggio di tipo 1. Anche nel modello a più nastri limitare la dimensione dei nastri di input ad una dimensione proporzionale all'input lo regtringe ai linguaggi di tipo 1.

Alternativamente possiamo pensare di avere un nastro di lavoro read only, in cui si ha una memoria strutturata a pila. Questo è detto automa a pila e riconosce i linguaggi di tipo 2.

Automi con nastri finiti, two way e read only e memoria finita riconoscono invece ancora i linguaggi di tipo 3. Questi automi sono detti automi two way.

Nella nostra analisi degli automi a stati finiti supporremo che l'input sia racchiuso da due delimitatori: $\triangleright$ e $\triangleleft$. E supporremo che l'automa per accettare deva spostare la testina oltre l'end marker ed andare in uno stato speciale chiamato q_F.

Definiamo il linguaggio

$$L_n = (a+b)^* a (a+b)^{n-1}$$

il linguaggio in cui la lettera n-esima da destra è una a.
Avevamo visto che un DFA ha almeno 2^n stati e un NFA $n+1$.
In automa two way si riesce a riconoscere questo linguaggio con un numero proporzionale ad n, rimanendo deterministici.
Supponiamo di voler verificare che $w = \triangleright abbaabb \triangleleft$. Definiremo δ come

$$\begin{aligned}
\delta(q_0, a) &= (q_0, +1) \\
\delta(q_0, b) &= (q_0, +1) \\
\delta(q_0, \triangleleft) &= (q_1, -1) \\
\delta(q_1, a) &= (q_2, -1) \\
\delta(q_1, b) &= (q_2, -1) \\
\delta(q_2, a) &= (q_3, -1) \\
\delta(q_2, b) &= (q_3, -1) \\
\delta(q_3, a) &= (q_F, +1) \\
\delta(q_F, a) &= (q_F, +1) \\
\delta(q_F, b) &= (q_F, +1) \\
\delta(q_F, \triangleleft) &= (q_F, +1)
\end{aligned}$$

Dove $+1$ indica che ci sposiamo a destra e -1 che ci spostiamo a sinistra.
È facile vedere che questo automa è deterministico ed ha $n+2$ stati.

Definiamo il linguaggio

$$K_n = (a+b)^* a (a+b)^{n-1} a (a+b)^*$$

dove ci sono due a a distanza n.
Con un automa two way è più semplice riconoscere questo linguaggio. Ad esempio data la strina $\triangleright baabbbbabbaaabb \triangleleft$ vogliamo vedere se questo appartiene a K_3. Generalmente basta che per ogni carattere, se questo è una a vado avanti di n simboli e controllo se anch'esso è una a, e se è vero vado oltre la testina, alternativamente controllo il prossimo elemento.
Generalmente servirà uno stato per saltare le b finché si trova una a, n stati nel caso si incontri una a, q_F e $n-1$ stati per tornare indietro. Quindi sostanzialmente gli stati sono all'incirca $2n+1$.
Alternativamente si può pensare di viaggiare più volte da sinistra a destra facendo salti da n elementi e ricordando solo l'ultimo simboli visto. Quindi ad esempio per K_3 controllo in una prima passata $0, 3, 6, \ldots$, e alla seconda passata $1, 4, 7, \ldots$, eccetera. Questo viene chiamato automa sweep ed utilizza $O(n^2)$ stati (ma si può ridurre ad $O(n)$).

Definiamo ora gli automi two way non deterministici 2NFA come una tupla

$$M = \langle Q, \Sigma, \delta, q_0, q_F \rangle$$

Con $\triangleright, \triangleleft \notin \Sigma$, e

$$\delta : Q \times (\Sigma \cup \{\triangleleft, \triangleright\}) \to 2^{Q \times \{-1,+1\}}$$

Le la macchina non può superare gli end marker salvo quando è nello stato q_F.

La configurazione inizale abbiamo q_0 e siamo posizionati sul primo carattere, nella configurazione accettante invece ci troviamo in q_F oltre $\triangleleft$.

Il modello è deterministico se la funzione δ ha al massimo una possibilità.

