

Inf2A: CFGs and PDAs

Stuart Anderson

School of Informatics
University of Edinburgh

October 29, 2009

Context-free Languages

- Context-free languages are widely used to describe simple languages utilizing parenthesis structure.
- They are very widely studied and many properties of the class are known.
- Algorithms for fast parsing of such languages have been studied extensively.
- Over the next three weeks we consider some of the work on context-free languages.

A simple CFG

$S \rightarrow NP VP$	$Det \rightarrow \text{the}$	$N \rightarrow \text{cat}$	$V \rightarrow \text{sleeps}$
$NP \rightarrow Det N$	$Det \rightarrow \text{a}$	$N \rightarrow \text{dog}$	$V \rightarrow \text{barks}$
$VP \rightarrow V Mod$	$Det \rightarrow \text{my}$	$Mod \rightarrow \text{well}$	$Mod \rightarrow \text{loudly}$

A derivation

$S \Rightarrow NP VP$

$\Rightarrow Det N VP$

$\Rightarrow the N VP$

$\Rightarrow the\ dog\ VP$

$\Rightarrow the\ dog\ V\ Mod$

$\Rightarrow the\ dog\ barks\ Mod$

$\Rightarrow the\ dog\ barks\ loudly$

using $S \rightarrow NP VP$

using $NP \rightarrow Det N$

using $Det \rightarrow the$

using $N \rightarrow dog$

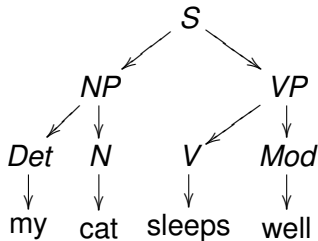
using $VP \rightarrow V Mod$

using $V \rightarrow barks$

using $Mod \rightarrow loudly.$

Another Derivation and its Tree

$S \Rightarrow NP VP$
 $\Rightarrow Det N VP$
 $\Rightarrow my N VP$
 $\Rightarrow my cat VP$
 $\Rightarrow my cat V Mod$
 $\Rightarrow my cat sleeps Mod$
 $\Rightarrow my cat sleeps well.$



Languages that are C-F but are not Regular

- $\{a^n b^n \mid n \geq 0\}$,
- palindromes over $\{a, b\}^*$,
- sequences of palindromes each of length greater than one,
- balanced strings of parentheses.

Can you demonstrate they are not Regular? (How?)

Can you demonstrate they are Context-Free? (How?)

Getting rid of ϵ and unit Productions

- Unclear whether we are “making progress” when we apply a production. Without these productions we either increase the number of terminal symbols or increase the length of the string each time we apply a production.
- If we have a CFG $G = (N, \Sigma, P, S)$ then we make a new set of productions \hat{P} that includes P and also \hat{P} satisfies two rules:
 - If $A \rightarrow \epsilon$ is in \hat{P} and $B \rightarrow \alpha A \beta$ is also in \hat{P} then $B \rightarrow \alpha \beta$ is in \hat{P} .
 - If $A \rightarrow C$ is in \hat{P} and $B \rightarrow \alpha A \beta$ is also in \hat{P} then $B \rightarrow \alpha C \beta$ is in \hat{P} .
- ϵ - and unit productions in \hat{P} are *redundant* because for any derivation using such a production we can find a shorter derivation without that use (Kozen 21.3)

Converting to Griebach Normal Form (sketch - see Kozen Ch 21)

- Starting with a grammar $G_0 = (N_0, \Sigma, P_0, S_0)$ convert it to a grammar $G = (N, \Sigma, P, S)$ in Chomsky Normal Form (possibly $L(G_0) - L(G) = \{\varepsilon\}$)
- Write $\alpha \xrightarrow[G]{L} \beta$ if β can be derived from α in G by rewriting only the *leftmost* symbol in any sentential form.
- Define $R_{A,a} = \{\beta \in N^* \mid A \xrightarrow[G]{L} a\beta\}$ and observe that this is a regular set.
- Define a grammar $G_{A,a}$ with start symbol $T_{A,a}$ with $L(G_{A,a}) = R_{A,a}$ and all the rules are of the form $X \rightarrow BY$ or $X \rightarrow \varepsilon$ where X and Y are nonterminals of $G_{A,a}$ and B is a nonterminal of G .

Conversion to Griebach Normal Form (ctd)

- Assume all the nonterminals of G and $G_{A,a}$ are disjoint and combine all the grammars into a new grammar G_1 with the same start symbol as G . This generates the same language as G because the new rules are not accessible from the start symbol. The rules in G_1 are all of the form $X \rightarrow b$, $X \rightarrow \varepsilon$, or $X \rightarrow BY$ where B is a non-terminal of G
- Derive a new grammar G_2 from G_1 by removing each production of the form $X \rightarrow BY$ with the set of productions $\{X \rightarrow bT_{B,b}Y \mid b \in \Sigma\}$.
- Derive a new grammar G_3 from G_2 by removing ε -productions and the resulting grammar is in GNF (because all the productions except ε productions contain terminals).

