Assignment 5
Context-free Grammars

Josefin Ulfenborg
940806-5960
yunalescca@gmail.com

2017-05-14

1 Task 1

Given grammar: $S \to abb|abSb|SS$, over the alphabet $\{a, b\}$. In order to prove the following two things:

- For any $w \in \mathcal{L}(S)$, w starts with an a and ends with a b
- $2 * \#_a(w) = \#_b(w)$

I will use the five steps of proof by induction.

Step 1 (State the property P):
P(n): if S $\Rightarrow^n$ w, then $2 * \#_a(w) = \#_b(w)$, and w will start with an a, and end with a b.

Step 2 (State method):
Course-of-value (strong induction) on the length of the derivation.

Step 3 (State and prove base cases):
Base case: My base case is for n = 1.

P(1) $\equiv S \Rightarrow^1 abb$. This is the smallest word we can get from just one derivation from the starting symbol. It's clear that the word has twice as many b's as a's, and it's clear that the word starts with an a and ends with a b, hence the base case holds.

Step 4 (State the inductive hypothesis):
Assume some $i, j \in \mathbb{N}$. Then assume

- P(i): if S $\Rightarrow^i w_1$, then $2 * \#_a(w_1) = \#_b(w_1)$, and w_1 will start with an a, and end with a b.
- P(j): if S $\Rightarrow^j w_2$, then $2 * \#_a(w_2) = \#_b(w_2)$, and w_2 will start with an a, and end with a b.

in at most $0 < i, j \leq n$ steps.

Step 5 (State and prove inductive steps):
Prove that if S $\Rightarrow^{n+1}$ w (with $n > 0$), then $2 * \#_a(w) = \#_b(w)$, and w will start with an a, and end with a b. Then we have two cases:

- $S \Rightarrow abSb \Rightarrow^n abw_1b : S \Rightarrow^n w_1$ and by the IH: $2 * \#_a(w_1) = \#_b(w_1)$, hence
  $2 * \#_a(w) = 2 * \#_a(abw_1b) = 2 * (1 + \#_a(w_1)) = 2 + 2 * \#_a(w_1)$
  $= \#_b(w) = \#_b(abw_1b) = 2 + \#_b(w_1)$
  $\Rightarrow 2 * \#_a(w_1) = \#_b(w_1)$
- $S \Rightarrow SS \Rightarrow^n w_1w_2 : S \Rightarrow^i w_1$ and $S \Rightarrow^j w_2$ with $i, j < n$. By the IH, $2 * \#_a(w_1) = \#_b(w_1)$ and $2 * \#_a(w_2) = \#_b(w_2)$ hence
  $2 * \#_a(w) = 2 * \#_a(w_1) + 2 * \#_a(w_2) = \#_b(w_1) + \#_b(w_2) = \#_b(w)$ $\square$

2 Task 2

Given language: $\{a^i b^j c^k \mid 0 < i \leq k \text{ or } 0 < k \leq j\}$.

a) What this grammar tells me is:

- At least as many c's as a's, and at least one a. Any number of b's allowed.
- **or** at least as many b's as c's, and at least one c. Any number of a's allowed.

This means that at least one condition has to be met in order for the word to be in the language. Then example of words that are allowed are:

- ac, acc, abc (from first condition)
- bc, bbc, aabbc (from the second condition)

Example of words that are **not** allowed are:

- ε, a, b, c, ab

This gives me the following context-free grammar:

$$S \to XC \mid AY$$

$$X \to aXc \mid aBc \mid ac$$

$$Y \to bYc \mid bBc \mid bc$$

$$A \to aA \mid \varepsilon$$

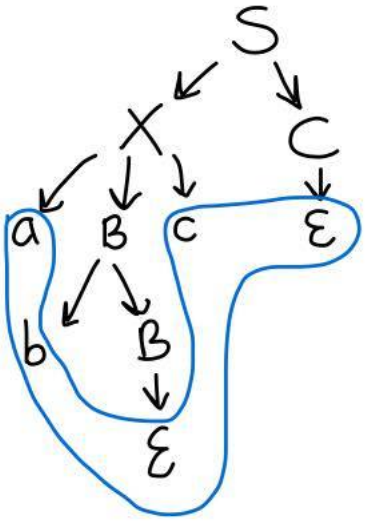
$$B \to bB \mid \varepsilon$$

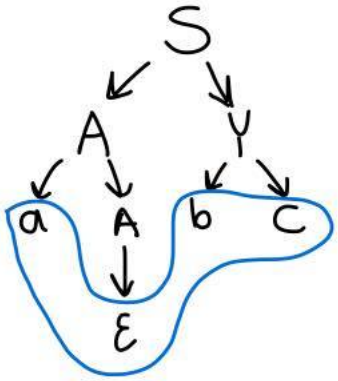
$$C \to cC \mid \varepsilon$$

b) Explanation of grammar: From the starting symbol S, you can either choose to generate the string XC or AY. By reading XC, you guarantee the smallest word ac, but you will also have the possibility to fill this words with any number of B's, and then finish by generating as many c's as you want. This part of the grammars works because it meets the first condition in the language, that for every a you will have one c, but you may also have any number of b's and even more c's.

