CSE2315 Slides week 3

Matthijs Spaan Stefan Hugtenburg

Algorithmics group

Delft University of Technology

4 March 2020

This lecture

- Grammars, derivations, parse trees
- Context-free Grammars (CFGs) and their formal definition
- Language of a grammar
- Designing CFGs
- Ambiguity
- Closure properties of context-free languages
- Chomsky Normal Form (CNF)
- Simplifying CFGs: elimination of ε and unit productions
- Push down automata (PDAs) and their formal definition
- Construction of a PDA from a CFG
- Construction of a CFG from a PDA (informally)
- Pumping lemma for context-free languages (informally)

Grammars

- A grammar describes the formal aspects of (natural) languages, which is called their syntax.
- Besides this there is also semantics, the study of the meaning of (natural) languages.
- A grammar describes what are correct “sentences” in a language. Here, a “sentence” is a program (in some programming language) or a logical formula, or a word from a language.

Applications of grammars

Originally: Linguistics and natural language (Noam Chomsky)

In computer science, they are used (for instance)

- to define programming languages,
- in compiler construction: recognizing the structure (parsing) of a program,
- to describe document formats, such as XML, and
- to describe communication protocols.

Grammars

(See also video Intro to Context-Free Grammars and Languages)

A grammar is a 4-tuple $G = (V, \Sigma, R, S)$, where:

- V is a finite set of variables,
- Σ is a finite set of terminal symbols,
- S is the starting variable ($S \in V$), and
- R is a finite set of rules of the form

$$x \to y,$$

with $x \in (V \cup \Sigma)^+$ and $y \in (V \cup \Sigma)^*$.

Context-free grammars and context-free languages

Definition: A grammar $G = (V, \Sigma, R, S)$ is context free iff all rules in R are of the following form:

$$A \to x,$$

where $A \in V$ and $x \in (V \cup \Sigma)^*$.

Definition: A language L is context free iff there exists a context-free grammar G such that $L = L(G)$.

Example

Let $G = (V, \Sigma, R, S)$ with $V = \{S, A, B, C, D\}$, $\Sigma = \{a, b, c, d\}$ and R the following set of rules:

$$
\begin{aligned}
S &\rightarrow AB \mid DC \\
A &\rightarrow \varepsilon \mid Aa \\
B &\rightarrow \varepsilon \mid bBc \\
C &\rightarrow \varepsilon \mid Cc \\
D &\rightarrow \varepsilon \mid aDb
\end{aligned}
$$

Notation: "$X \rightarrow x \mid y$" represents two rules, $X \rightarrow x$ and $X \rightarrow y$.

Derivations in grammars

If $x \to y$ is a rule, then from the word $w = uxv$ the word $z = uyv$ can be derived.

Notation: $w \Rightarrow z$ if z can be derived (immediately) from w.

Notation: If $w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$, this is written as $w_1 \overset{*}{\Rightarrow} w_n$.

Note: The $*$ indicates 0 or more derivation steps are used, so we also have $w \overset{*}{\Rightarrow} w$.

Example

$$
\begin{aligned}
S &\rightarrow AB \mid DC \\
A &\rightarrow \varepsilon \mid Aa \\
B &\rightarrow \varepsilon \mid bBc \\
C &\rightarrow \varepsilon \mid Cc \\
D &\rightarrow \varepsilon \mid aDb
\end{aligned}
$$

Then:

$$S \Rightarrow D\underline{C} \Rightarrow \underline{D}Cc \Rightarrow a\underline{D}bCc \Rightarrow ab\underline{C}c \Rightarrow abc$$

and so $S \overset{*}{\Rightarrow} abc$.

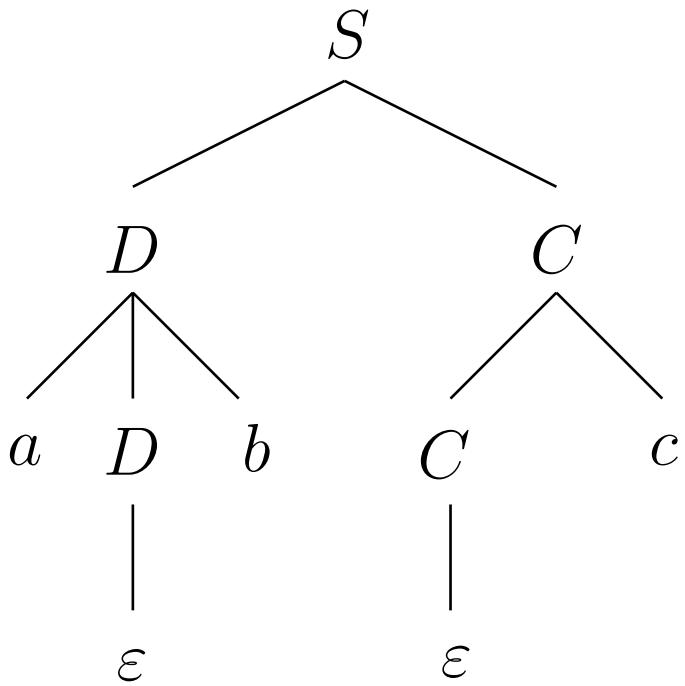
Parse trees

Each derivation has a matching parse tree B: the root of B is S and for every application of a rule $A \to x_1 x_2 \ldots x_n$, with $A \in V$ and $x_1, x_2, \ldots, x_n \in V \cup \Sigma$ there is a node in B with (only) the children $x_1, x_2, \ldots, x_n$, from left to right.

