4.3 Context free languages Mark Scheme

Q1.

(a) **Marks are for AO1 (understanding)**

Real number	Valid? (Yes/No)
87.000	Yes
97+12	No
12.31E+12	Yes

A. alternative indicators for Yes/No eg Y/N.

Mark as follows:
One mark per correct row

3

(b) **Marks are for AO2 (apply)**

`<natural> ::= <digit> | <digit> <natural>`
A. alternative names for `<natural>`
A. recursive and non-recursive cases swapped around

Mark as follows:
1 mark: correct recursive case
1 mark: correct non-recursive case
MAX 1 if any errors in answer eg missing |

2

[5]

Q2.

(a) One mark per correct response.

Construct	Example	Valid? (Yes/No)
identifier	`Game_Over`	No;
parameter	`ref x,y:bool`	Yes;
procedure-def	`procedure square(s:float)`	Yes;
procedure-def	`procedure rect(w:int,h:int)`	No;

A. Alternative clear indicators of Yes/No such as Y/N, True/False and Tick/Cross.

4

(b) The `<type>` rule is missing type `string`;
The `<procedure-def>` rule does not allow a procedure without parameters // cannot be just an identifier;

Accept answers comparing the figures the other way around, ie

- The type rule has an extra type string
- The procedure does not have to have parameters / can be just an identifier

2

(c) Required as there can be a list of parameters // required as there can be more than one parameter;
BNF does not support iteration // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise // recursion allows for more than one parameter;
MAX 1
A. Input for parameter
NE. Rule needs to loop

1

[7]

Q3.

(a) Syntax diagram;
A. Railroad diagram

1

(b) 3

1

(c) **Mark is conditional upon a correct answer to (b)**
It requires that a signed binary number starts with a + or a - // it won't allow a signed binary number that starts with a bit/digit/1 or 0 // + and – are not optional /are required;
A. 'String' or 'number' for 'signed binary number'

1

(d) Some example correct regular expressions are listed below but award a mark for any regular expression that would correctly represent the language accepted by the FSA.

a ((ba)|c)* // a((ba)*|c*)* // a((ba)*|c)* // a((ba)|c*)* // a((ba)*|c*)+ // a(c*(ba)?)

A. Missing brackets around ba as BOD

1

[4]

Q4.

(a) **Mark is for AO1 (understanding)**

Original state	Input	New state
S3	0	S4
S3	1	S2

1 mark: Table completed as above
I order of rows

1

(b) **All marks AO2 (analyse)**

(0|1)*((00)|(11))(0|1)*

Mark as follows:
1 mark: (0|1)* at start;
1 mark: (00)|(11);
1 mark: (0|1)* at end;

Or

Alternative answer
(0|1)*(11(0|1)*)|(00(0|1)*)

Mark as follows:
1 mark: (0|1)* at start;
1 mark: (11(0|1)*);
1 mark: |(00(0|1)*) at end;

Maximum 2 marks: If final answer not correct.

A any regular expression that correctly defines the language.

3

(c) **Mark is for AO2 (apply)**

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

1 mark: All values in the table have been completed correctly.

1

(d) **1 mark for AO2 (analyse) and 1 mark for AO3 (design)**

1 mark for AO2 (analyse): There is no non-recursive / base case;
1 mark for AO3 (design): `<word> ::= <char><word> | <char>;`

2

[7]

Q5.

(a)

<variable>	Valid? (Tick any number of rows)
a	✓
money-paid	
taxrate2	✓
2ndPlayerName	

1 mark for ticks in the correct two rows and other rows left blank.

A Use of alternative symbol for tick
A Use of two symbols - one to indicate validity and one to indicate invalidity, so long as the meaning of the symbols is clear e.g. a tick and a cross or a Y and an N.

1

(b) Required as an integer can contain any number of digits;
NE More than one digit
A "numbers" for "digits" as **BOD**
BNF does not support iteration / looping // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise;
NE Rule needs to loop
MAX 1

1

(c) Variable may not have been declared;
Variable may be of inappropriate type;
Position of statement within program may be invalid;
Rightmost integer may be a lower value than the leftmost one;
One of the numbers / limits may be outside of the range of valid integers;
A Examples of any of the above
MAX 1

1

[3]

Q6.

