suffix e Forme Normali - Slide

Gabriel Rovesti

1 Esercizio 1: Chiusura di suffix(L) per i CFL

Definizione 1.1. Per un linguaggio qualunque $L \subseteq \Sigma^*$, definiamo:

$$\text{suffix}(L) \;=\; \{\, v \in \Sigma^* \;\mid\; \exists\, u \in \Sigma^* : uv \in L\}.$$

Teorema 1.2. *Se L è un linguaggio context-free, allora* suffix(L) *è anch'esso context-free.*

Dimostrazione (Schema). Sia $L \subseteq \Sigma^*$ un linguaggio context-free. Allora esiste un PDA (Pushdown Automaton) M o, equivalentemente, una grammatica G che lo genera. Mostriamo come costruire un PDA (o una grammatica) che riconosca tutte le stringhe che sono suffissi di qualche stringa in L.

Costruzione con PDA: Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA che riconosce L. Vogliamo un nuovo PDA M' che, su input v, "indovini" in quale punto di v avrebbe potuto iniziare la parte u (cioè la parte iniziale di una stringa $uv \in L$).

(i) Inizialmente, M' può nondeterministicamente "consumare" un certo numero di simboli dell'input (simulando come se fossero la parte u), senza realmente spingere alcunché nello stack in modo vincolato.

(ii) A un certo punto, sempre in modo nondeterministico, M' decide che comincia la parte effettiva v. Da quel momento, M' simula il comportamento di M *come se* stesse leggendo la coda finale di una stringa uv.

(iii) Se a fine input M' si trova in uno stato accettante (e lo stack è gestito coerentemente), allora v è suffisso di qualche stringa in L.

Poiché questa costruzione è realizzabile all'interno del modello dei PDA (grazie al nondeterminismo), il linguaggio riconosciuto è context-free. Dunque suffix(L) è context-free.

Osservazione: Un'argomentazione analoga si può impostare a livello di grammatiche context-free, trasformando la grammatica in modo che una parte iniziale (corrispondente a u) venga "saltata" con produzioni che non-deterministicamente ignorano una sezione di stringa prima di iniziare la vera derivazione. □

2 Esercizio 2: Inerente Ambiguità di $\{a^i b^j c^k \mid i = j$ oppure $j = k\}$

Teorema 2.1. *Il linguaggio*

$$L = \{ a^i b^j c^k \mid i = j \text{ oppure } j = k\}$$

è inerentemente ambiguo*, cioè non esiste alcuna grammatica context-free non ambigua che lo generi.*

Idea della Dimostrazione. Osserviamo che

$$L = L_1 \cup L_2,$$

dove

$$L_1 = \{ a^n b^n c^k \mid n, k \geq 0\}, \quad L_2 = \{ a^i b^n c^n \mid i, n \geq 0\}.$$

Entrambi L_1 e L_2 sono linguaggi context-free. Inoltre, la loro intersezione è:

$$L_1 \cap L_2 = \{ a^n b^n c^n \mid n \geq 0\},$$

che è un noto linguaggio context-free *ambiguo* (infatti $\{a^n b^n c^n\}$ non può essere generato da una grammatica deterministica e si mostra anche che ogni grammatica per esso introduce ambiguità in un contesto di unione).

L'idea standard per provare l'inerente ambiguità è supporre, per assurdo, che L sia generato da una grammatica non ambigua G. Se ciò fosse vero, allora $(L \cap R)$ sarebbe non ambiguo per ogni linguaggio regolare R. Scegliendo opportunamente un automa finito che "filtra" esattamente le stringhe $a^n b^n c^n$ in L, si ottiene una grammatica non ambigua anche per $\{a^n b^n c^n\}$. Ciò contraddice la dimostrata ambiguità intrinseca di $\{a^n b^n c^n\}$. Concludiamo che L è *inerentemente ambiguo.* □

3 Esercizio 3: Trasformazione in Forma Normale di Chomsky (CNF)

Consideriamo la grammatica

$$G_6: \quad \begin{cases} S \;\to\; A\,S\,A \;\mid\; a\,B, \\ A \;\to\; B \;\mid\; S, \\ B \;\to\; b \;\mid\; \varepsilon. \end{cases}$$

Vogliamo trasformarla in Forma Normale di Chomsky (CNF). Ricordiamo che in CNF ogni regola ha la forma $A \to BC$ oppure $A \to a$, con l'unica eccezione della possibile regola $S_0 \to \varepsilon$ se il linguaggio ammette la stringa vuota.