The word produced through a parse tree is the word consisting of all the symbols that "hang" from the tree as leaves, from left to right.

Example parse tree

Parse tree



belonging to the derivation

$$S \Rightarrow D\underline{C} \Rightarrow \underline{D}Cc \Rightarrow a\underline{D}bCc \Rightarrow ab\underline{C}c \Rightarrow abc$$

Exercise

Suppose we have a CFG with the following rules:

$$\begin{aligned} S &\to AB \mid \varepsilon, \\ A &\to aB, \\ B &\to Sb. \end{aligned}$$

Give a derivation and a parse tree for the word $aabbbb$.

Solution

$$
\begin{aligned}
S &\Rightarrow \underline{A}B \\
&\Rightarrow a\underline{B}B \\
&\Rightarrow aSb\underline{B} \\
&\Rightarrow a\underline{S}bSb \\
&\Rightarrow aABb\underline{S}b \\
&\Rightarrow a\underline{A}Bbb \\
&\Rightarrow aa\underline{B}Bbb \\
&\Rightarrow aaSb\underline{B}bb \\
&\Rightarrow aa\underline{S}bSbbb \\
&\Rightarrow aab\underline{S}bbb \\
&\Rightarrow aabbbb
\end{aligned}
$$

Languages of grammars

Let $G = (V, \Sigma, R, S)$ be a grammar. Then the language $L(G)$ generated by G is given by:

$$\boxed{L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}}$$

Two grammars G_1 and G_2 are equivalent iff $L(G_1) = L(G_2)$.

Exercise

Design a context-free grammar for the (nonregular!) language

$$L = \{a^n b^n \mid n \geq 0\}$$

Solution

$$S \rightarrow \varepsilon \mid aSb$$

(See also video Context-Free Grammar Example)

Exercise: the language of palindromes

A palindrome is a word that stays the same when read left to right and right to left.

Examples: refer, noon, racecar.

Formally: Let Σ be an alphabet. Then

$$L_{pal} = \{w \in \Sigma^* \mid w = w^R\}.$$

Exercise:

1. Give an inductive definition of L_{pal}.
2. Is L_{pal} a regular language?

Solution

1. Define L_{pal} inductively such that:
   - Basis: $\varepsilon \in L_{pal}$ and $\Sigma \subseteq L_{pal}$.
   - Inductive clause: For all $a \in \Sigma$: if $w \in L_{pal}$ then $awa \in L_{pal}$.
   - Exclusion: No other string is a word in L_{pal}.
2. No; use the pumping lemma.

Exercise

Give a context-free grammar for the language L_{pal} of palindromes over the alphabet $\{a, b\}$.

(see Sipser, Section 2.1, Designing Context-Free Grammars)

Solution

Follow the inductive definition:

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$$

Useful exercise: Give an inductive proof that this grammar does indeed generate L_{pal}. That is, prove that $L(G) = L_{pal}$, where G represents the above grammar.

Hint: Proving set equality entails proving $\subseteq$ and $\supseteq$. Prove $\subseteq$ by induction over the length of the derivation. Prove $\supseteq$ by induction over the construction of a palindrome (use the inductive definition of L_{pal}).

Exercise

Give a context-free grammar for the following language:

$$L = \{a^k b^m c^n \mid k = m \text{ or } m = n\}.$$

For instance: $aaabbbcc \in L$ en $aabbbccc \in L$ but $aabbbcc \notin L$.

A solution

$$
\begin{aligned}
S &\rightarrow AB_1 \mid B_2C \\
A &\rightarrow \varepsilon \mid Aa \\
C &\rightarrow \varepsilon \mid Cc \\
B_1 &\rightarrow bB_1c \mid \varepsilon \\
B_2 &\rightarrow aB_2b \mid \varepsilon
\end{aligned}
$$

Leftmost and rightmost derivations

Definition: A derivation is a leftmost derivation if in every step of the derivation, each time the leftmost variable is rewritten.
If in each step the rightmost variable is rewritten, we call it a rightmost derivation.

Theorem: Every word generated by a CFG has a leftmost derivation.

Note: This does not hold in general for (context-sensitive) grammars!

Parsing and ambiguity

(See also video Kinds of Context-Free Languages)

The sentence:

> “They saw the girl with binoculars.”

has two different, conventional readings. Besides this there are also two “strange” readings.

Check this!

Ambiguity

Definition: A CFG G is called ambiguous if there exists a $w \in L(G)$ with at least two different leftmost or two different rightmost derivations in G.
Or, equivalently, a G is ambiguous if there exists a $w \in L(G)$ with at least two different parse trees.

Important: Also in non-ambiguous grammars a word can have different parse trees, but not different leftmost derivations.

Example ambiguity (1)

Let $G = (V, \Sigma, R, E)$ be a CFG with $V = \{E\}$ and rules:

$$E \quad \rightarrow \quad a \mid b \mid c \mid (E) \mid E + E \mid E * E$$

The word $a + b * c \in L(G)$ has two different leftmost derivations:

$$\underline{E} \Rightarrow \underline{E} + E \Rightarrow a + \underline{E} \Rightarrow a + \underline{E} * E \Rightarrow a + b * \underline{E} \Rightarrow a + b * c,$$

$$\underline{E} \Rightarrow \underline{E} * E \Rightarrow \underline{E} + E * E \Rightarrow a + \underline{E} * E \Rightarrow a + b * \underline{E} \Rightarrow a + b * c.$$

Example ambiguity (2)

Equivalent unambiguous CFG $G' = (V', \Sigma, R', E)$ with $V' = \{E, T, F, I\}$ and R':

$$\begin{aligned} E &\rightarrow T \mid E + T \\ T &\rightarrow F \mid T * F \\ F &\rightarrow I \mid (E) \\ I &\rightarrow a \mid b \mid c \end{aligned}$$

Make a leftmost derivation for $a + b * c$.

