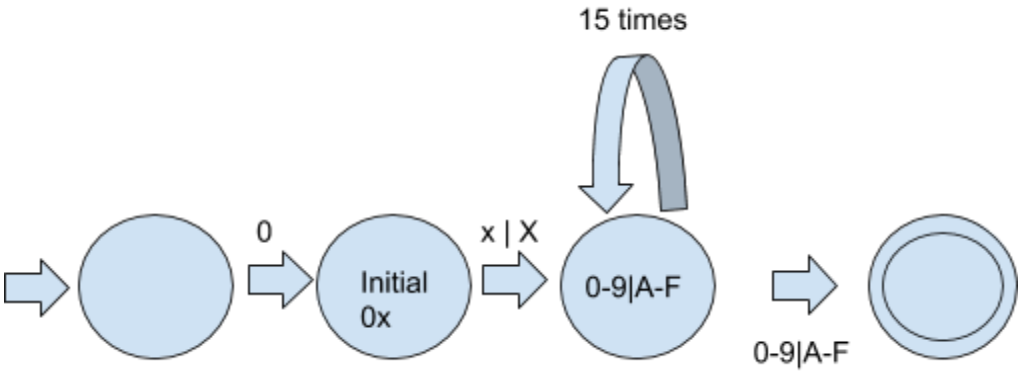
Gentry Atkinson
CS5318 Spring, 2019
Homework 2

1) Write a regular expression to capture Java's long literals in base 16 and then construct a finite-state automation for it.

0[xX] [0-9|A-F]{16, 16}



2) Consider the following grammar:

<E> ::= a <B> | b <A> | ε
<A> ::= a <E> | b <A> <A>
<B> ::= b <E> | a <B> <B>

1. **Write a leftmost derivation for the string a b a a b b.**

   <E>

   a <B>

   ab<E>

   aba<B>

   abaa<B><B>

   abaab<E><B>

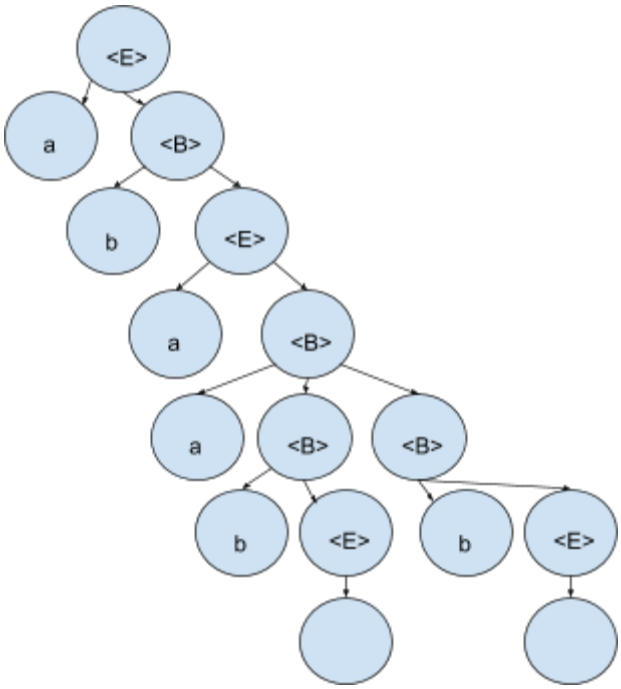
   abaabε<B>

   abaabb<E>

abaabbε

abaabb

2. **Show the parse tree for the string a b a a b b.**



3. **Describe in English the language that the grammar generates.**

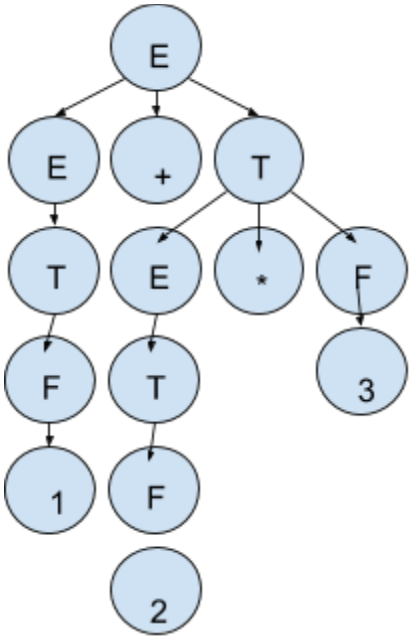
   String of any length containing only a's and b's with a balanced number of a's and b's.

3) Consider the following grammar for expressions:

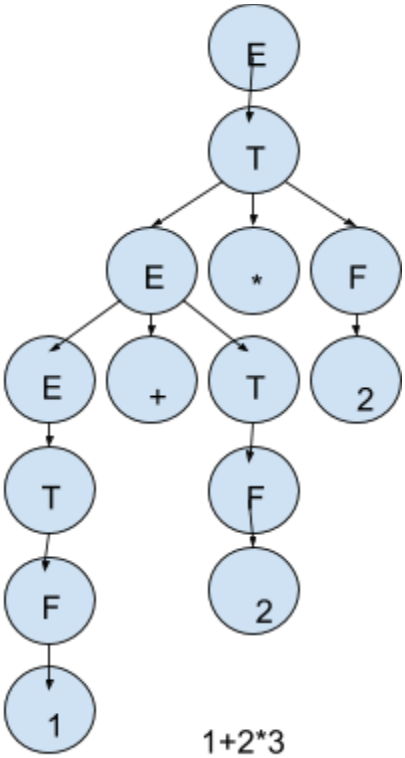
<expr> ::= <expr> + <term> | <term>
<term> ::= <expr> * <factor> | <factor>
<factor> ::= id

1.Show that the grammar is ambiguous.

Two parse trees exist for the expression 1 + 2 * 3, so the language must be ambiguos.



1+2*3



1+2*3

2.Provide an alternate unambiguous grammar that defines the same set of expressions.

<expr> ::= <expr> + <term> | <term>
<term> ::= <term> * <factor> | <factor>
<factor> ::= (<expr>) | id

4) Consider the grammar

S -> (L) | a
L -> L, S | S

1. Re-write the grammar in the EBNF notation.

S -> (L) | a

L -> [L,] {S,}* S

2. Write a recursive-descent parser based on the EBNF notation.

void S() {

```
        if (lookahead == a)
                nexttoken();
        else if (lookahead == ()
                nexttoken();
                L();
                nexttoken();
}
void L() {
        if nexttoken == L
                L();
                nexttoken();
        else while (nexttoken == S)
                S();
                if lookahead == ,
                        nexttoken();
}
```

5)

(a) A number consists of digits from 0 to 9. For example, 1234 is a number. Give two different grammars to define a number, one using a left-recursive rule and the other using a right-recursive rule.

Left: <digit> -> <digit>id | id

Right: <digit> -> id<digit> | id

(b) A decimal number can be defined as a number followed by a decimal point (.) and another number. For example, 123.045 is a decimal number. Define attributes and write an attribute grammar to compute the value of any decimal number.

<number> -> <left>.<right>

value = leftValue + rightValue

leftPower = 1, rightPower - 0.1

<left> -> id<left> | id

leftValue += id * leftPower

leftPower = leftPower * 10

<right> -> <right>id | id

rightValue += id * rightPower

rightPower *= 0.1