In questo modello la computazione può entrare in un loop infinito, in quel caso la parola non è accettata. Questo è il caso in cui si raggiunge per una seconda volta uno stesso stato nella stessa posizione.

Nel caso non deterministico può accadere che ci si ritrovi in uno stesso stato nella stessa posizione, ma questo vuol dire che la computazione può essere accorciata.

A volte si permette di restare nella stessa posizione utilizzando una funzione δ

$$\delta : Q \times (\Sigma \cup \{\lhd, \rhd\}) \to 2^{Q \times \{-1,0,+1\}}$$

ma questo può essere simulato con due mosse.

Un'altra variazione è quella senza end marker in cui il nastro contiene solo l'input. In questo caso però l'automa cambia, ed esempio, K_n possiamo ancora riconoscerlo con il primo metodo, ma non con il secondo.

Teorema 1. *Gli automi two way riconoscono i linguaggio regolari, o alternativamente può essere trasformato ad un automa a stati finiti*

Dimostrazione. Vediamo nel caso determinstico, ma nel caso non deterministico si fa nello stesso modo.

Il concetto generale è che se l'automa two way fa una serie di giri per passare dallo stato q allo stato p, nell'automa one way abbiamo un modo per passare direttamente da q a p. Ovviamente questo non possiamo farlo ricordandoci l'intero input, perché la memoria è finita. Introduciamo le *tabelle di transizione*: da una stringa $w \in \Sigma^*$, a questa è associata una tabella

$$\tau_w : Q \times Q \to \{0, 1\}$$

Questa rappresenta lo stato da qui si esce se si suppone di essere nello stato q all'ultimo simbolo di $\rhd w$. Quindi $\tau(p, q) = 1$ sse essite una sequenza di mosse che inizia sul simbolo più a destra di una porzione del nastro che contiene $\rhd w$ nello stato p e raggiunge la prima cella a destra di tale porzione nello stato q Da notare è che il numero di tabelle è finito, perché il numero di stati è finito.

Iniziamo a calcolare τ_ϵ, questa contiene un uno per tutte le regole per cui $\delta(p, \rhd) = q$.

Ora per un $\sigma \in \Sigma, w \in \Sigma?^*$ dato τ_w calcoliamo $\tau_{w\sigma}$. Ci possono essere due casi:

- se $\delta(p, \sigma) = (q, +1)$, allora $\tau(p, q) = 1$
- se $\delta(p, \sigma) = (r_1, -1)$, allora, dopo un numero arbitrario di passaggi nella stringa w, torneremo al simbolo σ in uno stato p_1, e $\tau_w(r_1, p_1) = 1$. Ora se $\delta(p_1, \sigma) = (r_2, -1)$, allora si ripete il passo di sopra.

Quindi $\tau_{w\sigma}(p,q) = 1$ sse esisono $p_0, p_1, \ldots, p_k, r_1, r_2, \ldots, r_k \in Q$ con $k \geq 0$, tali per cui

$$\begin{aligned} p_0 &= p \\ \delta(p_{i-1}, \sigma) &= (r_i, -1) \\ \tau_w(r_i, p_i) &= 1 \\ \delta(p_k, \sigma) &= (q, +1) \end{aligned}$$

Se ci mettiamo $k = 0$, abbiamo il caso di sopra.

La proprietà importante di questa tabella è che se $\tau_w = \tau_{w'}$, alora

$$\forall \sigma \in \Sigma \mid \tau_{w\sigma} = \tau_{w'\sigma}$$

quindi se due stringhe hanno la stessa tabella, aggiungere un altro simbolo restituisce di nuovo la stessa tabella, questo perché la costruzione della tabella non dipende dalla composizione di w.

Quindi esiste una funzione che associa le tabelle a tabelle

$$F_\sigma : (Q \times Q \to \{0,1\}) \to (Q \times Q \to \{0,1\})$$

che $\forall w \in \Sigma^*$

$$\tau_{w\sigma} = F_\sigma(\tau_w)$$

calcolta utilizzando le regole di sopra.

■