Exercise

Let G be a CFG with the following rules:

$$\begin{aligned} S &\rightarrow AB \mid aaB \\ A &\rightarrow Aa \mid a \\ B &\rightarrow b \end{aligned}$$

Show that G is ambiguous and construct an equivalent unambiguous grammar.

Solution

The word aab can be derived in two different ways:

1. $S \Rightarrow AB \Rightarrow AaB \Rightarrow aaB \Rightarrow aab$
2. $S \Rightarrow aaB \Rightarrow aab$

An unambiguous grammar could be:

$$S \rightarrow aS \mid ab$$

An alternative solution:

$$\begin{aligned} S &\rightarrow aB \\ B &\rightarrow aB \mid b \end{aligned}$$

Inherent ambiguity

Definition: If every grammar for a context-free language L is ambiguous, then L is said to be inherently ambiguous.

Example: The language $\{a^k b^m c^n \mid k = m \text{ or } m = n\}$ is inherently ambiguous. However, proving this is complicated.

Exercise: closure properties

Question: Is the class of context-free languages closed under union, concatenation and star?

Solution

Yes! (See also video Facts About Context-Free Languages)

Proof: (union)
Let L_1 and L_2 be context-free languages. Then there exist CFGs $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ such that $L(G_1) = L_1$ and $L(G_2) = L_2$.
Suppose $V_1 \cap V_2 = \emptyset$ and $S \notin V_1 \cup V_2$ (rename variables where necessary).

Then $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R, S)$ is a CFG that generates $L_1 \cup L_2$, where

$$R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}.$$

■

Chomsky Normal Form

(See also video Chomsky Normal Form)

Definition: A CFG $G = (V, \Sigma, R, S)$ is in Chomsky normal form (CNF) iff every rule in R is of one of the following forms:

1. $S \to \varepsilon$
2. $A \to a$
3. $A \to BC$

where $a \in \Sigma$ and $A, B, C \in V$ with B and C not equal to S.

Problematic rules:

1. Rules of the form $A \to xSy$
2. ε rules (of the form $A \to \varepsilon$)
3. Unit rules (of the form $A \to B$)
4. Rules of the form $A \to x$ with $|x| > 2$

Chomsky normal form

Theorem: Every CFG G is equivalent to some CFG in Chomsky normal form.

Proof idea: Let $G = (V, \Sigma, R, S)$ be a CFG. Transform G into an equivalent CFG in Chomsky normal form by performing the following steps:

1. Add a new start variable S_0 to V and the rule $S_0 \to S$ to R.
2. Eliminate all ε rules except possibly $S_0 \to \varepsilon$.
3. Eliminate all unit rules.
4. Convert all rules to the right form.

The resulting CFG is in Chomsky normal form and is also equivalent to G.

Note: First eliminate ε rules and only then the unit rules!

Algorithm for eliminating ε rules

While there are rules of the form $A \to \varepsilon$ (with A not equal to the start variable):

- Take such a rule $A \to \varepsilon$ and remove it from R.
- For all rules of the form $B \to xAy$: add the rule $B \to xy$ to R.
- For all rules of the form $B \to xAyAz$: add the rules $B \to xyAz$, $B \to xAyz$ and $B \to xyz$ to R.
- ...
- If while doing this the rule $B \to \varepsilon$ would be created, only add it if it was not removed in an earlier stage.

Exercise

Eliminate the ε rules from the following context-free grammar:

$$
\begin{aligned}
S &\rightarrow Abb \mid B \\
A &\rightarrow \varepsilon \mid BA \\
B &\rightarrow A \mid AcB
\end{aligned}
$$

Solution

$$
\begin{aligned}
S &\rightarrow Abb \mid B \mid bb \mid \varepsilon \\
A &\rightarrow BA \mid B \mid A \\
B &\rightarrow A \mid AcB \mid cB \mid Ac \mid c
\end{aligned}
$$

Unit rules

A unit rule is a rule of the form $A \to B$, where A and B are variables ($A, B \in V$).

Algorithm for eliminating unit rules

While there are unit productions of the form $A \to B$ (with A and B being variables):

- Take a unit rule $A \to B$ and remove it.
- For all rules of the form $B \to x$, add the rule $A \to x$ to R, unless this is a unit rule that was previously removed.

Note: Rules of the form $A \to A$ can be removed immediately!

