4.4.3 Backus-Naur Form

Backus-Naur Form

Backus-Naur form, like regular expressions, is another way of representing languages.

Unlike regular expressions, **BNFs can make use of recursion** which allows it to represent more languages.

Production Rules

BNF can use production rules to describe a language.

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

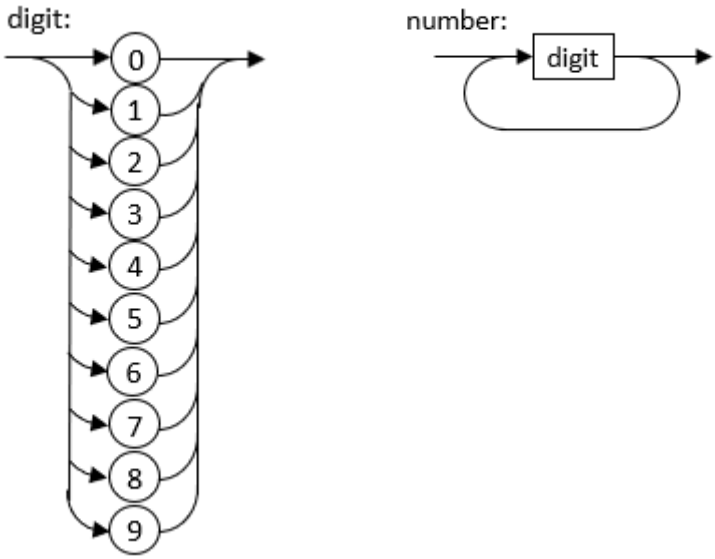
<Number> :: = <Digit> | <Digit><Number>

In the example above:

- The definition of a Digit uses | as an 'or' and is defined as any value from 0 to 9.
- The definition of a Number uses recursion to build numbers with one or more digits

Syntax Diagrams

Syntax diagrams given a visual representation of production rules.



A circle represents a terminal symbol.
A rectangle represents a non-terminal.

The digit non-terminal is used when defining a number.

BNF recursion

Using recursion BNF can describe languages that could not be described using regular expressions.

An example of this would be a language that has an ‘a’ enclosed between any number of opening and closing brackets.

{ a, (a), ((a)), (((a))), ((((a)))), … }

There is no way of representing this language using a regular expression but using recursion in Backus-Naur form:

<Term> ::= a | (<Term>)

Caution

While using recursion allows Backus-Naur form to define languages that a regular expression cannot, just because a production rule is recursive doesn’t mean that a regular expression could not define the same language.

<Symbol> ::= a | b

<Word> ::= <Symbol> | <Symbol><Word>

The production for Word is recursive but describes the same language as the regular expression $(a|b)^+$.