IN2080

Context-Free Languages: Pushdown Automata

Einar Broch Johnsen

Universitety of Oslo

15th February 2022



Department of Informatics



University of Oslo

Repetition from Last Week

- Context-free grammars (CFGs)

  ... contain rules of the form $A \to X_1 X_2 \cdots X_n$ where A is a variable and X_i are variables or terminals

  Example:

  $$S \to 0S1$$
  $$S \to \varepsilon$$

  $$L = \{0^n 1^n \mid n \geq 0\}$$

- CFGs generate *context-free languages*
- We have seen that {Regular Languages} $\subsetneq$ {Context-free Languages}

Computational Models

- We have seen that {Regular Languages}$\subsetneq$ {Context-free Languages}
- NFA/DFA/RE/GNFA were all equivalent computational models that describe/accept regular languages
- What computational model accepts context-free languages?

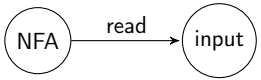
  $\rightarrow$ Pushdown automata!

Pushdown Automata (PDA)

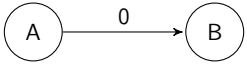
- DFA/NFA/RE/GNFA were all computational models with *finite* memory
- We now add a (very limited) form of infinite memory:
  a stack (LIFO principle: last in, first out)
- So, essentially, PDAs are NFAs with an additional stack

Pushdown Automata (PDA)

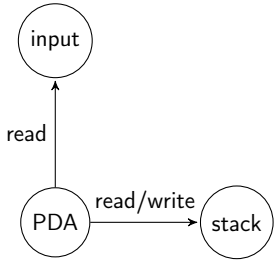
- NFAs transitioned from one state to another according to the current input



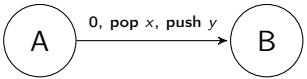
Transitions:



- PDAs can access a stack in addition to the input:

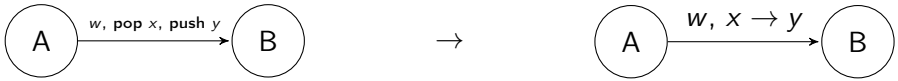


Transitions:



Pushdown Automata (PDA)

For notational ease, we draw these transitions as follows:



- $w, x \to y$: read input w, pop x from stack, push y on to stack
- $w, x \to \varepsilon$: read input w, pop x from stack
- $w, \varepsilon \to y$: read input w, push y on to stack
- $\varepsilon, x \to y$: read no input, perform stack operations as described above (x and/or y may be ε)

Pushdown Automata (PDA)

- So what does the stack let us do?
- Counting?
- Store read input?
- ...more?

Pushdown Automata (PDA)

Consider the grammar:

$$S \to 0S1$$
$$S \to \varepsilon$$

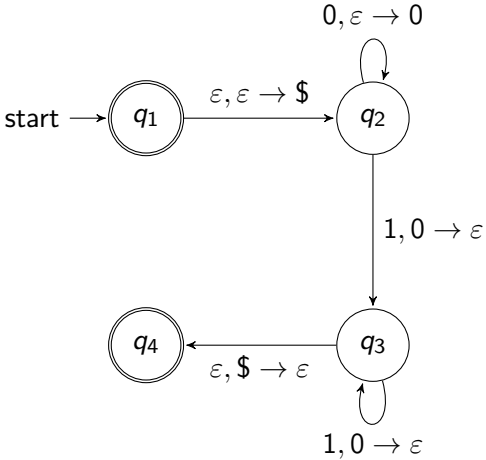
- Generates all 0^*1^* words with equal number of 0s and 1s.
- Can we define a PDA that accepts this language?

- **Vague idea:**
  keep track of number of 0s using the stack, compare with number of 1s

Pushdown Automata (PDA)

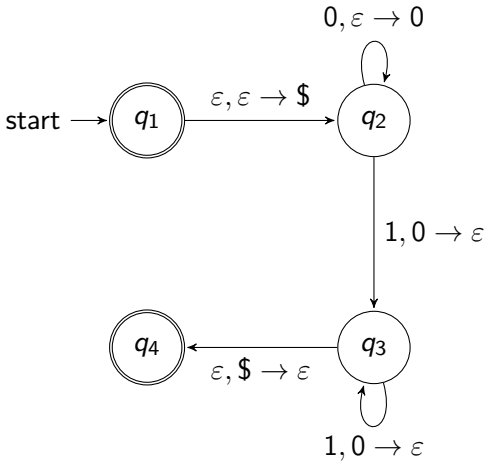
Slightly less vague idea:

- For each 0 read, push a 0 on to the stack.
- Then, for each 1 read, pop a 0 from the stack.
- If the stack is empty after reading the input, accept.
- The only stack operations allowed are: pop/push! How to check if a stack is empty with these operations?
- Before doing anything else, push a special symbol, \$, on to stack. If we ever read this symbol again, the stack is empty!