By reading AY, you guarantee that the smallest word bc will be generated. Likewise, you have the option to read even more b's and before this you have the option to read any number of a's. This part of the grammar will also work since it meets the second condition in the language.

c) Yes, this answer is ambiguous because the word 'abc' has two different parse trees. This can be seen in the image below:





d) The assignment here is to construct the word 'abbccc' by recursive inference, leftmost derivation and a parse tree. My three solutions can be seen on the next page.

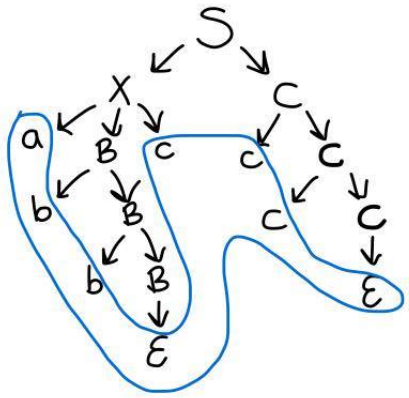
Recursive inference

	String inferred	For language of	Production used	String(s) used
(i)	ε	B	$B \to \varepsilon$	–
(ii)	b	B	$B \to bB$	(i)
(iii)	bb	B	$B \to bB$	(ii)
(iv)	ε	C	$C \to \varepsilon$	–
(v)	c	C	$C \to cC$	(iv)
(vi)	cc	C	$C \to cC$	(v)
(vii)	abbcc	X	$X \to aBc$	(iii)
(viii)	abbccc	S	$S \to XC$	(vii)

Leftmost derivation

$S \Rightarrow^{lm} XC \Rightarrow^{lm} aBcC \Rightarrow^{lm} abBcC \Rightarrow^{lm} abbBcC \Rightarrow^{lm} abb\varepsilon cC \Rightarrow^{lm} abbcC \Rightarrow^{lm} abbccC \Rightarrow^{lm} abbcccC \Rightarrow^{lm} abbccc\varepsilon \Rightarrow^{lm} abbccc$

Parse tree



3 Task 3

Given grammar, with start variable S:

$$S \to aSb \mid A \mid B$$

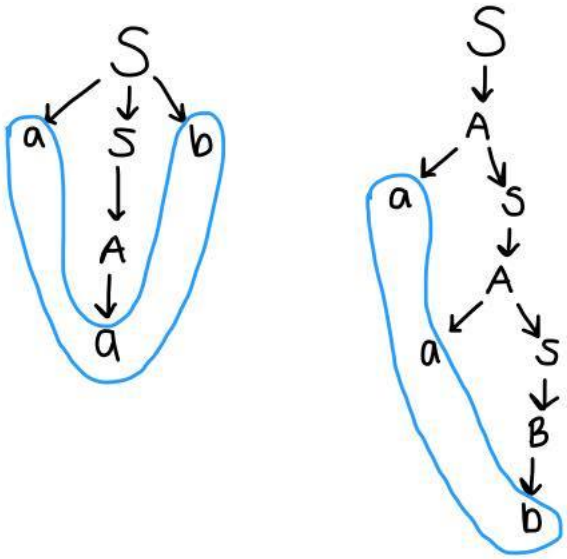
$$A \to aS \mid a$$

$$B \to Sb \mid b$$

a) The words in the language has to consist of at least one a or at least one b, i.e. it cannot be empty. Besides these, there are no restrictions on the words except for that if we have any word with both a's and b's, then the b's have to follow the a's. This gives the following formulation for the language like so:

$$\mathcal{L} = \left\{ a^n b^m \mid n > 0 \textbf{ or } m > 0 \right\}$$

b) In order to show that this language is ambiguous I will construct two parse trees for the word 'aab'. The result can be seen in the image below:



Clearly these are two different parse trees, hence the language is ambiguous.

c) The goal in constructing an unambiguous grammar is that, for instance, I only want one parse tree for the word 'aab'. That means I have to rewrite the grammar so that one of these two ways is not possible anymore.

The problem in this grammar is that from the starting variable S we can either read the string aSb, or go to either A or B. However, from both A and B we can go back to S again, which means we can jump back and forth between the productions, which will cause ambiguity.

When I constructed this unambiguous grammar, this is what I wanted to remove. I introduced a new variable T, and changed all three of S, A and B. Like so:

$$S \to A \mid B \mid AB$$
$$A \to aA \mid a$$
$$B \to bB \mid b$$

d) My unambiguous construction of this grammar is, as can see, completely different from what was first given to me. In order to construct this I my knowledge of what I knew of the characteristics of the words in the language, that is: the words in the language consist of at least one a or one b, or any number of a's and b's as long as the b's follow the a's.

That is, from S, you can either choose to create the words only consisting of a's, or the words only consisting of b's, or the words with both a's and b's.

This grammar is unambiguous because I have removed the back and forth-jumping as I mentioned above. When we derive from A, we can only read a's. There's no way going back to S or to B (and the same goes for the productions of B). Because of this, there is only one way to create each word in the language, which makes is unambiguous.