1. [5 points] The following grammar is ambiguous and not regular:

```
    e : NUM
      | e + e
      | e * e
      | (e)
```

a. Assume that we don’t care about precedence and associativity (there are no semantic actions; we just want to recognize strings). Devise a regular grammar that accepts the same language, except for expressions with more than one level of parenthesis nesting. For example, 1+2 and (3*4)+(1+2) are in the language, but not (1+(2*3))*4.

**Answer:**

```
    e1 : NUM
      | NUM "+" e1
      | NUM "*" e1

    t : NUM
      | "(" e1 ")"

    e2 : t
      | t "+" e2
      | t "*" e2
```

b. Devise a regular expression corresponding to the grammar of part (a). Feel free to use abbreviations in the style of Flex. Assume, in particular, that NUM is already defined for you as

```
NUM (\[0-9\]+)
```

and that you can embed it in another expression like this:

```
{NUM}:{NUM} is short for ([0-9]+):([0-9]+)
```

You can add additional such definitions as desired (no recursive definitions, however).

**Answer:**

```
E1  ({NUM}(\[+*\]{NUM})*)

T   ({NUM}\("\{E1\}\")

\{T\}(\[+\]{T})*
```
2. [5 points] Consider the following grammar (‘e’ is the only nonterminal):

1. $e : e + e$
2. $| e * e$
3. $| \text{NUM}$

Identify the productions by the numbers on the left (1–3). A leftmost derivation for a certain string consists of the following sequence of productions:

$1, 2, 3, 2, 3, 3, 1, 3, 1, 3, 3$

a. What is the parse tree?

Answer:

```
           e
           /|
          / |/
         /   |
        e    e
         / |
        /   |
       e    e
      /     /|
     NUM   * NUM
```

b. What is the string corresponding to the parse tree in (a)? (Just use ‘N’ to denote terminal symbols matching NUM.)

Answer:

$N * N * N + N + N + N$

c. What is the production sequence in the rightmost derivation corresponding to the same parse tree as in (a)?

Answer:

$1 1 1 3 3 2 2 3 3 3$

d. What is a leftmost derivation for a different parse tree matching the same string as in (b)?

Answer:

$2 3 2 3 1 3 1 3 1 3 3$
3. [1 point] In 1966, William Shatner appeared in a horror movie with a very unusual characteristic. What was unusual about it?

**Answer:**

It was in Esperanto (*Incubus*).

4. [5 points] Consider a calculator for postfix expressions that uses the following grammar:

**Answer:**

\[
\begin{align*}
    p & : e \quad \{ \text{print } \$1 \} \\
    e & : \text{NUM} \quad \{ $$ = \$1 \} \\
        & \mid e \ e + \{ $$ = \$1 + \$2 \} \\
        & \mid e \ e * \{ $$ = \$1 * \$2 \}
\end{align*}
\]

The idea is that, for example, ‘5 2 3 * 4 + *’ should print 50 (which is \(5 \times (2 \times 3 + 4)\)).

a. Complete the semantic rules (in the \{\} parts) to make a syntax-driven parser for this language. You can use Bison notation (as in the first rule) or Horn notation (adding ’=’ definitions to the right-hand side rules) as desired.

b. Explain why this grammar is not LL(\(k\)) for any finite lookahead \(k\).

**Answer:** To choose between the last two productions, one must look beyond two ‘e’s. But ‘e’ can produce an arbitrarily long string.

c. Is this grammar ambiguous? If so, give an input with multiple parses. Otherwise, give an argument demonstrating that it is unambiguous—why there is only one way to match any given input.

**Answer:** It is unambiguous. Consider the language whose sentences are the reverse of sentences in this grammar. Any parse tree for that language can clearly be converted to the parse tree for its reverse in this language, and vice-versa. Likewise the grammar for the reverse language is simply the prefix grammar

\[
\begin{align*}
e & : \text{NUM} \mid + e e \mid * e e
\end{align*}
\]

But the prefix grammar is clearly LL(1) (each alternative is headed by a unique terminal) and therefore unambiguous. The one-to-one correspondence between prefix and postfix parse trees therefore implies that the postfix grammar is unambiguous as well.
5. [5 points] Find a DFA with three states that recognizes the same binary strings as this NFA:

![Diagram of NFA]

It is *not* necessary to use the general NFA-to-DFA construction algorithm for this, although you may do so.

**Answer:**

![Diagram of DFA]
Login: