

Deterministic Push-Down Store Automata

1 Context-Free Languages

$$\begin{aligned}A &\longrightarrow B C D \dots \\A &\longrightarrow w_0 w_1 D \dots E\end{aligned}$$

The language $a^n b^n$:

$$\begin{aligned}S &\longrightarrow a S b \\S &\longrightarrow a b\end{aligned}$$

2 Finite-State Automata

3 Pushdown Automata

Pushdown automata (pda's) is an fsa with an auxiliary tape in addition to the input tape. This tape is a *stack* (old-fashioned term: pushdown store). Things are written onto and read off the stack. The order is always last in, first out [this is what makes it a stack, as opposed to a queue, last in, last out].

1. Finite set of states: K
2. Start state: q_0
3. Set of final states: F
4. Set of transitions Δ of the form $(q_i, a, A) \rightarrow (q_j, \gamma)$
 - (a) q_i and q_j are states
 - (b) a is a member of the input alphabet Σ
 - (c) A is a symbol of the stack alphabet Γ and γ is a string of symbols in the stack alphabet.

This means, when in state q_i , if symbol a is encountered in the input and the stack has symbol A on top, then move to state q_j and replace A with the symbols γ on the stack.

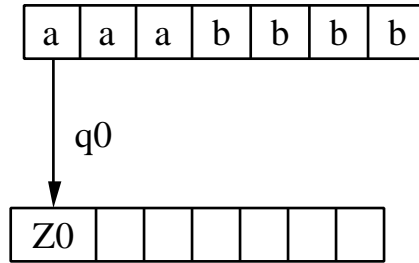


Figure 1: A Push-Down Automaton

Acceptance has 3 conditions:

1. The pda is in an accept state;
2. The entire input has been read;
3. The stack is empty.

A pda that accepts $a^n b^n$:

States:	$K = \{q_0, q_1\}$
Input alphabet:	$\Sigma = \{a, b\}$
Stack alphabet:	$\Gamma = \{A\}$
Initial state:	q_0
Final states:	$F = \{q_0, q_1\}$
Transitions	$\Delta = \left\{ \begin{array}{l} (q_0, a, e) \rightarrow (q_0, 1) \\ (q_0, b, 1) \rightarrow (q_1, e) \\ (q_1, b, 1) \rightarrow (q_1, e) \end{array} \right\}$

