Lecture 6:
Pushdown Automata and Context-Free Grammars

Lecture Outline

- Quick Review of Pushdown Automata and Context-Free Grammars
- Leftmost and Rightmost Derivations
- Constructing a PDA from a CFG

Reading

Chapter 2 of Brookshear

Chapters 6,7 of Martin, 2nd ed.

Chapter 6 of Revesz

Part II of Cohen.

Chapters 4,5,6 Hopcroft and Ullman

Leftmost and Rightmost Derivations

- Given a context-free grammar there may be several ways of rewriting its nonterminals to arrive at precisely the same string.

- For example, consider the grammar $\langle \Sigma_N, \Sigma_T, S, R \rangle$, where $\Sigma_N = \{S, A, B\}$, $\Sigma_T = \{a, b, c\}$ and R is given by:

$$\begin{aligned} S &\rightarrow aABb \\ A &\rightarrow bAb \\ A &\rightarrow c \\ B &\rightarrow aB \\ B &\rightarrow b \end{aligned}$$

- The string $abcbabb$ could be derived:

$$S \Rightarrow aABb \Rightarrow abAbBb \Rightarrow abcbBb \Rightarrow abcbaBb \Rightarrow abcbabb$$

  following a rule that always expands the leftmost nonterminal first. Such a derivation is a called a **leftmost derivation**.

- The string $abcbabb$ could be also derived:

$$S \Rightarrow aABb \Rightarrow aAaBb \Rightarrow aAabb \Rightarrow abAbabb \Rightarrow abcbabb$$

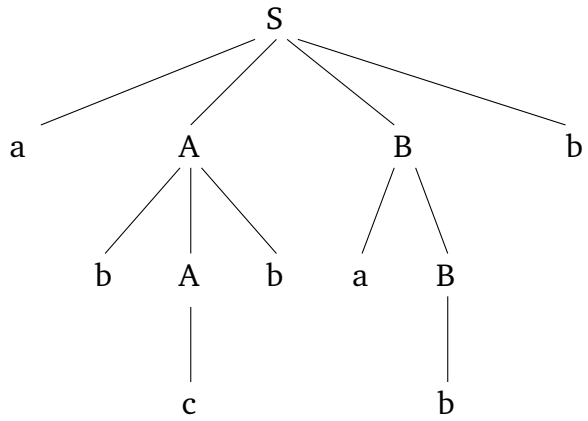
  following a rule that always expands the rightmost nonterminal first. Such a derivation is a called a **rightmost derivation**.

- Note that there are other derivations that are neither leftmost nor rightmost.

Parse Trees

- Derivations in generative grammars can be represented as **parse trees**.

- The derivation of the string $abcbabb$ according to the previous grammar can be represented by the tree:



- Note that the leftmost derivation corresponds to the construction of the parse tree left-branch-first while the rightmost derivation corresponds to the construction of the tree right-branch-first.

- However, both derivations result in precisely the same tree.

  Further, **any** derivation of this string from this grammar would result in the same parse tree.

- Thus the order of application of the rewrite rules does not affect the set of strings that are generated.

  So, if a string can generated by any derivation then it can be generated a leftmost derivation of the string.

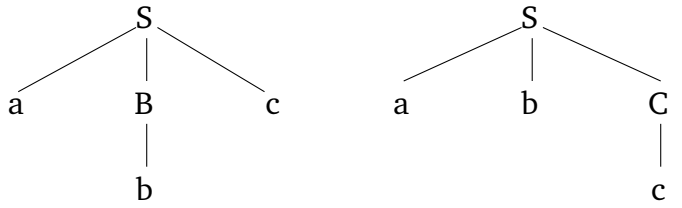
Parse Trees – Ambiguity

- Some grammars permit different parse trees to be constructed for the same string. Such strings are said to be **ambiguous** with respect to the grammar.

- For example, consider the trivial grammar:

$$\begin{aligned} S &\rightarrow aBc \\ S &\rightarrow abC \\ B &\rightarrow b \\ C &\rightarrow c \end{aligned}$$

  This grammar permits two parse trees to be constructed for the string abc:



- In such cases different rules are being used in the two derivations.

  Contrast this with the previous example of leftmost vs. rightmost derivations where in both cases the rules used are the same and simply the **order** of rule application is different.

