Linguaggi Regolari

INDICE

Automi a stati finiti (DFA)

Gli automi a stati finiti rappresentano il modello computazionale più semplice, hanno memoria limitata e un insieme finito di stati in cui possono trovarsi. Due di questi stati sono lo stato iniziale a finale. Per rappresentare un automa a stati finiti possiamo usare dei diagrammi di stato.

I DFA sono automi as tati finiti deterministici ovvero automi che, dato uno stesso input, restituiranno sempre lo stesso risultato.

Definizione formale di DFA

Un DFA può esseree rappresentato con una quintupla di valori:

$$(Q, \Sigma, \delta, q_0, F)$$

dove:

- Q è un insieme finito di **stati**
- Σ è un insieme finito che rappresenta **l'alfabeto**
- $\delta : Q \times \Sigma \to Q$ è la **funzione di transizione**
- $q_0 \in Q$ è lo **stato iniziale**
- $F \subseteq Q$ è l'insieme degli **stati di accettazione** (chiamati anche **stati finali**)

Prendiamo A come l'insieme di tutte le stringhe che la macchina M accetta: allora diciamo che A **è il linguaggio della macchina** M e scriviamo $L(M) = A$. Allo stesso modo diciamo che M accetta (o riconosce) A.

Una macchina può riconoscere più stringhe, ma riconosce sempre un solo linguaggio. Anche se non accetta stringhe, riconosce comunque il linguaggio $\emptyset$.

Definizione formale di computazione

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA, e sia $w = w_1 w_2 ... w_n$ una stringa dove ogni w_i è un membro dell'alfabeto Σ. Allora M accetta w se una sequenza di stati $r_0, r_1, ..., r_n$ in Q esiste con tre condizioni:

- $r_0 = q_0$ → La macchina inizia nello stato iniziale
- $\delta(r_i, w_{i+1}) = r_{i+1}$, per $i = 0, ..., n-1$ → La macchina cambia stato seguendo la funzione di transizione
- $r_n \in F$ → La macchina accetta l'input se finisce dello stato di accettazione.

Diciamo che M riconosce un linguaggio A se $A = \{w | M \text{ accepts } w\}$

> Un linguaggio è chiamato **Regolare** se qualche automa finito lo riconosce

Operazioni regolari

Definiamo tre operazioni sui linguaggi, dette **operazioni regolari**:

- Unione → $A \cup B = \{x | x \in A \vee x \in B\}$ → Operazione che prende tutte le stringhe di due linguaggi e le unisce in uno solo.
- Concatenazione → $A \circ B = \{xy | x \in A \wedge y \in B\}$ → Operazione che unisce una stringa di A davanti a una stringa di B in tutti i modi possibili per ottenere le stringhe di un nuovo linguaggio.
- Star → $A^* = \{x_1 x_2 ... x_k | k \geq 0 \text{ e ogni } x_i \in A\}$ → Operazione che si applica a un solo linguaggio. Unisce insieme un numero qualunque di stringhe di A per ottenere una stringa in un nuovo linguaggio.

Teorema 1.25

> La classe dei linguaggi regolari è chiusa sotto l'operazione di unione. Quindi presi due linguaggi regolari A_1 e A_2, allora anche $A_1 \cup A_2$ sarà un linguaggio regolare.

PROOF IDEA:

Sapendo che un linguaggio regolare ha sempre un automa finito che lo riconosce, costruiremo un DFA M che riconosce il linguaggio $A_1 \cup A_2$. Quindi è una dimostrazione per costruzione.

Definiamo M come $M = (Q, \Sigma, \delta, q_0, F)$ dove:

- $Q = \{(r_1, r_2) | r_1 \in Q_1 \wedge r_2 \in Q_2\}$ → Questo è il prodotto cartesiamo degli insiemi Q_1 e Q_2 dei due linguaggi regolari iniziali.
- Σ è lo stesso alfabeto dei due automi M_1 e M_2 che riconoscono i linguaggi inziali. Se anche i due alfabeti inziali fossero diversi, il teorema vale comunque per $\Sigma = \Sigma_1 \cup \Sigma_2$.
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$ → La funzione di transizione parte da uno stato di M (rappresentato come uno stato di M_1 e uno di M_2) e, con un simbolo in input, passa al prossimo stato.
- q_0 è la coppia di stati (q_1, q_2), ovvero gli stati iniziali dei due automi iniziali.
- F è l'insieme di tutte le coppie in cui uno dei due è uno stato accettante di M_1 o M_2

Teorema 1.26

> La classe dei linguaggi regolari è chiusa sotto l'operazione di unione. Quindi presi due linguaggi regolari A_1 e A_2, allora anche $A_1 \circ A_2$ sarà un linguaggio regolare.

