Discrete Structures for Computer Science

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU CSci 341

Subsection 1

Context-Free Languages

Context-Free Grammars

- A **context-free grammar** is a grammar whose productions are of the form
$$S \to w,$$
where S is a nonterminal and w is any string over the alphabet of terminals and nonterminals.
- Example: The grammar
$$S \to \Lambda | aSb$$
is context-free.
- Any regular grammar is context-free.

Context-Free Languages

- A language is **context-free** if it is generated by a context-free grammar.
- Example: The language $\{a^n b^n : n \geq 0\}$ is a context-free language since it is generated by the context-free grammar $S \to \Lambda | aSb$.
- Regular languages are context-free.
- The language $\{a^n b^n : n \geq 0\}$ is context-free but not regular.
- Therefore the set of all regular languages is a proper subset of the set of all context-free languages.

Terminology

- The term "context-free" comes from the requirement that all productions contain a single nonterminal on the left.
- When this is the case, any production $S \to w$ can be used in a derivation without regard to the "context" in which S appears.
- Example: We can use the rule $S \to w$ to make the following derivation step: $aS \Rightarrow aw$.
- A grammar that is not context-free must have some production whose left side is a string of two or more symbols.
- Example: The production $Sc \to w$ cannot be part of a context-free grammar.

  Any derivation that uses this production can replace the nonterminal S only in a "context" that has c on the right.
- Example: We can use the rule $Sc \to w$ to make the following derivation step: $aSc \Rightarrow aw$.

Combining Context-Free Languages

- Suppose M and N are context-free languages whose grammars have disjoint sets of nonterminals (rename them if necessary).
- Suppose also that the start symbols for the grammars of M and N are A and B, respectively.
- Then we have the following new languages and grammars:
  1. The language $M \cup N$ is context-free, and its grammar starts with the two productions $S \to A|B$.
  2. The language MN is context-free, and its grammar starts with the production $S \to AB$.
  3. The language M^* is context-free, and its grammar starts with the production $S \to \Lambda|AS$.

Subsection 2

Pushdown Automata

Pushdown Automata

- A **pushdown automaton** (**PDA**) is formally defined as a 7-tuple: $M = \langle A, \Gamma, Z, S, q_0, F, \delta \rangle$, where
  - A is a finite set, called the **input alphabet**;
  - Γ is a finite set, called the **stack alphabet**;
  - $Z \in \Gamma$ is the **initial stack symbol**;
  - S is a finite set of states;
  - $q_0 \in S$ is the **start state**;
  - $F \subseteq S$ is the **set of final states**;
  - $\delta \subseteq S \times (A \cup \{\Lambda\}) \times \Gamma \times \{\mathrm{pop}, \mathrm{push} \times (\Gamma - \{Z\}), \mathrm{nop}\} \times S$ is the **transition function**.

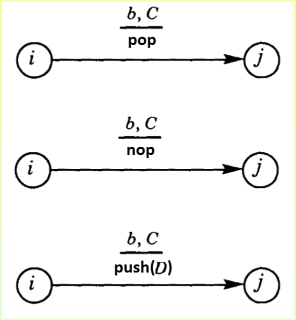
Graphical Representation

- We look at the meaning of some operations:

  (a) The first operation is (i, b, C, pop, j).
  The meaning is at state i, with input b and C on top of the stack, pop C and transition to state j.

  (b) The second operation is (i, b, C, nop, j).
  The meaning is at state i, with input b and C on top of the stack, neither pop nor push any symbol from the stack and transition to state j.

  (c) The third operation is $(i, b, C, \text{push}(D), j)$.
  The meaning is at state i, with input b and C on top of the stack, push D on the stack and transition to state j.



Nondeterminism

- In a PDA there are two types of nondeterminism that may occur:
  - A state i may emit two or more edges labeled with the same input symbol b and the same stack symbol C.
    In other words, there are at least two 5-tuples with first three components i, b, C.
    Example: The following represents nondeterminism:
    $$(i, b, C, \text{pop}, j),\ (i, b, C, \text{push}(D), k).$$
  - A state i may emit two edges labeled with the same stack symbol C, where one input symbol is Λ and the other input symbol is not.
    Then the machine has the option of consuming the input letter or leaving it alone.
    Example: The following represents nondeterminism:
    $$(i, \Lambda, C, \text{pop}, j),\ (i, b, C, \text{push}(D), k).$$

The Language of a Pushdown Automaton

- Let P be a PDA with input alphabet A.
- A string is **accepted** by the PDA if there is some computation (sequence of moves) from the start state that ends up in a final state with all letters of the string consumed.
- Otherwise, the string is **rejected** by the PDA.
- The **language of the PDA** P, denoted $L(P)$, is the set of strings that it accepts.