3.1 Step 1: Aggiunta di un nuovo simbolo iniziale

Aggiungiamo una nuova variabile iniziale S_0 che non compare altrove e poniamo

$$S_0 \;\to\; S.$$

In questo modo, ci assicuriamo che la vecchia variabile S non compaia mai sul lato destro come simbolo iniziale di produzione (requisito utile in CNF).

3.2 Step 2: Eliminazione di regole ε-produzioni

La regola $B \to \varepsilon$ è una ε-produzione. Dobbiamo eliminarla "propagando" l'effetto di ε in tutte le produzioni dove B compare a destra.

Le regole originali che contengono B:

$$\begin{aligned} &S \to aB, \\ &A \to B \quad \text{(anche } A \to S\text{, ma non contiene } B\text{)}, \\ &B \to b \;\mid\; \varepsilon. \end{aligned}$$

- Dalla produzione $S \to aB$, sostituendo B con ε otteniamo anche $S \to a$. - Dalla produzione $A \to B$, sostituendo B con ε otteniamo $A \to \varepsilon$. Poiché A non è il nuovo simbolo iniziale, dobbiamo continuare a eliminare ε-produzioni se si creano nuovi casi.

In definitiva, la regola $A \to \varepsilon$ va eliminata a sua volta, propagandone gli effetti: - Nel corpo di $S \to ASA$, potremmo sostituire uno o entrambi gli A con ε. Quindi emergono nuove produzioni:

$$S \to SA, \quad S \to AS, \quad S \to S, \quad \text{(attenzione a quest'ultima } S \to S \text{ è inutile)},$$

e in alcuni contesti potrà ridursi ulteriormente. - Nella regola $A \to S$, sostituire S non ci dà direttamente ε, ma potrà influire su altre produzioni.

Alla fine di questo step, si rimuovono tutte le ε-produzioni. Le stringhe che si ottengono possono essere molte, ma si mantiene l'equivalenza (a parte la stringa vuota, se non era già generata).

3.3 Step 3: Eliminazione delle regole unitarie ($A \to B$, $A \to S$, etc.)

Ora si rimuovono le produzioni unitarie, come $A \to B$, $A \to S$. L'idea è: se $A \to B$ e $B \to u$ (dove u è una stringa di terminali e/o variabili), allora aggiungiamo $A \to u$ e rimuoviamo $A \to B$. Stessa cosa per $A \to S$.

3.4 Step 4: Messa in forma $A \to BC$ o $A \to a$

Alla fine, si sostituiscono tutte le produzioni di lunghezza ≥ 2 di terminali e/o variabili con catene di variabili binarie. Inoltre, si rimpiazza ogni terminale isolato in una produzione lunga con una variabile ad hoc (es. $X_a \to a$).

Risultato Finale (Esempio di CNF): Potremmo arrivare a un insieme di produzioni simile a:

$$\begin{aligned}
S_0 &\to S, \\
S &\to X_a B \mid SA \mid AS \mid \dots \\
A &\to B \mid S \mid \dots \\
B &\to X_b \ \dots \\
X_a &\to a, \\
X_b &\to b.
\end{aligned}$$

dove, passo dopo passo, abbiamo eliminato ε, poi le unitarie, e poi messo in forma binaria (Chomsky). A seconda di come vengono effettuati i passaggi di eliminazione e di come si gestiscono le produzioni ridondanti, il risultato finale può avere più regole, ma tutte del tipo $A \to BC$ o $A \to$ (terminale). L'unica ε-produzione ammessa sarebbe eventualmente $S_0 \to \varepsilon$ se il linguaggio originale poteva generare la stringa vuota.

Riferimenti

- Hopcroft, Motwani, Ullman. *Introduction to Automata Theory, Languages, and Computation.*

- M. Sipser. *Introduction to the Theory of Computation.*