Exercise

Eliminate the unit productions from the following CFG:

$$
\begin{aligned}
S &\rightarrow Abb \mid B \mid bb \mid \varepsilon \\
A &\rightarrow BA \mid B \mid A \\
B &\rightarrow A \mid AcB \mid cB \mid Ac \mid c
\end{aligned}
$$

Solution

$$
\begin{aligned}
S &\rightarrow Abb \mid bb \mid BA \mid AcB \mid cB \mid Ac \mid c \mid \varepsilon \\
A &\rightarrow BA \mid AcB \mid cB \mid Ac \mid c \\
B &\rightarrow AcB \mid cB \mid Ac \mid c \mid BA
\end{aligned}
$$

Algorithm for putting rules in the right form

Let $G = (V, \Sigma, R, S)$ be a CFG without ε or unit rules (with the exception of $S \to \varepsilon$).
Construct $G' = (V', \Sigma, R', S)$ as follows:

1. For every symbol $a \in \Sigma$, add a variable U_a to V.
2. For every $a \in \Sigma$, add the rule $U_a \to a$ to R.
3. For $n \geq 3$, replace every rule $A_0 \to x_1 \cdots x_n$ in R with the rules $A_0 \to x_1 A_1$, $A_1 \to x_2 A_2$, $\ldots$, $A_{n-2} \to x_{n-1} x_n$, with all $x_i \in V \cup \Sigma$ and $A_1, \ldots, A_{n-2}$ being new variables.
4. In every rule $A \to x$ in R with $|x| = 2$, replace every occurrence of a terminal symbol $a \in \Sigma$ in x by U_a.

Example

Put the following rule into the right form:

$$S \rightarrow Abb$$

Result:

$$\begin{aligned} S &\rightarrow AX_1 \\ X_1 &\rightarrow U_b U_b \\ U_b &\rightarrow b \end{aligned}$$

Exercise

Suppose we have the following grammar G:

$$\begin{aligned} S &\to SS \mid BA \\ A &\to \varepsilon \mid a \mid Ab \\ B &\to \varepsilon \mid b \end{aligned}$$

Give a CFG in Chomsky normal form equivalent to G.

Solution

A possible equivalent CFG in CFN is given by $G' = \{V', \Sigma, R', S_0\}$ with $V' = \{S_0, S, A, B\}$ and where R' consists of the following rules:

$$
\begin{aligned}
S_0 &\rightarrow \varepsilon \mid SS \mid BA \mid a \mid AB \mid b \\
S &\rightarrow SS \mid BA \mid a \mid AB \mid b \\
A &\rightarrow a \mid AB \mid b \\
B &\rightarrow b
\end{aligned}
$$

G' is in Chomsky normal form and is equivalent to the CFG G.

Note: for this exercise, other than the start variable S_0, no new variables were needed to put rules in the right form.

Pushdown automata

(See also video PDAs: Pushdown Automata)

Pushdown automata are a machine model for context-free languages.

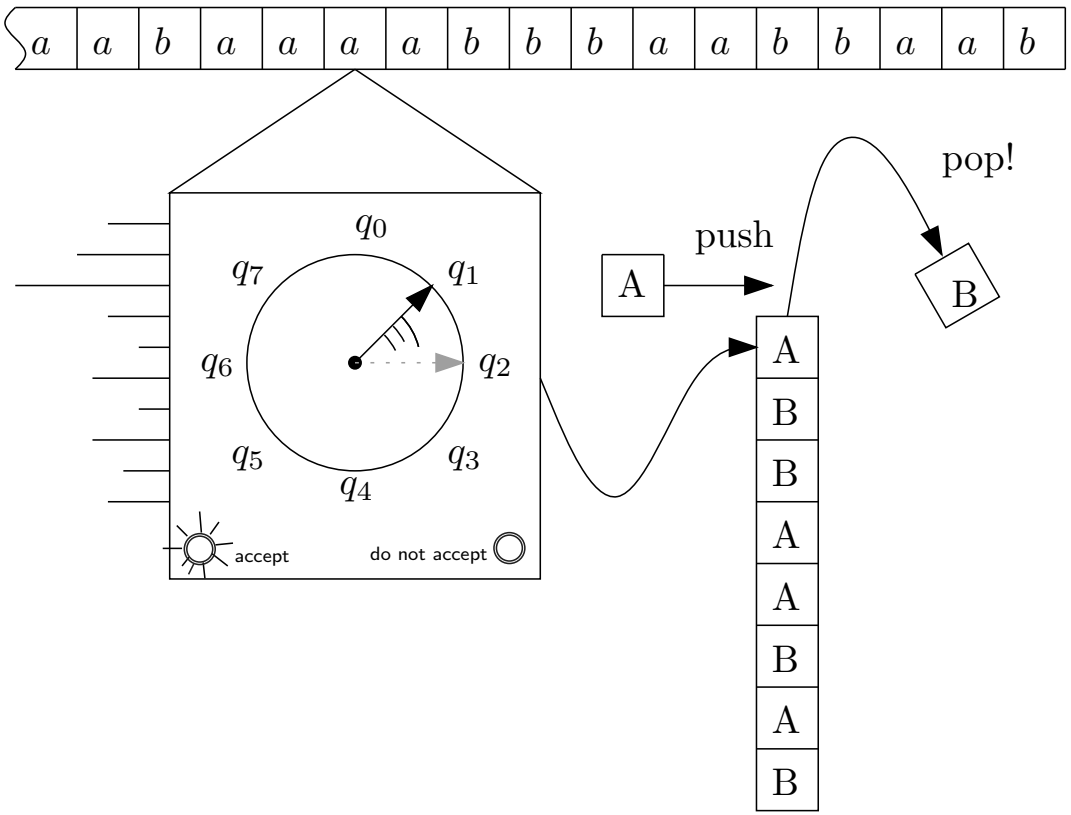
Intuitively: A pushdown automaton is an NFA with a stack for memory.