Representing a Computation of a Pushdown Automaton

- Let P be a PDA.
- A triple of the form

$$(\text{current state}, \text{unconsumed input}, \text{stack contents})$$

  is called an **instantaneous description** or an **ID**.
- We represent a **computation** as a sequence of IDs.
- Example: The ID

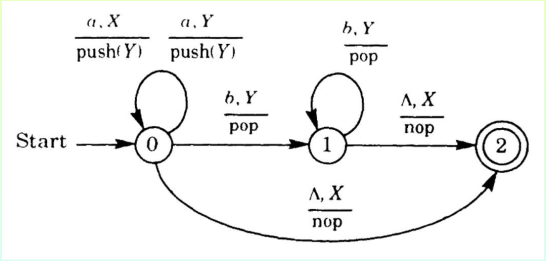
$$(i, abc, XYZW)$$

  means that:
  - The PDA is in state i;
  - It is currently reading the letter a of the unconsumed input abc;
  - X is at the top of the stack whose contents are $XYZW$.

Example

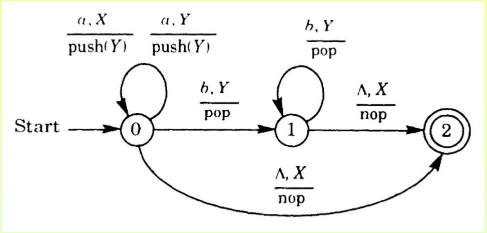
- The language $\{a^n b^n : n \geq 0\}$ can be accepted by a PDA.
  The PDA works as follows:
  - It keeps track of the number of a's in an input string by pushing the symbol Y onto the stack for each a.
  - Then it changes state and starts to pop the stack for each b encountered.

  The following PDA will do the job, where X is the initial symbol on the stack:



Example (Cont'd)

- We give a formal representation.
  The components are $P = \langle A, \Gamma, Z, S, q_0, F, \delta \rangle$, where:
  - $A = \{a, b\}$;
  - $\Gamma = \{X, Y\}$;
  - $Z = X$;
  - $S = \{0, 1, 2\}$;
  - $q_0 = 0$;
  - $F = \{2\}$;
  - $\delta = \{(0, \Lambda, X, \text{nop}, 2), (0, a, X, \text{push}(Y), 0), (0, a, Y, \text{push}(Y), 0), (0, b, Y, \text{pop}, 1), (1, b, Y, \text{pop}, 1), (1, \Lambda, X, \text{nop}, 2)\}$



- An accepting computation on *aabb* is as follows:

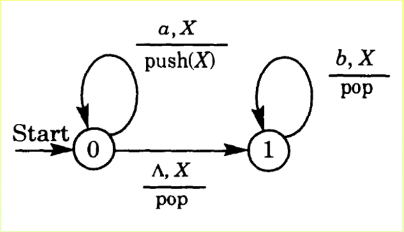
$$(0, aabb, X) \to (0, abb, YX) \to (0, bb, YYX)$$
$$\to (1, b, YX) \to (1, \Lambda, X) \to (2, \Lambda, X).$$

Equivalent Forms of Acceptance

- There are two types of acceptance in PDAs:
  - Final-state acceptance:
    A string is accepted if it has been consumed and the PDA is in a final state.
  - Empty stack acceptance:
    This requires that the input string be consumed and the stack be empty, with no requirement that the machine be in any particular state.
- We show that these definitions of acceptance are equivalent:
  The class of languages accepted by PDAs that use empty stack acceptance is the same as the class of languages accepted by PDAs that use final-state acceptance.

