Context-Free Languages

Regular languages

- Regular expressions — recursive definition mirrors recursive definition of set of regular languages. Clear that regular languages are those that can be generated by regular expressions.
- Finite automata (FA’s, NFA’s, NFA-A’s) — simple machines that recognize regular languages (Kleene’s Theorem). Nondeterminism is inessential (Theorem 4.3).

Context-free languages

- Context-free grammars (CFG’s) — rough first approximation; much like recursive definitions of sets. Context-free languages are, by definition, those that can be generated by a CFG.
- Pushdown automata — an NFA-A with auxiliary memory (an unbounded stack). Nondeterministic PDA’s recognize context-free languages (Theorems 7.2k/7.4). Deterministic PDA’s correspond to a proper subset of context-free languages.

Example
Recall the language \textit{pal} over \{a, b\}: The smallest subset of \{a, b\} \* s.t.

1. \(A, a, b \in \text{pal}\).
2. For all \(x \in \text{pal}\), \(axa, bxb \in \text{pal}\).

Compare — a CFG for \textit{pal}:

\[
S \rightarrow A \\
A \rightarrow aSa \\
S \rightarrow bSb
\]

Example derivations:

\[
S \Rightarrow aSa \\
S \Rightarrow abSb \\
S \Rightarrow abSb \Rightarrow abba
\]

Another representation of this CFG for \textit{pal}:

\[
S \rightarrow A \\
A \rightarrow aSa \\
S \rightarrow bSb
\]

We’ll make these ideas precise momentarily…

Consider the complement of \textit{pal}, which we’ll call \textit{napal} for now.

We can give a recursive definition of \textit{napal} over \{a, b\} as follows: \textit{napal} is the least subset of \{a, b\} \* s.t.

1. For all \(x \in \{a, b\} \ast\), \(axb, bxa \in \text{napal}\).
2. For all \(x \in \text{napal}\), \(axa, bxb \in \text{napal}\).

The first step in constructing a CFG for \textit{napal} over \{a, b\} is to construct a CFG for \{a, b\} \* s.t. Here is one:

\[
A \rightarrow A \\
A \rightarrow aA \\
A \rightarrow bA
\]

Using this, we can construct a CFG for \textit{napal} similar to the recursive definition above:

\[
S \rightarrow aAb \mid bAs \\
A \rightarrow Aa \\
S \rightarrow aSa \\
S \rightarrow bSb
\]

Example derivations:

\[
S \Rightarrow aAb \Rightarrow ab \\
S \Rightarrow aSa \Rightarrow axa \Rightarrow axa \\
S \Rightarrow aAb \Rightarrow abA \Rightarrow abaa
\]

Definition
A context-free grammar (CFG) is a 4-tuple \((V, \Sigma, S, P)\)

where

- \(V\) and \(\Sigma\) are disjoint finite sets of symbols
- \(S \in V\)
- \(P\) is a finite set of expressions of the form \(A \rightarrow \alpha\)

s.t. \(A \in V\) and \(\alpha \in (V \cup \Sigma) \ast\).

The elements of \(V\) are called \textit{variables} or \textit{nonterminal symbols}.

The elements of \(\Sigma\) are called \textit{terminals} or \textit{terminal symbols}.

(Note: we will see that a CFG with set \(\Sigma\) of terminal symbols generates a language over \(\Sigma\).)

\(S\) is called the \textit{start symbol}.

The elements of \(P\) are called \textit{grammar rules} or \textit{productions}.

You’ll notice that in the examples we’ve considered thus far the productions are not of the prescribed syntactic form…
Productions

\[ A \rightarrow a_1 \]
\[ A \rightarrow a_2 \]
\[ \vdots \]
\[ A \rightarrow a_n \]

are often abbreviated

\[ A \rightarrow a_1 \mid a_2 \mid \cdots \mid a_n \].

Derivation Steps

Given a CFG \( G = (V, \Sigma, S, P) \),

for any \( \alpha, \beta, \gamma \in (V \cup \Sigma)^* \) and \( A \in V \), we write

\[ \alpha A \gamma \Rightarrow_\alpha \alpha \beta \gamma \]

to mean that there is a production

\[ A \rightarrow \beta \]

in \( P \).

(Often the subscript \( G \) is suppressed, if it is clear from context.)

If \( \alpha \Rightarrow_\gamma \beta \), we say that \( \alpha \) derives \( \beta \) in one step.

Notice:

\[ \alpha \Rightarrow_\gamma \beta \]

iff, for all \( \gamma_1, \gamma_2 \in (V \cup \Sigma)^* \),

\[ \gamma \alpha \gamma_2 \Rightarrow_\gamma \gamma \beta \gamma_2 \].

Derivations, CFL’s

For any \( \alpha, \beta \in (V \cup \Sigma)^* \), we write

\[ \alpha \Rightarrow_0 \beta \]

if

- \( \alpha = \beta \), or
- there is a \( \gamma \in (V \cup \Sigma)^* \) s.t. \( \alpha \Rightarrow_\gamma \gamma \) and \( \gamma \Rightarrow_\gamma \beta \).

That is, if we can derive \( \beta \) from \( \alpha \) in zero or more steps.

(Again, the subscript \( G \) is often suppressed.)

It will also be convenient to write

\[ \alpha \Rightarrow_n \beta \]

to say that there is an \( n \) step derivation of \( \beta \) from \( \alpha \).

For any CFG \( G = (V, \Sigma, S, P) \), the language generated by \( G \) is

\[ L(G) = \{ x \in \Sigma^* \mid S \Rightarrow_0^x \} \].