Stack:

- Stack elements on top of each other.
- Stack operations: pop and push.
- Last-in-First-out (lifo) principle.

Note: A stack is memory with a limited form of access. Other forms of memory are quite possible.

Pushdown automata



Pushdown automata: formal definition

Definition: A pushdown automaton (PDA) is a 6-tuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where:

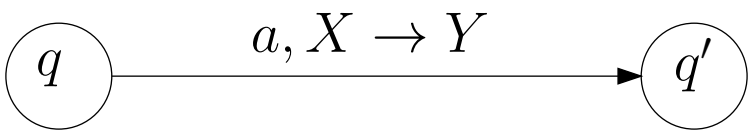
- Q a finite set of states,
- Σ and Γ are the input and stack alphabet, respectively (both of them finite),
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \mathscr{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ is the transition function,
- q_0 is the start state,
- $F \subseteq Q$ is set of accept states.

Transition function of a pushdown automaton

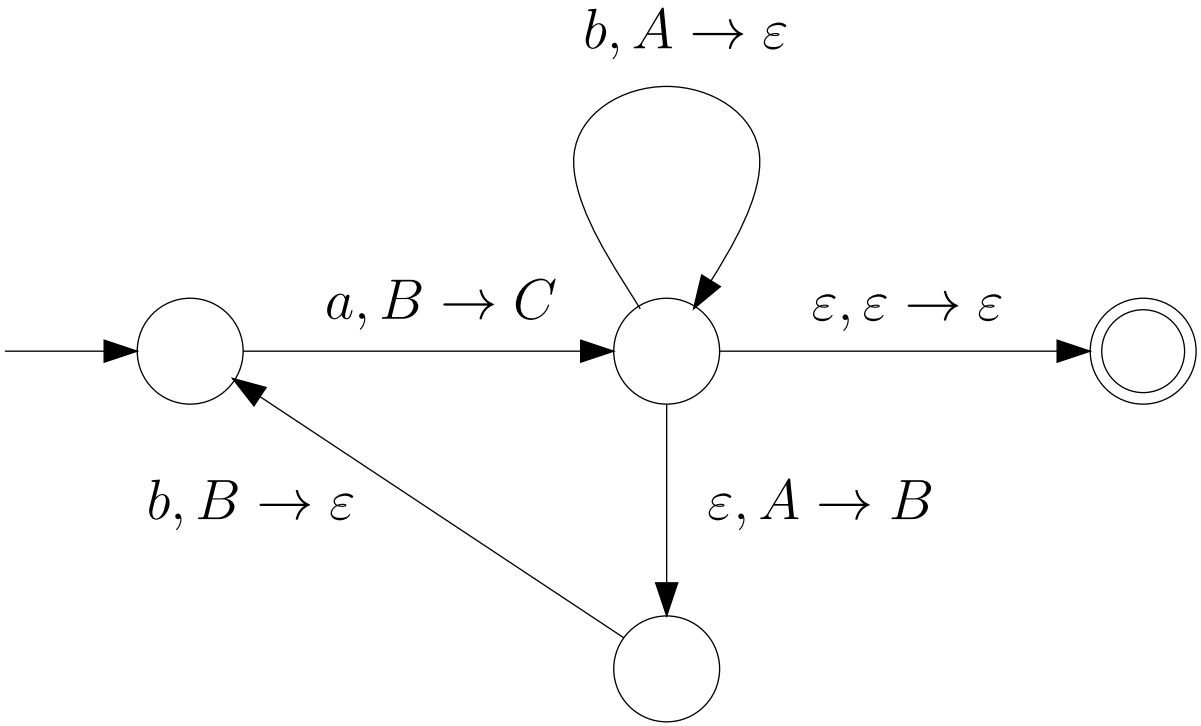
Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ with $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ and $X, Y \in \Gamma \cup \{\varepsilon\}$:

If $(q', Y) \in \delta(q, a, X)$, then P can perform the following transition:

1. P in state q reads a as the first symbol of the word to be processed,
2. P in state q reads X as the top symbol on the stack,
3. P pops X (from the stack),
4. P pushes Y (onto the stack),
5. P transitions to state q',



Transition graphs for pushdown automata



State descriptions

A state description is a triple $(q, w, Z_0 \cdots Z_n)$ with

- q a state,
- $w \in \Sigma^*$ and
- $Z_0 \cdots Z_n$ the stack.

Transition between state descriptions:

$$(q_1, aw, XZ_0 \cdots Z_n) \vdash_P (q_2, w, YZ_0 \cdots Z_n)$$

iff

$$(q_2, Y) \in \delta(q_1, a, X).$$

Notation: $\vdash^*$ for zero or more steps of $\vdash$.

