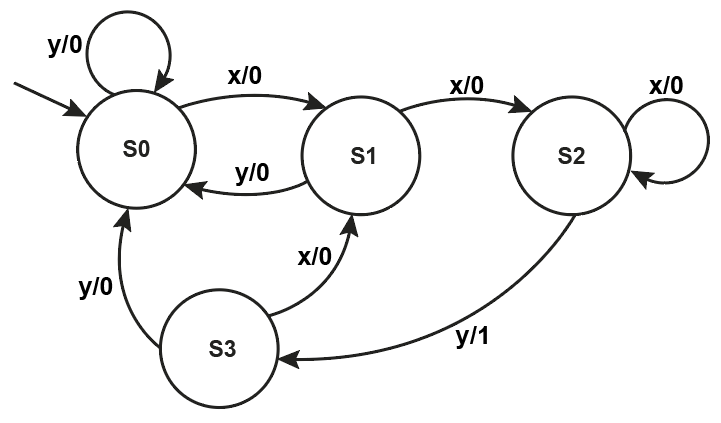
# **Assessment test Answers**

1. An alphabet consists of only the letters ‘x’ and ‘y’. A Mealy machine will read any number of x or y.

Here is a state transition diagram for a Mealy machine that uses this alphabet.



1. Complete the state transition table equivalent to this state transition diagram. [3]

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Current state** | **Output** | **Next state** |
| x | S0 | 0 | S1 |
| y | S0 | 0 | S0 |
| x | S1 | 0 | S2 |
| y | S1 | 0 | S0 |
| x | S2 | 0 | S2 |
| y | S2 | 1 | S3 |
| x | S3 | 0 | S1 |
| y | S3 | 0 | S0 |

1. Complete the state sequence table to verify the behaviour of the machine using an input string x y x x x x y x. [4]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | x | y | x | x | x | x | y | x |  |
| State | S0 | S1 | S0 | S1 | S2 | S2 | S2 | S3 | S1 |
| Output | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  |

(c) State the sequence of letters which will result in a 1 being output. [1]

xxy

1. Here are definitions of seven sets.

|  |  |  |  |
| --- | --- | --- | --- |
| A = {0, 1} | B = {1, 2} | C = {3, 4, 5, 6} | D = {1, 3, 5, 7} |
| E = {6, 7, 8, 9} | F = {7, 9} | G = {(0n1n+1 | n ≥ 1} |  |

1. Describe set G, by listing the first 4 members of the set. [1]

G = {011, 00111, 0001111, 000011111, …}

1. Describe set D, by completing this set comprehension. [1]

D = {a | a ∈ **N** ...ᴧ a is odd ᴧ a <= 7}

1. List the members of the set D ∩ C. [1]

{3, 5}

1. Using sets from those listed above, construct an expression which results in the set {2, 4, 6} [1]

(B ∪ C) \ D

1. Using sets from those listed above, construct an expression which results in the set (1, 3, 5} [1]

D \ F

1. Show the members of the set (A x B) ∩ (B x B) [1]

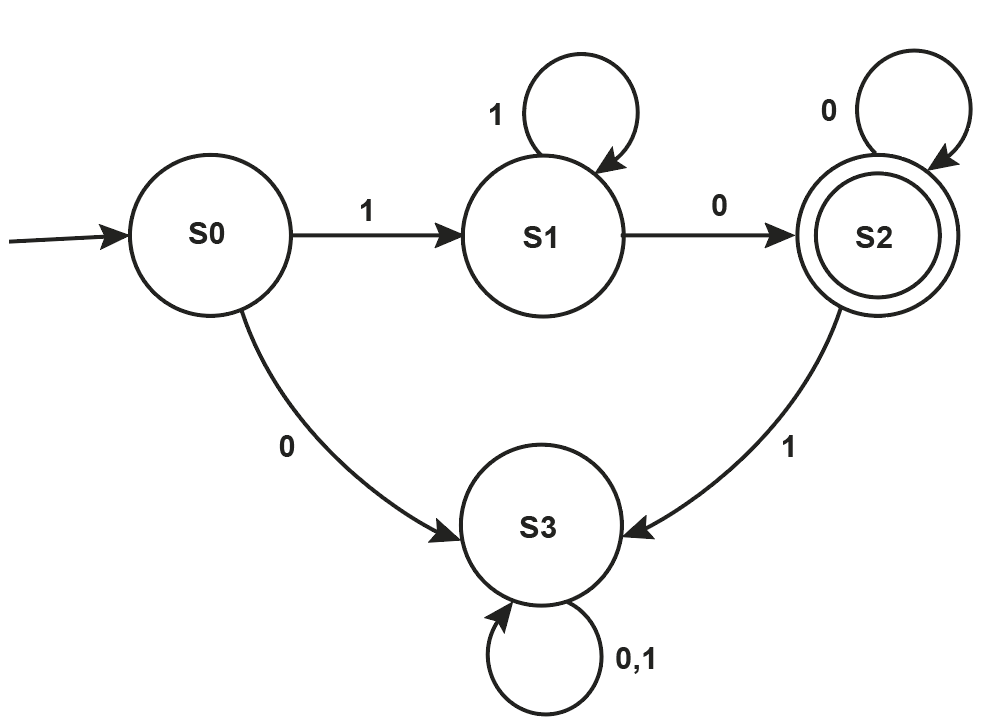
{(0,1), (0,2), (1,1), (1,2)} ∩ {(1,1), (1,2), (2,1), (2,2)} = {(1,1), (1,2)}

