Practical Sheet 10
Language Processing

1 Regular Expressions

Exercise 1.1: For each of the following regular expression

- Give an example matching string
- Describe the regular expression

1. `(x|y|z)(1|2|3)`

2. `(Mr|Ms|Mrs)(Smith|Jones)`

3. `(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*`

Exercise 1.2: Regular Expressions in Python

Look at the documentation of the Python RE library.

- Use the findall function to find names in a string of text.
- A name starts with 'Mr' or 'Mrs' (etc) and is then followed by a first and second name, starting with capitals.
- Elaborate the rules for names and the regular expression matching them.

2 Finite State Machines

Exercise 2.1: Draw a FSM to recognise:

- A binary string with at least 2 '1' bits in succession
  - E.g. 11 is accepted
  - E.g. 111 is accepted
  - E.g. 10101 is not accepted
- A 6 bit binary sequence with even parity
  - E.g. 101101 is accepted
  - E.g. 101100 is not accepted

Exercise 2.2: Draw a FSM to recognise string specified by the regular expressions:

1. `(x|y|z)(1|2|3)`

2. `(a|b)*a`

Exercise 2.3: Translate the FSM from exercise 2.2 into a Python if statement (see slide) and test it.

3 Syntax and Parsing

<u>Exercise 3.1</u>: Look at the following grammar:

```
<expression> ::= <factor> | <factor> * <factor> | <factor> / <factor>
<factor> ::= <term> | <term> + <term> | <term> - <term>
<term> ::= - <expression> | <number>
<number> ::= <digit> | <digit> <number>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Draw parse trees for the following expressions

- 123
- 1/2
- 1*2+3
- 88 + 3 * 2

<u>Exercise 3.2</u>: Explain the difference between a parse tree and an abstract syntax tree.

- Draw abstract syntax trees for the expressions above
- Suggest how the trees could be represented in Python

4 The Simple Interpreter for Expressions

The interpreter tokens are defined by the following lexical rules:

```
word ::= number | operator
number ::= digit*
operator ::= + | - | * | /
```

and the following grammar:

```
exp ::= factor (('+' | '-') factor)*
factor ::= term (('*' | '/') term)*
term ::= number | '(' exp ')'
```

<u>Exercise 4.1</u>: Consider the following input strings and:

- List the tokens
- Draw the (abstract) syntax tree
- Show how the tree is evaluated

1. `12 * 44`
2. `8 + (11 * 4)`
3. `(2 * 4 * 5) / 3`

<u>Challenge Exercise 4.2</u>: Enhance the interpreter to handle variables and assignment, e.g: v1 = 10 v2 = v1 * 2 v2 * 3

Answer **all** questions in the spaces provided.

1 An operating system is designed to hide the complexities of the hardware from the user and to manage the hardware and other resources.

Give **three** different types of management of either hardware or other resources that are performed by an operating system.

1 ....

....

2 ....

....

3 ....

....

(3 marks)

3

2 **Figure 1** shows some production rules that have been used to define the syntax of valid mathematical expressions in a particular programming language.

Figure 1

```
<expression> ::= <factor> | <factor> * <factor> | <factor> / <factor>
<factor> ::= <term> | <term> + <term> | <term> - <term>
<term> ::= - <expression> | <number>
<number> ::= <digit> | <digit> <number>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

2 (a) What notation method has been used in **Figure 1**?

....

(1 mark)

Do not write outside the box

2 (b) Complete **Table 1** by writing **Yes** or **No** in the empty column to indicate whether or not the strings are valid examples of the statement types from **Figure 1**.

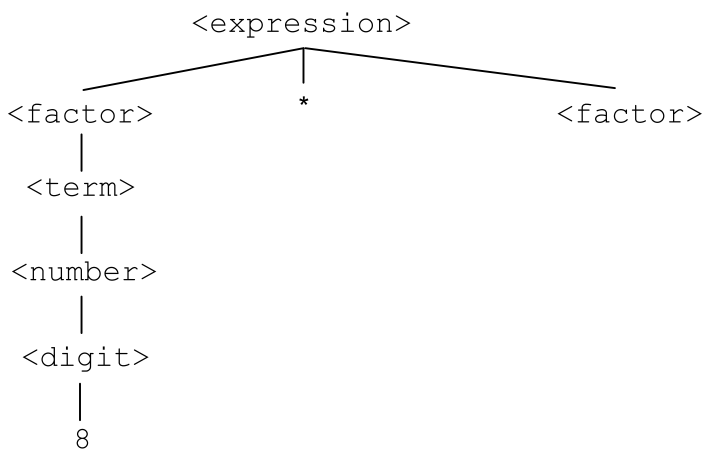
Table 1

Statement type	String	Valid (Yes/No)
`<number>`	`129.376`	
`<factor>`	`23 + 17`	

(2 marks)

2 (c) A tree can be used to demonstrate that an `<expression>` is valid. This is known as a parse tree.

