GRAMMAR: AN ALTERNATIVE METHOD FOR SPECIFYING A LANGUAGE

- It use auxiliary symbols *set of variables*, in additional to the alphabet symbols Σ of the language, instead of states in PDA and FSA.
- It uses *substitution rules* instead of transitions in PDA or FSA.
- The most general form of grammar is equivalent to Turing Machine in terms of the capability to specify a language.

CONTEXT-FREE GRAMMAR

Example. Start symbol: S, Terminal Symbols $T = \{a, b\}$

(Substitution) Rules: (i) $S \to ab$

(ii) $S \to aSb$

Context-Free Grammar G**:**

- T = a finite set of *terminal* symbols, which are denoted by lower case letters $a, b, c, \cdots$.
- V = a finite set of *non-terminal* symbols or variables, which are denoted by capital letters $X, Y, \cdots$. The start-symbol $S \in V$.
- The leftside of a rule is a variable, and the rightside of a rule is a string in $(V \cup T)^+$; no λ-rule for now. The number of rules is *finite*.

Rule Application:

- Application of a rule $X \to y$ to a string $uXv \in (V \cup T)^+$ gives the string uyv, denoted by $uXv \Rightarrow uyv$.
- We form strings in T^+ by repeated application of rules, beginning with the start-symbol S.

(1) $S \Rightarrow ab$

(2) $S \Rightarrow aSb \Rightarrow aabb$

(3) $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$

The language $L(G)$ **of a grammar** G**:** $L(G) = \{x \in T^+ \colon S \Rightarrow^+ x\}$.

For the above grammar: $L(G) = \{a^n b^n \colon n \geq 1\} = L_{a^n b^n}$.

Why Context-Free:

- Application of a rule "$X \to y$" to a string uXv containing X does not depend on the contexts u and v of X in uXv.

ANOTHER CFG FOR $L_{a^n b^n}$

Example. $G' = \{S \to aB, B \to b, S \to aC, C \to Sb\}$

- A derivation of *aabb*: $S \Rightarrow aC \Rightarrow aSb \Rightarrow aaBb \Rightarrow aabb$

A Compact Notation: $G' = \{S \to aB \mid aC, B \to b, C \to Sb\}$.

- The rules $\{S \to aB, B \to b\}$ amount to the rule $S \to ab$
- The rules $\{S \to aC, C \to Sb\}$ amount to the rule $S \to aSb$.
- $L(G) = L(G')$.

A context-free grammar is an alternate way of specifying a context-free language.

EXERCISE

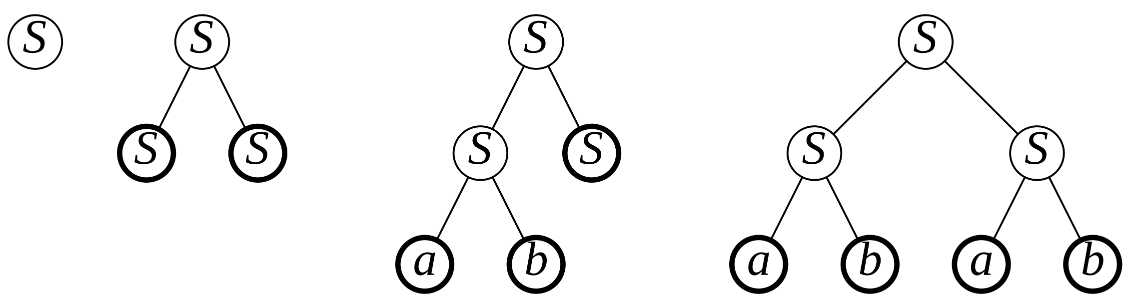
1. Is the grammar $G = \{S \to ab \mid aSb \mid SS\}$ context-free? What is the language $L(G)$ for this grammar?
2. How many derivations of *abab* are there for the grammar in Problem 1? In what way, the role of the third rule differs from that of the other two rules?

PARSE-TREES

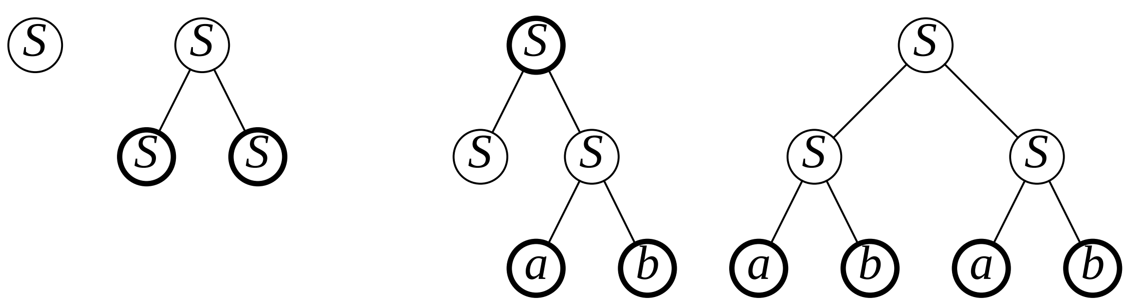
Parse-tree:

- Shows which part of the string $x \in L(G)$ is derived from which variable symbol in the form of a tree-structure.
- Each intermediate tree-node is a variable, whose children (taken in the left to right order) form the rightside of a rule for that variable.
- Each terminal node is a terminal-symbol; these taken together in the left to right order give x.