Example

- The language $\{a^n b^n : n \geq 0\}$ can be accepted by a PDA that accepts by empty stack.
  The following PDA will do the job, where X is the initial symbol on the stack:



- The formal description of δ is:
  $\delta = \{(0, a, X, \text{push}(X), 0), (0, \Lambda, X, \text{pop}, 1), (1, b, X, \text{pop}, 1)\}$;
- An accepting computation on *aabb* is:

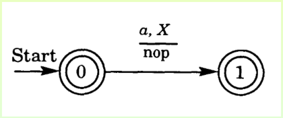
$$(0, aabb, X) \to (0, abb, XX) \to (0, bb, XXX)$$
$$\to (1, bb, XX) \to (1, b, X) \to (1, \Lambda, \Lambda).$$

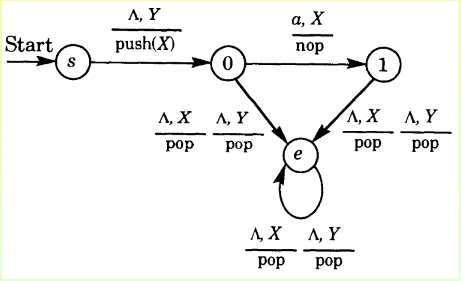
From a Final State PDA to an Empty Stack PDA

- Given a PDA that uses final state acceptance, with X the initial stack symbol.
  1. Create a new start state s, a new "empty stack" state e, and a new stack symbol Y that is at the top of the stack when the new PDA starts its execution.
  2. Connect the new start state to the old start state by an edge labeled $\frac{\Lambda, Y}{\text{push}(X)}$.
  3. Connect each final state to the new "empty stack" state e with one edge for each stack symbol.
     Label the edges with the expressions of the following form, where Z denotes any stack symbol, including Y: $\frac{\Lambda, Z}{\text{pop}}$.
  4. Add new edges from e to e labeled with the same expressions as in Step 3.

Example

- A PDA accepting the language $\{\Lambda, a\}$ by final state, with initial stack symbol X, is shown on the left:





Following the algorithm we transform it into a PDA accepting by empty stack, with initial stack symbol Y.

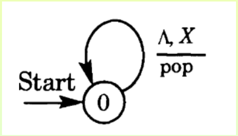
This is shown on the right.

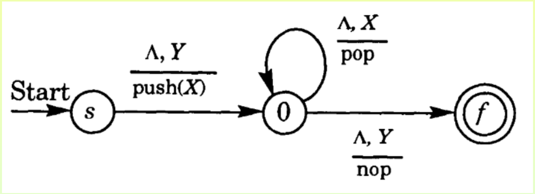
From an Empty Stack PDA to a Final State PDA

- The idea is to create a new final state that can be entered when an empty stack occurs in the given PDA.
  - We create a new start state with a new stack symbol Y.
  - Then add a Λ edge from the new start state to the old start state that pushes the old initial stack symbol X onto the stack.
  - An empty stack of the given PDA is detected whenever Y appears at the top of the stack.
- The algorithm to construct a final-state PDA from an empty stack PDA, with initial stack symbol X:
  1. Create a new start state s, a new final state f, and a new stack symbol Y, set as the initial stack symbol of the new PDA.
  2. Connect the new start state to the old start state by an edge labeled $\dfrac{\Lambda, Y}{\text{push}(X)}$.
  3. Connect each state of the given PDA to the new final state f, and label each of these new edges with the expression $\dfrac{\Lambda, Y}{\text{nop}}$.

Example

- The PDA on the left accepts $\{\Lambda\}$ by empty stack, with initial stack symbol X.





We get a PDA accepting $\{\Lambda\}$ by final state with initial stack symbol Y.

This is shown on the right.

Subsection 3

Context-Free Languages and Pushdown Automata

Allowing Multiple Stack Operations

- To shorten a PDA's description, we allow the operation field of a PDA instruction to hold a list of stack instructions.
- The 5-tuple

$$(i, a, C, \langle \mathrm{pop}, \mathrm{push}(X), \mathrm{push}(Y)\rangle, j)$$

  is executed by performing the operations $\mathrm{pop}, \mathrm{push}(X), \mathrm{push}(Y)$.
- This generalization does not alter the "power" of PDAs:

  For each generalized PDA, there exists an ordinary (single stack instruction) PDA that recognizes the same language.
- We can implement the generalized instructions in an ordinary PDA by placing enough new symbols on the stack at the start of the computation to make sure that any sequence of pop operations will not empty the stack if it is followed by a push operation.