A language \( L \) is context-free if there is a CFG \( G \) s.t. \( L(G) = L \).
Consider the CFG

\[ G = ([S, X], \{a, b\}, S, P) \]

where \( P \) is

\[
S \to X | a \\
X \to SS | b
\]

What strings over \( \{S\} \) are derivable from \( S \)?

(Recall: \( S \Rightarrow^*_G S \))

What strings over \( \{a\} \) are derivable from \( S \)?

What strings over \( \{b\} \) are derivable from \( S \)?

In light of this, what is \( L(G) \)?

Claim If \( \alpha =_G^* \beta \),

then, for all \( \gamma_1, \gamma_2 \in (V \cup \Sigma)^* \),

\[
\gamma_1 \alpha \gamma_2 =_G^* \gamma_1 \beta \gamma_2.
\]

This follows from the stronger claim that if \( \alpha =_G^* \gamma \)

then, for all \( \gamma_1, \gamma_2 \in (V \cup \Sigma)^* \),

\[
\gamma_1 \alpha \gamma_2 =_G^* \gamma_1 \beta \gamma_2,
\]

which should be easy to prove by induction on \( n \).

It follows, for instance, that if \( \alpha =_G^* \gamma \beta \gamma_2 \)

and \( \beta =_G^* \gamma \),

then \( \alpha =_G^{\text{trans}} \gamma \beta \gamma_2 \).

Example Consider the CFG we used for \( \text{npal} \) over \( \{a, b\} \):

\[ G = ([S, A], \{a, b\}, S, P) \]

where \( P \) consists of

\[
S \to aSa | bSb | aAb | bAa \\
A \to A | aA | bA
\]

Let's prove that \( L(G) = \text{npal} \).

The proof consists of two parts.

1. We show that \( L(G) \subseteq \text{npal} \). More precisely, we show that, for all \( \alpha \in \{a, b\}^* \), if \( S \Rightarrow^*_G \alpha \)

then \( \alpha \in \text{npal} \).

2. We show that \( \text{npal} \subseteq L(G) \). For this part we use structural induction on the recursive definition of \( \text{npal} \).
1. For all $x \in \{a, b\}^*$, $axb, bxa \in \text{npal}$.
2. For all $x \in \text{npal}$, $axa, bxa \in \text{npal}$.

$$
S \rightarrow aSa | bSb | aAb | bAa \\
A \rightarrow A | aA | bA
$$

First we show that $L(G) \subseteq \text{npal}$. What we’ll show is that, for every $n \in \mathbb{N}$, for all $x \in \{a, b\}^n$, if $S \Rightarrow^n x$, then $x \in \text{npal}$.

Proof is by strong induction on $n$.

III: For all $m < n$ and all $x \in \{a, b\}^*$, if $S \Rightarrow^m x$, then $x \in \text{npal}$.

Assume that $S \Rightarrow^n x$. Notice that $n \geq 2$, since $x \in \{a, b\}^*$. Now consider two cases:

Case 1: First step of the derivation yields either $aAb$ or $bAa$.

We can conclude that $x$ has either the form $ayb$ or $bya$ for some $y \in \{a, b\}^*$. It follows from the recursive definition of $\text{npal}$ that $x \in \text{npal}$.

Case 2: First step of the derivation yields either $aSa$ or $bSb$.

We can conclude that $x$ has either the form $aya$ or $byb$ for some $y \in \{a, b\}^*$. Assume that $x$ has the form $aya$. (The other case is similar.) Since $S \Rightarrow aSa \Rightarrow a^n a$, we can conclude that $S \Rightarrow a^ny$. It follows by the III that $y \in \text{npal}$, and then by the recursive definition of $\text{npal}$ that $x = aya \in \text{npal}$ also.

Another standard example of a non-regular language is

$$
L = \{a^n b^n \mid n \in \mathbb{N}\}.
$$

Claim: $L$ is generated by the CFG $G = (\{S, \{a, b\}, S, P\}$ where $P$ consists of $S \rightarrow aSb | \Lambda$.

Proof. We show by induction on $n$ that, for every $n \in \mathbb{N}$,

$$
[w \in \{a, b\}^* \mid S \Rightarrow^n w] = \{a^n b^n, a^{n+1} b^{n+1}\}.
$$

Base case: It is clear that the only strings over $\{a, b\}$ that can be derived in one step are $\Lambda = a^0 b^0$ and $aSb$.

Ind. step: $\text{III: } [w \in \{a, b, S\}^* \mid S \Rightarrow^n w] = \{a^n b^n, a^{n+1} b^{n+1}, a^{n+1} b^{n+2}\}$.

To show:

$$
[w \in \{a, b, S\}^* \mid S \Rightarrow^{n+2} w] = \{a^{n+1} b^{n+1}, a^{n+2} b^{n+2}\}.
$$

By the III, only $a^n b^n$ and $a^{n+1} b^{n+1}$ can be derived from $S$ in $n + 1$ steps. It follows that the only $n + 2$ step derivations from $S$ are given by

and

$$
S \Rightarrow a^n b^n Sb^{n+1} \Rightarrow a^{n+1} b^{n+1} \\
S \Rightarrow a^n b^n Sb^{n+1} \Rightarrow a^{n+2} b^{n+2}.
$$

Closure properties of CFL’s

We have seen that regular languages are closed under set operations, as well as concatenation and Kleene star. What about CFL’s?

Recall: For regular languages, closure under union, concatenation and Kleene star follows immediately from the recursive definition. There was a simple FA construction establishing closure under set complement (in light of Kleene’s Theorem).

For CFL’s, it seems none of these closure properties follow immediately from the definition (a CFL is a language generated by a CFG).