- Ambiguity is a pervasive feature of natural language. Consider sentences such as:

  He saw the boy with a telescope.
  Time flies like an arrow.
  Flying planes made her duck.

  These exemplify both **structural** *ambiguity (what modifies what) and* **lexical** *ambiguity (which of several word meanings is intended).*

Constructing a PDA from a CFG

- One of the central results of automata theory is the following:

  The languages generated by context free-grammars are exactly the languages accepted by pushdown automata.

- This claim may be broken down into two simpler claims:

  1. For any CFG G there is a PDA M such that $L(M) = L(G)$.
  2. For any PDA M there is a CFG G such that $L(G) = L(M)$.

- The first of these claims is of practical interest because the proof of it will give us a means of constructing a machine for recognising a language given a grammar for the language.

  Such machines (parsers) are a key component of compilers for programming languages.

- We introduce a slight modification to the notation for transitions to allow single transitions to push multiple symbols onto the stack.

  E.g.

  $$(p, a, s; q, xyz)$$

  where this is taken to mean that z, y and x are to be pushed onto the stack, in that order (so after the transition x is on top of the stack, with y under it and z beneath it).

  This does not enhance the power of our PDAs since any transition of this form could be written as the sequence of transitions:

  $$(p, a, s; q_1, z),\ (q_1, \lambda, \lambda; q_2, y),\ (q_2, \lambda, \lambda; q, x)$$

Constructing a PDA from a CFG (cont)

Proposition 6.1 *For any context-free grammar G there is a pushdown automaton M such that $L(M) = L(G)$.*

Proof Given a CFG $G = (\Sigma_N, \Sigma_T, S, R)$ construct a PDA $(Q, \Sigma, \Gamma, T, \iota, F)$ as follows:

1. Let $\Sigma = \Sigma_T$, i.e. let the machine's alphabet be grammar's terminal symbols.

2. Let $\Gamma = \Sigma_T \cup \Sigma_N \cup \{\#\}$, i.e. let the machine's stack symbols be grammar's terminal and nonterminal symbols plus the bottom of stack symbol (assume $\# \notin \Sigma_T \cup \Sigma_N$).

3. Let the states of M, $Q = \{\iota, p, q, f\}$ where ι is the initial state and f is only accept state (so $F = \{f\}$).

4. Let T contain the following transitions:

   (a) $(\iota, \lambda, \lambda; p, \#)$;

   (b) $(p, \lambda, \lambda; q, S)$;

   (c) for each grammar rule $N \rightarrow w \in G$ a transition of the form $(q, \lambda, N; q, w)$ (note that w may contain 0 or more terminals and nonterminals);

   (d) for each terminal x in G a transition of the form $(q, x, x; q, \lambda)$;

   (e) $(q, \lambda, \#; f, \lambda)$.

Constructing a PDA from a CFG (cont)

The behaviour of the automaton may be described as follows:

1. First, it marks the bottom of the stack with $\#$.

2. Then it pushes S, the start symbol of the grammar onto the stack and enters state q.

3. Until $\#$ returns to the top of the stack the automaton either
   (a) pops a nonterminal from the top of the stack and replaces it with the righthand side of a rewrite rule for this nonterminal; or
   (b) pops a terminal from the top of the stack while reading the same terminal from the input.

4. When $\#$ returns to the top of the stack the automaton shifts into its accept state.

The symbols making up the right hand side of a rewrite rule are pushed onto the stack from right to left (i.e. rightmost symbol will be bottommost).

Thus, if the rule contains nonterminals, the leftmost one will come to the top of the stack first, and will in turn be replaced by the righthand side of a rewrite rule.