Example. Let $G = \{S \to ab, S \to SS\}$ and $x = abab$.



$S \Rightarrow SS \Rightarrow abS \Rightarrow abab$

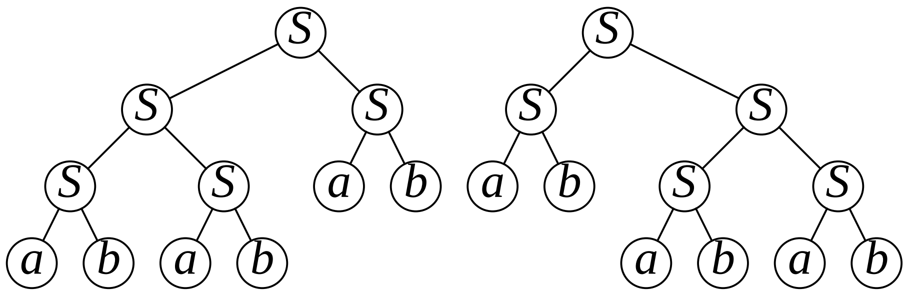


$S \Rightarrow SS \Rightarrow abS \Rightarrow abab$

Two different derivations of $x = abab$ giving the same parse-tree.

EXERCISE

1. How many derivations are there for x = *ababab* for each of the parse-trees below?



2. What is $L(G)$ for this grammar?
3. Give a different CFG G' for the language $L(G)$ such that each $x \in L(G') = L(G)$ has exactly one derivation.

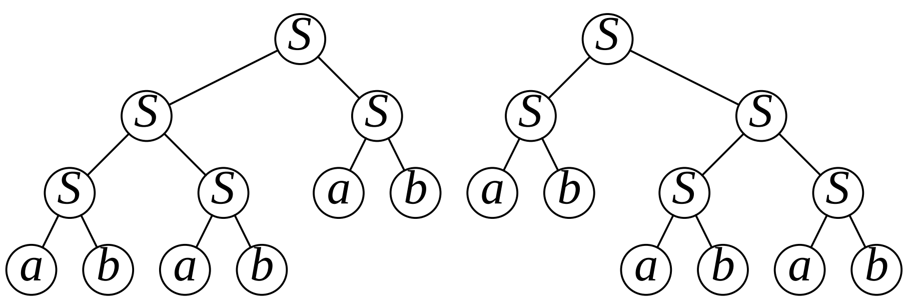
LEFTMOST DERIVATION

Each parse-tree for a string x gives a distinct leftmost derivation of x.

Leftmost Derivation:

- It is related to the *pre-order* traversal of the parse-tree.
- Use the derivation step for a node before those for its children.
- Use the derivation steps for the subtrees in the left to right order.

Example: Consider the parse-trees at the bottom of the previous page.



First parse-tree: $S \Rightarrow SS \Rightarrow SSS \Rightarrow abSS \Rightarrow ababS \Rightarrow ababab$
Second parse-tree: $S \Rightarrow SS \Rightarrow abS \Rightarrow abSS \Rightarrow ababS \Rightarrow ababab$

Ambiguous Grammar:

- A CFG- G is *ambiguous* if some $x \in L(G)$ has more than one parse-tree (i.e., more than one leftmost derivation).
- A CFL L is *ambiguous* if every CFG for it is ambiguous.

AN UNAMBIGUOUS CFG FOR $L_{bal\text{-}par}$

Key Observations:

- If $x \in L_{bal\text{-}par}$ has no non-empty prefix which is also balanced, then $x = ab$ or $x = ayb$, where $y \in L_{bal\text{-}par}$. These strings are generated by starting with the first two productions shown below.
- Otherwise, $x = yz$, where y is the smallest prefix which is in $L_{bal\text{-}par}$; z is also in $L_{bal\text{-}par}$. These strings are generated by starting with the last two productions shown below.

$$G_{bal\text{-}par} = \{S \to ab,\ S \to aSb,\ S \to abS,\ S \to aSbS\}.$$

Examples of unique leftmost derivations:

$x = abab$: $S \Rightarrow abS \Rightarrow abab$

$x = aabbab$: $S \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbab$

a non-leftmost derivation: $S \Rightarrow aSbS \Rightarrow aSbab \Rightarrow aabbab$

$x = ababab$: $S \Rightarrow abS \Rightarrow ababS \Rightarrow ababab$

An unambiguous CFG for $L_{\lambda+bal\text{-}par}$:

- Here, the variable A plays the same role as S before.

$$S \to \lambda,\ S \to A,$$
$$A \to ab,\ A \to aAb,\ A \to abA,\ A \to aAbA.$$

Convention:

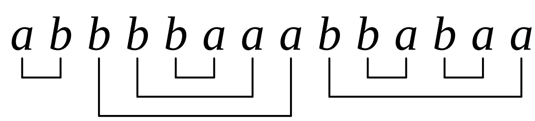
- If $\lambda \in L(G)$, then $S \to \lambda$ is the only rule with λ on the right side and S does not appear on the right side of any rule.