Example. Consider CFG $G_1$ with start symbol $S_1$ and productions

$$
S_1 \rightarrow S_1 a \mid b
$$

and CFG $G_2$ with start symbol $S_2$ and productions

$$
S_2 \rightarrow S_2 b \mid a.
$$

Of course $L(G_1)$ and $L(G_2)$ are CFL’s. What about $L(G_1) \cup L(G_2)$?
\[ S_1 \rightarrow S_2 \alpha \mid \emptyset \]
\[ S_2 \rightarrow S_3 \mid a \]

One way to show that the language \( L(G_1) \cup L(G_2) \) is a CFL is by constructing a CFG to generate it...

We can similarly show that \( L(G_1) \cup L(G_2) \) is a CFL...

And another construction shows that \( L(G_1)^* \) is a CFL...

\[ S \rightarrow S_4 S \mid \Lambda \]

**Theorem 6.1** If \( L_1 \) and \( L_2 \) are context-free languages over \( \Sigma \), then so are \( L_1 \cup L_2 \), \( L_1 L_2 \) and \( L_1^* \).

**Proof.** Consider CFG's

\[ G_1 = (V_1, \Sigma, S_1, P_1) \]

and

\[ G_2 = (V_2, \Sigma, S_2, P_2) \]

s.t. \( V_1 \cap V_2 = \emptyset, L(G_1) = L_1 \) and \( L(G_2) = L_2 \).

**Union:** Take

\[ G_a = ([S_i] \cup V_1 \cup V_2, \Sigma, S_a, P_a) \]

where \( S_a \notin V_1 \cup V_2 \) and

\[ P_a = [ S_i \rightarrow S_i \mid S_a ] \cup P_1 \cup P_2. \]

To see that \( L(G_a) = L_1 \cup L_2 \), notice that, for any \( x \in \Sigma^* \),

\[ S_a \Rightarrow^*_a x \]

iff either

\[ S_a \Rightarrow^*_a S_i \Rightarrow^*_a x \quad \text{or} \quad S_a \Rightarrow^*_a S_a \Rightarrow^*_a x \]

iff either

\[ S_i \Rightarrow^*_a x \quad \text{or} \quad S_a \Rightarrow^*_a x. \]

**Concatenation:** Take

\[ G_1 = (V_1, \Sigma, S_1, P_1) \quad G_2 = (V_2, \Sigma, S_2, P_2) \]

where \( S_i \notin V_1 \cup V_2 \) and

\[ P_i = [ S_i \rightarrow S_i S_i ] \cup P_1 \cup P_2. \]

To see that \( L(G_1) \cap L(G_2) = L_1 L_2 \), notice that, for any \( x \in \Sigma^* \),

\[ S_i \Rightarrow^*_i x \]

iff there are \( y, z \in \Sigma^* \) s.t. the following three conditions hold:

- \( x = yz \),
- \( S_i \Rightarrow^*_i y \),
- \( S_i \Rightarrow^*_i z \).

**Kleene star:** Take

\[ G_1 = (V_1, \Sigma, S_1, P_1) \]

where \( S_a \notin V_1 \) and

\[ P_a = [ S_a \rightarrow S_1 S_a ] \cup P_1. \]

Claim: \( L(G_a) = L_1^* \).

In this case, a convincing proof seems a bit harder...

Key observation: For any \( y \in A^* \),

\[ S_a \Rightarrow^*_a S_a^* \]

\[ G_1 = (V_1, \Sigma, S_1, P_1) \]

where \( S_i \notin V_1 \) and

\[ P_i = [ S_i \rightarrow S_i ] \cup P_1. \]

Claim: \( L(G_i) = [A]^* \).

In this case, a convincing proof seems a bit harder...
We have looked at the constructions used in the proof of...

**Theorem 6.1** If $L_1$ and $L_2$ are context-free languages over $\Sigma$, then so are $L_1 \cup L_2$, $L_1 L_2$, and $L_1^*$.

This theorem yields the following corollary.

**Corollary 6.1** Every regular language is context-free.

*Proof.* Notice that the “basic” languages $\emptyset$, $\{\lambda\}$, and $\{a\}$ (for all $a \in \Sigma$) are all context-free. The set $L_R$ of regular languages over $\Sigma$ is the smallest set of languages over $\Sigma$ that contains these “basic” languages and is closed under the operations of union, concatenation and Kleene star. Since the set $L_C$ of context-free languages over $\Sigma$ also contains the “basic” languages and, by Theorem 6.1, is closed under the operations of union, concatenation and Kleene star, we can conclude that $L_R \subseteq L_C$.

And since we have seen that $\{a^n b^n \mid n \in \mathbb{N}\}$, for instance, is context-free but not regular, $L_R$ is a proper subset of $L_C$.

---

**Example** Consider the language

$$L = \{0^i 1^j 0^k \mid i,j,k \in \mathbb{N}, j = i+k\}.$$

$L$ can be obtained by the concatenation $L_1 L_2$ of

$$L_1 = \{0^n 1^n \mid n \in \mathbb{N}\}$$

and

$$L_2 = \{1^n 0^n \mid n \in \mathbb{N}\}.$$

Since $L_1$ is generated by

$$A \rightarrow 0A1 \mid \lambda$$

and $L_2$ is generated by

$$B \rightarrow 1B0 \mid \lambda,$$

it follows that $L$ is generated by

$$S \rightarrow AB$$

$$A \rightarrow 0A1 \mid \lambda$$

$$B \rightarrow 1B0 \mid \lambda.$$

---

**Example** Consider the language represented by the regular expression

$$(011 + 1)^*.$$

Notice that the production

$$A \rightarrow 011 \mid 1$$

can be used to generate the language represented by $011 + 1$. Then, using the Kleene star construction from Theorem 6.1, we can generate the language represented by $(011 + 1)^*$ using the productions

$$B \rightarrow AB \mid A$$

$$A \rightarrow 011 \mid 1$$

The same approach yields the productions

$$D \rightarrow DC \mid A$$

$$C \rightarrow 01$$

for regular expression $(01)^*$. Then, using the concatenation construction from Theorem 6.1, we obtain the following set of productions for $(011 + 1)^*(011)^*$:

$$S \rightarrow BD$$

$$B \rightarrow AB \mid A$$

$$A \rightarrow 011 \mid 1$$

$$D \rightarrow DC \mid A$$

$$C \rightarrow 01$$

---

**Theorem 61** shows that the set of CFL’s is closed under union, concatenation and Kleene star.

**Question:** What about intersection and complement?

Although we’re not quite ready to show it, the language

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

is not context-free. (We’ll use that fact here anyway.)