(a) One mark per correct response.

Construct	Example	Valid?
identifier	`Player2name`	No;
parameter	`x, y:bool`	Yes;
procedure-def	`procedure square(s:real)`	No;
procedure-def	`procedure rect(w:int,h:int)`	No;

A alternative clear indicators of Yes / No such as Y / N, True / False and Tick /

Cross.

4

(b) (i) The <type> rule has an extra type char;
The <procedure-def> rule does not allow a procedure without parameters // cannot be just an identifier;

A answers comparing the figures the other way around, i.e.
- The type rule does not allow a char
- The procedure does not have to have parameters / can be just an identifier

2

(ii) Required as there can be a list of parameters // required as there can be more than one parameter;
BNF does not support iteration // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise // recursion allows for more than one parameter;
Max 1
A Input for parameter
NE Rule needs to loop

1

[7]

Q7.

(a) Backus-Naur (Form);
A Backus Normal (Form), BNF, Extended Backus-Naur (Form), Augmented Backus-Naur (Form), ABNF
A Misspellings of Backus-Naur
A Format for Form and the word "Notation"
R BN

1

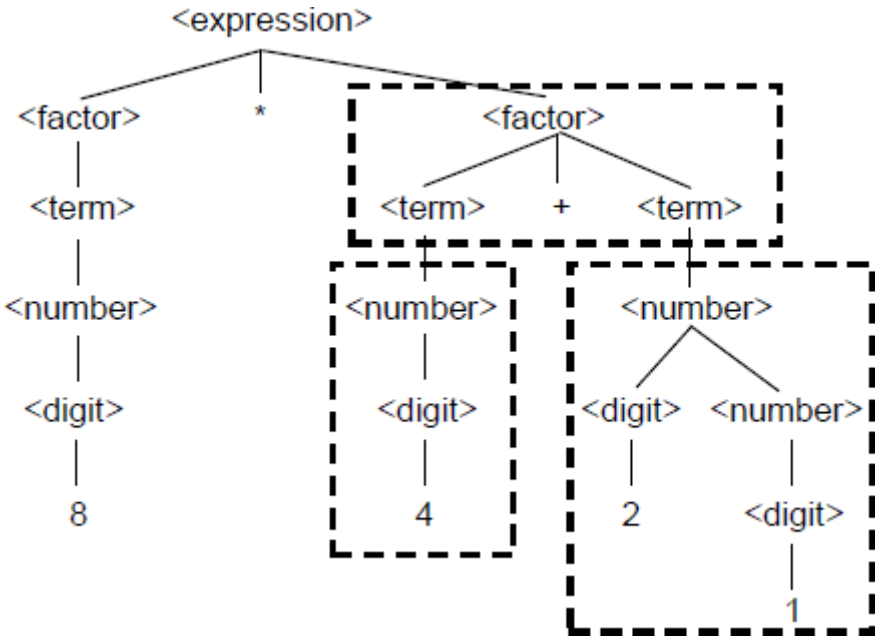
(b)

Statement Type	String	Valid (Yes / No)
<number>	129.376	*No;*
<factor>	23 + 17	*Yes;*

A Alternative clear indicators of Yes or No e.g. Y / N, Tick / Cross, Valid / Invalid, True / False

2

(c)



1 mark for each area surrounded by a rectangle

A missing chevrons
DPT Arrows drawn instead of lines

3

[6]

Q8.

(a)

Real number	Yes / No
203.412	**Yes**
–12.87	**No**
12.43E–12	**Yes**

1 mark per correct Yes / No

A other indicators that clearly mean Yes / No e.g. True/False, Tick / Cross.