PROOF IDEA:

Questso teorema può essere dimostrato seguendo lo stesso procedimento del teorema precedente

---

Nondeterminismo

Il nondeterminismo è un concetto molto utile per la descrizione degli automi. Se il con **determinismo** un automa può passare da uno stato all'altro dato un solo input, con il **nondeterminismo** ci potrebbero essere più scelte possibili per passare da uno stato all'altro.

Negli NFA può essere presente una funzione di transizione ϵ, ovvero una transizione che fa cambiare lo stato dell'automa ma senza consumare alcun input. Con le transizioni ϵ l'automa si divide in più possibili ramificazioni: se almeno una di queste termina in uno stato di accettazione, allora la macchina accetta l'input.

Definizione formale di NFA

Un NFA può essere rappresentato come una quintupla di valori:

$$(Q, \Sigma, \delta, q_0, F)$$

dove:

- Q è un insieme finito di **stati**
- Σ è un insieme finito che rappresenta **l'alfabeto**
- $\delta : Q \times \Sigma_\epsilon \to P(Q)$ è la **funzione di transizione**
- $q_0 \in Q$ è lo **stato iniziale**
- $F \subseteq Q$ è l'insieme degli **stati di accettazione** (chiamati anche **stati finali**)

La dicitura $P(Q)$ indica **l'insieme potenza** di Q, ovvero l'insieme di tutti i sottoinsiemi di Q, mentre l'alfabeto Σ_ϵ indica l'alfabeto $\Sigma \cup \{\epsilon\}$.

Equivalenza fra NFA e DFA (Teorema 1.39)

Diciamo che due automi sono **equivalenti** se riconoscono lo stesso linguaggio.

> Ogni NFA ha sempre un DFA equivalente

PROOF IDEA:

Se un linguaggio è riconosciuto da un NFA allora dobbiamo dimostrare che esiste un DFA che riconosce lo stesso linguaggio. Dobbiamo quindi convertire un NFA in un DFA: ad esempio se l'NFA ha un insieme con k stati, allora il DFA avrà un insieme con 2^k stati. Più nel dettaglio:

- Ogni stato di M è un insieme di stati di N (ovvero l'insieme potenza di Q).
- Se R è uno stato di M, allora è anche un insieme di stati di N. Se M legge un simbolo a nello stato R, mostra dove a porta ogni stato in R. Siccome ogni prossimo stato è un insieme di stati prendiamo l'unione di tutti gli insiemi, ovvero:

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

- Lo stato iniziale di M corrisponde all'insieme contenente solo gli stati iniziali di N.
- M accetta l'input se uno dei possibli stati in cui può trovarsi N in quel momento è accettante.
- Per considerare le transizioni ϵ introduciamo la collezione di stati che possono essere raggiunti dallo stato R solo attraverso quelle transizioni (definito con $E(R)$).
- L'ultima cosa che manca è modificare lo stato iniziale, spostandolo in $E(\{q_0\})$.

A questo teorema si collegano un corollario

> Un linguaggio è regolare se e sono se qualche NFA lo riconosce

Chiusura sotto le operazioni regolari (Nondeterminismo)

Attraverso gli NFA e il non determinismo è possibile semplificare di molto le dimostrazioni della chiusura dei linguaggi regolari sotto le operazioni regolari:

- Per dimostrare la chiusura sotto l'operazione di unione basta prendere due automi N_1 e N_2 e aggiungere uno stato inziale (esterno a entrambi) collegandolo ad ambo gli automi con una transizione ϵ.
- Per dimostrare la chiusura sotto la concatenazione basta collegare gli stati finali del primo automa con lo stato iniziale del secondo usando delle transizioni ϵ.
- Per dimostrare la chiusura sotto lo star utilizziamo due passaggi:
    1. Inseriamo un nuovo stato iniziale, che sarà anche stato di accettazione, e colleghiamolo al vecchio stato iniziale con una transizione ϵ.
    2. Colleghiamo gli stati finali dell'automa con il suo vecchio stato iniziale usando transizioni ϵ.

Espressioni regolari

Come in aritmetica usiamo le classiche operazioni di somma e moltiplicazioni per costruire delle espressioni, possiamo utilizzare le operazioni regolare per costruire delle **espressioni regolari.**

Il valore di un'espressione regolare è un linguaggio. Per esempio l'espressione $(0 \cup 1)0^*$ ha come valore un linguaggio che consiste di tutte le stringhe che iniziano per 0 o per 1 seguite da un qualunque numero di 0. Il simbolo di concatenazione è implicito, quindi l'espressione precedente può essere riscritta come $(0 \cup 1) \circ 0^*$.