But the languages

$$L_1 = \{a^n b^n c^n \mid m, n \in \mathbb{N}\}$$

and

$$L_2 = \{a^n b^n c^n \mid m, n \in \mathbb{N}\}$$

are context-free. For instance, $L_2$ is generated by

$$S \rightarrow AC$$

$$A \rightarrow aAb \mid \lambda$$

$$C \rightarrow cC \mid \lambda.$$

Since $L_1 \cap L_2 = L$, we conclude that the set of context-free languages is not closed under intersection.

And since the set of CFL’s is closed under union, adding complement would give us intersection $(L_1 \cup L_2^c) = L_1 \cap L_2^c$, so context-free languages are also not closed under complement.
Pushdown Automata
A pushdown automaton (PDA) is essentially an NFA-Λ with an auxiliary stack (see LIFO).

Rough sketch of a PDA $M$ to recognize $\{a^n b^n \mid n \geq 0\}$:
$M$ starts in state $q_0$ with an "empty" stack. State $q_2$ is the only accepting state; $M$ accepts input string $x$ if it ends up in $q_2$ after having read all of $x$.

In state $q_0$:
1. If $M$ reads an $a$, $M$ stays in $q_0$ and pushes $a$ onto the stack.
2a. If $M$ reads a $b$ with an "empty" stack, $M$ "crashes."
2b. If $M$ reads a $b$ with a "nonempty" stack, $M$ goes to $q_2$ and pops the top symbol off the stack.
3. No $\Lambda$-transitions from state $q_0$.

In state $q_1$:
1. If $M$ reads an $a$, $M$ "crashes."
2a. If $M$ reads a $b$ with an "empty" stack, $M$ "crashes."
2b. If $M$ reads a $b$ with a "nonempty" stack, $M$ stays in $q_1$ and pops the top symbol off the stack.
3. When in state $q_1$ with an empty stack, $M$ may go to state $q_0$ without reading anything. (A $\Lambda$-transition!)

In state $q_2$:
1, 2. If $M$ reads a symbol when in state $q_2$, $M$ "crashes."

Example PDA recognizing $\{a^n b^n \mid n \geq 0\}$.
$M = \{(q_0, q_1, q_2), \{a, b\}, \{a, Z_0\}, q_0, Z_0, \{q_2\}, \delta\}
where $\delta$ is defined as follows,

$\delta(q_0, \Lambda, Z_0) = \epsilon$
$\delta(q_0, a, a) = \epsilon$
$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa)\}$
$\delta(q_0, b, Z_0) = \epsilon$
$\delta(q_0, b, a) = \{(q_1, A)\}$
$\delta(q_1, A, Z_0) = \{(q_2, Z_0)\}$
$\delta(q_1, \Lambda, a) = \epsilon$
$\delta(q_2, a, Z_0) = \epsilon$
$\delta(q_2, a, a) = \epsilon$
$\delta(q_2, b, a) = \{(q_2, A)\}$
$\delta(q_2, A, Z_0) = \epsilon$
$\delta(q_2, a, Z_0) = \epsilon$
$\delta(q_2, a, a) = \epsilon$
$\delta(q_2, b, Z_0) = \epsilon$

In the previous example, moves were determined by
- current state
- symbol to be read (or $\Lambda$)
- what's on the stack (or whether the stack is empty)

A move may change the state and/or pop/push the stack.

Definition A pushdown automaton (PDA) is a 7-tuple $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ where

- $Q$ is a finite set (states)
- $\Sigma$ and $\Gamma$ are finite sets of symbols (input and stack alphabets)
- $q_0 \in Q$ (initial state)
- $Z_0 \in \Gamma$ (initial stack symbol)
- $A \subseteq Q$ (accepting states)
- $\delta : Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \rightarrow \{\text{finite subsets of } Q \times \Gamma\}$.

Example Computation of $M$ on string $aabb$.

$(q_0, aabb, Z_0) \vdash_M (q_0, aabb, aZ_0)$
$\vdash_M (q_0, aabb, aZ_0)$
\vdash_M (q_1, a, Z_0)$
$\vdash_M (q_2, A, Z_0)$

Example Computation of $M$ on string $abb$.

$(q_0, aabb, Z_0) \vdash_M (q_0, aabb, aZ_0)$
$\vdash_M (q_0, aabb, aZ_0)$
$\vdash_M (q_2, A, Z_0)$

Example Computation of $M$ on string $aabb$.

$(q_0, aabb, Z_0) \vdash_M (q_0, aabb, aZ_0)$
$\vdash_M (q_0, aabb, aZ_0)$
$\vdash_M (q_2, A, Z_0)$
**Configurations**

**Definition** A configuration is a triple

\[(q, x, \alpha)\]

where \(q \in Q, x \in \Sigma^*, \alpha \in \Gamma^*\).

- \(q\) is the current state.
- \(x\) is the unread portion of the input string.
- \(\alpha\) is the stack contents—top of stack corresponds to beginning of string \(x\).