EXERCISE

1. Find an unambiguous CFG for the language $L_{\#a=\#b}$. (Keep the CFG as simple as possible in terms of the number of non-terminals and the productions.) You may find the following properties of the strings in $L_{\#a=\#b}$ helpful; these properties are similar to, but slightly more general than, the properties for balanced parenthetical strings.

   (i) Any string $x \in L_{\#a=\#b}$ can be decomposed uniquely as $x = x_1 x_2 \cdots x_k$, where each $x_i \in L_{\#a=\#b}$ and no proper prefix of x_i belongs to $L_{\#a=\#b}$.

   (ii) If x_i begins with *a*, then it ends with *b*; call such an x_i of type *ab*. Similarly, if it begins with *b*, then it ends with *a*; call such an x_i of type *ba*. (This together with (i) gives us a unique way of matching *a*'s with *b*'s.)



   (iii) If x_i is of type *ab* and $x_i = ay_ib$, then either y_i is also of type *ab* (and has no further decomposition) or its decompositions consists of *ab* type strings only. Similarly for *ba* type strings.

   Run the CFG-simulator for strings of length ≤ 6.

2. Find an unambiguous CFG for the language $L_{sym} = \{x \in (a+b)^+ : x = x^r\}$. Thus, *aa* and $aabbaa \in L_{sym}$, but $ab \notin L_{sym}$.

3. Find an unambiguous CFG for $L_{m \geq n} = \{a^m b^n : m \geq n \geq 1\}$. Do the same for $L_{m \neq n} = \{a^m b^n : m \neq n \text{ and } m, n \geq 1\}$.

4. Find an unambiguous CFG for $L_{m.n,m+n} = \{a^m b^n c^{m+n} : m, n \geq 1\}$.

5. Give an induction argument to show that each string generated by the grammar $S \to ab \mid aSb \mid SS$ has equal number of *a*'s and *b*'s. Also, give an induction argument to show that each string *x*

generated by the grammar has the property that any prefix of x has at least as many a's as the number of b's.

6. Show that the complement of $L_{a^n b^n} = \{a^n b^n \colon n \geq 1\}$, i.e., $(a+b)^* - \{a^n b^n \colon n \geq 1\}$ equals the union of the following languages: a^*, $b(a+b)^*$, $a^+ b^+ a(a+b)^*$, and $L_{m \neq n} = \{a^m b^n \colon m \neq n,\ m, n \geq 1\}$. Use this information to obtain an unambiguous CFG for the complement of $L_{a^n b^n}$.

7. Give an unambiguous CFG for D_2.

8. Argue that the following CFG correctly generates the strings over $\{a, b, c, d\}$ which represent the binary trees with ≥2 nodes (the binary tree with one node corresponds to the string λ). Explain in English what each rule does in relation to the binary trees.

$$
\begin{array}{lll}
S \rightarrow L & L \rightarrow ab & R \rightarrow cd \\
S \rightarrow R & L \rightarrow aSb & R \rightarrow cSd \\
S \rightarrow LR & &
\end{array}
$$

9. Simplify the grammar in Problem 8 by eliminating the variable L; the variables in the new grammar should be only $\{S, R\}$. Then, further simplify the grammar by eliminating the variable R as well, leaving S as the only variable. Explain in English what each rule does in relation to the binary trees.

10. Which of the grammars in Problems 8 and 9 are unambiguous?

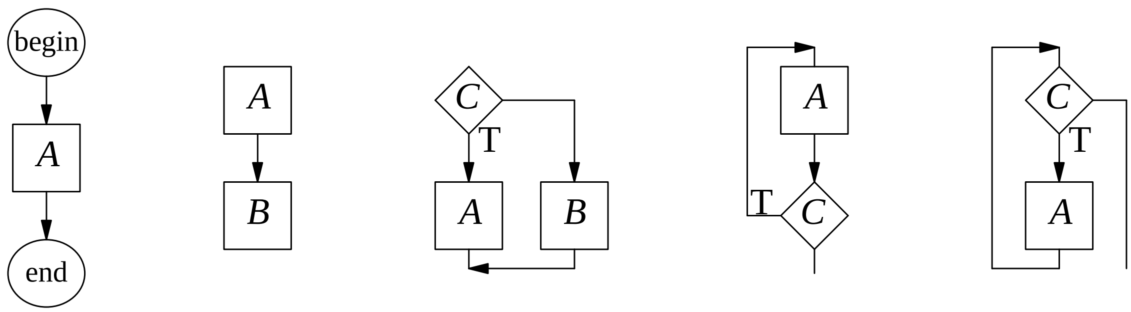
11. Give a CFG for $L_{skyline}$; $\lambda \notin L_{skyline}$.