Conversion to Griebach Normal Form - Example

- $S \rightarrow AB \mid AC \mid SS, \quad C \rightarrow SB, \quad A \rightarrow (, \quad B \rightarrow)$
- $R_{S,(} = (B + C)S^*, \quad R_{C,(} = (B + C)S^*B, \quad R_{A,(} = R_{B,(} = \{\varepsilon\}$
rest are \emptyset
- $T_{S,(} \rightarrow BX \mid CX, \quad X \rightarrow SX \mid \varepsilon \dots$
- $S \rightarrow (T_{A,(}B \mid (T_{A,(}C \mid (T_{S,(}S, \quad C \rightarrow (T_{S,(}B, \quad A \rightarrow (, \dots$

Derivations with GNF Grammars

- Consider the GNF grammar G with rules:
 $S \rightarrow (SA \mid \#, \quad A \rightarrow$
- Suppose we want to check that $(((\#)))$ is in $L(G)$
- $S \Rightarrow (SA \Rightarrow ((SAA \Rightarrow (((SAAA \Rightarrow (((\#AAA \Rightarrow (((\#)AA \Rightarrow$
 $(((\#))A \Rightarrow (((\#)))$

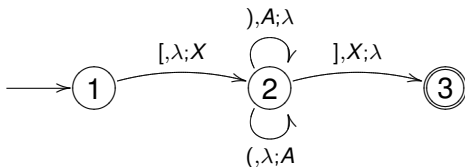
Extending Automata

- FSAs are not adequate to recognise context-free languages (pumping lemma).
- How can we generalise FSAs to accept context-free languages?
- We extend FSAs with a single *stack*, this is called a pushdown automaton.
- Now we meet pushdown automata and see how we can use them, systematically, to create a recogniser for any language described by a CFG.

What is a Pushdown Automaton (PDA)?

- We augment an FSA (deterministic/non-deterministic, with or without ϵ -transitions) with a single *stack* of symbols drawn from some alphabet.
- The automaton is augmented so it can read the next input symbol and the top of stack symbol.
- The automaton can also manipulate the stack by pushing and popping symbols from the stack.
- A *transition* becomes more complex because we have more to do. Using the current state, input symbol, and top of stack symbol we decide what to do and that comprises the specification of the next state and what we do to the stack (either push, pop, or leave unchanged)

Example



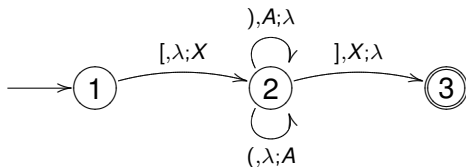
- Transitions have the form $a, A; \beta$ where a, A is the current input symbol (or ε) and the current top of stack symbol (or λ) and β is the new sequence of stack symbols to be written on the top of the stack.
- We use λ as the empty stack symbol so there are three kinds of changes we can make to the stack:
 - $a, A; \lambda$ pops an A symbol from the stack,
 - $a, \lambda; \beta$ pushes the string β onto the stack, and
 - $a, A; \beta$ replaces the A on top of the stack with β .

Configurations and Computations

- A *configuration* of a PDA P is written (q, x, γ) where q is the current state, x is the remaining input to be read, and γ is the stack written as a string with the leftmost symbol on the top of stack.
- A computation of a PDA P is any sequence of configurations c_1, \dots, c_n where c_{i+1} is derived from c_i by applying a transition to it.
- For example $c' = (q', x, \beta\gamma)$ is derived from $c = (q, ax, A\gamma)$ in P if there is a transition labelled $a, A; \beta$ between q and q' in P .

Configurations and Computations - continued

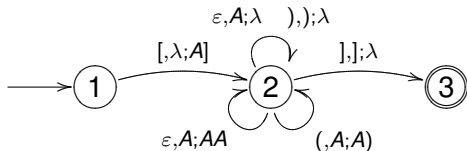
- A string w is accepted by PDA P if there is a computation from (q_0, w, \perp) (where $\perp \in \Gamma$ is the sole initial symbol on the stack) whose last configuration is $(q_f, \varepsilon, \gamma)$ where q_0 is an initial state of P and q_f is a final state.
- In some cases we may choose a variant definition of acceptance - a string is accepted on *empty stack* if it causes the machine to remove the initial instance of \perp from the stack.



- Consider the initial configuration: $(1, [(()), \lambda)$
- $(2, ()), X$
- $(2, ()), AX$
- $(2,))], AAX$
- $(2,)], AX$
- $(2,], X$
- $(3, \varepsilon, \lambda)$
- What is the language recognised? What is a failing string?

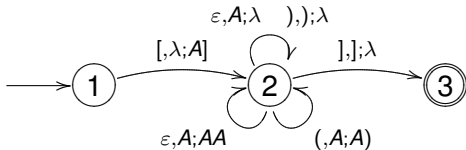
Example: CFGs and PDAs

- Consider a grammar with productions:
 $S \rightarrow [A]; A \rightarrow \varepsilon \mid AA \mid (A)$
- Consider constructing a three state PDA:



- Note, we have introduced ε transitions here
- Also we have allowed the PDA to push more than one symbol at a time.
- This is a non-deterministic PDA - what challenges does that have for the notion of acceptance in practice?

Example: Computation



- $(1, [((())], \perp)$
- $(2, ((())), A] \perp)$
- $(2, ()), A] \perp)$
- $(2,))], A))] \perp)$
- $(2,))],))] \perp)$
- $(2,)],)] \perp)$
- $(2,],] \perp)$
- $(2, \varepsilon, \perp)$

Definition of A Pushdown Automaton

- A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$
- Q is the set of states.
- Σ is the input alphabet.
- Γ is the stack alphabet.
- $\delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \cup \{\lambda\} \times Q \times \Gamma^*$ is the transition relation
- q_0 is the initial state.
- $\perp \in \Gamma$ is the initial stack symbol.
- F is the set of final states.