If \(q_0\) is the initial state of \(M\), \(x \in \Sigma^*\), and \(Z_0\) is the initial stack symbol of \(M\), then

\[(q_0, x, Z_0)\]

is the initial configuration of \(M\) on input \(x\).

If \(q\) is an accepting state of \(M\) and \(\alpha \in \Gamma^*\), then

\[(q, \Lambda, \alpha)\]

is an accepting configuration of \(M\).

**One-step and multi-step transitions**

**Definition** For any \(x \in \Sigma^*\) and \(\alpha \in \Gamma^*\), we can write

\[(q, x, \alpha) \rightarrow_M (r, x, \beta \alpha)\]

if \((r, \beta) \in \delta(q, \alpha)\).

For any \(x \in \Sigma^*\) and \(\alpha \in \Gamma^*\), we can write

\[(q, x, \beta \alpha) \rightarrow_M (r, x, \beta \alpha)\]

if \((r, \beta) \in \delta(q, \Lambda, \beta)\).

**Definition** For any \(q \in Q, x \in \Sigma^*\) and \(\alpha \in \Gamma^*\), we can write

\[(q, x, \alpha) \rightarrow_M (q, x, \alpha)\]

For any \(p, r \in Q, x, z \in \Sigma^*\) and \(\alpha, \gamma \in \Gamma^*\), we can write

\[(p, x, \alpha) \rightarrow_M (r, z, \gamma)\]

if there exist \(q \in Q, \gamma \in \Sigma^*\) and \(\beta \in \Gamma^*\) s.t.

\[(p, x, \alpha) \rightarrow_M (q, \gamma, \beta)\]

and

\[(q, y, \beta) \rightarrow_M (r, z, \gamma)\].

**Example** Idea for a PDA recognizing \(\{xa^* | x \in \{a, b\}^*\}\).

- **Initial state** \(q_0\) read next symbol of input and push onto stack

  \[\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}\]

  \[\delta(q_0, a, a) = \{(q_0, aa)\}\]

  \[\delta(q_0, a, b) = \{(q_0, ab)\}\]

  \[\delta(q_0, b, Z_0) = \{(q_0, bZ_0)\}\]

  \[\delta(q_0, b, a) = \{(q_0, ba)\}\]

  \[\delta(q_0, b, b) = \{(q_0, bb)\}\]

  or guess "end of \(a^*\)" and go to \(q_1\) on \(\Lambda\)-transition

  \[\delta(q_0, \Lambda, Z_0) = \{(q_1, Z_0)\}\]

  \[\delta(q_0, \Lambda, a) = \{(q_1, a)\}\]

  \[\delta(q_0, \Lambda, b) = \{(q_1, b)\}\]

- **State** \(q_1\) roughly, read \(a^*\) - popping off stack to check against \(x\)

  \[\delta(q_1, a, a) = \{(q_1, A)\}\]

  \[\delta(q_1, b, b) = \{(q_1, A)\}\]

  when stack is empty, take \(\Lambda\)-transition to \(q_2\)

  \[\delta(q_1, \Lambda, Z_0) = \{(q_2, Z_0)\}\]

- **Accepting state** \(q_2\)
Complete specification of PDA for \( [xx^r \mid x \in \{a, b\}^*] \).

\[
M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, Z_0\}, q_0, Z_0, \delta)
\]

where \( \delta \) is defined as follows.

\[
\begin{align*}
\delta(q_0, A, Z_0) &= \{ (q_0, Z_0) \} \\
\delta(q_0, A, a) &= \{ (q_0, a) \} \\
\delta(q_0, A, b) &= \{ (q_0, b) \} \\
\delta(q_0, a, Z_0) &= \{ (q_0, aZ_0) \} \\
\delta(q_0, a, a) &= \{ (q_0, aa) \} \\
\delta(q_0, a, b) &= \{ (q_0, ab) \} \\
\delta(q_0, b, Z_0) &= \{ (q_0, bZ_0) \} \\
\delta(q_0, b, a) &= \{ (q_0, ba) \} \\
\delta(q_0, b, b) &= \{ (q_0, bb) \} \\
\delta(q_1, A, Z_0) &= \{ (q_1, Z_0) \} \\
\delta(q_1, A, a) &= \emptyset \\
\delta(q_1, A, b) &= \emptyset \\
\delta(q_1, a, Z_0) &= \emptyset \\
\delta(q_1, a, a) &= \{ (q_1, A) \} \\
\delta(q_1, a, b) &= \emptyset \\
\delta(q_1, b, Z_0) &= \emptyset \\
\delta(q_1, b, a) &= \emptyset \\
\delta(q_1, b, b) &= \{ (q_1, A) \}
\end{align*}
\]

Some slight modifications to the previous PDA are needed to accept

\[
pad = \{ x \in \{a, b\}^* \mid x = x^r \}.
\]

Roughly speaking, the PDA should also be allowed to guess if the input string is a palindrome of odd length. If so, then the middle character of the string will be ignored — neither pushed nor popped.

More precisely, one can add

\[
(q_0, a, Z_0) \in \delta(q_0, a, Z_0)
\]

\[
(q_0, a) \in \delta(q_0, a, a)
\]

\[
(q_1, b) \in \delta(q_0, a, b)
\]

and

\[
(q_0, a, Z_0) \in \delta(q_1, b, Z_0)
\]

\[
(q_0, a) \in \delta(q_1, b, a)
\]

\[
(q_1, b) \in \delta(q_1, b, b)
\]

It appears that nondeterminism is essential for a PDA accepting \( pad \) or even \( [xx^r \mid x \in \{a, b\}^*] \).