12. Give an unambiguous CFG for the language $\{10^m \times 10^n = 10^{m+n},\ m, n \geq 1\}$, which represents a special form of binary multiplications. (Hint: First find a CFG for the language $\{10^m \times 1 = 10^m \colon m \geq 1\}$.) Show the leftmost derivation and the parse tree for $10^2 \times 10 = 10^3$.

CFG FOR STRUCTURED-FLOWCHARTS

Structured flowchart:

- Start with (i) and successively expand a box using rules (ii)-(v).



(i) Simplest flowchart. (ii) Sequence. (iii) Decision; *B* may be absent. (iv) Do *A* while *C*. (v) While *C* do *A*.

Associated Grammar Rules (an initial attempt):

- Terminal symbols: d = decision, u = until (do-while), w = while, a = action, and parentheses symbols
- Rule (3.1) is for "if-then-else", with the two N's for "then" and "else" parts; rule (3.2) is for "if-then".

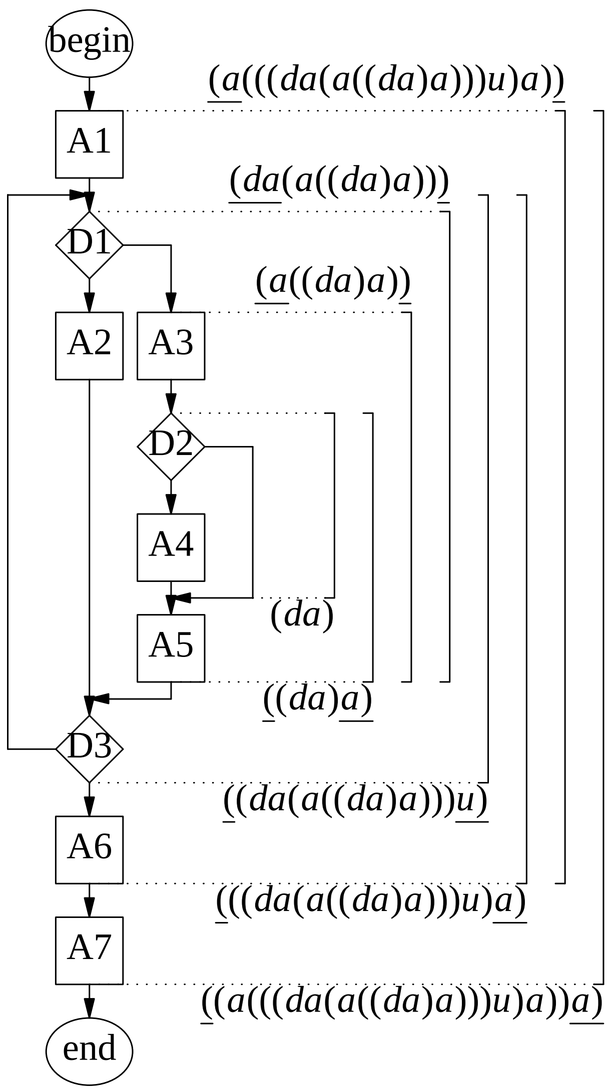
(1.1) $S \rightarrow bNe$ (3.1) $N \rightarrow (dNN)$ (4) $N \rightarrow (Nu)$

(1.2) $N \rightarrow a$ (3.2) $N \rightarrow (dN)$ (5) $N \rightarrow (wN)$

(2) $N \rightarrow (NN)$

AN APPLICATION

(1.1) $S \rightarrow bNe$ (3.1) $N \rightarrow (dNN)$ (4) $N \rightarrow (Nu)$
(1.2) $N \rightarrow a$ (3.2) $N \rightarrow (dN)$ (5) $N \rightarrow (wN)$
(2) $N \rightarrow (NN)$



A possible string representations:
$x = b((a(((da(a((da)a)))u)a))a)e$.
Here, the jth a corresponds to Aj in the flowchart. Similarly, for Dj's.

Two problems:

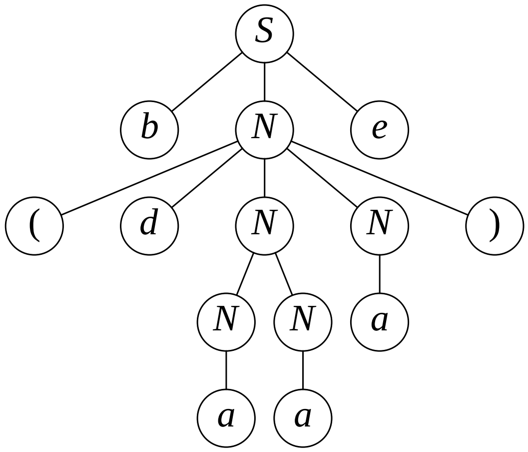
(1) $b(a(aa))e$ and $b((aa)a)e$ give the same flowchart.