Example

- Consider again the instruction
$$(i, a, C, \langle \text{pop}, \text{push}(X), \text{push}(Y)\rangle, j).$$
- We can execute this instruction in an ordinary PDA by the following sequence of instructions, where k and ℓ are new states:

$$\begin{array}{ll} (i, a, C, pop, k) & \\ (k, \Lambda, ?, \text{push}(X), \ell) & \quad (\text{? represents some stack symbol}) \\ (\ell, \Lambda, X, \text{push}(Y), j). & \end{array}$$

From a Context-Free Grammar to a PDA (Empty Stack)

- Given a context-free grammar G.
- We construct a (generalized) PDA accepting by emptystack with:
  - A single state 0;
  - Stack symbols the set of terminals and nonterminals;
  - Initial stack symbol the grammar's start symbol.
  - The transition function consists of the following instructions:
    1. For each terminal symbol a, the instruction $(0, a, a, \text{pop}, 0)$;
    2. For each production $A \rightarrow B_1 B_2 \ldots B_n$, where each B_i represents either a terminal or a nonterminal, create the instruction
       $$(0, \Lambda, A, \langle \text{pop}, \text{push}(B_n), \text{push}(B_{n-1}), \ldots, \text{push}(B_1) \rangle, 0).$$
    3. For each production $A \rightarrow \Lambda$, create the instruction $(0, \Lambda, A, \text{pop}, 0)$.

Example

- Suppose we have the following context-free grammar for the language $\{a^n b^n : n \geq 0\}$:
$$S \to aSb|\Lambda.$$
Apply the algorithm to obtain a PDA accepting by empty stack the same language.
    1. From the terminals a and b we create the two instructions
    $$(0, a, a, \text{pop}, 0), \quad (0, b, b, \text{pop}, 0).$$
    2. From the production $S \to aSb$, we create the instruction $(0, \Lambda, S, \langle \text{pop}, \text{push}(b), \text{push}(S), \text{push}(a) \rangle, 0)$.
    3. From the production $S \to \Lambda$ we create the instruction $(0, \Lambda, S, \text{pop}, 0)$.

  We have the PDA, with set of states $\{0\}$, input alphabet $\{a, b\}$, stack alphabet $\{S, a, b\}$, initial stack symbol S and transition $\{(0, a, a, \Lambda, 0), (0, b, b, \Lambda, 0),$
$(0, \Lambda, S, \langle \text{pop}, \text{push}(b), \text{push}(S), \text{push}(a) \rangle, 0), (0, \Lambda, S, \Lambda, 0)\}$.

Example (Cont'd)

- For the PDA constructed above, with transition $\{(0,a,a,\Lambda,0)$, $(0,b,b,\Lambda,0)$, $(0,\Lambda,S,\text{pop},\text{push}(b),\text{push}(S),\text{push}(a),0)$, $(0,\Lambda,S,\Lambda,0)\}$, write an accepting PDA computation for the input *aabb*.

ID	Instruction
$(0,aabb,S)$	
$(0,aabb,aSb)$	$(0,\Lambda,S,\langle\text{pop},\text{push}(b),\text{push}(S),\text{push}(a)\rangle,0)$
$(0,abb,Sb)$	$(0,a,a,\text{pop},0)$
$(0,abb,aSbb)$	$(0,\Lambda,S,\langle\text{pop},\text{push}(b),\text{push}(S),\text{push}(a)\rangle,0)$
$(0,bb,Sbb)$	$(0,a,a,\text{pop},0)$
$(0,bb,bb)$	$(0,\Lambda,S,\text{pop},0)$
$(0,b,b)$	$(0,b,b,\text{pop},0)$
$(0,\Lambda,\Lambda)$	$(0,b,b,\text{pop},0)$

From a PDA (Empty Stack) to a Context -Free Grammar

- Given a PDA that accepts strings by empty stack, with start state s and initial stack symbol E.
- For each stack symbol B and each pair of states i and j of the PDA, we construct a nonterminal of the grammar and denote it by B_{ij}.