What about \( [a^n b^n \mid n \in \mathbb{N}] \)?
On the other hand, one can easily verify that our previous PDA for $[a^n b^n | n > 0]$ is deterministic.

**Question** Is $[a^n b^n | n ∈ A]$ a DCFL?

**Example** DPDA recognizing $[a^n b^n | n ∈ A]$.

$M = ([q_0, q_0, q_1], \{a, b\}, \{a, Z_0\}, q_0, Z_0, [q_0, q_1], \delta)$

where $\delta$ is defined as follows. (Only nonempty moves shown.)

\[
\begin{align*}
\delta(q_0, a, Z_0) &= ([q_1, a Z_0]) \\
\delta(q_1, a, a) &= ([q_1, aa]) \\
\delta(q_1, b, a) &= ([q_1, A]) \\
\delta(q_2, A, Z_0) &= ([q_2, Z_0]) \\
\delta(q_2, b, a) &= ([q_2, A])
\end{align*}
\]

Computation of $M$ on input $aab$.

\[
\begin{align*}
(q_0, aabb, Z_0) &\xrightarrow{M} (q_1, aabb, aZ_0) \\
&\xrightarrow{M} (q_1, bb, aaZ_0) \\
&\xrightarrow{M} (q_2, b, aZ_0) \\
&\xrightarrow{M} (q_2, a, A) \\
&\xrightarrow{M} (q_2, A, Z_0)
\end{align*}
\]

**Example** For CFG

$G = ([S], \{a, b\}, S, \{S → aS | A\})$

we construct PDA

$M = ([q_0, q_0, q_1], \{a, b\}, \{S, a, b, Z_0\}, q_0, Z_0, [q_0, q_1], \delta)$

where the nonempty moves of $\delta$ are:

\[
\begin{align*}
\delta(q_0, A, Z_0) &= ([q_0, SZ_0]) \\
\delta(q_0, A, S) &= ([q_0, aS], (q_0, A]) \\
\delta(q_0, a, a) &= ([q_0, A]) \\
\delta(q_1, b, b) &= ([q_0, A]) \\
\delta(q_2, A, Z_0) &= ([q_2, Z_0])
\end{align*}
\]

**Embedding CFG’s in PDA’s**

**Theorem 7.2** For any CFG $G = (V, \Sigma, S, P)$ there is a PDA that accepts $L(G)$.

Take

$M = ([q_0, q_1], \Sigma, V' \cup \Sigma, [Z_0], q_0, Z_0, [q_0, q_1], \delta)$

where $Z_0 \notin V' \cup \Sigma$ and $\delta$ has the following nonempty moves.

- $\delta(q_0, A, Z_0) = ([q_0, SZ_0])$
- For all $A \in V'$, $\delta(q_0, A, a) = ([q_0, aA])$.
- For all $a \in \Sigma$, $\delta(q_0, a, a) = ([q_0, A])$.
- $\delta(q_1, A, Z_0) = ([q_2, Z_0])$

Two computations of $M$ on $aab$:

\[
\begin{align*}
(q_0, aabb, Z_0) &\xrightarrow{M} (q_1, aabb, SZ_0) \\
&\xrightarrow{M} (q_1, aabb, aSZ_0) \\
&\xrightarrow{M} (q_1, aabb, SbZ_0) \\
&\xrightarrow{M} (q_1, aabb, bZZ_0) \\
&\xrightarrow{M} (q_1, b, bZZ_0) \\
&\xrightarrow{M} (q_1, b, bZ_0) \\
&\xrightarrow{M} (q_2, A, Z_0) \\
&\xrightarrow{M} (q_2, A, Z_0) \quad \text{(accept)}
\end{align*}
\]

\[
\begin{align*}
(q_0, aabb, Z_0) &\xrightarrow{M} (q_1, aabb, SZZ_0) \\
&\xrightarrow{M} (q_1, aabb, ZZ_0) \\
&\xrightarrow{M} (q_1, aabb, Z_0) \\
&\xrightarrow{M} (q_1, aabb, Z_0) \quad \text{(crash)}
\end{align*}
\]
Correspondence between PDA's and CFL's

**Theorem 7.2** For any CFG $G = (V, \Sigma, S, P)$ there is a PDA that accepts $L(G)$.

Take

$$M = ([q_0, q_1], \Sigma, V \cup \Sigma \cup \{Z_0\}, q_0, Z_0, \delta)$$

where $Z_0 \notin V \cup \Sigma$ and $\delta$ has the following nonempty moves:

1. $\delta(q_0, \Lambda, Z_0) = [(q_1, Z_0)]$
2. For all $A \in V$, $\delta(q_0, A, A) = [(q_1, A) \mid A \to A \in P]$.
3. For all $a \in \Sigma$, $\delta(q_1, a, a) = [(q_1, A)]$.
4. $\delta(q_1, \Lambda, Z_0) = [(q_0, Z_0)]$

On the other hand, we can conclude from Theorems 7.3 and 7.4 in the book that:

For any PDA $M$, there is a CFG $G$ s.t. $L(G) = L(M)$.

In light of this result and Theorem 7.2, we have the same kind of correspondence between PDA's and CFL's that we had for FA's and regular languages.

---

**Theorem 8.1a** (Pumping Lemma for CFL's) For any CFL $L$ over $\Sigma$, there is an $n \in \mathbb{N}$ s.t. for all $u \in L$, if $|u| \geq n$, then there are $v, w, x, y, z \in \Sigma^*$ s.t.

1. $u = vwxyz$
2. $|wy| > 0$
3. $|wxy| \leq n$
4. For all $m \in \mathbb{N}$, $vw^mxy^mz \in L$.

