Esempi di Dimostrazioni di Chiusura per Linguaggi Regolari

Gabriel Rovesti

March 29, 2025

1 Introduzione

Presentiamo alcune dimostrazioni formali relative alla chiusura della classe dei linguaggi regolari rispetto a varie operazioni: *NOPREFIX*, *NOEXTEND*, *avoids*, *perfect shuffle* e *shuffle*. Richiamiamo innanzitutto la definizione di *linguaggio regolare*.

Definizione 1 (Linguaggio Regolare)**.** Un linguaggio $L \subseteq \Sigma^*$ si dice *regolare* se esiste un *automa a stati finiti* (DFA o NFA) che riconosce tutte e soltanto le stringhe di L, oppure se esiste un'espressione regolare che lo descrive.

2 Operazioni NOPREFIX e NOEXTEND

Definizione 2. Dato un linguaggio $A \subseteq \Sigma^*$, definiamo:

$$\text{NOPREFIX}(A) = \{\, w \in A \;\mid\; \text{nessun prefisso proprio di } w \text{ appartiene ad } A\},$$
$$\text{NOEXTEND}(A) = \{\, w \in A \;\mid\; w \text{ non è prefisso proprio di alcuna stringa in } A\}.$$

Ricordiamo che un *prefisso proprio* di w è una stringa u tale che $w = uz$ con $z \neq \varepsilon$ e $u \neq w$.

2.1 Chiusura di NOPREFIX(A)

Teorema 1. *Se A è un linguaggio regolare, allora* NOPREFIX(A) *è regolare.*

Dimostrazione (traccia). L'idea è costruire un DFA (o NFA) che riconosce A, quindi *modificarlo* in modo da rifiutare ogni stringa w che abbia un prefisso proprio in A. In dettaglio:

(i) Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce A.

(ii) Vogliamo accettare w soltanto se:

$$w \in A \quad \text{e} \quad \forall u \text{ prefisso proprio di } w,\ u \notin A.$$

(iii) In M, una stringa u è accettata se la computazione di u da q_0 termina in uno stato di F. Dunque, **se esiste un prefisso proprio** u di w **accettato** da M, w va rifiutata.

(iv) Possiamo costruire un nuovo DFA M' partendo da M e aggiungendo un *controllo di memoria finita*:

- Se durante la lettura di w troviamo che un prefisso (terminato in un certo punto) è già stato accettato da M, entriamo in uno stato trappola che *rifiuta* tutto il resto dell'input.
- Alla fine, per accettare w, occorre che lo stato finale corrisponda a "$w \in A$" e non ci sia mai stato un prefisso accettato prima del termine.

(v) Formalmente, si può costruire la *macchina prodotto* fra M e una "spia" booleana che monitori se *qualche* prefisso precedente fosse accettante. Se la "spia" si attiva, andiamo in uno stato di rifiuto permanente.

(vi) Infine, si definiscono gli stati finali come quei nodi in cui la spia è *spenta* (nessun prefisso in A) e la componente di M è in uno stato di F (cioè l'intera stringa w *è* in A).

In questo modo otteniamo un DFA che riconosce esattamente $w \in \text{NOPREFIX}(A)$, dimostrando la regolarità. □

2.2 Chiusura di NOEXTEND(A)

Teorema 2. *Se A è regolare, allora* NOEXTEND(A) *è regolare.*

Idea della dimostrazione. Definizione:

$$\text{NOEXTEND}(A) = \{\, w \in A \mid \text{non esiste } x \neq \varepsilon \text{ con } wx \in A\}.$$

In termini di automi, possiamo dire che w è in A, ma non è possibile *prolungare* w con altri simboli per rimanere in A.

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA per A. Quando *accetta* w, si trova in uno stato $f \in F$. Vogliamo accettare w soltanto se *nessuna* estensione di w

rimane in A. Cioè, da quello stato f, **tutte** le transizioni $\delta(f, a)$, per ogni $a \in \Sigma$, devono condurre a stati *non accettanti* (o in un percorso che non porterà mai più a uno stato accettante).

La costruzione formale:

- Conosciamo per ciascuno stato $q \in Q$ se, a partire da q, ci sia un percorso che conduce in uno stato finale.
- Ci basta individuare quegli stati $f \in F$ tali che *nessuna* estensione da f ritorna in un altro stato finale.
- Marcando tali stati come "nuovi stati finali", e rimuovendo dal DFA tutti gli altri stati finali, otteniamo un nuovo automa M' che riconosce $NOEXTEND(A)$.

L'algoritmo per determinare se "da f si può tornare in finale" è una classica visita sul grafo degli stati, o una *reverse search* sugli stati finali originari. Tutto ciò è eseguibile in modo finito, producendo un DFA. Dunque NOEXTEND(A) è regolare. □

3 Operatore *avoids*

Definizione 3 (*avoids*)**.** Date due lingue $A, B \subseteq \Sigma^*$, definiamo

$A \,\text{avoids}\, B \;=\; \{\, w \in A \;\mid\; w\text{non contiene alcuna stringa di } B \text{ come sottostringa}\}.$

Teorema 3. *Se A e B sono linguaggi regolari su Σ, allora $A \,\text{avoids}\, B$ è regolare.*

Dimostrazione (schema). L'idea tipica è di vedere la *condizione "nessuna stringa di B come sottostringa"* come un insieme di pattern da *evitare*. Se B è finito, è più semplice:

$$A \,\text{avoids}\, B = A \;\cap\; \bigcap_{x \in B} \overline{\Sigma^* \, x \, \Sigma^*}.$$

Ognuno dei linguaggi $\Sigma^* x \Sigma^*$ (che è "contenere x come substring") è regolare, e quindi il suo complementare è regolare (proprietà di chiusura). Intersecando un numero finito di regolari si ottiene un regolare.