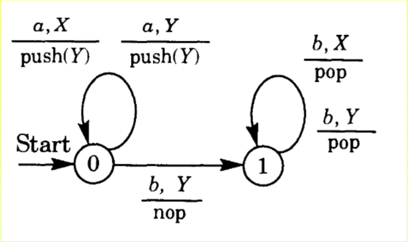
  We think of B_{ij} as deriving all strings that cause the PDA to move, in one or more steps, from state i to state j in such a way that the stack at state j is obtained from the stack at state i by popping B.
- We create one additional nonterminal S to denote the start symbol for the grammar.
  1. For each state j of the PDA, construct a production $S \to E_{sj}$;
  2. For each instruction of the form (p, a, B, pop, q), construct a production $B_{pq} \to a$.
  3. For each instruction of the form (p, a, B, nop, q), and each state j, construct a production $B_{pj} \to aB_{qj}$.
  4. For each instruction of the form $(p, a, B, \text{push}(C), q)$, and all states i and j construct a production $B_{pj} \to aC_{qi}B_{ij}$.

Example

- The following PDA accepts the language

  $$\{a^n b^{n+2} : n \geq 1\}$$

  by empty stack, where X is the initial stack symbol:


$\frac{a, X}{\text{push}(Y)}$
$\frac{a, Y}{\text{push}(Y)}$
$\frac{b, X}{\text{pop}}$
$\frac{b, Y}{\text{pop}}$
Start
0
1
$\frac{b,\ Y}{\text{nop}}$

  We have the nonterminals $S, X_{00}, X_{01}, X_{10}, X_{11}, Y_{00}, Y_{01}, Y_{10}, Y_{11}$.

  We construct the following productions:

  $S \to X_{00}, S \to X_{01}$
  $X_{11} \to b, Y_{11} \to b$
  $Y_{00} \to bY_{10}, Y_{01} \to bY_{11}$
  $X_{00} \to aY_{00}X_{00}, X_{01} \to aY_{00}X_{01}, X_{00} \to aY_{01}X_{10}, X_{01} \to aY_{01}X_{11}$
  $Y_{00} \to aY_{00}Y_{00}, Y_{01} \to aY_{00}Y_{01}, Y_{00} \to aY_{01}Y_{10}, Y_{01} \to aY_{01}Y_{11}$.

Example (Cont'd)

- We constructed the grammar:

  $S \to X_{00}, S \to X_{01}$
  $X_{11} \to b, Y_{11} \to b$
  $Y_{00} \to bY_{10}, Y_{01} \to bY_{11}$
  $X_{00} \to aY_{00}X_{00}, X_{01} \to aY_{00}X_{01}, X_{00} \to aY_{01}X_{10}, X_{01} \to aY_{01}X_{11}$
  $Y_{00} \to aY_{00}Y_{00}, Y_{01} \to aY_{00}Y_{01}, Y_{00} \to aY_{01}Y_{10}, Y_{01} \to aY_{01}Y_{11}.$

  Many derivations are unusable (X_{10} and Y_{10} do not appear on the left hand side of any rule).

  We can shrink the grammar by discarding unusable rules:

$$\begin{array}{rcl} S & \to & X_{01} \\ X_{01} & \to & aY_{01}X_{11} \\ Y_{01} & \to & aY_{01}Y_{11} | bY_{11} \\ X_{11} & \to & b \\ Y_{11} & \to & b. \end{array}$$

Subsection 4

Chomsky Normal Form

Removing Λ Productions

- Given a grammar, such that Λ does not occur in the language generated by the grammar.
- Then we can write a grammar for the same language without Λ-productions by using the following algorithm:
    1. Find the set of all nonterminals N such that N derives Λ.
    2. For each production of the form $A \to w$, create all possible productions of the form $A \to w'$, where w' is obtained from w by removing one or more occurrences of the nonterminals found in Step 1.
    3. The desired grammar consists of the original productions together with the productions constructed in Step 2, minus any productions of the form $A \to \Lambda$.

Example

- Consider the grammar

$$S \to aDaE \quad D \to bD|E \quad E \to cE|\Lambda$$

The language of the grammar does not contain Λ.
Apply the algorithm to eliminate Λ-productions.

1. The nonterminals deriving Λ are D and E: $E \Rightarrow \Lambda$ and $D \Rightarrow E \Rightarrow \Lambda$.
2. Original productions together with corresponding new productions:

$S \to aDaE$	$S \to aaE\|aDa\|aa$
$D \to bD$	$D \to b$
$D \to E$	$D \to \Lambda$
$E \to cE$	$E \to c$
$E \to \Lambda$	Discarded

3. New grammar:

$$\begin{array}{rcl} S & \to & aDaE|aaE|aDa|aa \\ D & \to & bD|b|E \\ E & \to & cE|c \end{array}$$

Chomsky Normal Form

- A context-free grammar is in **Chomsky normal form** if the right side of each production is either:
  - a single terminal or
  - a string of two nonterminals.