In the proof of this theorem, we learn that for any sufficiently long $u \in L$, there are derivability relations

$$S \Rightarrow^n vAw \Rightarrow^m vwAyz \Rightarrow^n vwxyz = u.$$

As we observed previously, this alone is enough to guarantee that for any $m \in \mathbb{N}$, $vw^mxy^mz \in L$. But we also learn that $|wy| > 0$ and $|wxy| \leq n$. These additional constraints make the result useful for showing that some languages are not context-free.

The proof of the Pumping Lemma uses "derivation trees," and also uses the fact that any context-free language can be generated by a CFG in "Chomsky normal form" (that is, the language minus $\Lambda$).

---

**Pumping Lemma for CFL's**

For regular languages, we observed that any sufficiently long string causes an FA to enter some state more than once. As a consequence, any such string $x$ accepted by the FA can be written $uxw$ in such a way that for all $n \in \mathbb{N}$, $ux^n w$ is also accepted by the FA.

A similar idea works for CFG's...

Consider derivability relations

$$S \Rightarrow^n vAz \Rightarrow^n uwAyz \Rightarrow^m uwxyz$$

where $v, w, x, y, z \in \Sigma^*$.

It follows that

$$A \Rightarrow^n w^3Ay^3 \Rightarrow^n w^3Ay^3 \ldots$$

and also that

$$A \Rightarrow^n x.$$

We can conclude that for every $n \in \mathbb{N}$,

$$S \Rightarrow^n vw^nxy^n z.$$

---

**Derivation Trees**

In what follows we will rely on previous experience with trees...

First, let's try an example or two. Recall the CFG for $npad$ with the following productions.

$$S \rightarrow aAb \mid bAa \mid aSa \mid bSb$$

$$A \rightarrow A \mid aA \mid bA$$

Example derivations:

$$S \Rightarrow aAb \Rightarrow aAb$$

$$S \Rightarrow aSa \Rightarrow aaAb \Rightarrow aaAba$$

$$S \Rightarrow aAb \Rightarrow abAb \Rightarrow abaaAb \Rightarrow abaaAba$$

Let's consider how to represent these derivations as trees...
Here's a definition of such derivation trees.

**Definition** Given a CFG $G = (V, \Sigma, S, P)$, a **derivation tree** (wrt $G$) is a tree with

- each interior node labeled with a variable,
- each leaf labeled with a variable, a terminal, or $\Lambda$

s.t. each interior node corresponds to a rule application — that is, if the label of the node is $A$, then either

- the node has a single child labeled $A$, and $P$ includes the production $A \rightarrow A$,

or

- no child of the node is labeled $A$ and the labels of the children of the node when concatenated left-to-right form a string $\alpha$ s.t. $P$ includes the production

$$A \rightarrow \alpha.$$  

**Chomsky normal form**

**Definition** A CFG is in **Chomsky normal form** if all of its productions have one of the forms

$$A \rightarrow BC$$

$$A \rightarrow a$$

where $A, B, C$ are variables and $a$ is a terminal symbol.

**Theorem 6.6** Any CFG $G$ can be converted into a CFG $G'$ in Chomsky normal form s.t. $L(G) = L(G') = \{\Lambda\}$.

**Remarks on the proof of the Pumping Lemma for CFL's:**

Derivation trees for CFG's in Chomsky normal form are almost binary trees — each interior node has either two children labeled with variables or a single child (a leaf) labeled with a terminal. Consequently, the derivation tree for a string of terminals of length $2^p$ will have a path from root to leaf (a branch) of length $p + 1$ or greater.

So if there are $p$ variables in the CFG, and there is a derivation tree for a string of length at least $2^p$, then there will be a branch in that tree in which at least one label is repeated.

Of course, each subtree of a derivation tree is itself a derivation tree. And the key to obtaining the upper bound on $|wxy|$ is that sufficiently large subtrees will have branches with repeated labels. On the other hand, the nonemptiness of $wz$ is obtained from the fact that there are no $\Lambda$ or unit productions in a CFG in Chomsky normal form.
Using the Pumping Lemma for CFL's

Example Take
\[ L = \{ a^i b^i c^i \mid i \in \mathbb{N} \}. \]
Suppose that \( L \) is context-free. Let \( n \) be the natural number guaranteed by the Pumping Lemma. Take
\[ u = a^n b^n c^n. \]
So by the Pumping Lemma, there are \( v, w, x, y, z \in \{ a, b, c \}^* \) s.t.
1. \( u = vwxz \)
2. \( |wy| > 0 \)
3. \( |vxy| \leq n \)
4. for any \( m \in \mathbb{N} \), \( vw^mxy^mz \in L \).

There are two cases to consider.

Case 1: \( a \) occurs in \( wxy \).
Since \( |wxy| \leq n \), it follows that \( c \) does not occur in \( wxy \). And since \( |wy| > 0 \), at least one \( a \) or \( b \) occurs in \( wxy \). It follows that \( vw^0xy^0z \notin L \), since \( vw^0xy^0z \) will have either fewer \( a \)'s than \( c \)'s or fewer \( b \)'s than \( c \)'s.

Case 2: \( a \) does not occur in \( wxy \).
And since \( |wy| > 0 \), at least one \( b \) or \( c \) occurs in \( wxy \). It follows that \( vw^0xy^0z \notin L \), since \( vw^0xy^0z \) will have either fewer \( b \)'s than \( a \)'s or fewer \( c \)'s than \( a \)'s.

So in both cases we conclude that \( vw^0xy^0z \notin L \), which contradicts the fact that for any \( m \in \mathbb{N} \), \( vw^mxy^mz \in L \).

We can conclude that \( L \) is not a CFL.