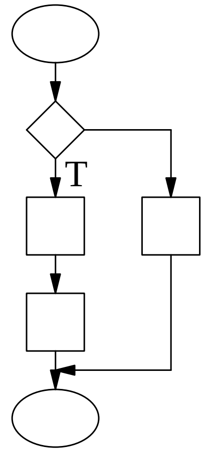
(2) Some of the parentheses is unnecessary.

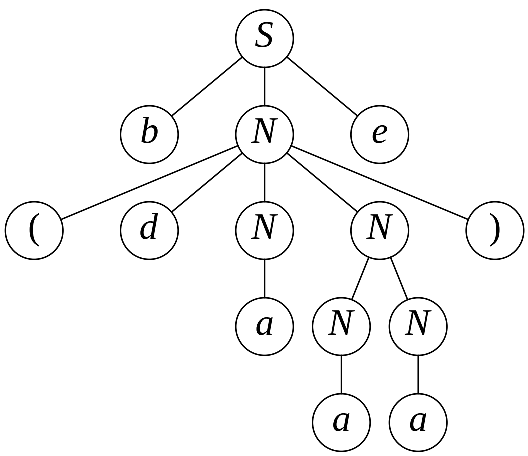
Replacing $N \rightarrow (NN)$ by $N \rightarrow NN$ is not a solution:

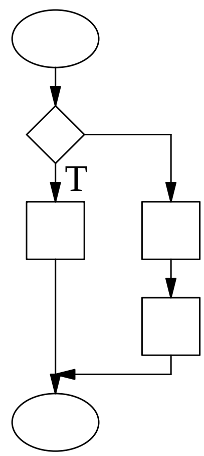
- The string $b(daaa)e$ has four leftmost derivations (parse-trees), giving three different flowcharts.

SOME PARSE-TREES FOR *b*(*daaa*)*e*.









Question:

•? Give another fbwchart and its parse-trees for the same string.

EXERCISE

1. Show all structured flowcharts with three nodes, in addition to the special "begin" and "end" nodes.
2. Show the flowchart diagram, a parse-tree, and the leftmost derivation of $x = b((w(((a(d(a((daa)a))))u)a))a)e$. Label the boxes in the flowchart as A1, A2, etc so that Aj corresponds to the jth a in x; label the branch-nodes in a similar way (Dj corresponding to jth d or u or w).
3. If we replace $N \to (wN)$ by $N \to wN$ and $N \to (Nu)$ by $N \to Nu$ but keep $N \to (NN)$ as it is, does it give rise to the problem of the same string having different parse-trees and giving different flowcharts?
4. Why did we avoid putting specific node names like A1, A2, D1, etc. in our string representation?
5. Argue that the rules $S \to bNe$, $N \to a \mid NN \mid dN{:}\, N) \mid dN) \mid \; (Nu \mid wN)$ avoid both the problems (1)-(2) indicated in the figure. Also, modify this grammar in a simple way to make it unambiguous. (The new grammar will have the property that all flowchart-strings obtained by n aplication of rules will have a total $n+1$ nodes in the flowchart, including "begin" and "end".)
6. Obtain a CFG for flowcharts where we do not have two or more boxes in a sequence such as A6 and A7 on page 10. Keep the grammar unambiguous; there should be a unique string-representation of each structured flowchart with the given restriction.

SIMULATING LEFTMOST DERIVATIONS BY A PUSH-DOWN AUTOMATA

Assume:

- $\lambda \notin L(G)$
- Each production has the form: $B \to bw$, where $b \in T$ and $w \in (V \cup T)^*$. (Such a grammar is said to be in *Greibach normal form*.)

PDA Operation vs. An Application of $B \to bw$**:**

(1) Match the symbol b from the input and replace the top symbol B in the stack by w^r so that the leftmost symbol in w becomes the top of the stack, if $w \neq \lambda$.

(2) If the first symbol in w is a terminal symbol c (and $B \to bw$ was part of a successful derivation of the input), then the next symbol in the input is c and the next move of PDA matches it off with c from the top of stack.

Relationship of Stack with Leftmost Derivation of $x = yz \in L(G)$**:**

- There is a leftmost derivation $S \Rightarrow^+ yw \Rightarrow^+ yz$.
- There is a successful (accepting) processing of x where after reading the initial part y the stack = w^r (leftmost symbol in w being the top of stack).
- The PDA will have only two states q_0 and q_1, and $q_1 \in F$; q_0 is also a finial state if and only if $\lambda \in L(G)$.
- The PDA is in state q_1 after the first move.

EXAMPLE OF SIMULATION

- $G = \{S \rightarrow ab, S \rightarrow aSb, S \rightarrow abS, S \rightarrow aSbS\}$ and $x = aabbab$.

(Pretend initially stack = S.)

