CS 341 Homework 12
Parse Trees

1. Consider the grammar G = ({+, *, (,), id, T, F, E}, {+, *, (,), id}, R, E}, where

R = {$E \rightarrow E+T$, $E \rightarrow T$, $T \rightarrow T * F$, $T \rightarrow F$, $F \rightarrow (E)$, $F \rightarrow id$}.

Give two derivations of the string id * id + id, one of which is leftmost and one of which is not leftmost.

2. Draw parse trees for each of the following:

(a) The simple grammar of English we presented in class and the string "big Jim ate green cheese."

(b) The grammar of Problem 1 and the strings id + (id + id) * id and (id * id + id * id).

3. Present a context-free grammar that generates ∅, the empty language.

4. Consider the following grammar (the start symbol is S; the alphabets are implicit in the rules):

$S \rightarrow SS \mid AAA \mid \varepsilon$

$A \rightarrow aA \mid Aa \mid b$

(a) Describe the language generated by this grammar.

(b) Give a left-most derivation for the terminal string abbaba.

(c) Show that the grammar is ambiguous by exhibiting two distinct derivation trees for some terminal string.

(d) If this language is regular, give a regular (right linear) grammar generating it and construct the corresponding FSM. If the language is not regular, prove that it is not.

5. Consider the following language : L = {$w^R w''$: $w \in \{a, b\}^*$ and w" indicates w with each occurrence of a replaced by b, and vice versa}. Give a context-free grammar G that generates L and a parse tree that shows that aababb ∈ L.

6. (a) Consider the CFG that you constructed in Homework 11, Problem 2 for {wcw^R : $w \in \{a, b\}^*$}. How many derivations are there, using that grammar, for the string aabacabaa?

(b) Show parse tree(s) corresponding to your derivation(s). Is the grammar ambiguous?

7. Consider the language L = {$w \in \{a, b\}^*$: w contains equal numbers of a's and b's}

(a) Write a context-free grammar G for L.

(b) Show two derivations (if possible) for the string aabbab using G. Show at least one leftmost derivation.

(c) Do all your derivations result in the same parse tree? If so, see if you can find other parse trees or convince yourself there are none.

(d) If G is ambiguous (i.e., you found multiple parse trees), remove the ambiguity. (Hint: look out for two recursive occurrences of the same nonterminal in the right side of a rule, e.g, $X \rightarrow XX$)

(e) See how many parse trees you get for aabbab using the grammar developed in (d).

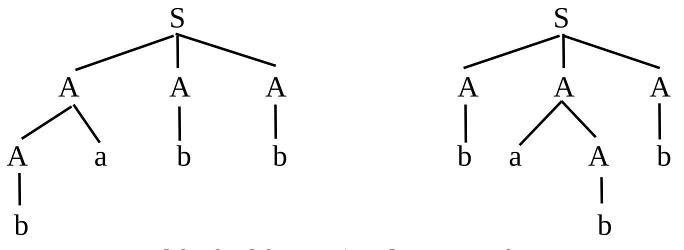
Solutions

3. G = ({S} ∪ Σ, Σ, R, S), where R is any set of rules that can't produce any strings in Σ*. So, for example, R = {$S \rightarrow S$} does the trick. So does R = ∅.

4. (a) (a*ba*ba*ba*)*

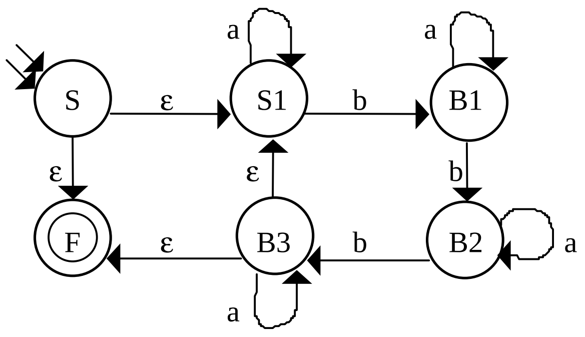
(b) $S \Rightarrow AAA \Rightarrow aAAA \Rightarrow abAA \Rightarrow abAaA \Rightarrow abbaA \Rightarrow abbaAa \Rightarrow abbaba$

(c)

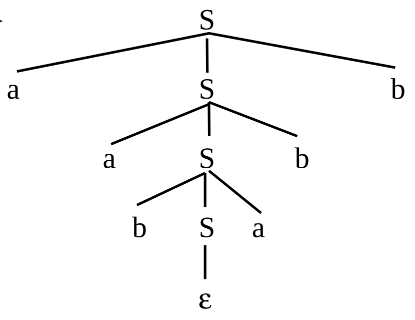


(d) $G = (\{S, S_1, B_1, B_2, B_3, a, b\}, \{a, b\}, R, S)$, where R = {

$S \rightarrow \varepsilon$	$B_1 \rightarrow aB_1$	$B_3 \rightarrow aB_3$
$S \rightarrow S_1$	$B_1 \rightarrow bB_2$	$B_3 \rightarrow \varepsilon$
$S_1 \rightarrow aS_1$	$B_2 \rightarrow aB_2$	$B_3 \rightarrow S_1$
$S_1 \rightarrow bB_1$	$B_2 \rightarrow bB_3$	



5. $G = (\{S, a, b\}, \{a, b\}, R, S)$, $R = \{ S \rightarrow aSb, S \rightarrow bSa, S \rightarrow \varepsilon\}$



6. (a) The grammar is $G = (V, \Sigma, R, S)$ with $V = \{S, a, b, c\}$, $\Sigma = \{a, b, c\}$, $R = \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow c\}$. There is a single derivation:

$S \Rightarrow aSA \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabaSabaa \Rightarrow aabacabaa$

(b) There is a single parse tree. The grammar is unambiguous.

