Teoria dei Linguaggi

Indice

1. Lezione 22 [23/05]

In questa lezione parleremo di troppe cose: toccheremo tutta la gerarchia di Chomsky, esclusi i linguaggi di tipo 0, considerando alfabeti particolari e macchine riconoscitrici diverse dal solito. Andremo poi avanti anche con le grammatiche di tipo 2.

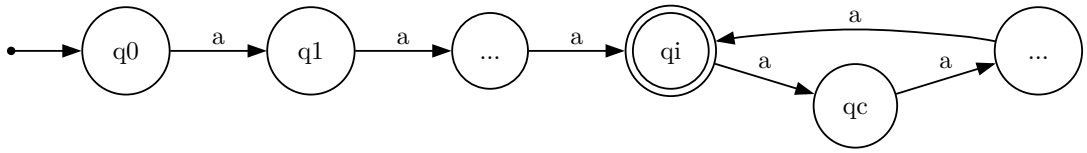
1.1. Alfabeti unari

Partiamo con gli **alfabeti unari**: questi sono alfabeti molto particolari formati da un solo carattere, ovvero sono nella forma

$$\Sigma = \{a\}.$$

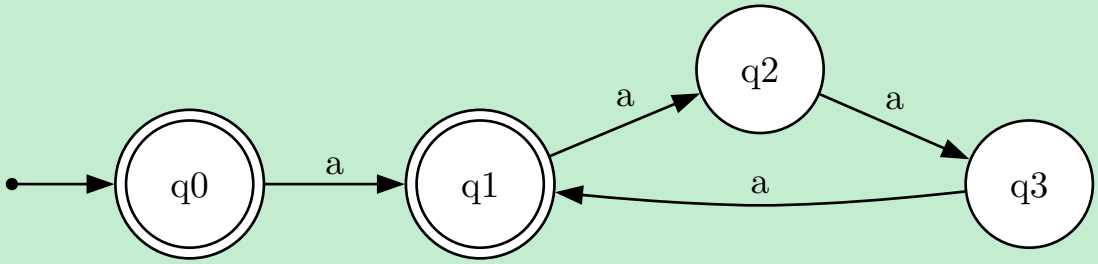
1.1.1. Linguaggi regolari

Riprendiamo in mano, dopo tanto tempo, gli **automi a stati finiti**. Se rimaniamo nel caso deterministico, da ogni stato di un **DFA** può uscire un solo arco con una certa etichetta, ovvero non posso avere più di 2 archi uscenti con la stessa etichetta. Avendo ora un solo carattere in Σ quello che abbiamo è una sequenza (opzionale) di stati che prima o poi sfocia in un **ciclo** (opzionale).



Notiamo come l'informazione sulle parole diventa **informazione sulla lunghezza** di esse, visto che possiamo riconoscere delle stringhe che seguono un certo pattern di lunghezze.

Esempio 1.1.1.1: Vediamo un esempio di automa a stati finiti unario.



Con questo automa riconosciamo ε, a e poi quest'ultima a cui aggiungiamo un numero di a uguali alla lunghezza del ciclo, ovvero

$$L = \{\varepsilon\} \cup \{a^{1+3k} \mid k \geq 0\} = \varepsilon + a(a^3)^*.$$

Dal punto di vista matematico, possiamo vedere questi automi come delle **successioni numeriche/aritmetiche**, ovvero delle successioni che hanno una parte iniziale e poi un periodo che viene ripetuto.

Nel caso di **NFA** invece abbiamo un grafo arbitrario, che per essere trasformato in DFA richiede meno dei 2^n classici della **costruzione per sottoinsiemi**, ovvero ci costa

$$e^{n \ln(n)}.$$

Come vediamo, è una quantità **subesponenziale** ma comunque **superpolinomiale**. Inoltre, questo bound non può essere migliorato, è la soluzione ottimale.

Esempio 1.1.1.2: Definiamo tre linguaggi

$$L_1 = a^{28}(a^3)^* \qquad L_2 = a^{11}(a^3)^* \qquad L_3 = a^{37}(a^3)^*$$

Cosa aggiungono questi linguaggi al linguaggio L dell'Esempio 1.1.1.1?