1. Using the fewest number of sets from the list above, construct an expression to generate the set of even numbers between 1 and 9. [2]

(B ∪ C) ∪ E \ (D ∪ F)

{1, 2, 3, 4, 5, 6, 7, 8, 9} \ {1, 3, 5, 7, 9} = {2, 4, 6, 8}

1. Regular expressions can be used to search for strings.
2. For each of the following regular expressions, describe the set of strings that they would find. [2]
   1. a + b\* One or more ‘a’ followed by zero or more ‘b’
   2. (ab)?c Zero or one occurrence of ‘ab’ followed by a single ‘c’
3. Write a regular expression to generate the following sets. [2]
   1. {a, b, aa, ba, aaa, baa, aaaa, baaa, …} (a | b) a\*
   2. {ba, baba, bababa, …} (ba)+
4. Draw a four-state FSM that accepts the same language as is recognised by the regular expression 1+0+ [4]



1. A particular Turing machine has states S0, S1, S2, and S3. S0 is the start state and S3 is the stop state. The machine uses one tape which is infinitely long in one direction to store data. The machine’s alphabet is 0, 1, and . The symbol is used to indicate a blank cell on the tape.

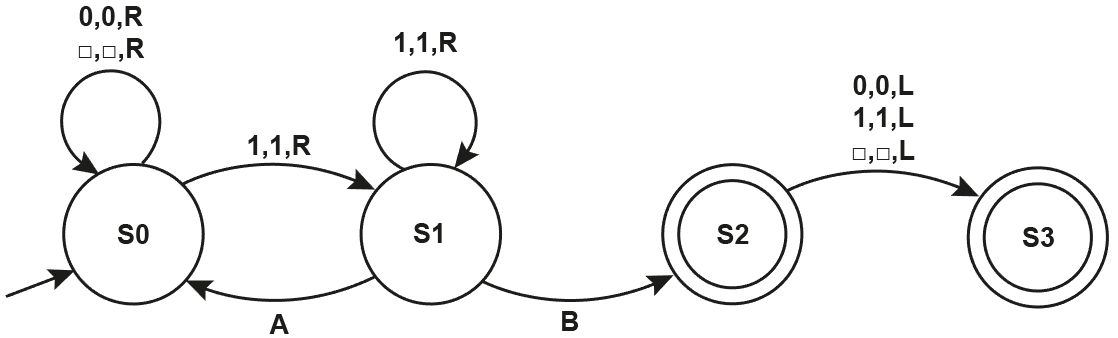
The transition rules for this Turing machine can be expressed as a transition function δ. Rules are written in the form:

δ (Current state, Input symbol) = (Next state, Output symbol, Head move)

The machine’s transition function, δ, is defined by:

|  |  |  |
| --- | --- | --- |
| δ (S0, 0) = (S0, 0, R)  δ (S0, 1) = (S1, 1, R)  δ (S0, ) = (S0, , R) | δ (S1, 0) = (S0, 0, R)  δ (S1, 1) = (S1, 1, R)  δ (S1, ) = (S2, , R) | δ (S2, 0) = (S3, 0, L)  δ (S2, 1) = (S3, 1, L)  δ (S2, ) = (S3, , L) |

Here is a finite state diagram for the transition function of this machine.



