Automi a Pila e Grammatiche Context-Free

Riassunto dei concetti fondamentali

Tutorato 6 - PDA, CFG e Primo Compitino

Gabriel Rovesti
Università degli Studi di Padova

Anno Accademico 2024-2025

Contents

1 Automi a Pila (PDA)

1.1 Definizione Formale

Concetto chiave

Un automa a pila (Pushdown Automata, PDA) è una sestupla $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ dove:

- Q è l'insieme finito di stati
- Σ è l'alfabeto di input
- Γ è l'alfabeto della pila
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to 2^{Q \times \Gamma_\varepsilon}$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati accettanti

I PDA estendono gli automi a stati finiti (FSA) aggiungendo una memoria a pila potenzialmente infinita. Questa struttura dati permette di riconoscere linguaggi context-free che non possono essere riconosciuti da FSA, come $\{0^n 1^n \mid n \geq 0\}$.

1.2 Funzionamento della Pila

La pila è un dispositivo di memoria LIFO (Last In, First Out) che permette due operazioni fondamentali:

- **Push**: scrive un nuovo simbolo in cima alla pila e "spinge giù" gli altri
- **Pop**: legge e rimuove il simbolo in cima alla pila (top)

Suggerimento

La pila permette di avere memoria infinita (ad accesso limitato). A differenza degli FSA, che hanno memoria limitata al numero di stati, i PDA possono tenere traccia di strutture annidate arbitrariamente profonde.

1.3 Accettazione per Pila Vuota

Un PDA accetta la parola w per pila vuota se esiste una computazione che:

- Consuma tutto l'input
- Termina con la pila vuota ($s_m = \varepsilon$)

Procedimento di risoluzione

Per verificare l'accettazione di una stringa w da parte di un PDA:

1. Iniziare nello stato q_0 con il simbolo iniziale nella pila
2. Seguire le transizioni applicabili, consumando input e/o manipolando la pila
3. Verificare se, dopo aver consumato tutto l'input, il PDA ha svuotato la pila

2 Grammatiche Context-Free (CFG)

2.1 Definizione e Componenti

Concetto chiave

Una grammatica context-free è una quadrupla $G = (V, \Sigma, R, S)$ dove:

- V è un insieme finito di variabili (o non-terminali)
- Σ è un insieme finito di simboli terminali, disgiunto da V
- R è un insieme di regole di produzione, ciascuna della forma $A \to \alpha$ dove $A \in V$ e $\alpha \in (V \cup \Sigma)^*$
- $S \in V$ è la variabile iniziale

Le CFG sono più espressive dei linguaggi regolari e permettono di rappresentare linguaggi con strutture annidate come parentesi bilanciate, che sono tipici nei linguaggi di programmazione.

2.2 Derivazioni e Alberi Sintattici

Una derivazione è una sequenza di sostituzioni che, partendo dalla variabile iniziale S, porta a una stringa di terminali. Ad esempio:

$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000\#111$

Un albero sintattico rappresenta graficamente la struttura di una derivazione, dove:

- La radice è la variabile iniziale
- I nodi interni sono variabili
- Le foglie sono terminali o ε

Suggerimento

Per verificare se una grammatica genera una stringa data, si può procedere in due modi:

- Forward: applicare le regole partendo da S fino a ottenere la stringa
- Backward: costruire tutte le possibili derivazioni della stringa e verificare se una porta a S

2.3 Forma Normale di Chomsky (CNF)

Concetto chiave

Una grammatica context-free è in Forma Normale di Chomsky (CNF) se ogni regola è della forma:

- $A \to BC$ dove B e C sono variabili non iniziali, oppure
- $A \to a$ dove a è un terminale

Inoltre, può esistere la regola $S \to \varepsilon$ per la variabile iniziale S.

Procedimento di risoluzione

Passi per convertire una grammatica in Forma Normale di Chomsky:

1. Aggiungere una nuova variabile iniziale S_0 e la regola $S_0 \to S$
2. Eliminare le ε-regole (tranne $S \to \varepsilon$ se necessario)
3. Eliminare le regole unitarie del tipo $A \to B$
4. Trasformare le regole restanti nelle forme appropriate:
   - Spezzare regole con più di due variabili
   - Sostituire terminali in regole miste con nuove variabili

La CNF è utile per algoritmi come CYK (Cocke-Younger-Kasami) che permettono di determinare in tempo polinomiale se una stringa appartiene a un linguaggio context-free.

