Example: recognizing a CFL

Prefix expressions:
\[ E \rightarrow \text{num} \mid + E E \mid \ast E E \]
top-down
Postfix expressions
\[ E \rightarrow \text{num} \mid E E + \mid E E \ast \]
top-down bottom-up

Construction Details

Overall construction
\[ a, a \rightarrow \varepsilon \text{ for term. } a \]
\[ \varepsilon, \varepsilon \rightarrow \$ \]
\[ \varepsilon, \varepsilon \rightarrow \$ \]
\[ \varepsilon, \$ \rightarrow \varepsilon \]
\[ \varepsilon, A \rightarrow w \text{ for rule } A \rightarrow w \]
Implementing production rule \( A \rightarrow u_1 u_2 \ldots u_n \); pop one symbol \((A)\) from stack, push \(n\) symbols
\[ \varepsilon, A \rightarrow u_1 \]
\[ \varepsilon, A \rightarrow u_2 \]
\[ \varepsilon, A \rightarrow u_n \]

From PDA to CFG

Transform PDA to simplified form
\[ \varepsilon, \varepsilon \rightarrow \varepsilon \]
\[ \varepsilon, \Gamma \rightarrow \varepsilon \]
\[ \varepsilon, \$ \rightarrow \varepsilon \]

Each transition either pushes or pops, not both
replace \( q_s \overset{a,b,c}{\rightarrow} q_f \) with \( q_s \overset{a,b,c}{\rightarrow} q_i \overset{c}{\rightarrow} q_f \)
replace \( q_s \overset{a,c}{\rightarrow} q_f \) with \( q_s \overset{a,c,b}{\rightarrow} q_i \overset{c}{\rightarrow} q_f \) for some \( b \in \Gamma \)
PDA to CFG: State Pair to Nonterminal

Recall NFA → GNFA → regex:
express (regex) all strings taking automaton between two states

Idea: characterize all strings that take automaton from
(p, empty stack) to (q, empty stack).

Then: same for arbitrary stack, will not be touched

Two cases:

1. will empty stack somewhere in between (some state r)
   can express as \( A_{pq} \rightarrow A_{pr}A_{rq} \)
2. stack stays nonempty: first: push u, last: pop u (same)
   1. push u on input a: \( a_a / u \rightarrow r \)
   2. \( r \rightarrow s \) without touching stack (\( A_{rs} \))
   3. pop u on input b: \( b_u / \epsilon \rightarrow q \)

Thus we have rule \( A_{pq} \rightarrow A_{pr}A_{rq} \)

PDA to CFG Construction

Consider PDA \( P = (Q, \Sigma, \Gamma, q_0, \{q_{acc}\}) \).

Construct grammar \( G \) with variables \( A_{pq} | p, q \in Q \)

1. Add rule \( A_{pq} \rightarrow a A_{pr}b \) if \( p \xrightarrow{a/\text{push } u} r \) and \( s \xrightarrow{b/\text{pop } u} q \).
2. Add rule \( A_{pq} \rightarrow A_{pr}A_{rq} \) for any \( p, q, r \in Q \).
3. Add rule \( A_{pq} \rightarrow \epsilon \) for any \( p \in Q \).
   (no action = empty production)

Start symbol is \( A_{q_0,q_{acc}} \).

PDA to CFG Proof (1)

If \( A_{pq} \) generates \( x \), then \( x \) can bring \( P \) from state \( p \) with empty stack to state \( q \) with empty stack.

**Induction** by number of steps in derivation \( A_{pq} \Rightarrow x \)

**Base case:** \( k = 1 \) step ⇒ RHS has no variables. Only rules are \( A_{pp} \rightarrow \epsilon \).

Clearly, with no action, \( P \) stays in same state with empty stack.

**Inductive step:** Assume \( A_{pq} \Rightarrow x \) in \( k + 1 \) steps.

Case 1: \( A_{pq} \Rightarrow a A_{rs} b \). Then \( x = ayb \) and \( A_{rs} \Rightarrow y \) in \( k \) steps.

Thus, \( P \) goes on \( y \) from \( r \) to \( s \) on empty stack.

By construction, \( p \xrightarrow{a/\text{push } u} r \) and \( s \xrightarrow{b/\text{pop } u} q \) for some symbol \( u \), thus we go from \( p \) to \( r \) to \( s \) to \( q \) on empty stack.

Case 2: \( A_{pq} \Rightarrow A_{pr} A_{rq} \).

Then \( x = yz \) with \( A_{ps} \Rightarrow y \) and \( A_{qs} \Rightarrow z \) in \( \leq k \) steps.

Thus \( P \) goes from \( p \) to \( r \) by \( y \) on empty stack, then from \( r \) to \( q \) by \( z \) on empty stack.

PDA to CFG Proof (2)

If \( x \) can bring \( P \) from state \( p \) with empty stack to state \( q \) with empty stack, then \( A_{pq} \) generates \( x \).

**Induction** by number of steps in \( P' \) computation \( p \rightarrow p \)

**Base case:** \( k = 0 \). \( P \) does nothing reads no input) ⇒ \( x = \epsilon \)

This is achieved by rule \( A_{pp} \rightarrow \epsilon \)

**Inductive step:** Consider computation of length \( k + 1 > 0 \).

Case 1: stack only empty at start and end ⇒ must start/end with push \( u / \text{pop } u \). Then we have \( p \xrightarrow{a/\text{push } u} r \) and \( s \xrightarrow{b/\text{pop } u} q \), \( x = ayb \).

String \( y \) brings \( P \) from \( r \) to \( s \) without touching stack (not emptied).

\( y \) takes \( k - 1 \) steps, so \( A_{rs} \Rightarrow y \), thus \( A_{pq} \Rightarrow ayb = x \)

Case 2: stack becomes empty at some state \( r \).

\( p \rightarrow r \) and \( r \rightarrow q \) have each \( \leq k \) steps.

Then we have \( A_{ps} \Rightarrow y \) and \( A_{qs} \Rightarrow z \)

Since we have rule \( A_{pq} \rightarrow A_{ps} A_{qs} \) we are done.