AQA Computer Science A-Level

4.4.3 Context-free languages

Intermediate Notes

Specification:

4.4.3.1 Backus-Naur Form (BNF)/syntax diagrams

Be able to check language syntax by referring to BNF or syntax diagrams and formulate simple production rules

Be able to explain why BNF can represent some languages that cannot be represented using regular expressions

Context-free languages

A context-free language is a set of strings and symbols that follow a set of rules called production rules. A production rule is a simple as replacing one character for another. The table below shows three examples of production rules.

a → ab	a → aa	b → a
This production rule specifies that the a character can be replaced by the two characters ab.	This production rule describes that the character a can be replaced by two a characters.	This production rule specifies that a b character can be replaced by an a character.

Backus-Naur form

Backus-Naur form is a way of notating context-free languages. It uses statements in which the left hand side is defined by the right hand side.

Non-terminals

Text which is placed inside of angle brackets represents something called a non-terminal (these are sometimes also called meta-components). Non-terminals can be broken down further into either more non-terminals, terminals or a combination of the two. For example, the non-terminal `<FullName>` could be defined by three more non-terminals as follows:

```
<FullName> ::= <Title><Forename><Surname>
```

Terminals

Text without any brackets represents a terminal. Terminals cannot be broken down any further and must be taken to be the written value. For example, the letter a is a terminal which is taken to mean the character “a”. For example, the non-terminal <Address> could be defined as follows:

```
      <Address> ::= <Number><Street>
<Number> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Number is a non-terminal which is defined by nine terminals from 1 to 9. The straight vertical line represents the OR operator.

Recursion in Backus-Naur form

Backus-Naur form is capable of representing some strings that cannot be represented by regular expressions as regular expressions cannot support recursion like Backus-Naur form can.

For example, the example below is the definition for a number.

```
<Number> ::= <Digit>|<Digit><Integer>
<Digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Synoptic Link

Something which is **recursive** is defined in terms of itself.

Recursive techniques are covered in more detail in **fundamentals of programming**.

In plain English, these definitions read as *“A number is defined as a digit or as a digit followed by a number”* and *“A digit is defined as one the the numbers from 0 to 9”*. The definitions mean that the strings “2”, “53” and “78230137” all qualify as valid numbers.

Example: Representing numbers

The definitions below allow for numbers with or without a decimal part, to be defined.

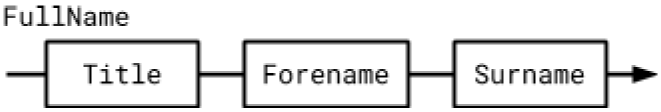
```
    <Digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
    <DigitString> ::= <Digit>|<Digit><Integer>
    <Number> ::= <DigitString>|<DigitString>.<DigitString>
```

The definitions above allow for the definition all of the following numbers.

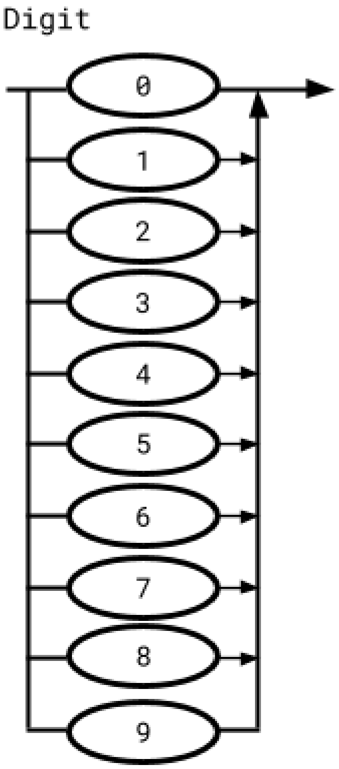
```
     7         45.332         86         553.3         9.009
```

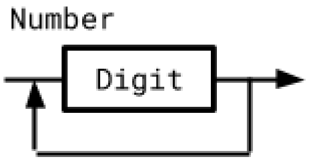
Syntax Diagrams

A syntax diagram is a visual representation of a regular language. Syntax diagrams use rectangles to represent non-terminals and ellipses to represent terminals. These shapes are joined by arrows which indicate how strings can be formed from the definitions. Each non-terminal is defined by its own syntax diagram.



The syntax diagram above shows the definition of `FullName` as a formation of the three non-terminals `Title`, `Forename` and `Surname`.





The example above shows two definitions: one for `Digit` and another for `Number`. `Number` is defined by the non-terminal `Digit` which is, in turn, defined by ten terminals.