Pushdown Automata (PDA)

1. $w = 000111$
   Stack: \$ (leftmost item is on top of stack)
2. $w = \overset{\downarrow}{0}00111$
   Stack: 0\$
3. $w = 0\overset{\downarrow}{0}0111$
   Stack: 00\$
4. $w = 00\overset{\downarrow}{0}111$
   Stack: 000\$
5. $w = 000\overset{\downarrow}{1}11$
   Stack: 00\$
6. $w = 0001\overset{\downarrow}{1}1$
   Stack: 0\$
7. $w = 00011\overset{\downarrow}{1}$
   Stack: \$
8. $w = 000111$
   Stack:

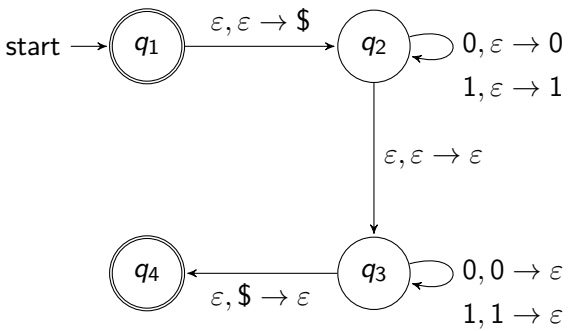


Pushdown Automata (PDA)

- So we can sort of count (compare numbers of 0's and 1's)
- What about tracking input?
- Consider language $L_2 = \{ww^R \mid w \in \{0,1\}^*\}$
- $01011010 \in L_2$, $0110 \in L_2$. L_2 is the language of even length $0,1$ palindromes
- How to design a PDA that accepts L_2?

Pushdown Automata (PDA)

- **Intuition:** to check if input is of the form ww^R, push input on the stack, then at some point compare the stack items to input and, if equal, pop.
- First, add special symbol \$ to stack.
- If we read a 0 push a 0, if we read a 1 push a 1.
- Nondeterministically guess when w has been read and w^R begins
- Then if we read a 0 pop 0, if we read a 1 pop 1.
- If the stack is empty, accept.



PDA

Definition (PDA)

A PDA is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ, F are finite states and

1. Q is a set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accepting states.

This lecture

We want to show:

Theorem

A language is context-free if and only if some pushdown automaton recognizes it.

Step by step.... first we show:

Lemma

If a language is context-free then there exists a pushdown automaton that recognizes it.

In other words: given a CFG G, construct a PDA that recognizes $L(G)$!

CFG→PDA

Lemma

If a language is context-free then there exists a pushdown automaton that recognizes it.

Proof idea:

- Construct a PDA that for input w determines if it has a valid derivation.
- Use stack to store *how far we've gotten in our derivation*, i.e., the intermediate strings

$$S \rightsquigarrow aSb \rightsquigarrow aaSbb \rightsquigarrow aa\varepsilon bb \rightsquigarrow aabb$$

- PDA needs to find variables in order to substitute them (recall: can only access top of stack) so if terminal is on top of stack, we compare to input
- If top of stack is variable, we pop the variable and replace with the righthand side of a rule

CFG→PDA

Lemma

If a language is context-free then there exists a pushdown automaton that recognizes it.

Proof idea: Intuitively, the PDA works as follows:

- Place special symbol \$ on the stack
- Repeat the following:
- If the top of stack is a variable A, nondeterministically choose a rule $A \to w$, pop A and push w
- If top of stack is a terminal a, compare with next input symbol.
  If they match, pop a and repeat.
  If they do not match, reject this branch.
- If top of stack is \$, enter accept state.