Definizione formale di espressione regolare

Diciamo che R è un'espressione regolare se è:

1. a per qualche a nell'alfabeto Σ → Rappresenta il linguaggio $\{a\}$
2. ϵ → Rappresenta il linguaggio $\{\epsilon\}$
3. $\emptyset$ → Rappresenta il linguaggio vuoto
4. $(R_1 \cup R_2)$ dove R_1 e R_2 sono espressioni regolari → Rappresenta il linguaggio dato dall'unione di due linguaggi
5. $(R_1 \circ R_2)$ dove R_1 e R_2 sono espressioni regolari → Rappresenta il linguaggio dato dall'intersezione di due linguaggi
6. (R_1^*) dove R_1 è un'espressione regolare → Rappresenta il linguaggio dato dallo star di un linguaggio

Importante ricordare la differenza tra ϵ e $\emptyset$: il primo rappresenta la stringa vuota, mentre il secondo rappresenta il linguaggio che non contiene stringhe.

Da notare anche che non incappiamo in una **definizione circolare**, siccome i due linguaggi R_1 e R_2 sono entrambi più piccoli di R. Di conseguenza evitiamo la circolarità e stiamo effettuando una **definizione induttiva**, ovvero definiamo un'espressione in termini di espressioni più piccole.

Equivalenza con un automa finito

Le espressioni regolare e gli automi a stati finiti sono equivalenti nel loro potere descrittivo.

> Un linguaggio è regolare se e solo se qualche espressione regolare lo descrive.

PROOF IDEA:

- Prendiamo la prima direzione: "se un linguaggio è descritto da un'espressione regolare, allora è regolare".

Prendiamo un'espressione R che descrive un qualche linguaggio A. Dimostriamo come convertire R in un NFA che riconosce il linguaggio. Dato il corollario del teorema 1.39 sappiamo che se un NFA riconosce un linguaggio, allora il linguaggio è regolare.

- Prendiamo ora la seconda direzione: “se un linguaggio è regolare, alloa è descritto da un’espressione regolare”.

Dobbiamo dimostrare che se un linguaggio A è regolare, un’espressione regolare lo descrive. Siccome A è regolare, allora è accettata da un DFA. Di conseguenza basta descrivere una procedura che converta un DFA in un’espressione regolare equivalente.

Notazioni utili

- Utilizziamo la notazione R^+ per indicare RR^*, ovvero il linguaggio che ha tutte le stringhe che sono una o più concatenazioni di stringhe di R.
- Utilizziamo R^k per indicare la concatenazione di R k-volte.
- Quando vogliamo distinguere l’espressione dal linguaggio regolare che essa descrive utilizziamo la notazione $L(R)$.

Linguaggi non regolari

Passiamo ora a definire le limitazioni degli automi a stati finiti. Dimostriamo quindi che certi linguaggi non possono essere riconosciuti da alcun automa a stati finiti. Prendiamo per esempio il linguaggio $B = \{0^n1^n | n \geq 0\}$. Se cerchiamo di trovare un DFA che lo riconosca, notiamo che l’automa dovrebbe ricordare quanti 0 e 1 sono stati letti in input. Siccome però il numero di 0 e 1 può essere infinito, la macchina dovrebbe ricordare infiniti stati.

Il metodo di provare la non regolarità viene da un teorma sui linguaggi regolari, tradizionalmente chiamato **pumping lemma**. Questo teorema dimostra che tutti i linguaggi regolari hanno una speciale proprietà: se dimostriamo che un linguaggio non ha tale proprietà allora sappiamo che non è regolare.

Questa proprietà dice che tutte le stringhe nel linguaggio hanno un valore chiamato **lunghezza di pumping**, ovvero una sezione che può essere rièetuta un numero qualsiasi di volte con la stringa risultante ancora presente nel linguaggio.

Teorema 1.70 - Pumping lemma

> Se A è un linguaggio regolare, allora c’è un numero p (la lunghezza di pumping) tale che data s una stringa in A di lunghezza almeno p, allora s può essere divisa in 3 pezzi $s = xyz$, che soddisfano 3 condizioni:

1. Per ogni $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$
3. $|xy| \leq p$

La notazione $|y|$ rappresenta la lunghezza della stringa data, y^i indica che i-copie della stringa y sono state concatenate insieme e $y^0 = \epsilon$.

PROOF IDEA:

Preso un DFA che riconosce un linguaggio A, assegnamo la lunghezza di pumping uguale al numero di stati dell'automa. A questo punto dimostriamo che ogni stringa $s \in A$ di lunghezza almeno p (lunghezza di pumping) può essere divisa in 3 pezzi xyz che soddisfano le condizioni del teorema.

Se non ci sono stringhe di lunghezza p allora dimostrarlo diventa ancora più semplice.