3

(b) <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<whole-number> ::= <digit> | <digit> <whole-number>
<integer> ::= <whole-number> | + <whole-number> |
- <whole-number>

1 mark for each correct rule

Alternative for integer (1 mark, accept in either order):
<symbol> ::= + | -
<integer> ::= <whole-number> | <symbol> <whole-number>
A <whole-number> defined with recursion other way around, i.e.
<whole-number> ::= <digit> | <whole-number> <digit>
A non-terminal names e.g. digit not enclosed in <> signs

A spaces in non-terminal names e.g. whole number
A terminal names enclosed in quotation marks e.g. “0”, ‘0’.
A any sensible symbol for assignment e.g. ←, :=, =, :
A ; as end-of-rule marker;
A any type of slash e.g. / for alternatives but **R** "or"
A use of EBNF extensions for repetition and optional terms:
<whole-number> ::= <digit> { <digit> }
<integer > ::= [+ | -] <whole-number> **A** () for [] but **R** {}
R rules that have additional options e.g. more than ten digits
DPT addition of chevrons or other symbols such as brackets to terminal symbols/rules unless they make meaning unclear

3

[6]

Examiner reports

Q1.

This question was generally well-answered by students with most able to identify which of the numbers in Table 1 were valid according to the rules defined in the syntax diagram. The second part of the question was harder with many students correctly defining the base case but not the recursive case; a number of answers were not written using BNF even though an example of a rule was given in the question.

Q3.

This question was about different formal methods of defining a language. For part (a) some candidates correctly identified that a syntax diagram had been used, but many incorrectly identified the notation as Bakus-Naur Form (BNF).

Three quarters of candidates identified that language definition 3 was different to the other two for part (b) and a pleasing number were able to explain that for definition 3 the sign before the binary number was optional in response to part (c). Some candidates gave an almost correct answer to part (c), recognising that the difference related to the sign, despite having identified the wrong language definition in part (b).

For part (d) candidates had to write a regular expression to recognise the same language as the finite statue automaton on the question paper. This question part was quite difficult and was poorly tackled. The most common mistake was to assume that any repetitions of ba always had to be made before any repetitions of c and to give an answer like a(ba)*c*.

Q5.

This question was about the use of BNF to recognise language syntax. Slightly over half of students achieved the mark for part (a).

For part (b), just under half of students achieved the mark. Good responses recognised that an integer could contain an unlimited number of digits and that as BNF does not support iteration, recursion had to be used to achieve this. Responses that stated that more than one digit might be used were not enough for a mark as they did not make clear that the number of digits was unlimited.

For part (c) students had to explain why a For loop that met the BNF syntax definition might produce an error during compilation. Just under half of students achieved a mark for this, with good responses including that the number used for the first limit might be higher than the second, that the count variable might not have been declared or might be an inappropriate type, or that count might be a reserved word in the language.

Q6.

Candidates demonstrated a pleasing understanding of the use of syntax diagrams and Backus Naur Form to specify language syntax.

For (a), the overwhelming majority of candidates scored at least three of the four available marks. Candidates had most trouble identifying that the third example `procedure square (s:real)` was not valid, perhaps because they just assumed that real was a valid type rather than checking it against the diagrams.

For (b)(i), the majority of candidates recognised that the BNF definitions incorrectly

included a new “char“ data type and almost half also identified that the BNF definitions did not allow for a procedure to have no parameters.

Part (b)(ii) was well answered with most candidates achieving a mark for recognising that there could be any number of parameters. Pleasingly, some also went on to explain that recursion had to be used because BNF does not support iteration. The most commonly seen incorrect response was to simply define what recursion was instead of addressing the specific question.

Q7.

Part (a): Approximately three quarters of students correctly recognised that Backus-Naur Form had been used. There was a considerable variation in the spelling of the term but, as long as the meaning was clear, spelling mistakes were not penalised in this question part.

Part (b): The vast majority of students correctly identified which of the two statements was valid.

Part (c): This question part introduced the idea of using a parse tree to demonstrate that an expression was valid. How the parse tree was built was explained in the question and the tree was partially completed to further aid students. Most students made a reasonable attempt at completing the rest of the tree, with nearly half achieving at least two of the three available marks. The most commonly made error was to treat “21” a single digit instead of decomposing it into two digits.

Q8.

Part (a): The vast majority of candidates scored full marks for this question part, correctly identifying whether or not the numbers were syntactically valid.

Part (b): There were some good responses to this question part, although some candidates clearly found it quite difficult. This is the first question that has been asked on this topic so, on this occasion, we were lenient when faced with minor syntactical errors. A small number of candidates confused Backus-Naur Form with regular expressions.