3 Equivalenza tra PDA e CFG

3.1 Da CFG a PDA

Dato un linguaggio context-free, esiste sempre un PDA che lo riconosce.

Procedimento di risoluzione

Per convertire una CFG $G = (V, \Sigma, R, S)$ in un PDA P:

1. Costruire un PDA con tre stati: q_0 (iniziale), q_1 (centrale), q_2 (finale)

2. Aggiungere transizioni:

   - $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, S)\}$ (inizializza la pila con S)
   - Per ogni regola $A \to \alpha$ in R, $\delta(q_1, \varepsilon, A) = \{(q_1, \alpha)\}$
   - Per ogni $a \in \Sigma$, $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$ (consuma input e pila)
   - $\delta(q_1, \varepsilon, \$) = \{(q_2, \varepsilon)\}$ (transizione finale)

L'idea è di seguire la leftmost derivation, espandendo le variabili e verificando la corrispondenza con l'input.

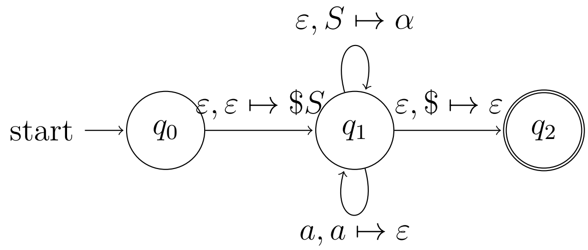


Figure 1: Schema generale di un PDA che simula una CFG

3.2 Da PDA a CFG

Ogni linguaggio riconosciuto da un PDA può essere generato da una CFG.

Procedimento di risoluzione

Per convertire un PDA in una CFG:

1. Creare una variabile A_{pq}^X per ogni coppia di stati p, q e simbolo di pila X

2. A_{pq}^X genera tutte le stringhe che, partendo dallo stato p con X in cima alla pila, portano allo stato q consumando esattamente X dalla pila

3. Costruire regole che simulano le transizioni del PDA

La costruzione è più complessa e richiede l'uso di variabili ausiliarie per gestire tutti i possibili comportamenti del PDA.

4 Proprietà dei Linguaggi Context-Free

4.1 Chiusura per Operazioni

I linguaggi context-free sono chiusi rispetto a:

- Unione
- Concatenazione
- Chiusura di Kleene
- Sostituzione

Ma **non** sono chiusi rispetto a:

- Intersezione
- Complemento

Errore comune

Un errore comune è assumere che l'intersezione di due linguaggi context-free sia ancora context-free. Questo non è vero in generale. Ad esempio, l'intersezione di $\{0^n1^n \mid n \geq 0\}$ e $\{0^n1^{2n} \mid n \geq 0\}$ non è context-free.

4.2 Operazioni Speciali su Linguaggi

Come dimostrato negli esercizi del compitino, esistono operazioni che preservano la classe dei linguaggi regolari o context-free:

Concetto chiave

Se L è context-free, anche i seguenti linguaggi sono context-free:

- $dehash(L) = \{dehash(w) \mid w \in L\}$ dove $dehash(w)$ è la stringa ottenuta eliminando tutti i simboli $\#$ da w.
- $stutter(L) = \{stutter(w) \mid w \in L\}$ con *stutter* definito ricorsivamente.

Per dimostrare queste proprietà di chiusura, si costruisce una nuova grammatica o un nuovo automa che simula il comportamento dell'originale applicando le trasformazioni appropriate.