Se consideriamo L_1 notiamo che a^{28} può essere anche riconosciuto facendo uno step con una a e poi facendo 9 cicli da 3, quindi riusciamo a riconoscerlo anche con L. Possiamo fare un discorso praticamente simile con L_3. Ma allora questi due linguaggi non aggiungono niente.

Considerando invece L_2 questo aggiunge qualcosa ad L perché riusciamo a riconoscere la stringa $a^{10} = a(a^3)^3$ con L ma poi rimane una a fuori, che ci manda in q_2 e quindi ci fa accettare di più, o comunque qualcosa di diverso rispetto a L.

Avremmo aggiunto altre informazioni considerando un linguaggio

$$L = a^k(a^3)^* \mid k \bmod 3 = 0.$$

Notiamo che, fissato un periodo, non possiamo unire tanti linguaggi, ma solo quelli che rimangono all'interno delle **classi di resto del periodo**.

1.1.2. Equivalenza tra linguaggi regolari e CFL

Vediamo ora come si comportano i CFL. Sia L un **CFL unario**, ovvero

$$L \subseteq a^*.$$

Applichiamo il **pumping lemma** a questo linguaggio. Prendiamo N la **costante del pumping lemma** per i CF per L. Questo ci dice che

$$\forall z \in L \mid |z| \geq N$$

noi possiamo decomporre z come $z = uvwxy$ con:

1. $|vwx| \leq N$;
2. $vw \neq \varepsilon$;
3. $\forall i \geq 0 \quad uv^iwx^iy \in L$.

Le stringhe di L sono formate da sole a, quindi se scambiamo dei fattori nella stringa non lo notiamo. Modifichiamo l'ultima condizione del pumping lemma con

$$\forall i \geq 0 \quad uwy(vx)^i \in L.$$

Mettendo insieme le prime due condizioni possiamo dire che

$$1 \leq |vx| \leq N.$$

La stringa z la possiamo dividere in una **parte fissa** e in una **parte pompabile**, ovvero

$$|z| = |uwy| + |vx| = s_z + t_z.$$

Grazie alla terza condizione sappiamo che

$$\forall i \geq 0 \quad a^{s_z}(a^{t_z})^i \in L \Longrightarrow a^{s_z}(a^{t_z})^* \in L.$$

Possiamo fare un'ulteriore divisione, stavolta sulle stringhe di L: infatti, possiamo scrivere L come unione di due insiemi L' e L'' tali che

$$L = L' \cup L''.$$

Nell'insieme L' mettiamo tutte le stringhe che non fanno parte del pumping lemma, ovvero

$$L' = \{z \in L \mid |z| < N\}.$$

Nell'insieme L'' mettiamo invece tutte le stringhe pompate, ovvero

$$L'' = \{z \in L \mid |z| \geq N\} \subseteq \bigcup_{z \in L''} a^{s_z}(a^{t_z})^*.$$

Analizziamo separatamente i due insiemi:

- L' è un linguaggio **finito**, quindi lo possiamo riconoscere con un automa a stati finiti;
- L'' invece sembra un'unione infinita, ma abbiamo visto che il periodo t_z del pumping lemma è boundato con le classi di resto, ovvero

  $$1 \leq t_z \leq N,$$

  quindi questo linguaggio, che è unione finita di linguaggi regolari, è anch'esso **finito**.

Ma allora il linguaggio L è **regolare**.

Teorema 1.1.2.1: Sia $L \subseteq a^*$ un CFL. Allora L è regolare.

Questo va d'accordo con quello che abbiamo fatto la lezione scorsa: i CFL hanno la **ricorsione**, ma se abbiamo un solo carattere non possiamo aprire e chiudere le parentesi, quindi collassiamo nei linguaggi regolari.

1.1.3. Teorema di Parikh

Vediamo, per finire, una serie di concetti un po' strani e che non dimostreremo.