4 Another pda

This pda accepts the language

$$\{xx^R \mid x \in \{a, b\}^*\}$$

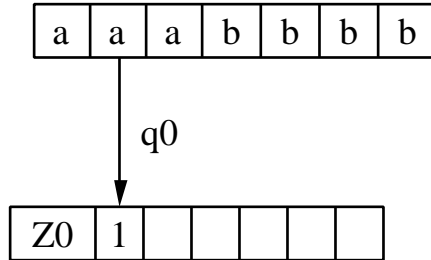


Figure 2: A Push-Down Automaton for counting as and bs: :Step 1

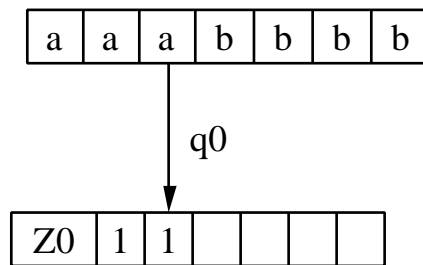


Figure 3: Step 2

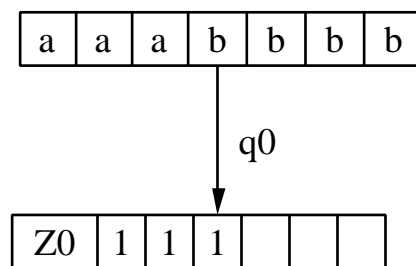


Figure 4: Step 3

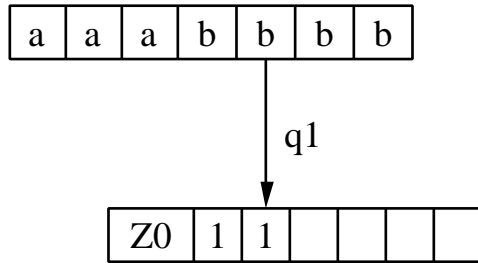


Figure 5: Step 4

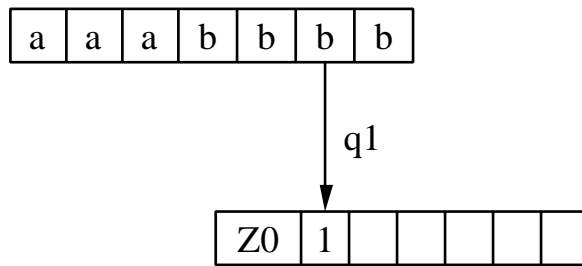


Figure 6: Step 5: Popping the last thing off the stack

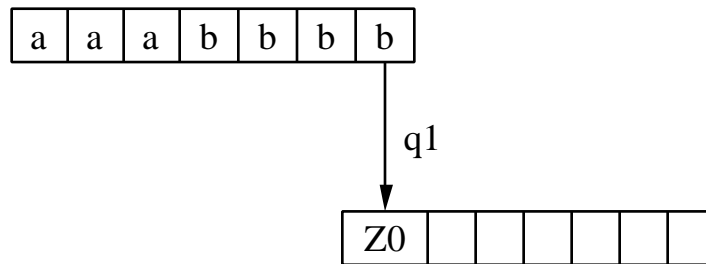


Figure 7: An unhappy ending: the stack must have a 1 at the top when we read a b

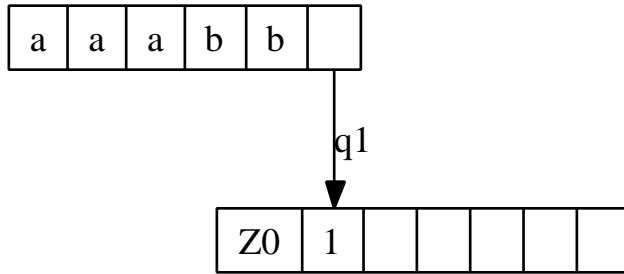


Figure 8: Another unhappy ending: must have empty stack at end of input

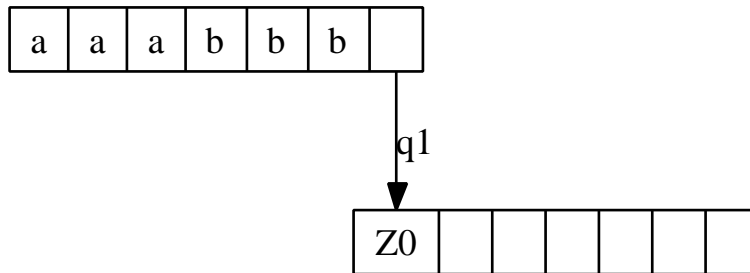
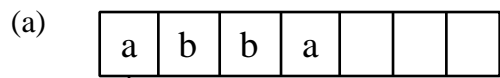


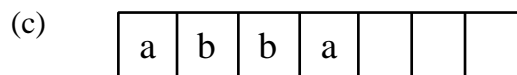
Figure 9: A happy ending

$$\begin{array}{ll}
\text{States:} & K = \{q_0, q_1\} \\
\text{Input alphabet:} & \Sigma = \{a, b\} \\
\text{Stack alphabet:} & \Gamma = \{A, B\} \\
\text{Initial state:} & q_0 \\
\text{Final states:} & F = \{q_0, q_1\} \\
\text{Transitions} & \Delta = \left\{ \begin{array}{cc} \text{pushing} & \text{popping} \\ \hline (1) & (q_0, a, e) \rightarrow (q_0, A) & (3) & (q_0, a, A) \rightarrow (q_1, e) \\ (2) & (q_0, b, e) \rightarrow (q_0, B) & (4) & (q_0, b, B) \rightarrow (q_1, e) \\ & & (5) & (q_1, a, A) \rightarrow (q_1, e) \\ & & (6) & (q_1, b, B) \rightarrow (q_1, e) \end{array} \right\}
\end{array}$$

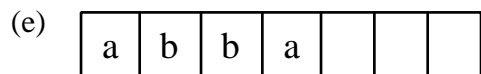
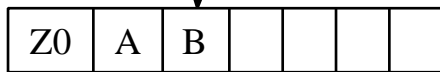
1. Non-deterministic: In state q_0 , reading an a with A on the stack, either instruction (1) or (3) is legal.
2. Likewise, for instructions (2) and (4), reading a b with B on the stack,
3. Pushing rules push something **onto** the stack; popping rules pop something **off** the stack. q_0 is the pushing state; q_1 the popping state.
4. In rules (3) and (4) the machines moves from pushing to popping states. In effect, when it uses these transitions, it is guessing that it has reached the exact middle of the palindrome. From now on, it needs to use rules (5) and (6) and read its way through the rest of the string to an empty stack.
5. What is the effect of having q_0 be a final state?