Example Take
\[ L = \{ a^n b^n \} \in \{ a, b \}^* \].
Suppose that \( L \) is context-free. Let \( n \) be the natural number guaranteed by the Pumping Lemma. Take
\[ u = a^n b^n c^n. \]
So by the Pumping Lemma, there are \( v, w, x, y, z \in \{ a, b \}^* \) s.t.
1. \( u = vwxz \)
2. \( |wy| > 0 \)
3. \( |vxy| \leq n \)
4. for any \( m \in \mathbb{N} \), \( vw^mxy^mz \in L \).

Consider three cases:

Case 1. \( wxy \) contains no characters from the second half of \( u \).
Since \( |wy| > 0 \), \( vw^0xy^0z \) will have the form \( a^i b^j c^0 \) with \( i + j < 2n \), so \( vw^0xy^0z \notin L \).

Case 2. \( wxy \) contains no characters from the first half of \( u \).
Since \( |wy| > 0 \), \( vw^0xy^0z \) will have the form \( a^i b^0 c^j \) with \( i + j < 2n \), so \( vw^0xy^0z \notin L \).

Case 3. \( wxy \) contains characters from both halves of \( u \).
Since \( |wxy| \leq n \), \( wxy \) contains no \( a \)'s from the first half of \( u \) and no \( b \)'s from the second half. Since \( |wy| > 0 \), \( vw^0xy^0z \) will have the form \( a^i b^j c^0 \) with \( i + j < 2n \), so \( vw^0xy^0z \notin L \).

So in all three cases we conclude that \( vw^0xy^0z \notin L \), which contradicts the fact that for any \( m \in \mathbb{N} \), \( vw^mxy^mz \in L \).

We can conclude that \( L \) is not a CFL.
Example. Take
\[ L = \{ x \in [a, b, c]^* \mid n_b(x) < n_1(x) \text{ and } n_c(x) < n_a(x) \}. \]

Suppose that \( L \) is context-free. Let \( n \) be the natural number guaranteed by the Pumping Lemma. Take
\[ u = a^n b^n c^n. \]

So by the Pumping Lemma, there are \( v, w, x, y, z \in [a, b, c]^* \) s.t.
1. \( u = vwyz \)
2. \( |wy| > 0 \)
3. \( |wxy| \leq n \)
4. for any \( m \in \mathbb{N} \), \( vw^mxy^mz \in L \).

Consider two cases. (As usual, we contradict 4 in each case.)

Case 1. \( w \) contains an \( a \).
Hence, \( vw^2yz \) has at least \( n + 1 \) \( a \)’s. And since \( |wyz| \leq n \),
\( wxyz \) contains no \( c \)’s. Hence, \( vw^2yz \) has only \( n + 1 \) \( c \)’s. So \( vw^2yz \notin L \).

Case 2. \( w \) contains no \( c \)’s.
Hence, \( vw^2yz \) has \( n \) \( a \)’s. And since \( |wyz| > 0 \), \( wyz \) contains a \( b \) or a \( c \). Hence, \( vw^2yz \) either has fewer than \( n + 1 \) \( b \)’s or fewer than \( n + 1 \) \( c \)’s. So \( vw^2yz \notin L \).

\[ (n + ip)^2 = n^2 + iq \]

Now, since \( |wy| > 0 \), \( p + q \neq 0 \). Taking \( i = 1 \), we obtain
\[ n^2 + q = (n + p)^2 = n^2 + 2pm + p^2 \]
from which we conclude that
\[ q = 2pm + p^2. \]

Taking \( i = 2 \), we obtain
\[ n^2 + 2q = (n + 2p)^2 = n^2 + 4pm + 4p^2 \]
from which we conclude that
\[ q = 2pm + 2p^2. \]

It follows that \( p^2 = 2p^2 \), which shows that \( p = 0 \). But then we see that \( q = 0 \) also, which contradicts the fact that \( p + q \neq 0 \).

We can conclude that \( L \) is not a CFL.

Example. Take
\[ L = \{ x \in [a, b]^* \mid n_a(x)^2 - n_b(x) \}. \]

Suppose that \( L \) is context-free. Let \( n \) be the natural number guaranteed by the Pumping Lemma. Take
\[ u = a^n b^n. \]

So by the Pumping Lemma, there are \( v, w, x, y, z \in [a, b]^* \) s.t.
1. \( u = vwyz \)
2. \( |wy| > 0 \)
3. \( |wyz| \leq n \)
4. for all \( m \in \mathbb{N} \), \( vw^mwy^mz \in L \).

Let \( p = n_a(wy) \) and let \( q = n_b(wy) \). So for all \( i \in \mathbb{N} \),
\[ n_a(vw^{i+1}yx^{i+1}z) = n + ip \]
and
\[ n_b(vw^{i+1}yx^{i+1}z) = n^2 + iq. \]

And since \( vw^{i+1}yx^{i+1}z \in L \) for all \( i \in \mathbb{N} \), we can conclude, by the definition of \( L \) that
\[ (n + ip)^2 = n^2 + iq \]
for all \( i \in \mathbb{N} \).

More closure properties of CFL’s

Recall that CFL’s are closed under union, concatenation and Kleene star.
Recall also that they are not closed under intersection and complement.

Theorem 8.4 If \( L_1 \) is a CFL over \( \Sigma \) and \( L_2 \) is a regular language over \( \Sigma \), then \( L_1 \cap L_2 \) is a CFL.

Proof idea. Take the PDA for \( L_1 \) and the FA for \( L_2 \) and use something very like the intersection construction for FA’s to construct a PDA for \( L_1 \cap L_2 \).

Claim If \( L \) is a DCFL, then so is \( L' \).

Proof idea. Adapt the complement construction used for FA’s.
It appears that this is not so straightforward though...