CFG→PDA

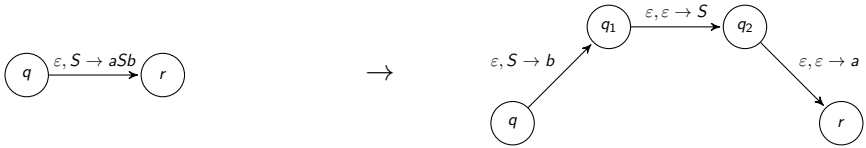
- Place special symbol \$ on the stack, push start variable S on to stack.
- Repeat the following:
- If the top of stack is a variable A, nondeterministically choose a rule $A \to w$, pop A and push w
- If top of stack is a terminal a, compare with next input symbol.
  If they match, pop a and repeat.
  If they do not match, reject this branch.
- If top of stack is \$, enter accept state.

$S \rightsquigarrow aSb \rightsquigarrow aaSbb \rightsquigarrow aa\varepsilon bb \rightsquigarrow aabb$

1. input: $aabb$
   Stack: \$
2. input: $aabb$
   Stack: S\$
3. input: $\overset{\downarrow}{a}abb$
   Stack: aSb\$
4. input: $a\overset{\downarrow}{a}bb$
   Stack: Sb\$
5. input: $a\overset{\downarrow}{a}bb$
   Stack: $aSbb$\$
6. input: $aabb$
   Stack: Sbb\$
7. input: $aa\overset{\downarrow}{b}b$
   Stack: bb\$
8. compare all terminals in stack to input

Note how in the previous slide we added multiple symbols the stack in one step!

We popped S, and pushed aSb.



We will use this shorthand notation during the proof.

CFG→PDA: Proof

Proof:

- Given CFG $G = (V, \Sigma, R, S)$, we want to construct a PDA $(Q, \Sigma, \Gamma, \delta, q_0, F)$ that accepts $L(G)$.
- Let $Q = \{q_0, q_{\text{loop}}, q_{\text{accept}}\} \cup E$, where E is the set of help states needed for our shorthand notation (previous slide).
- Before anything else, push \$ and the start variable on to stack: $\delta(q_0, \varepsilon, \varepsilon) = \{(q_{\text{loop}}, S\$)\}$.

CFG→PDA: Proof

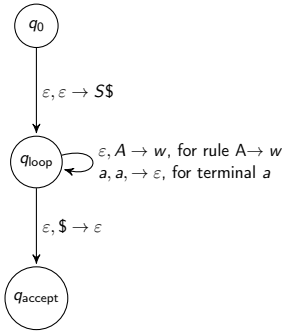
Recall intuition:

- If the top of stack is a variable A, nondeterministically choose a rule $A \to w$, pop A and push w
- If top of stack is a terminal a, compare with next input symbol. If they match, repeat. If they do not, reject this branch.
- If top of stack is \$, enter accept state.

- $\delta(q_{\text{loop}}, \varepsilon, A) = \{(q_{\text{loop}}, w) \mid A \to w \text{ is a rule in } R\}$
- $\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \varepsilon)\}$
- $\delta(q_{\text{loop}}, \varepsilon, \$) = \{(q_{\text{accept}}, \varepsilon)\}$

CFG→ PDA

State diagram (without help states from shorthand notation):



PDA→CFG

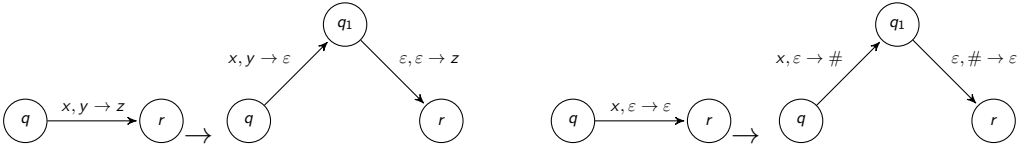
The other direction is much more involved....

For convenience, we assume our PDA has the following properties

- It has a single accept state q_{accept}
  (create a new accepting state with ε transitions from all previous accept states)
- It empties its stack before accepting
  (make a loop on q_{accept} that empties the stack without reading an input)
- Each transition either pushes a symbol or pops a symbol,
  but does not do both at the same time.

PDA→CFG

- Each transition either pushes a symbol or pops a symbol, but does not do both at the same time.



PDA→CFG

Idea:

Given a PDA $(Q, \Sigma, \Gamma, \delta, q_0, F)$, create a grammar where variables are A_{pq} for states $p, q \in Q$.