Se B è infinito, esiste comunque un metodo classico (costruzione Aho–Corasick o simili) che produce un NFA in grado di "monitorare" simultaneamente tutte le possibili substring di B. L'NFA scarta non appena incontra una stringa di B. Quella costruzione è sempre *finita* se B è regolare, dimostrando la chiusura. □

4 Perfect Shuffle e Shuffle

Definizione 4 (Perfect Shuffle)**.** Dati due linguaggi A e B sullo stesso alfabeto Σ, il *perfect shuffle* è definito come

$$A \otimes B = \Big\{ w \mid w = a_1 b_1 \cdots a_k b_k,\ \ a_1 \cdots a_k \in A,\ \ b_1 \cdots b_k \in B,\ k \geq 0 \Big\}.$$

In altre parole, si "mescolano" le stringhe $a_1 \ldots a_k$ e $b_1 \ldots b_k$ in modo *perfettamente alternato* (un simbolo da A, poi un simbolo da B, ecc.).

Teorema 4. *Se A e B sono regolari, allora $A \otimes B$ è regolare (perfect shuffle).*

Idea della costruzione. Siano M_A e M_B due DFA (o NFA) che riconoscono rispettivamente A e B. Vogliamo un NFA che, dato w, "estragga" i simboli in posizione dispari di w e li sottoponga a M_A, ed "estragga" i simboli in posizione pari di w e li sottoponga a M_B. Un modo standard:

- L'NFA legge un simbolo e lo manda alla componente "A" (cambia stato nella macchina M_A), poi legge il simbolo successivo e lo manda alla componente "B" (cambia stato in M_B), alternando.
- Se la lunghezza di w è dispari, l'ultimo simbolo letto va alla macchina M_A.
- L'NFA accetta se e solo se entrambe le macchine M_A e M_B terminano in uno stato finale (al netto di eventuali differenze di lunghezza).

Formalmente si costruisce il prodotto degli stati con un indice *mod* 2 che dice "tocca ad A o a B". Lo stato finale è $(q_A, q_B, \text{parità})$ dove $q_A \in F_A$ e $q_B \in F_B$ e la parità coincide con la lunghezza del giro "perfetto". L'insieme delle transizioni rispecchia il passaggio giusto a M_A o M_B a seconda che ci si trovi in uno step di tipo "dispari" o "pari".

Trattandosi di un costrutto *finitamente* implementabile, $A \otimes B$ resta regolare. □

Definizione 5 (Shuffle)**.** Dati due linguaggi A e B, lo *shuffle* (senza la parola "perfect") è

$$A\ B\ =\ \Big\{ w \mid w = a_1 b_1 \cdots a_k b_k \text{ con } a_1 \cdots a_k \in A, b_1 \cdots b_k \in B,\ \text{gli } a_i, b_j \in \Sigma^* \Big\}.$$

Qui non è richiesta l'alternanza stretta simbolo-per-simbolo, bensì *blocchi* di simboli da A mescolati con blocchi di simboli da B.

Teorema 5. *Se A e B sono regolari, allora $A\ B$ è regolare.*

Idea del riconoscitore NFA. La macchina a stati finiti può "nondeterministicamente" decidere quanto leggere come pezzo di A e quando passare a leggere un pezzo di B, e così via, finché consuma l'intera stringa. Formalmente:

- Abbiamo un NFA con due *copie* di un automa per A e uno per B.
- A ogni step, l'NFA sceglie se continuare a mandare i simboli letti nella componente "A" o se passare (via ε-transizione) alla componente "B", e così via.
- Il disco di "controllo" di A deve accettare la concatenazione di tutti i blocchi $a_i \in \Sigma^*$, mentre il controllo di B deve accettare la concatenazione di tutti i blocchi $b_i \in \Sigma^*$.

Alla fine, se entrambe le componenti si trovano in stati accettanti (si gestiscono con un prodotto incrociato) e l'intero input è stato consumato in un modo compatibile, l'NFA accetta. Da qui la regolarità di AB. $\square$

5 Conclusioni e Riferimenti

Abbiamo visto esempi di come la classe dei linguaggi regolari sia chiusa rispetto a svariati operatori, anche se alcuni (come NOPREFIX, NOEXTEND o avoids) richiedono costruzioni meno *immediate* rispetto a quelle canoniche (unione, intersezione, complemento, ecc.). Le costruzioni di perfect shuffle e shuffle si basano su macchine in *parallelo* o *nondeterministiche* che smistano gli input nei diversi sottolinguaggi.

Riferimenti classici:

- Hopcroft, Motwani, Ullman, *Introduction to Automata Theory, Languages, and Computation.*
- M. Sipser, *Introduction to the Theory of Computation.*
- Per la costruzione avoids con insiemi infiniti, vedi l'algoritmo *Aho–Corasick* (AC Automaton), *Commun. ACM* 1975.