Derivation step ($y. w$)		Stack	State	Remainder of input string (= z)
	S	λ	q_0	$aabbab$
(rule: $S \rightarrow aSbS$)	$a{\cdot}SbS$	SbS	q_1	$abbab$
(rule: $S \rightarrow ab$)	$aa{\cdot}bbS$	Sbb	q_1	$bbab$
	$aab{\cdot}bS$	Sb	q_1	bab
	$aabb{\cdot}S$	S	q_1	ab
(rule: $S \rightarrow ab$)	$aabba{\cdot}b$	b	q_1	b
	$aabbab{\cdot}\lambda$	λ	q_1	λ

Formal description of the PDA-transitions:

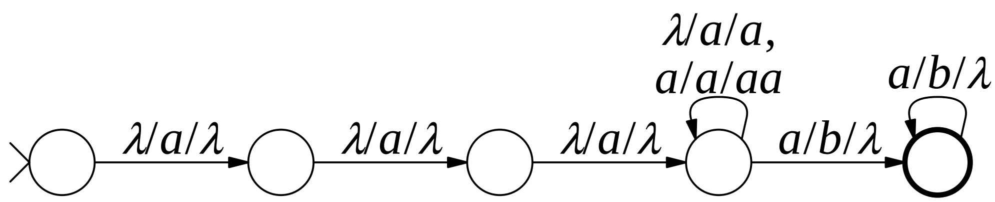
- An input string x is accepted by this PDA in as many ways as the number of leftmost derivations of x.

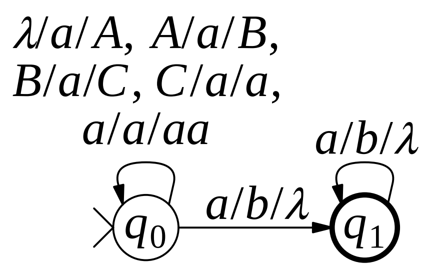
Production	Transitions	Comment
$S \rightarrow a$	$\delta(\lambda, q_0, a) = (q_1, \lambda)$	Stack still remains empty
	$\delta(S, q_1, a) = (q_1, \lambda)$	S removed from stack
$S \rightarrow aw$	$\delta(\lambda, q_0, a) = (q_1, w^r)$	w^r is added to empty stack
	$\delta(S, q_1, a) = (q_1, w^r)$	w^r replaces top(stack)=S
$B \rightarrow b$	$\delta(B, q_1, b) = (q_1, \lambda)$	top(stack) = B is removed
$B \rightarrow bw$	$\delta(B, q_1, b) = (q_1, w^r)$	B in stack is replaced by w^r
	$\delta(c, q_1, c) = (q_1, \lambda)$	c = top(stack) is removed

- The minimization of the number of states for a PDA is no longer meaningful. (The use of stack eliminates the problem.)

REDUCING STATES OF A PDA BY USING THE STACK

Two PDAs for $L_{m=n+3}$ (the second one has 2 states):





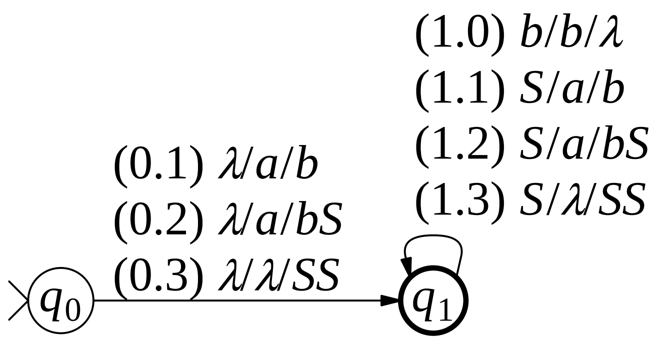
Stack	State	Input (remaining)
λ	q_0	$aaaaabb$
A	q_0	$aaaabb$
B	q_0	$aaabb$
C	q_0	$aabb$
a	q_0	abb
aa	q_0	bb
a	q_1	b
λ	q_1	λ

- Given any PDA, there is a CFG which gives the same language.
- Given any CFG, there is a CFG for that language which is in Greibach Normal Form.
- Given any Greibach Normal Form CFG, there is a PDA for that language with at most 2 states.

λ-MOVES IN PDA AND NON-GREIBACH FORM RULES

λ-move: A move (transition) when no input symbol is consumed. One or both of the stack and the state may be altered in the process.

(1) $S \to ab$
(2) $S \to aSb$
(3) $S \to SS$



- Simulation by PDA for the derivastion:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbSS \Rightarrow aabbabS \Rightarrow aabbabab.$$

(Two λ-moves for two applications of $S \to SS$.)