  Exception: If the language of the grammar contains Λ, then $S \to \Lambda$ is allowed, where S is the start symbol.
- Any context-free grammar can be written in Chomsky normal form.
- The Chomsky normal form is useful for many reasons:
  - Any string of length $n > 0$ can be derived in $2n - 1$ steps;
  - The derivation trees are binary trees.

Converting into Chomsky Normal Form

- Given a context-free grammar.
  1. If the start symbol S of the given grammar occurs on the right side of some production, then create a new start symbol S' and add $S' \to S$.
  2. If there is a production $A \to \Lambda$, where A is not the start symbol, then use algorithm to remove all productions that contain Λ. If this process removes a Λ production from the start symbol, then add it back.
  3. For each pair of nonterminals A and B, if $A \to B$ is a unit production or if there is a derivation $A \Rightarrow^+ B$, then add all productions of the form $A \to w$, where $B \to w$ is not a unit production.
     Now remove all the unit productions.
  4. For each production whose right side has two or more symbols, replace all occurrences of each terminal a with a new nonterminal A, and also add the new production $A \to a$.
  5. Replace each production of the form $B \to C_1C_2 \cdots C_n$, where $n > 2$, with the following two productions, where D is a new nonterminal:
     $B \to C_1D$ and $D \to C_2 \cdots C_n$.
     Continue this step until all right sides have two nonterminals.

Example (Steps 1 and 2)

- Write the following grammar in Chomsky normal form:

$$\begin{aligned} S &\to R|aTa \\ R &\to S|b \\ T &\to R|c \end{aligned}$$

1. Since there is S on the right of a production, we introduce a new start symbol S' and the production $S' \to S$:

$$\begin{aligned} S' &\to S \\ S &\to R|aTa \\ R &\to S|b \\ T &\to R|c \end{aligned}$$

2. There are no occurrences of Λ on the right.

   So this step leaves the grammar unchanged.

Example (Step 3)

- In Step 2, we got the grammar on the left.

$S' \to S$	$S' \to aTa \mid b$	
$S \to R \mid aTa$	$S \to aTa \mid b$	$S' \to aTa \mid b$
$R \to S \mid b$	$R \to aTa \mid b$	$T \to aTa \mid b \mid c$
$T \to R \mid c$	$T \to aTa \mid b \mid c$	

3. We have the unit productions

$$S' \Rightarrow S|R, \quad S \to R, \quad R \to S, \quad T \Rightarrow R|S.$$

The symbols S and R appear on the right and the nonunit productions they give are $S \to aTa$ and $R \to b$.
So we must add the productions

$$S' \to aTa|b, \quad S \to b, \quad R \to aTa, \quad T \to aTa|b.$$

This gives the grammar in the middle above.
Eliminating unused productions, we get the grammar on the right.

Example (Steps 4 and 5)

- In Step 3 we got the grammar

$$\begin{aligned} S' &\rightarrow aTa|b \\ T &\rightarrow aTa|b|c \end{aligned}$$

4. Replace the letter a in aTa by A and add the new production $A \rightarrow a$:

$$\begin{aligned} S' &\rightarrow ATA|b \\ T &\rightarrow ATA|b|c \\ A &\rightarrow a \end{aligned}$$

5. Replace $S' \rightarrow ATA$ by $S' \rightarrow AB$ and $T \rightarrow ATA$ by $T \rightarrow AB$, where $B \rightarrow TA$.
   We obtain the Chomsky normal form:

$$\begin{aligned} S' &\rightarrow AB|b \\ T &\rightarrow AB|b|c \\ B &\rightarrow TA \\ A &\rightarrow a \end{aligned}$$

Subsection 5

Properties of Context-Free Languages

The Pumping Lemma for Context-Free Languages

- **Pumping Lemma for Context-Free Languages**

  Let L be an infinite context-free language.
  There is a positive integer m such that for all strings $z \in L$ with $|z| \geq m$, z can be written in the form $z = uvwxy$, where the following properties hold:

$$\begin{array}{l} |vx| \geq 1 \\ |vwx| \leq m \\ uv^k wx^k y \in L, \text{ for all } k \geq 0. \end{array}$$

- The positive integer m depends on the grammar for the language L.
- It must be large enough to ensure a recursive derivation of any string of length m or more.