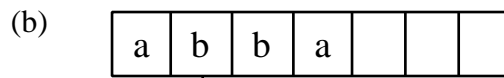
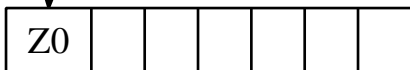
q0



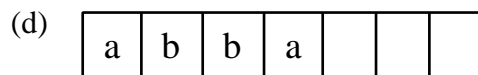
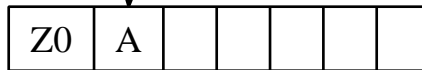
q0



q1



q0



q1



5 Facts

1. The class of deterministic pda languages (Deterministic Context Free or LR(1) languages), of particular interest to people who work on programming languages and compilers, is a subset of the class of context-free languages.
2. a^n, b^n is a deterministic pda language.
3. xx^R is a nondeterministic pda language.
4. The class of pda languages is equivalent to the class of context-free languages.

6 Context-free grammars

In a context free grammar, every rule is of the form

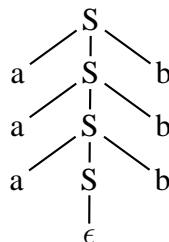
$$\begin{aligned} A &\rightarrow \psi \\ \psi &\in V_T \cup V_N \\ \psi &\text{ may be the empty string} \end{aligned}$$

Examples

$$\begin{aligned} S &\rightarrow NP VP \\ NP &\rightarrow Det N \\ PP &\rightarrow P NP \\ NP &\rightarrow NP PP \quad \text{Recursive!} \\ N &\rightarrow \text{dog} \quad \text{Lexical} \\ V &\rightarrow \text{walks} \quad \text{Lexical} \\ S &\rightarrow a S a \quad \text{also okay} \end{aligned}$$

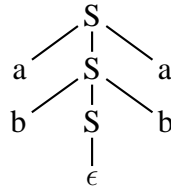
A grammar for $a^n b^n$:

$$\begin{aligned} S &\rightarrow a S b \\ S &\rightarrow \epsilon \end{aligned}$$



A grammar for xx^R , $x \in \{a, b\}^*$:

$$\begin{aligned} S &\rightarrow a S a \\ S &\rightarrow b S b \\ S &\rightarrow \epsilon \end{aligned}$$



For every pda there is a context-free grammar and for every context free grammar a pda. Proof not given.

Procedure: We go from a CFG G to a pda M .

1. Let $G = \langle V_N, V_T, S, R \rangle$
2. The states of M are q_0 and q_1 .
3. q_1 is the sole final state.
4. The input alphabet is V_T
5. The stack alphabet is $V_T \cup V_N$
6. **Start Symbol:** M contains the instruction $(q_0, e, e) \rightarrow (q_1, S)$ Put the start symbol on the stack and enter state q_1 .
7. **Rule:** For each rule $r = A \rightarrow \psi$, M contains

$$(q_1, e, A) \rightarrow (q_1, \psi)$$

Replace the symbol A on the stack with the right hand side of the rule.

8. **Terminal Symbol:** For each symbol $a \in V_T$, M contains the instruction:

$$(q_1, a, a) \rightarrow (q_1, e)$$

Pop a terminal off the stack if it matches the input.

The pda operates very much like (if not exactly like) a top down parser/recognizer.

Example: The pda generated from the above grammar for $a^n b^n$ is

States: $K = \{q_0, q_1\}$
 Input alphabet: $\Sigma = \{a, b\}$
 Stack alphabet: $\Gamma = \{S, a, b\}$
 Initial state: q_0
 Final states: $F = \{q_1\}$

Transitions $\Delta = \left\{ \begin{array}{l} \text{Instruction} \\ \text{Justification} \\ \hline (1) \quad (q_0, e, e) \rightarrow (q_1, S) \quad \text{Start symbol} \\ (2) \quad (q_1, e, S) \rightarrow (q_1, aSb) \quad S \rightarrow aSb \\ (3) \quad (q_1, e, S) \rightarrow (q_1, e) \quad S \rightarrow \epsilon \\ (4) \quad (q_1, a, A) \rightarrow (q_1, e) \quad \text{Terminal symbol} \\ (5) \quad (q_1, b, B) \rightarrow (q_1, e) \quad \text{Terminal symbol} \end{array} \right\}$

Derivation:

	Read	State	To be read	Stack	Instruction
(a)	ϵ	q_0	$aabb$	ϵ	
(b)	ϵ	q_1	$aabb$	S	(1)
(c)	ϵ	q_1	$aabb$	aSb	(2)
(d)	a	q_1	abb	Sb	(4)
(e)	a	q_1	abb	$aSbb$	(2)
(f)	aa	q_1	bb	Sbb	(4)
(g)	aa	q_1	bb	bb	(?)
(h)	aab	q_1	b	b	(?)
(i)	$aabb$	q_1	ϵ	ϵ	(?)