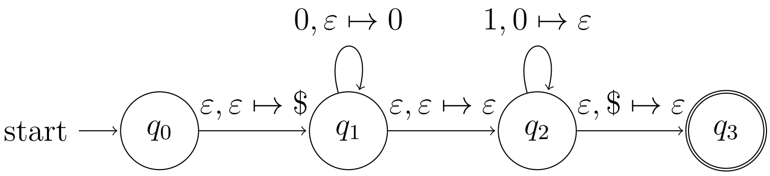
5 Esempi e Strategie per il Compitino

5.1 Esempi di Linguaggi Context-Free e PDA Corrispondenti

5.1.1 Linguaggio $\{0^n1^n \mid n \geq 0\}$

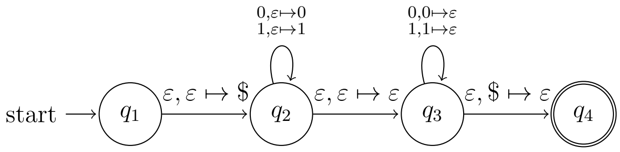
Grammatica: $S \to 0S1 \mid \varepsilon$

PDA:



5.1.2 Linguaggio $\{w \in \{0,1\}^* \mid w = w^R$ e la lunghezza di w è dispari$\}$

PDA: Usa la pila per memorizzare la prima metà della stringa, poi confronta con la seconda metà.



5.2 Strategie per la Risoluzione degli Esercizi

Suggerimento

Tecniche per dimostrare che un linguaggio NON è regolare:

- Pumping Lemma per linguaggi regolari
- Costruzione di controesempi specifici

Tecniche per costruire PDA:

- Identificare la struttura "context-free" del linguaggio (bilanciamento, conteggio, palindromi)
- Usare la pila per memorizzare informazioni che non possono essere gestite da un FSA
- Per il linguaggio $\{0^n 1^n \mid n \geq 0\}$, memorizzare gli 0 nella pila e consumarli con gli 1

Suggerimenti per convertire CFG in CNF:

- Procedere step-by-step seguendo l'algoritmo
- Fare attenzione alle regole con ε e alle regole unitarie
- Verificare che ogni regola finale sia della forma $A \to BC$ o $A \to a$

5.3 Errori Comuni da Evitare

Errore comune

- Confondere le condizioni di accettazione dei PDA (stato finale vs. pila vuota)
- Dimenticare di gestire i casi base nelle dimostrazioni per induzione
- Assumere erroneamente che proprietà dei linguaggi regolari si estendano ai context-free
- Introdurre ambiguità non intenzionali nelle grammatiche

6 Esempi di Esercizi per il Compitino

6.1 NOPREFIX(L)

Dato un linguaggio regolare L, dimostrare che il linguaggio

$$NOPREFIX(L) = \{w \in L \mid \text{ nessun prefisso proprio di } w \text{ appartiene ad } L\}$$

è regolare.

Procedimento di risoluzione

Costruire un DFA A' che accetta $NOPREFIX(L)$ partendo dal DFA A che riconosce L:

1. Aggiungere uno stato "pozzo" non accettante q_s
2. Modificare la funzione di transizione in modo che da qualsiasi stato finale, ogni transizione porti allo stato pozzo
3. Lo stato iniziale e gli stati finali rimangono invariati

Questo garantisce che solo le stringhe in L che non hanno prefissi propri in L vengano accettate da A'.

6.2 Dimostrare che $L_2 = \{uvu \mid u, v \in \{0,1\}^*\}$ non è regolare

Procedimento di risoluzione

Usiamo il Pumping Lemma per dimostrare che il linguaggio non è regolare:

1. Supponiamo per assurdo che L_2 sia regolare con lunghezza di pompaggio k
2. Consideriamo la parola $w = 0^k110^k \in L_2$
3. Qualsiasi decomposizione $w = xyz$ con $|xy| \leq k$ e $|y| > 0$ implicherebbe che y sia composto solo da 0
4. Per il Pumping Lemma, $xy^2z \in L_2$, ma questa parola avrebbe la forma $0^{k+i}110^k$ che non è in L_2
5. Contraddizione, quindi L_2 non è regolare

Concetto chiave

Per dimostrare che i linguaggi context-free sono un sottoinsieme proprio dei linguaggi lineari:

- Ogni linguaggio regolare è generabile da una grammatica lineare (con regole del tipo $A \to aB$ o $A \to a$)
- Esistono linguaggi lineari che non sono regolari, come $\{0^n1^n \mid n \geq 0\}$