Example

- Show that the language $L = \{a^n b^n c^n : n \geq 0\}$ is not context-free.
  Assume L is context-free.
  Then by the Pumping Lemma we can pick a string $z = a^m b^m c^m$ in L, where m is the positive integer mentioned in the lemma.
  Since $|z| \geq m$, we can write it in the form $z = uvwxy$, such that $|vx| \geq 1$, $|vwx| \leq m$, and such that $uv^k wx^k y \in L$ for all $k \geq 0$.
  Neither v nor x can contain two distinct letters, since then v^2 or x^2 would contain letters out of order and $uv^2 wx^2 y \notin L$.
  Since $|vx| \geq 1$, we know that at least one of v and x is a nonempty string of the form a^i, or b^i, or c^i for some $i > 0$.
  Therefore the pumped string $uv^2 wx^2 y$ cannot contain the same number of a's, b's, and c's because one of the three letters a, b and c does not get pumped up.
  Thus $uv^2 wx^2 y$ cannot be in L, contradicting the Pumping Lemma.
  Therefore L is not context-free.

Context-Free Languages and Intersection

- Context-free languages are not closed under intersection.
- Example: Consider the languages

$$L_1 = \{a^n b^n c^k : n, k \in \mathbb{N}\}, \quad L_2 = \{a^k b^n c^n : n, k \in \mathbb{N}\}.$$

L_1 is context-free, since it is the language of the context-free grammar on the left.

$$\begin{array}{ll} S \to AC & S \to AB \\ A \to aAb|\Lambda & A \to aA|\Lambda \\ C \to cC|\Lambda & B \to bBc|\Lambda \end{array}$$

L_2 is also context-free, since it is the language of the context-free grammar on the right.

Finally, note that

$$L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}.$$

We saw that $L_1 \cap L_2$ is not context-free.

Context-Free Languages and Complement

- Context-free languages are not closed under complement.
  Suppose to the contrary that context-free languages are closed under complement.
  Consider again the context-free languages
  $$L_1 = \{a^n b^n c^k : n, k \in \mathbb{N}\}, \quad L_2 = \{a^k b^n c^n : n, k \in \mathbb{N}\}.$$
  Then, by hypothesis, L_1' and L_2' are context-free.
  Since the union of context-free languages are context-free, we then have that $L_1' \cup L_2'$ is context-free.
  Then, again by hypothesis,
  $$L_1 \cap L_2 = (L_1' \cup L_2')'$$
  is context-free.
  But $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$, which is not context-free, a contradiction.

Closure Properties of Context-Free Languages

- Context-free languages satisfy the following closure properties:
  1. The union of two context-free languages is context-free;
  2. The language product of two context-free languages is context-free;
  3. The closure of a context-free language is context-free;
  4. The intersection of a regular language with a context-free language is context-free.

Context-Free Language Morphisms

- Let $f : A^* \to A^*$ be a language morphism, i.e., such that
  - $f(\Lambda) = \Lambda$;
  - $f(uv) = f(u)f(v)$ for all strings u and v.

  Let L be a language over A.
  1. If L is context-free, then $f(L)$ is context-free;
  2. If L is context-free, then $f^{-1}(L)$ is context-free.

1. Since L is context-free, it has a context-free grammar.

   We create a context-free grammar for $f(L)$ as follows:

   > Transform each production $A \to w$ into a new production of the form $A \to w'$, where w' is obtained from w by replacing each terminal a in w by $f(a)$.

   The new grammar is context-free, and any string in $f(L)$ is derived by this new grammar.

Example

- Show that $L = \{a^n b c^n d e^n : n \geq 0\}$ is not context-free.

  We can define a morphism

  $$f : \{a, b, c, d, e\}^* \to \{a, b, c, d, e\}^*$$

  by

  $$f(a) = a,\ f(b) = \Lambda,\ f(c) = b,\ f(d) = \Lambda,\ f(e) = c.$$

  Then $f(L) = \{a^n b^n c^n : n \geq 0\}$.

  If L is context-free, then we must also conclude that $f(L)$ is context-free.

  But we know that $f(L)$ is not context-free.

  Therefore L is not context-free.