The language of a pushdown automaton

Definition: The language $L(P)$ accepted by a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is defined as:

$$w \in L(P)$$

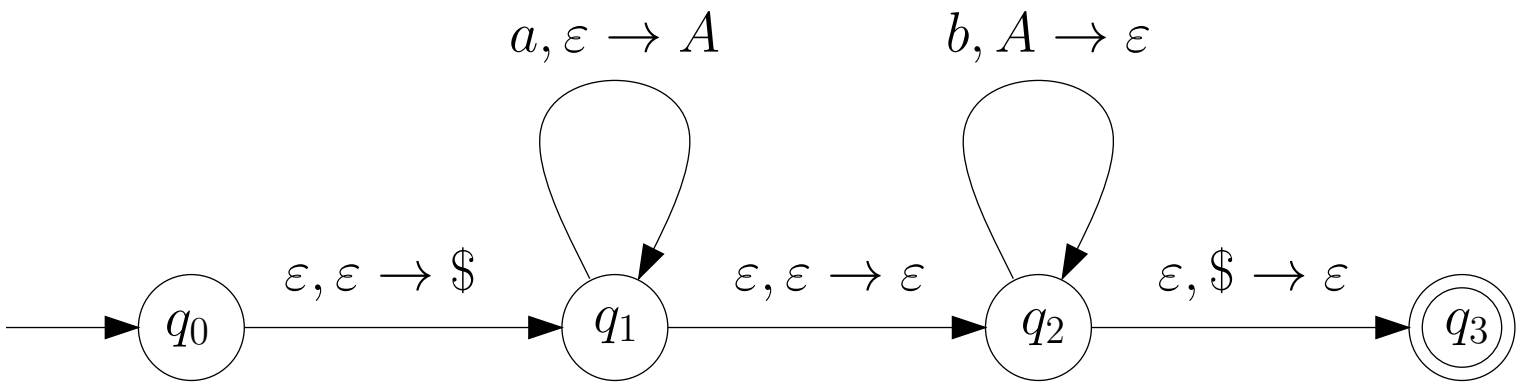
iff

there exist $q \in F$ and $u \in \Gamma^*$ with $(q_0, w, \varepsilon) \vdash_P^* (q, \varepsilon, u)$.

Note: When accepting a word, the stack does not have to be empty. Of course the whole word must have been read.

Example of a pushdown automaton

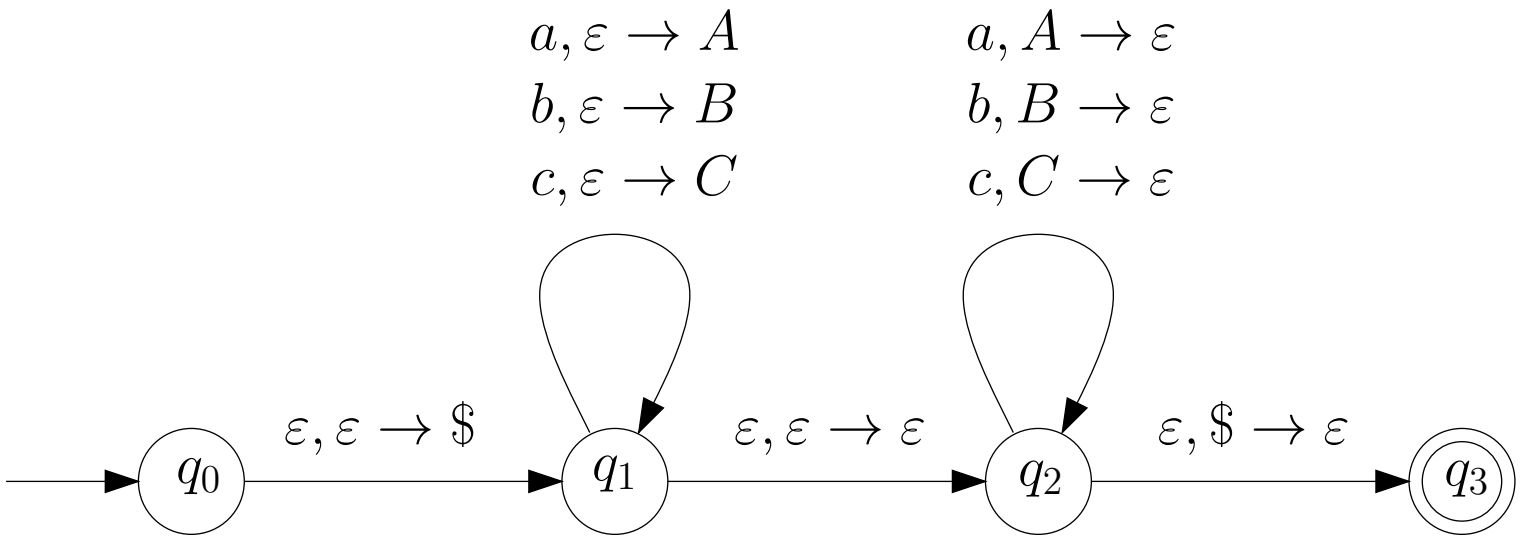
The following PDA recognizes the language $\{a^n b^n \mid n \geq 0\}$:



Exercise

1. Construct a PDA P that accepts the language $\{ww^R \mid w \in \{a, b, c\}^*\}$.
2. Show that $abccba \in L(P)$.

Solution 1


$a, \varepsilon \to A$
$b, \varepsilon \to B$
$c, \varepsilon \to C$
$a, A \to \varepsilon$
$b, B \to \varepsilon$
$c, C \to \varepsilon$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
q_0
q_1
q_2
q_3

Solution 2

$$
\begin{aligned}
(q_0, abccba, \varepsilon) &\vdash (q_1, abccba, \$) \\
&\vdash (q_1, bccba, A\$) \\
&\vdash (q_1, ccba, BA\$) \\
&\vdash (q_1, cba, CBA\$) \\
&\vdash (q_2, cba, CBA\$) \\
&\vdash (q_2, ba, BA\$) \\
&\vdash (q_2, a, A\$) \\
&\vdash (q_2, \varepsilon, \$) \\
&\vdash (q_3, \varepsilon, \varepsilon).
\end{aligned}
$$