7. (a) $G = (V, \Sigma, R, S)$ with $V = \{S \}$, $\Sigma = \{a, b \}$, R = {

$S \rightarrow aSb$
$S \rightarrow bSa$
$S \rightarrow \varepsilon$
$S \rightarrow SS$ }

(b) (i) $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aaSbbS \Rightarrow aabbaSb \Rightarrow aabbab$ /* This is the leftmost derivation of the most "sensible" parse.

(ii) $S \Rightarrow SS \Rightarrow SSS \Rightarrow aSbSS \Rightarrow aaSbbSS \Rightarrow aabbSS \Rightarrow aabbaSbS \Rightarrow aabbabS \Rightarrow aabbab$ /* This is the leftmost derivation of a parse that introduced an unnecessary S in the first step, which was then eliminated by rewriting it as ε in the final step.

(c) No. The two derivations shown here have different parse trees. They do, however, have the same bracketing, [a[ab]b][ab].(In other words, they have similar essential structures.) They differ only in how S is introduced and then eliminated. But there are other derivations that correspond to additional parse trees, and some of them correspond to a completely different bracketing, [a[ab][ba]b]. One derivation that does this is

(ii) $S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aabSb \Rightarrow aabbab$

(d) This is tricky. Recall that we were able to eliminate ambiguity in the case of the balanced parentheses language just by getting rid of ε except at the very top to allow for the empty string. If we do the same thing here, we get R = { $S \rightarrow \varepsilon$

$S \rightarrow T$
$T \rightarrow ab$
$T \rightarrow aTb$
$T \rightarrow ba$
$T \rightarrow bTa$
$T \rightarrow TT$

But aabbab still has multiple parses in this grammar. This language is different from balanced parens since we can go back and forth between being ahead on a's and being ahead on b's (whereas, in the paren language, we must always either be even or be ahead on open paren). So the two parses correspond to the bracketings [aabb][ab] and [a [ab] [ba] b]. The trouble is the rule $T \rightarrow TT$, which can get applied at the very top of the tree (as in the case of the first bracketing shown here), or anywhere further down (as in the case of the second one). We clearly need some capability for forming a string by concatenating a perfectly balanced string with another one, since, without that, we'll get no parse for the string abba. Just nesting won't work. We have to be able to combine nesting and concatenation, but we have to control it. It's tempting to think that maybe an unambiguous grammar doesn't exist, but it's pretty easy to see how to build a deterministic pda (with a bottom of stack symbol) to accept this language, so there must be an unambiguous grammar. What we need is the notion of an A region, in which we are currently ahead on a's, and a B region, in which we are currently ahead on b's. Then at the top level, we can allow an A region, followed by a B region, followed by an A region and so forth. Think of switching between regions as what will happen when the stack is empty and we're completely even on the number of a's and b's that we've seen so far. For example, [ab][ba] is one A region followed by one B region. Once we enter an A region, we stay in it, always generating an a followed (possibly after something else embedded in the middle) by a b. After all, the definition of an A region, is that we're always ahead on a's. Only when we are even, can we switch to a B region. Until then, if we want to generate a b, we don't need to do a pair starting with b. We know we're ahead on a's, so make any desired b's go with an a we already have. Once we are even, we must either quit or move to a B region. If we allow for two A regions to be concatenated at the top, there will be ambiguity between concatenating two A regions at the top vs. staying in a single one. We must, however, allow two A regions to be concatenated once we're inside an A region. Consider [a[ab][ab]b] Each [ab] is a perfectly balanced A region and they are concatenated inside the A region whose boundaries are the first a and the final b. So we must distinguish between concatenation within a region (which only happens with regions of the same type, e.g, two A's within an A) and concatenation at the very top level, which only happens between different types.

Also, we must be careful of any rule of the form $X \rightarrow XX$ for another reason. Suppose we have a string that corresponds to XXX. Is that the first X being rewritten as two, or the second one being rewritten as two. We need to force a single associativity.

All of this leads to the following set of rules R:

$S \rightarrow \varepsilon$
$S \rightarrow T_a$ /* start with an A region, then optionally a B, then an A, and so forth
$S \rightarrow T_b$ /* start with a B region, then optionally an A, then a B, and so forth

$T_a \rightarrow A$ /* just a single A region
$T_a \rightarrow AB$ /* two regions, an A followed by a B
$T_a \rightarrow ABT_a$ /* we write this instead of $T_a \rightarrow T_aT_a$ to allow an arbitrary number of regions, but force associativity,
$T_b \rightarrow B$ /* these next three rules are the same as the previous three but starting with b
$T_b \rightarrow BA$
$T_b \rightarrow BAT_b$

$A \rightarrow A_1$/* this A region can be composed of a single balanced set of a's and b's
$A \rightarrow A_1A$ /* or it can have arbitrarily many such balanced sets.
$A_1 \rightarrow aAb$ /* a balanced set is a string of A regions inside a matching a, b pair
$A_1 \rightarrow ab$ /* or it bottoms out to just the pair a, b

$B \rightarrow B_1$ /* these next four rules are the same as the previous four but for B regions
$B \rightarrow B_1B$
$B_1 \rightarrow bBa$
$B_1 \rightarrow ba$

(e) The string aabbab is a single A region, and has only one parse tree in this grammar, corresponding to [[aabb][ab]]. You may also want to try abab, abba, and abaababb to see how G works.