Complete the parse tree below to show that `8 * 4 + 21` is a valid `<expression>`.



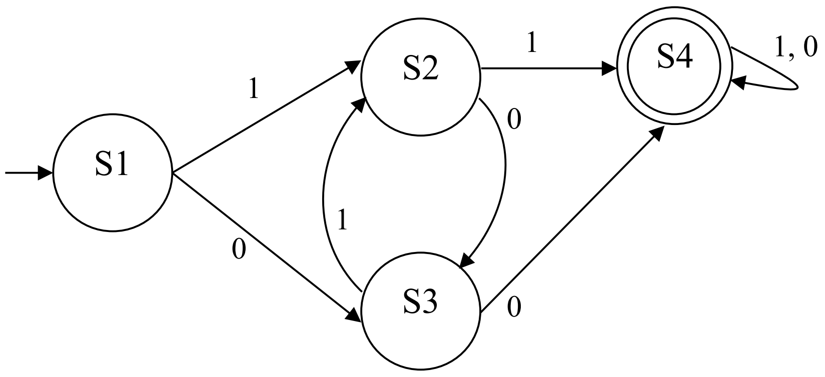
(3 marks)

6

Turn over for the next question

0 2 A finite state machine (FSM) can be used to define a language: a string is allowed in a language if it is accepted by the FSM that represents the rules of the language. **Figure 1** shows the state transition diagram for an FSM.

Figure 1



An FSM can be represented as a state transition diagram or as a state transition table. **Table 1** is an incomplete state transition table for **Figure 1**.

0 2 . 1 Complete **Table 1** and copy the table into the Electronic Answer Document.

Table 1

Original state	Input	New state
S3		
S3		

[1 mark]

Any language that can be defined using an FSM can also be defined using a regular expression.

The FSM in **Figure 1** defines the language that allows all strings containing at least, either two consecutive 1s or two consecutive 0s.

The strings 0110, 00 and 01011 are all accepted by the FSM and so are valid strings in the language.

The strings 1010 and 01 are not accepted by the FSM and so are not valid strings in the language.

0 2 . 2 Write a regular expression that is equivalent to the FSM shown in **Figure 1**.

[3 marks]

Question 2 continues on the next page

Turn over ▸

Backus-Naur Form (BNF) can be used to define the rules of a language.

Figure 2 shows an attempt to write a set of BNF production rules to define a language of full names.

Figure 2

Note: underscores (_) have been used to denote spaces.
Note: rule numbers have been included but are not part of the BNF rules.

Rule number

```
1       <fullname> ::= <title>_<name>_<endtitle> |
                       <name> |
                       <title>_<name> |
                       <name>_<endtitle>

2       <title> ::= MRS | MS | MISS | MR | DR | SIR

3       <endtitle> ::= ESQUIRE | OBE | CBE

4       <name> ::= <word> |
                   <name>_<word>

5       <word> ::= <char><word>

6       <char> ::= A | B | C | D | E | F | G | H | I |
                   J | K | L | M | N | O | P | Q | R |
                   S | T | U | V | W | X | Y | Z
```

BNF can be used to define languages that are not possible to define using regular expressions. The language defined in **Figure 2** could not have been defined using regular expressions.

0 2 . 3 Complete **Table 2** below by writing either a '**Y**' for **Yes** or '**N**' for **No** in each row.

Table 2

Rule number (given in Figure 2)	Could be defined using a regular expression
1	
2	
3	
4	
5	
6	

Copy your answer in **Table 2** into the Electronic Answer Document.

[1 mark]

There is an error in rule 5 in **Figure 2** which means that no names are defined by the language.

0 2 . 4 Explain what is wrong with the production rule and rewrite the production rule so that the language does define some names – the names ‘BEN D JONES’, ‘JO GOLOMBEK’ and ‘ALULIM’ should all be defined.

[2 marks]

Turn over for the next question

Turn over ▶

0 5 Convert the following Reverse Polish Notation expressions to their equivalent infix expressions.

0 5 . 1 `3 4 *`

[1 mark]

0 5 . 2 `12 8 + 4 *`

[1 mark]

Reverse Polish Notation is an alternative to standard infix notation for writing arithmetic expressions.

0 5 . 3 State **one** advantage of Reverse Polish Notation over infix notation.

[1 mark]

END OF SECTION A

Section B begins on page 14