7 Some more facts

1. CFLs are closed under union, concatenation, and Kleene star.
2. CFLs are not closed under intersection, for example:

$$L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$$

is context-free. And so is:

$$L_2 = \{a^k b^l c^l \mid k, l \geq 0\}$$

The grammar for L_1 is:

$S \rightarrow BC$
 $B \rightarrow aBb$
 $B \rightarrow \epsilon$
 $C \rightarrow cC$
 $C \rightarrow c$

The grammar for L_2 is very similar. The intersection of these two languages is:

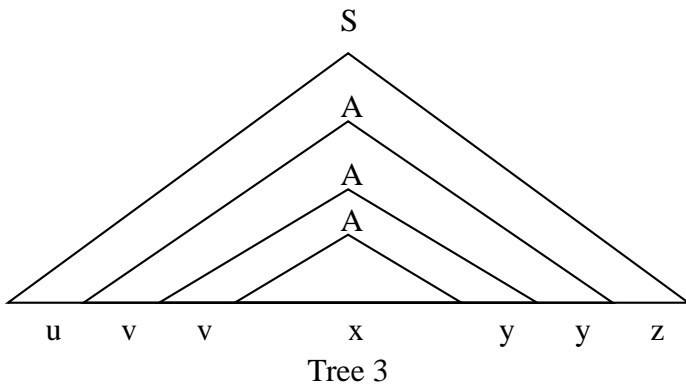
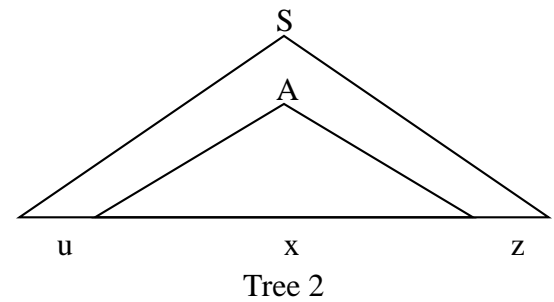
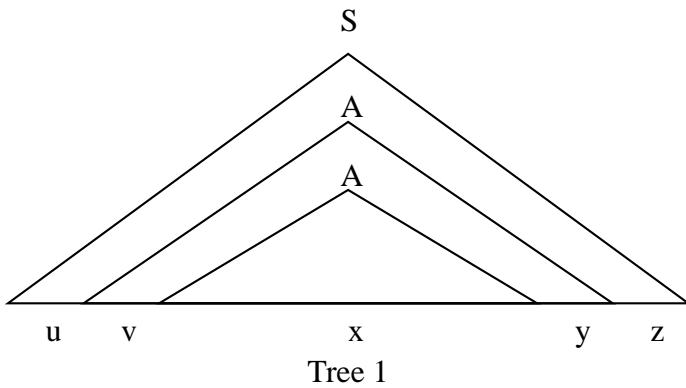
$$L_3 = \{a^i b^i c^i \mid i \geq 0\}$$

This is provably non-context-free (Pumping Theorem, 18.3).

3. The intersection of a CFL with a regular language is always a CFL. Useful for proofs. Is the language that contains all and only strings with the same number of a's, b's and c's context-free? No, because its intersection with regular language $a^*b^*c^*$ is the language L_3 above. [This trick will also be important when we turn to English].
4. The CFLs are not closed under complementation (this actually follows from the fact that they are not closed under intersection but are closed under union).

8 Pumping Lemma

Height of a tree	h	longest path extending continuously from root to leaves
Max width	n	The maximum number of symbols on the RHS of a rule in grammar G .
Width bound	n^h	The maximum width of a parse tree of height h is n^h
	m	$ V_N $
		Any parse tree of width greater than n^m must have at least one repeated non terminal in it (call it A).
		Consider such a parse tree and let x be the terminal string dominated by the lower occurrence of A and vx the terminal string dominated by the upper occurrence of A (tree 1)
		The grammar must also admit tree 2.
		The grammar must also admit tree 3.
		The grammar admits uv^nxy^nz , $n > 0$



The case $L = a^n b^n c^n, n > 0$.

1. Let K be the length bound past which a CFL tree must have repetitions.
2. Choose $a^K b^K c^K$.
3. If L is context-free then $a^K b^K c^K$ is factorable into $uvxyz$ such that for all $n > 0, uv^n xy^n z \in L$
 - (a) v cannot consist of both a's and b's because when pumped it would produce b's before a's.
 - (b) v cannot consist of both b's and c's because when pumped it would produce c's before b's.
 - (c) v must consist of just as , just bs , or just cs .
 - (d) Therefore when we pump v and y simultaneously, the result is strings not in L . For example, if v is a's and y is b's, then we have a string with more a's and b's than c's. And so on for all the other possibilities.