1. Write the transition rules for the arcs labelled: [2]

A: 0, 0, R

B: , , R

1. Trace the behaviour of this machine, given this input tape. The \* represents the location of the read-write head. [4]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | |  | |  | |  | |  | |  | | **Current state** | | |  |
| \* | 1 | | 0 |  | | 1 | | 1 | |  | |  | |  | | S0 |
|  | 1\* | | 0 |  | | 1 | | 1 | |  | |  | |  | | S0 |
|  | 1 | | 0\* |  | | 1 | | 1 | |  | |  | |  | | S1 |
|  | 1 | | 0 | \* | | 1 | | 1 | |  | |  | |  | | S0 |
|  | 1 | | 0 |  | | 1\* | | 1 | |  | |  | |  | | S0 |
|  | 1 | | 0 |  | | 1 | | 1\* | |  | |  | |  | | S1 |
|  | 1 | | 0 |  | | 1 | | 1 | | \* | |  | |  | | S1 |
|  | 1 | | 0 |  | | 1 | | 1 | |  | | \* | |  | | S2 |
|  | 1 | | 0 |  | | 1 | | 1 | | \* | |  | |  | | S3 |

1. What is the purpose of this Turing machine? [1]

After the start position, it finds the first occurrence of a blank with a 1 to the left. In other words, the first blank after a 1 to the right of the starting position.

1. Valid currency amounts and email addresses can take a variety of forms.

(a) A representation for a currency amount has the following requirements:

* One or more digits to the left of the decimal.
* Either 2 or 3 digits to the right of the decimal.
* Positive amounts have no sign.
* Negative amounts may have a negative sign ‘-‘.
* Alternatively, negative amounts can be written between brackets ( ).
* Valid monetary amounts: 11352.15, 12.789, -7.34, -17.283, (12.56), (8.527)

Write the BNF production rules for currency. [5]

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<pence> ::= <digit><digit> | <digit><digit><digit>

<pounds> ::= <digit> | <digit><pounds>

<pounds\_and\_pence> ::= <pounds> ‘.’<pence>

<currency> ::= <pounds\_and\_pence> | ‘-’ <pounds\_and\_pence> | (<pounds\_and\_pence>)

(b) Explain why compiler writers need Backus-Naur Form. [2]

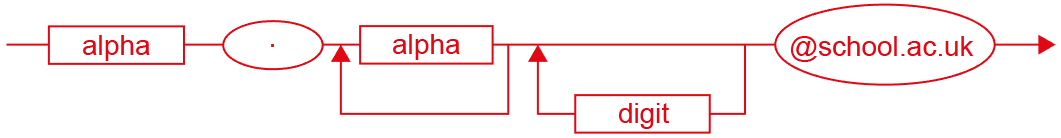
Programming languages cannot be ambiguous. A computer cannot make decisions based on context. BNF provides an unambiguous method for representing programming languages. In addition, some languages can’t be represented by other methods, such as regular expressions.

(c) An email address must conform to the following rules:

* It must begin with an alphabetic character (a … z), representing a person’s first initial.
* This is followed by a single dot ‘.’
* The person’s surname comes next and can only consist of alphabetic characters (a … z). It cannot be blank.
* This is followed by 0 or more digits (0 … 9) indicating how many clashes have previously occurred with this combination.
* Finally, the email address ends with the characters ‘@school.ac.uk’

Here are some valid email addresses: [c.jones@school.ac.uk](mailto:c.jones@school.ac.uk), [d.brown12@school.ac.uk](mailto:d.brown12@school.ac.uk), and [r.white123@school.ac.uk](mailto:r.white123@school.ac.uk)

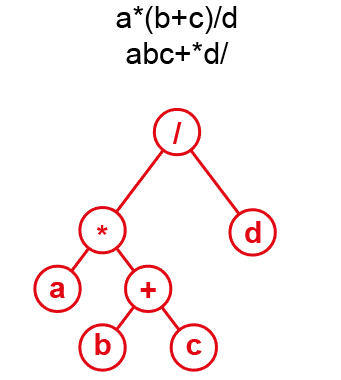
Draw a syntax diagram to describe the email addresses. Assume that **alpha** has been defined as {a … z} and that **digit** has been defined as {0 … 9}. [2]



1. Reverse Polish Notation is an alternative to standard infix notation for writing arithmetic expressions.
2. Convert the following infix expressions to their equivalent Reverse Polish Notation expressions.
3. a \* (b + c) abc+\* [1]
4. 16 + 4 / 20 16 4 20 / + [1]
5. Convert the following Reverse Polish Notation expressions to their equivalent infix expressions. [1]
   1. b c / a+ ( ( b c / ) a +) 🡪 b / c + a
   2. 5 2 ^ 10 / 3 + [1]

(((5 2 ^)10 /) 3 +) 🡪 5^2 / 10 + 3

1. Here is an infix expression: a \* (b + c) / d.   
     
   i. Complete the binary tree below to represent the expression. [4]



ii. Perform a post-order traversal to give the equivalent Reverse Polish Notation. [2]

The post-order traversal is a b c + \* d /, which is the Reverse Polish Notation.

[Total 50 marks]