Transition	Stack	State	Remaining input	
(0.3)	λ	q_0	$aabbabab$	(λ-move)
(1.2)	SS	q_1	$aabbabab$	
(1.1)	SbS	q_1	$abbabab$	
(1.0)	Sbb	q_1	$bbabab$	
(1.0)	Sb	q_1	$babab$	
(1.3)	S	q_1	$abab$	(λ-move)
(1.1)	SS	q_1	$abab$	
(1.0)	Sb	q_1	bab	
(1.1)	S	q_1	ab	
(1.0)	b	q_1	b	
	λ	q_1	λ	

REGULAR GRAMMAR

A special case of CFG:

- The rightside of a rule consists of a terminal followed by at most one variable (cf. GNF and CNF).

$$(1) \quad A \rightarrow a$$
$$(2) \quad A \rightarrow aB$$

- More general rules, having more than one terminal in (1) or in (2) preceding the variable, can be converted to the special form:

$A \rightarrow abc$	can be replaced by	$A \rightarrow aC,\ C \rightarrow bD,\ D \rightarrow c$
$A \rightarrow abcB$	can be replaced by	$A \rightarrow aE,\ E \rightarrow bF,\ F \rightarrow cB$

Caution: Do not mix right linear and left linear rules.

G_1:		G_2:
$S \rightarrow ab$ $S \rightarrow aB$ (right linear) $B \rightarrow Sb$ (left linear)	is equivalent to (gives the same language)	$S \rightarrow aA,\ A \rightarrow b$ $S \rightarrow aSb$

- Both G_1 and G_2 are CFG (but not RG), and $L(G_1) = L_{a^n b^n} = L(G_2)$.

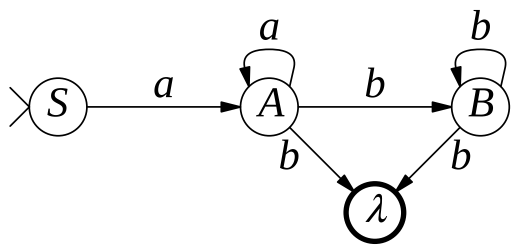
Regular Grammar for $L_{a^m b^n}$ ($m, n \geq 1$):

$$S \rightarrow aA$$
$$A \rightarrow aA \qquad A \rightarrow b \qquad A \rightarrow bB$$
$$B \rightarrow bB \qquad B \rightarrow b$$

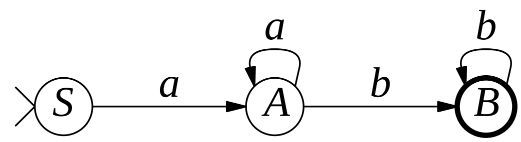
REGULAR GRAMMAR vs. FSA

Regular Grammar for $L_{a^m b^n}$ ($m, n \geq 1$):

$$S \to aA, \quad A \to b \mid aA \mid bB, \quad B \to b \mid bB$$



(i) NFSA for the regular grammar.



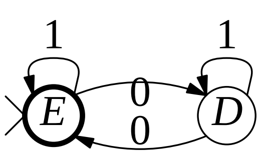
(ii) Deterministic form of (i).

- The states are the variables, including a special final-state = λ; S = start-state.
- There is one transition $\delta(A, a, B)$ for each type-(1) rule $A \to aB$.
- For each type-(2) rule $A \to a$ create a transition $\delta(A, a, \lambda)$ to a special final-state λ.

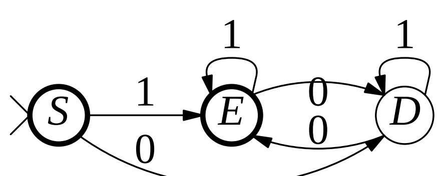
From FSA to Regular Grammar:

- Convert FSA to an equivalent (N)FSA with a new start-state (call it S) and no transition to start-state and also a special and the only final-state (call if "λ") from which there are no transitions.

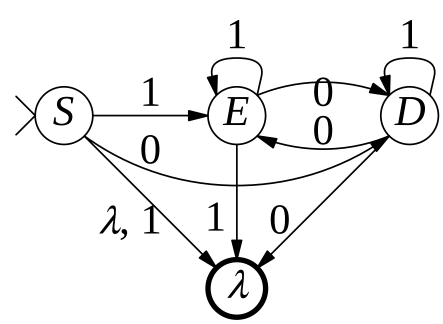
  Note that for each $\delta(q, a) = q' \in F$, there is $\delta(q, a) = \lambda$ now.
- Create the grammar rules accordingly



$M_{0\text{-}even}$



(ii) An intermediate form with no transition to start-state.



(iii) The final NFSA.

