CS 164 **Fall 2015**

WA2

Nikhil Unni (cs164-es), Section : Monday 3pm

1. Let L be the language consisting of all properly balanced forms of brackets in the alphabet "{, [,], (,), }". Write a context free grammar for the language L.

$$S \to \{S\}$$
$$S \to (S)$$
$$S \to [S]$$
$$S \to SS$$
$$S \to \epsilon$$

2. Consider the following grammar:

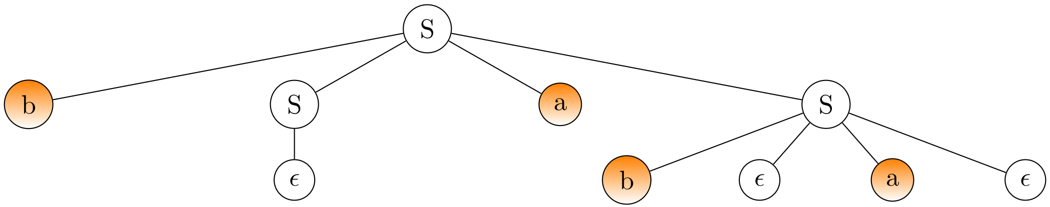
$$S \to aSbS$$
$$S \to bSaS$$
$$S \to \epsilon$$

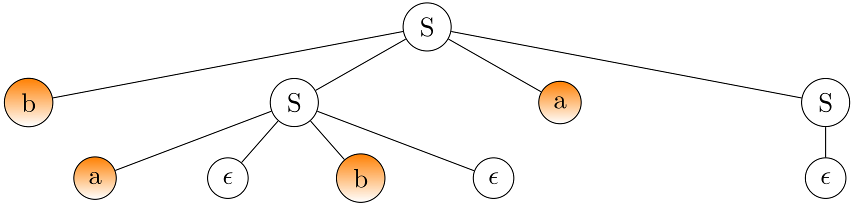
(a) Give a one-sentence description of the language generated by this grammar.

It is the language of all words $w \in (a|b)^*$, such that the number of a's is equal to the number of b's.

(b) Show that this grammar is ambiguous by giving a string that can be parsed in two different ways.

For the word "baba":





(c) Give an unambiguous grammar that accepts the same language as the grammar above.

$$S \rightarrow T$$
$$S \rightarrow \epsilon$$

$$T \rightarrow Same$$
$$T \rightarrow Diff$$

$$Same \rightarrow aB_2a$$
$$Same \rightarrow bA_2b$$

$$Diff \rightarrow aSb$$
$$Diff \rightarrow bSa$$

$$A_2 \rightarrow aaT$$
$$A_2 \rightarrow aTa$$
$$A_2 \rightarrow Taa$$
$$A_2 \rightarrow aa$$

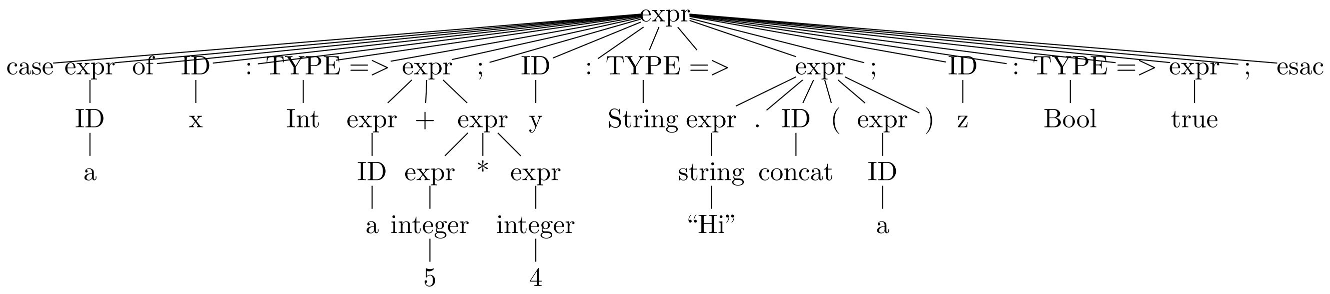
$$B_2 \rightarrow bbT$$
$$B_2 \rightarrow bTb$$
$$B_2 \rightarrow Tbb$$
$$B_2 \rightarrow Tbb$$
$$B_2 \rightarrow bb$$

(d) Give a grammar that accepts the same language as the grammar EXCEPT does not include the empty string. You are not allowed to use epsilon.

$$S \rightarrow ab$$
$$S \rightarrow ba$$
$$S \rightarrow aSbS$$
$$S \rightarrow bSaS$$
$$S \rightarrow aSb$$
$$S \rightarrow bSa$$
$$S \rightarrow abS$$
$$S \rightarrow baS$$

3. Using the context-free grammar for Cool given in Section 11 of the Cool manula, draw a parse tree for the following expression.

```
case a of
    x : Int => a + 5 * 4;
    y : String => "Hi".concat(a);
    z : Bool => true;
esac
```



4. What issues might a top down parser have with the grammar listed below? How would a bottom up parser avoid these issues?

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to \mathbb{Z}$$

The grammar is left recursive, which means that a top-down parser cannot parse it until we convert it to a right recursive grammar (and factor) that accepts the same language. A bottom-up parser avoids these issues, and can parse a left recursive grammar, because it builds up from the original string to the start symbol, so it doesn't matter if the first symbol in a transition is the original symbol itself (or, less confusingly, "left recursive").