Definizione 1.1.3.1 (Immagine di Parikh sulle stringhe): Sia $\Sigma = \{\sigma_1, ..., \sigma_n\}$ un alfabeto. L'**immagine di Parikh** sulle stringhe è la funzione

$$\psi : \Sigma^* \longrightarrow \mathbb{N}^{|\Sigma|}$$

tale che

$$\psi(x) = \big(\#_{\sigma_1}(x), ..., \#_{\sigma_n}(x)\big).$$

In poche parole, questa funzione conta le **occorrenze** di ogni lettera di Σ dentro la stringa x.

Esempio 1.1.3.1: Definiamo $\Sigma = \{a, b\}$. Data $z = aababa$, calcoliamo

$$\psi(z) = (4, 2).$$

Con l'immagine di Parikh sulle stringhe possiamo definire un insieme di queste immagini.

Definizione 1.1.3.2 (Immagine di Parikh): Dato L un linguaggio generico, l'**immagine di Parikh** è l'insieme

$$\psi(L) = \{\psi(x) \mid x \in L\}.$$

In poche parole, l'immagine di Parikh è l'insieme di tutte le immagini di Parikh sulle stringhe di L.

Esempio 1.1.3.2: Vediamo tre linguaggi e le loro immagini di Parikh associate.

Linguaggio	**Immagine di Parikh**
$L = \{a^n b^n \mid n \geq 0\}$	$\{(n, n) \mid n \geq 0\}$
$L = a^* b^*$	$\{(i, j) \mid i, j \geq 0\}$
$L = (ab)^*$	$\{(n, n) \mid n \geq 0\}$

Notiamo come il primo e il terzo insieme sono uguali, anche se vengono generati da due linguaggi gerarchicamente diversi: il primo è un tipo 2, il terzo è un tipo 3.

L'ultima osservazione fatta genera quello che è il **teorema di Parikh**.

Teorema 1.1.3.1 (Teorema di Parikh): Se L è un CFL allora $\exists R$ regolare tale che

$$\psi(L) = \psi(R).$$

In poche parole, se non ci interessa l'**ordine** con cui scriviamo i caratteri di una stringa, allora i **linguaggi regolari** e i **CFL** sono la stessa cosa, collassano nella stessa classe.

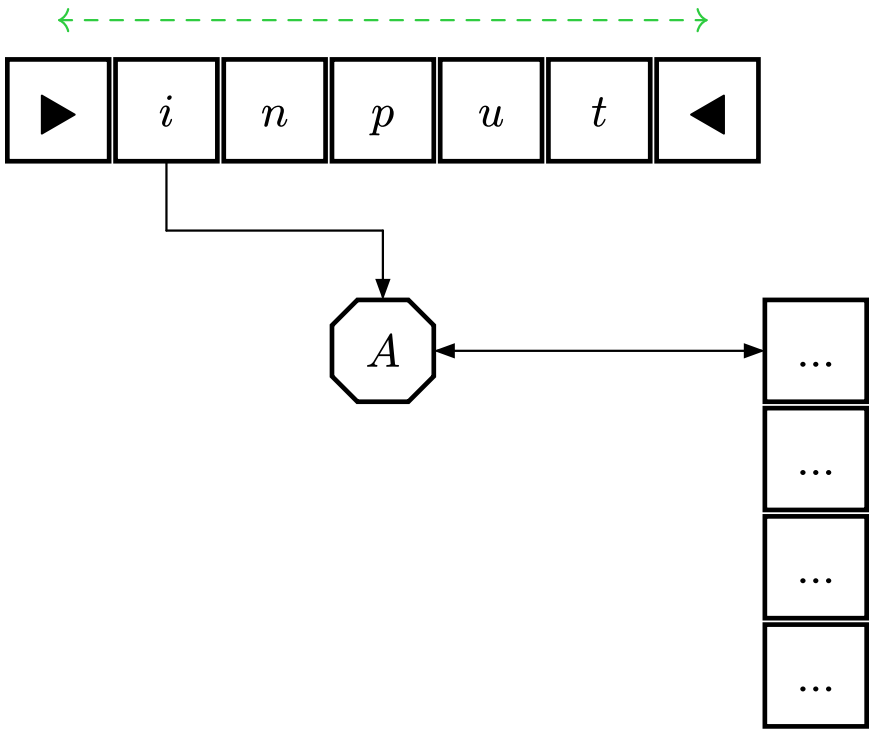
1.2. Automi a pila two-way

Modifichiamo un po' la macchina che stiamo usando da forse troppo tempo.