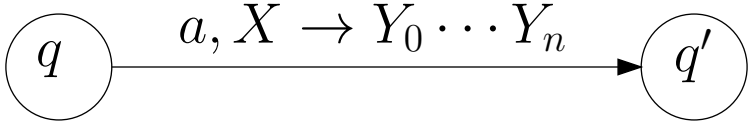
Generalizing the transition function

Instead of popping one symbol from the stack and pushing another, you are allowed to pop a symbol from the stack and push an arbitrary string of symbols.

$$(q, aw, XZ_0 \cdots Z_m) \vdash (q', w, Y_0 \cdots Y_n Z_0 \cdots Z_m)$$

iff

$$(q', Y_0 \cdots Y_n) \in \delta(q, a, X).$$



Exercise

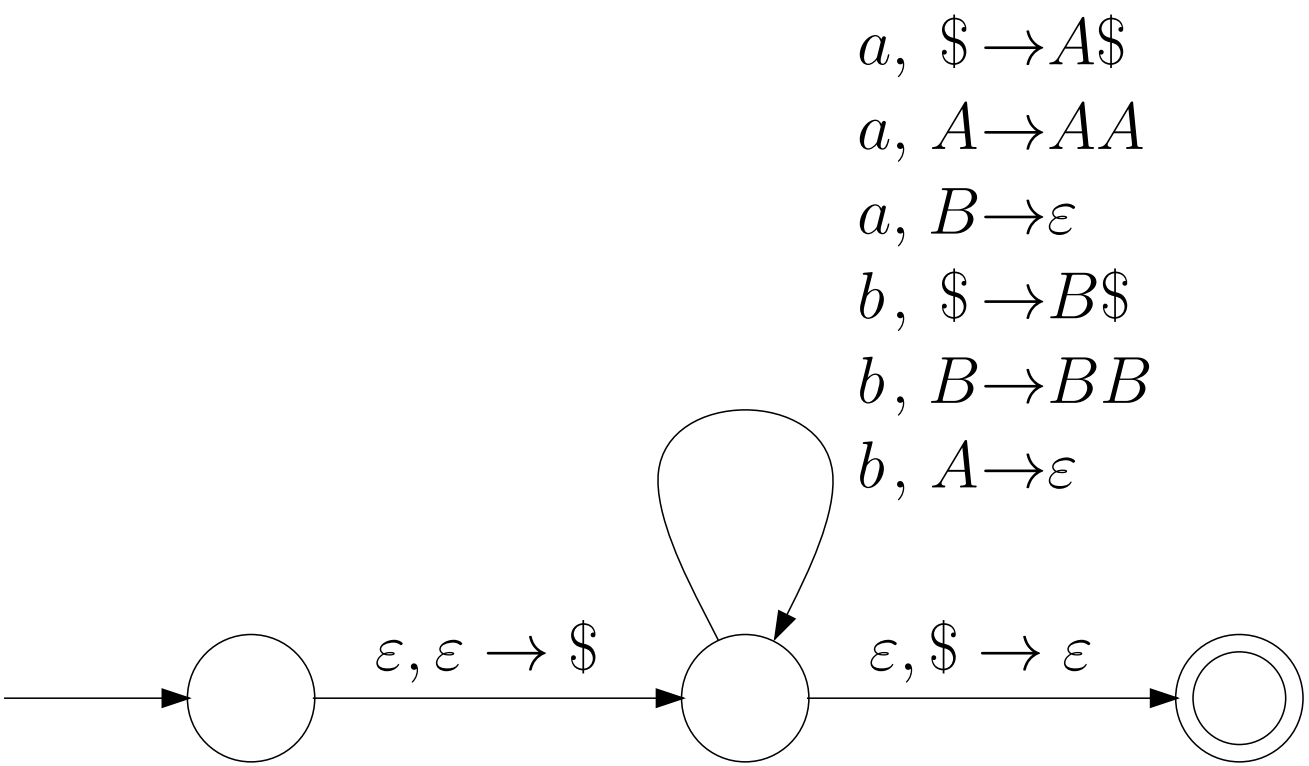
Let $\Sigma = \{a, b\}$. Construct a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

$$L(P) = \{w \in \Sigma^* \mid n_a(w) = n_b(w)\}.$$

Here $n_a(w)$ is the number of as and $n_b(w)$ the number of bs in w.

Exercise: Show that indeed $bbaaab \in L(P)$.

Solution


$a,\ \$ \to A\$$
$a,\ A \to AA$
$a,\ B \to \varepsilon$
$b,\ \$ \to B\$$
$b,\ B \to BB$
$b,\ A \to \varepsilon$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \$ \to \varepsilon$

PDAs recognize context-free languages

Theorem: A language L is context-free iff there exists a PDA P recognizing L.