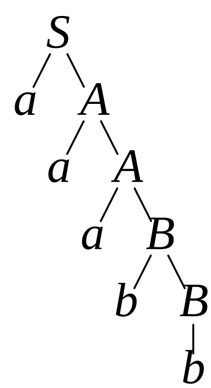
$S \to \lambda \mid 1 \mid 0D \mid 1E, \quad E \to 1 \mid 0D \mid 1E,$
$D \to 0 \mid 0E \mid 1D$

SIMULATING DERIVATION OF A REGULAR GRAMMAR BY A PDA

Another RG for {$a^m b^n$: m, $n \geq$ 1} = $a^+ b^+$:

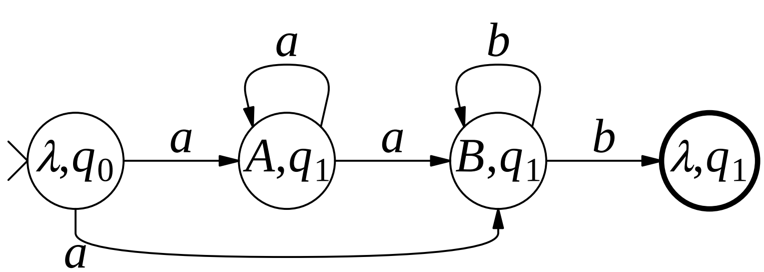
$S \to aA \mid aB, \quad A \to aA \mid aB, \quad B \to bB \mid b$

$S \Rightarrow aA \Rightarrow aaA \Rightarrow aaaB \Rightarrow aaab$
(Each derivation is now a leftmost derivation.)



- The stack now has ≤ 1 symbol at any time in the PDA-simulation.
- This stack information can be kept in a state of the FSM.

Stack	State	Remaining Input
λ	q_0	$aaabb$
A	q_1	$aabb$
A	q_1	abb
B	q_1	bb
B	q_1	b
λ	q_1	λ

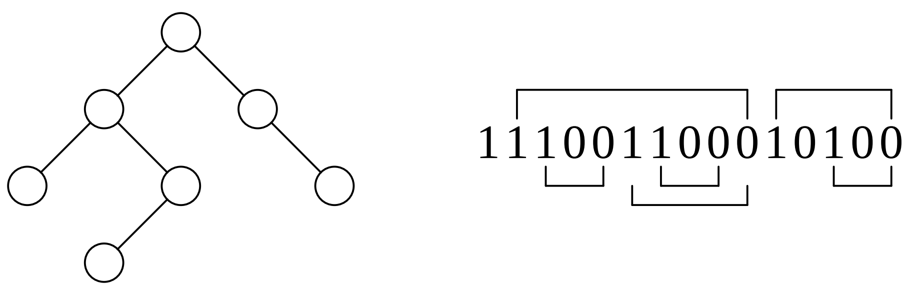


An NFSA

EXERCISE.

1. Consider the grammar $G = \{S \rightarrow ab,\ S \rightarrow SS\}$. Write the transitions for the PDA to simulate leftmost derivations in G. Show all possible move sequences leading to the acceptance of $x = ababab$.

2. Consider the string representation of a binary tree as illustrated below. This is obtained by a preorder traversal of the tree: node, left subtree, and right subtree, where we write for each node present and 0 for each missing child-node of node.



Design a PDA which accepts only those binary strings which represents a non-empty binary tree. Draw the trees corresponding to the valid strings of length ≤ 10. Verify your construction using the PDA-simulator, and show your program outputs (including the transitions). (The program takes quite a bit of time to print the strings of length > 6.)

3. Is there a PDA to test if the binary tree is symmetric? How about testing if the tree is completely balanced?

4. Given any context free grammar G, consider the language $L(G)$ and the PDA $P(G)$, which simulates the leftmost derivations of strings in $L(G)$. Let $Stack(x) = \{s\colon s =$ stack at some point in processing of $x\}$; here the first symbol in s is the bottom of stack. Thus, for the grammar $\{S \rightarrow ab$ and $S \rightarrow aSb$ for the language $L_{a^nb^n}$ (which does not contain λ) and $x = aaabbb$, $Stack(x) = \{\lambda, bS, bbS, bbb, bb, b\}$. Note that we consider $Stack(x)$ only for $x \in L(G)$. The set $Stack(x)$ has the property that it contains all prefixes of each string

in it; such a set of strings is called *prefix-closed*. Finally, we define $Stack(G)$ by

$$Stack(G) = \bigcup_{x \in L(G)} Stack(x).$$

Clearly, $Stack(G)$ is also prefix-closed. For the above grammar, $Stack(G) = b^* + b^+S$.

Show that $Stack(G)$ is regular for any CFG G by finding a regular grammar G' for $Stack(G)$. Assume for simplicity that each rule of G has the property that the righthand side of each rule begins with a terminal symbol and hence $\lambda \notin L(G)$. (Hint: In G', allow general rules of the form $A \rightarrow cde \cdots fB$ or $A \rightarrow cde \cdots f$, with more than one terminal symbols before the non-terminal symbol (if any) on the right. Be careful about determining your terminal and non-terminal symbols for G'. The regular grammar G' you are looking for is closely related to G, or more precisely, its parse-trees. Focus on the strings in $Stack(x)$, and in $Stack(G)$, that correspond to the situations when the stack grows. Once you obtain G' for these strings, and hence an FSM for them, you can easily modify that FSM to accept initial parts of those strings; the latter will cover the situations where the stack shrinks. You need to show how to create G' from G.)

Verify your method for, say, G for L_{sym} and $L_{bal\text{-}par}$.