1.2.1. Definizione

Negli automi a stati finiti, il movimento **two-way** non aumentava la potenza computazionale del modello. Ma cosa succede negli **automi a pila two-way**?

Vediamo prima di tutto una rappresentazione del modello.



Come nei 2DFA, mettiamo degli **end marker** per marcare i bordi della stringa, perché ora la nostra testina di lettura può andare **avanti e indietro** sul nastro.

Con questo modello possiamo fare **molto di più** dei classici automi a pila.

1.2.2. Esempi

Vediamo una serie di linguaggi non CFL che riusciamo a riconoscere con questo modello.

Esempio 1.2.2.1: Definiamo il linguaggio

$$L = \{a^n b^n c^n \mid n \geq 0\}.$$

Questo linguaggio non è CFL perché una volta che controlliamo le b con le a perdiamo l'informazione su n. Con un **2DPDA** possiamo controllare le a con le b, poi tornare all'inizio delle b e controllare le b con le c.

Esempio 1.2.2.2: Definiamo il linguaggio

$$L = \{a^{2^n} \mid n \geq 0\}.$$

Con il **pumping lemma** avevamo mostrato che questo linguaggio non è CFL. Ora che abbiamo la definizione di sequenza algebrica, possiamo dire che questo linguaggio non è una **sequenza algebrica** perché le a si allontanano sempre di più tra loro.

Dobbiamo controllare se l'input è una potenza di 2: per fare ciò continuiamo a dividere per 2, verificando di avere sempre resto zero, salvo alla fine, dove abbiamo per forza resto 1.

Se k è la lunghezza dell'input, possiamo eseguire i seguenti passi:

1. leggiamo l'input per intero, e ogni due a carichiamo una lettera sulla pila, caricando in totale $\frac{k}{2}$ caratteri. Con questa passata controlliamo se le a sono pari o dispari;
2. svuotiamo la pila, spostandoci di una posizione a sinistra ogni volta che togliamo un carattere. Con questa mezza passata ci troviamo, appunto, a metà della stringa, sul carattere in posizione $\frac{k}{2}$;
3. ricominciamo dal primo punto fino a quando non rimaniamo con un carattere solo, che mi dà per forza resto 1.

Anche questo, come quello di prima, è un **2DPDA**.

Esempio 1.2.2.3: Definiamo il linguaggio

$$L = \{ww \mid w \in \{a, b\}^*\}.$$

Anche questo linguaggio non è CFL, e lo avevamo mostrato con uno dei quattro criteri della scorsa lezione, non mi ricordo quale in questo momento.

Come prima, carichiamo nella pila un carattere ogni due caratteri letti dell'input completo. Con questa prima passata controlliamo anche se il numero di caratteri è pari o dispari, e in quest'ultimo caso ci fermiamo e rifiutiamo. Spostiamoci poi in mezzo alla stringa scaricando la pila fino al carattere iniziale che avevamo anche prima.

Chiamiamo w la parte a sinistra della posizione nella quale ci troviamo ora. Carichiamo w dal centro verso l'inizio: stiamo leggendo w^R, che caricata sulla pila diventa w.

Spostiamoci di nuovo a metà della stringa, mettendo un separatore # tra w e i caratteri che usiamo per spostarci. Ora che siamo a metà, togliamo # dal congelatore e, con w sulla pila, possiamo controllare se la seconda parte è uguale a w.

Anche questo è un fantastico **2DPDA**.

Esempio 1.2.2.4: Non lo facciamo vedere, ma il linguaggio

$$L = \{ww^R \mid w \in \{a, b\}^*\}$$

ha un **2DPDA** che lo riconosce in maniera molto simile a quelle precedenti.