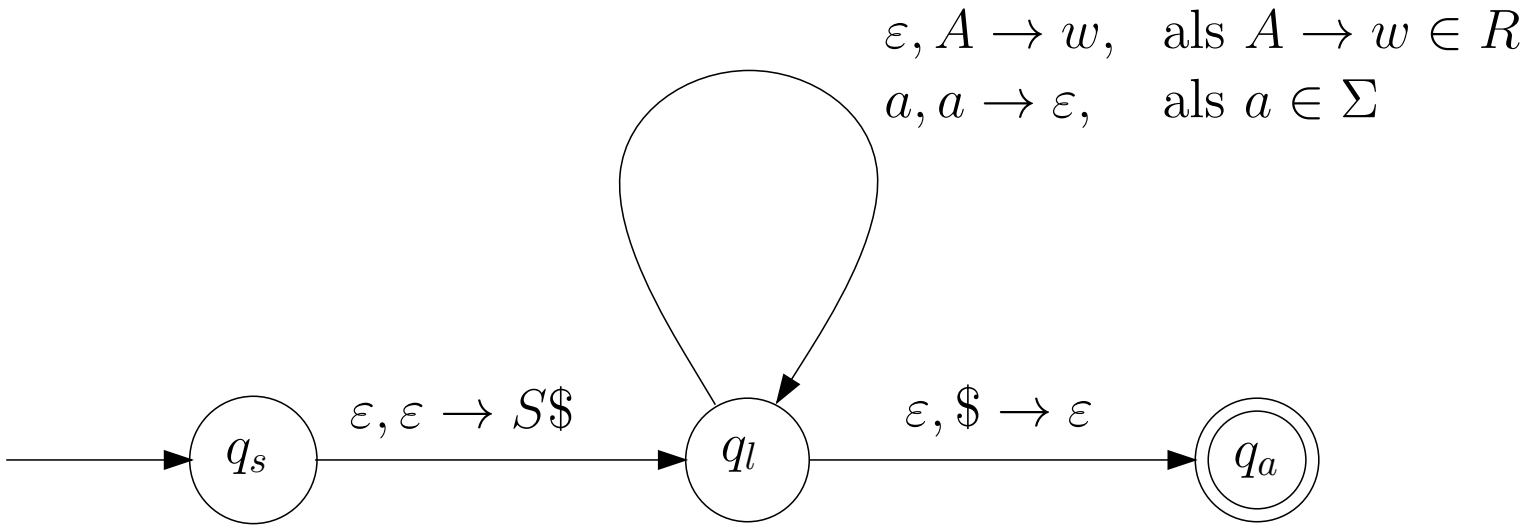
Proof sketch: Prove both directions separately.

1. If a language L is context free, then there exists a PDA recognizing L.
2. If P is a PDA, then $L(P)$ is context free.

For 1: Take a CFG and construct a PDA.
For 2: Take a PDA and formulate a CFG.

From CFG to PDA

(See also video Equivalence of CFGs and PDAs (part 1))



Idea: Alternate between: (1) processing input and (2) guessing the correct rule from the grammar on the stack until accepting is possible (or not).

From CFG to PDA: formally

Let $G = (V, \Sigma, R, S)$ be a CFG. Now define $P_G = (Q, \Sigma, \Gamma, \delta, q_s, F)$ such that:

$$\begin{array}{lcllcl} Q & = & \{q_s, q_l, q_a\} \cup E & \Gamma & = & \Sigma \cup V \cup \{\$\} \\ q_s & : & \text{start state} & F & = & \{q_a\} \end{array}$$

$$\delta(q,x,z) = \begin{cases} \{(q_l, S\$)\} & \text{if } q = q_s \text{ and } x = z = \varepsilon, \\ \{(q_a, \varepsilon)\} & \text{if } q = q_l,\ x = \varepsilon \text{ and } z = \$, \\ \{(q_l, \varepsilon)\} & \text{if } q = q_l \text{ and } x = z = a \in \Sigma, \\ \{(q_l, w) \mid A \to w \in R\} & \text{if } q = q_l,\ x = \varepsilon \text{ and } z = A, \\ \emptyset & \text{otherwise.} \end{cases}$$

NB: E is a set of auxiliary states (see Sipser).

Exercise

Fact: For every PDA P there is an equivalent P' that:

1. has exactly one accept state,
2. clears the stack before reaching the accept state, and
3. always either pushes or pops a symbol, but never both simultaneously.

Exercise: Show this.

Hint: First ensure that in the original automaton the stack is never completely cleared and then introduce a state that clears the stack before the accept state is reached.

Context-free grammar for a PDA

Theorem: Let P be a PDA. Then $L(P)$ is context-free.

Idea: We can assume that $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a PDA that

1. has exactly one accept state q_f,
2. clears the stack before the accept state is reached, and
3. always either pushes or pops a symbol, but never both simultaneously.

Construct a context-free grammar $G = (V, \Sigma, R, S)$ that for all $p, q \in Q$ has a variable A_{pq} such that for all $w \in \Sigma^*$:

$$A_{pq} \stackrel{*}{\Rightarrow} w \quad \text{iff} \quad (p, w, \varepsilon) \vdash^* (q, \varepsilon, \varepsilon)$$

Pumping lemma for context-free languages

Idea: If a word from a language is long enough, then there must be a rule in the derivation of that word that is applied more than once. This way the word can be "pumped up".

Variations on a theme

- Deterministic pushdown automata
- Chomsky hierarchy
- Turing machines
- Turing-Church Thesis

Chomsky hierarchy

Classification of grammars, corresponding classes of languages and machine models:

Type	Grammar	Languages	Machine model
0	unlimited	recursively enumerable	Turing machines
1	context-sensitive	context-sensitive	linear-bounded automata
2	context-free	context-free	PDAs (nondeterministic)
3	regular	regular	finite automata

(See also optional video Chomsky Hierarchy and Context-Sensitive Languages)