The variable A_{pq} will generate all strings that take P from p to q *with empty stacks*.

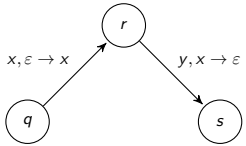
...what does that mean?

$\rightsquigarrow$ input w takes P from p to q with empty stacks if:

- starting in p with an empty stack, after parsing the input s the PDA P ends in state q.
- when P arrives at q, the stack is once again empty

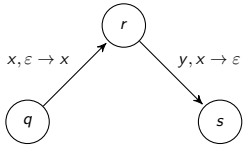
Then $A_{q_0 q_{\text{accept}}}$ will generate precisely the words the *PDA* accepts!

Example:



- string x does not take P from q to r with empty stacks,
  since the stack ends with an extra x
- string xy takes P from q to s with empty stacks,
  since the x that was pushed gets popped in the last transition to s.

Observation:



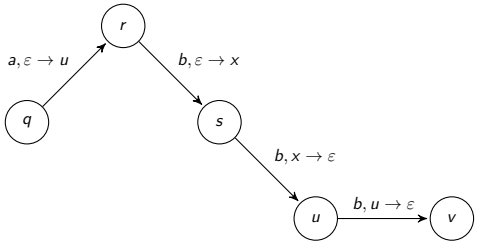
If P takes a string from p to q with empty stacks, then

- the first transition *must* be a push
- the last transition *must* be a pop

We have **two possibilities:**

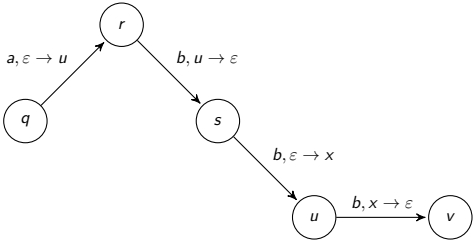
- Possibility 1: these transitions push/pop the same symbol
- Possibility 2: these transitions push/pop different symbols

Possibility 1 (example):



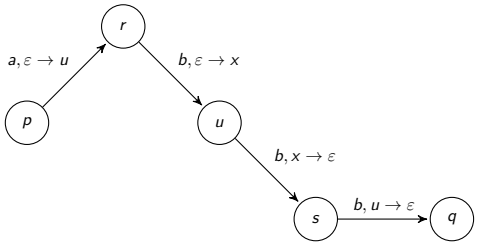
Input *abbb* takes P from q to v with empty stacks. Here the stack is never emptied in between.

Possibility 2 (example):



Input *abbb* takes P from q to v with empty stacks. Here the stack is emptied at state s.

Stack not emptied in between:

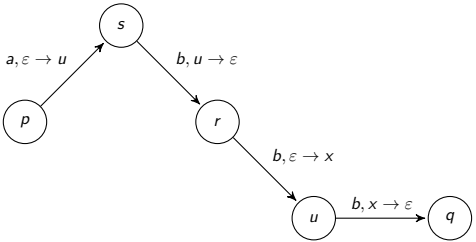


To address this case, add

$$A_{pq} \to aA_{rs}b$$

for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$
such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε).

Stack is emptied in between:



To address this case, add rule

$$A_{pq} \to A_{pr} A_{rq}$$

for all $p, q, r \in Q$

PDA→CFG

Finally, add rules $A_{pp} \to \varepsilon$ for all $p \in Q$.

Summary:

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- Start variable: $A_{q_0 q_{\text{accept}}}$

We are done if we can show:

Lemma

If A_{pq} generates x, then x brings P from p to q with empty stacks.

Lemma

If x can bring P from p to q with empty stacks, then A_{pq} generates x.

PDA→CFG

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- start variable: $A_{q_0 q_{\mathbf{accept}}}$

Lemma

If A_{pq} generates x then x brings P from p to q with empty stacks.

Proof by induction over number of derivation steps:

Base. Derivation has 1 step.

- A derivation with 1 step uses a rule with no variables on the righthand side.
- Only such rules are $A_{pp} \to \varepsilon...$ and ε takes P from p to p with empty stacks.✓

PDA→CFG

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- start variable: $A_{q_0 q_{\text{accept}}}$

Lemma

If A_{pq} generates x then x brings P from p to q with empty stacks.