Come vediamo, questo modello è **molto potente**, talmente potente che nessuno sa quanto sia potente: infatti, tutti gli esempi visti sono stati risolti con un **2DPDA**, quindi anche da un **2NPDA** che fa partire una sola computazione alla volta, ma non sappiamo se

$$\text{2NPDA} \stackrel{?}{=} \text{2DPDA} .$$

Inoltre, non si conosce la relazione che si ha con i linguaggi di tipo 1, che vediamo tra poco, ovvero non sappiamo se

$$\text{2DPDA} \stackrel{?}{=} \text{CS} .$$

1.3. Problemi di decisione dei CFL

Per finire questa lezione infinita, torniamo indietro ai linguaggi CFL e vediamo qualche **problema di decisione**. Per ora vedremo i problemi a cui sappiamo rispondere con quello che sappiamo, questo perché dei problemi di decisione richiedono conoscenze delle **macchine di Turing**, che per ora non abbiamo.

1.3.1. Appartenenza

Dato L un CFL e una stringa $x \in \Sigma^*$, ci chiediamo se $x \in L$.

Questo è molto facile: sappiamo che i CFL sono **decidibili** perché lo avevamo mostrato per i linguaggi di tipo 1. Come complessità come siamo messi?

Sia $n = |x|$. Esistono algoritmi semplici che permettono di decidere in tempo

$$T(n) = O(n^3).$$

L'**algoritmo di Valiant**, quasi incomprensibile, riconduce il problema di riconoscimento a quello di prodotto tra matrici $n \times n$, che con l'algoritmo di Strassen possiamo risolvere in tempo

$$T(n) = O\big(n^{\log_2(7)}\big) = O(n^{2.81\dots}).$$

L'algoritmo di Strassen in realtà poi è stato superato da altri algoritmi ben più sofisticati, che impiegano tempo quasi quadratico, ovvero

$$T(n) = O(n^{2.3\dots}).$$

Una domanda aperta si chiede se riusciamo ad abbassare questo bound al livello quadratico, e questo sarebbe molto comodo: infatti, negli algoritmi di parsing avere degli algoritmi quadratici è apprezzabile, e infatti spesso di considerano sottoclassi per avvicinarsi a complessità lineari.

1.3.2. Linguaggio vuoto e infinito

Sia L un CFL, ci chiediamo se $L \neq \varnothing$ oppure se $|L| = \infty$.

Vediamo un teorema praticamente identico a uno che avevamo già visto.

Teorema 1.3.2.1: Sia $L \subseteq \Sigma^*$ un CFL, e sia N la costante del pumping lemma per L. Allora:

1. $L \neq \varnothing \iff \exists z \in L \mid |z| < N$;
2. $|L| = \infty \iff \exists z \in L \mid N \leq |z| < 2N$.

Gli algoritmi per verificare la non vuotezza o l'infinità non sono molto efficienti: infatti, prima di tutto bisogna trovare N, e se ho una grammatica è facile (basta passare in tempo lineare per la FN di Chomsky), ma se non ce l'abbiamo è un po' una palla. Poi dobbiamo provare tutte le stringhe fino alla costante, che sono 2^N, e con questo rispondiamo alla non vuotezza. Per l'infinità è ancora peggio.

Si possono implementare delle tecniche che lavorano sul **grafo delle produzioni**, ma sono molto avanzate e (penso) difficili da utilizzare.

1.3.3. Universalità

Dato L un CFL, vogliamo sapere se $L = \Sigma^*$, ovvero vogliamo sapere se siamo in grado di generare tutte le stringhe su un certo alfabeto.

Nei linguaggi regolari passavamo per il complemento per vedere se il linguaggio era vuoto, ma nei CFL **non abbiamo il complemento**, quindi non lo possiamo utilizzare.

Infatti, questo problema **non si può decidere**: non esistono algoritmi che stabiliscono se un PDA riesce a riconoscere tutte le stringhe, o se una grammatica riesce a generare tutte le stringhe.