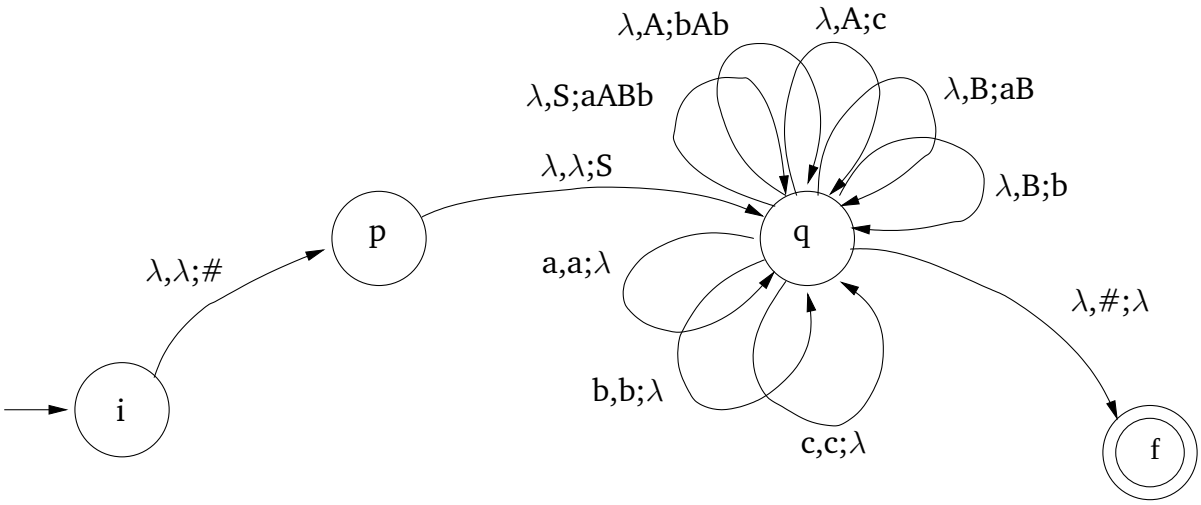
So, the machine performs a leftmost derivation according to the rules of the grammar G. Since if there is any derivation of a string in G there is a leftmost derivation, it follows that M accepts exactly the same language as generated by G. ∎

Constructing a PDA from a CFG – Example

- Recall the grammar

$$\begin{aligned} S &\rightarrow aABb \\ A &\rightarrow bAb \\ A &\rightarrow c \\ B &\rightarrow aB \\ B &\rightarrow b \end{aligned}$$

- An automaton constructed according to the procedure defined above looks like:



Constructing a PDA from a CFG – Example (cont)

In analysing the string $abcbabb$ the machine's behaviour could be represented thus (top of stack is at the left).

Contents of Stack	**Remaining Input**	**Transition Executed**
λ	$abcbabb$	$(\iota, \lambda, \lambda; p, \#)$
$\#$	$abcbabb$	$(p, \lambda, \lambda; q, S)$
$S\#$	$abcbabb$	$(q, \lambda, S; q, aABb)$
$aABb\#$	$abcbabb$	$(q, a, a; q, \lambda)$
$ABb\#$	$bcbabb$	$(q, \lambda, A; q, bAb)$
$bAbBb\#$	$bcbabb$	$(q, b, b; q, \lambda)$
$AbBb\#$	$cbabb$	$(q, \lambda, A; q, c)$
$cbBb\#$	$cbabb$	$(q, c, c; q, \lambda)$
$bBb\#$	$babb$	$(q, b, b; q, \lambda)$
$Bb\#$	abb	$(q, \lambda, B; q, aB)$
$aBb\#$	abb	$(q, a, a; q, \lambda)$
$Bb\#$	bb	$(q, \lambda, B; q, b)$
$bb\#$	bb	$(q, b, b; q, \lambda)$
$b\#$	b	$(q, b, b; q, \lambda)$
$\#$	λ	$(q, \lambda, \#; f, \lambda)$

Constructing a CFG from a PDA

Proposition 6.2 *For any pushdown automaton there is a M context-free grammar G such that $L(G) = L(M)$.*

The proof is somewhat involved and is traditionally omitted from courses on machines and languages. The details may be found in Martin or Brookshear.

Summary

- The same string may be derived in multiple ways from a CFG.

  Consistently expanding the leftmost nonterminal first leads to a **leftmost derivation**.

  Consistently expanding the rightmost nonterminal first leads to a **rightmost derivation**.

- If the same string may be derived in multiple ways using different sets of rules in the grammar (not just changing the order of their application) then the string is **ambiguous** with respect to the grammar.

- For each CFG there is a PDA which accepts just the strings that the CFG generates.

- Such a PDA can be constructed by pushing the righthand side of rewrite rules from the grammar onto the stack and then

  1. if the top symbol on the stack is a nonterminal, pushing the RHS of a further rule onto the stack;
  2. if the top symbol on the stack is a terminal, reading a matching terminal from the input and popping the stack.