Proof by induction over number of derivation steps:

Step. IH: assume true for derivations of length at most $k \geq 1$.

- Assume A_{pq} derives x in $k+1$ steps.
  Then the first rule application is either $A_{pq} \to aA_{rs}b$ or $A_{pq} \to A_{pr}A_{rq}$
- If $A_{pq} \to aA_{rs}b$ was used, let y be generated by A_{rs} such that $x = ayb$.
- A_{rs} derives y in at most k steps.
  By the induction hypothesis, y takes P from r to s with empty stacks.
- $\to a$ brings P from p to r by pushing u, y brings P from r to s on empty stacks, b brings P from s to q by popping u. ✓

PDA→CFG

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- start variable: $A_{q_0 q_{\mathbf{accept}}}$

Lemma

If A_{pq} generates x then x brings P from p to q with empty stacks.

Proof by induction over number of derivation steps:

Step. IH: assume true for derivations of length at most $k \geq 1$.

- Assume A_{pq} derives x in $k+1$ steps.
  Then the first rule application is either $A_{pq} \to aA_{rs}b$ or $A_{pq} \to A_{pr}A_{rq}$
- If $A_{pq} \to A_{pr}A_{rq}$ was used,
  let $x = yz$ be such that y is generated by A_{pr} and z is generated by A_{rq}.
- A_{pr} (resp. A_{rq}) generates y (resp. z) in at most k steps.
- By the induction hypothesis, y takes P from p to r with empty stacks,
  and z takes P from r to q with empty stacks. ✓

PDA→CFG

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- start variable: $A_{q_0 q_{\text{accept}}}$

Lemma

If x can bring P from p to q with empty stacks, A_{pq} generates x.

Proof by induction over computation steps:

Base. Computation has 0 steps.

- A computation with 0 steps stays in the same state p and cannot read any input. Since $A_{pp} \to \varepsilon$ is a rule in G, the base case is proved. ✓

PDA→CFG

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- start variable: $A_{q_0 q_{\text{accept}}}$

Lemma

If x can bring P from p to q with empty stacks, A_{pq} generates x.

Proof by induction over computation steps:

Step. IH: Assume true for computations of length at most k.

- Assume P has a computation where input x takes P from state p to q with empty stacks that takes $k + 1$ steps.
- Two cases: (case 1) either the stack is only empty at the beginning and end, or (case 2) it is emptied in between.

PDA→CFG

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- start variable: $A_{q_0 q_{\text{accept}}}$

Lemma

If x can bring P from p to q with empty stacks, A_{pq} generates x.

- **Case 1:** stack is empty only at beginning and end
- Symbol u pushed in the first step must be the same symbol popped in the last step
- Let a denote the input read in step 1, b the input in the last step, r the state after the first move, s the state before the last move.
- Then $A_{pq} \to aA_{rs}b$ is in G.
- Let $x = ayb$. Then y takes P from r to s with empty stacks in $k - 1$ steps
- Induction hypothesis $\Rightarrow$ A_{rs} generates y.
- $\Rightarrow$ A_{pq} generates x.

PDA→CFG

- $A_{pq} \to aA_{rs}b$ for every $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\varepsilon$ such that $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),
- $A_{pq} \to A_{pr}A_{rq}$ for every $p, q, r \in Q$,
- $A_{pp} \to \varepsilon$ for all $p \in Q$
- start variable: $A_{q_0 q_{\text{accept}}}$

Lemma

If x can bring P from p to q with empty stacks, A_{pq} generates x.

- **Case 2:** stack is empty inbetween
- Let r be a state where the stack empties other than at the beginning and end of the computation of x.
- Then the computations from p to r (let us call its parsed input y) and from r to q (let us call its parsed input z) contain at most k steps.
- Induction hypothesis $\Rightarrow$ A_{pr} (resp. A_{rq}) generates y (resp. z)
- Then A_{pq} generates $x = yz$. □

Summary

- PDA: NFA with stack
- Remember to start by pushing special symbol \$ on the stack
- Lemma: If a language is context-free, then there is a PDA that recognizes it
- Lemma: If a language is recognized by a PDA, then it is context-free
  - If A_{pq} generates x, then x brings P from p to q with empty stacks.
  - If x can bring P from p to q with empty stacks, then A_{pq} generates x.
- Theorem: A language is context-free if and only if some PDA recognizes it