3802ICT Programming Languages - Assignment 2

Jesse Schneider

September 19, 2020

Abstract

This report is targeted at investigating EBNF and parsing for the JavaScript Object Notation (JSON) data-interchange format. It includes EBNF definitions, a Haskell JSON Data Type, a JSON Lexer and Parser written in Haskell and Validation of the parser.

1 Task 1: JSON EBNF

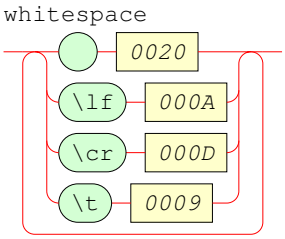
For this report, we have 2 different sections of EBNF defined: Lexical syntax and Context-free syntax. Our Lexical EBNF is used to define Lexical tokens that will be in the parsed content. The Context-free rules will define how we combine the Lexical tokens to define rules, in this instance defining how JSON will be interpreted.

1.1 Lexical Syntax Rules

Here is the Lexical EBNF and Railroad Diagrams drawn from those rules, to display the different Lexical Tokens within JSON:

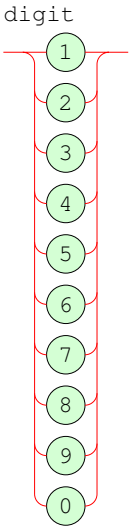
Whitespace - Spaces, Line Feeds, Carriage Returns, Tabs

```
whitespace ::= { " " $0020$| "\lf" $000A$ | "\cr" $000D$ | "\t" $0009$ }+ ;

level="lexical".
```



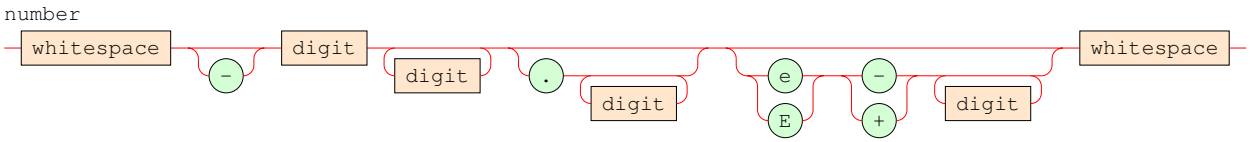
Digits - All digits from 0 - 9

```
digit ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "0";

level="lexical".
```



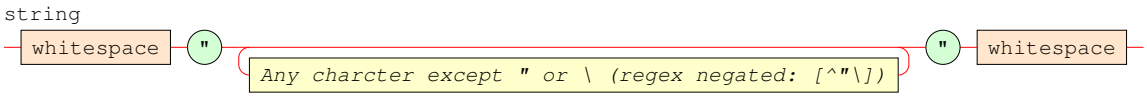
Numbers - positive and negative Integer, Decimal, Exponential

```
number ::= whitespace
    ["-"] digit { digit }
    ["." { digit }]
    [("e" | "E") ("-" | "+") {digit}] whitespace;

level="lexical".
```



Strings - A collection of any characters grouped together

```
string ::= whitespace "\""
    { $Any charcter except " or \\ (regex negated: [^"\])$ }
    "\"" whitespace;

level="lexical".
```

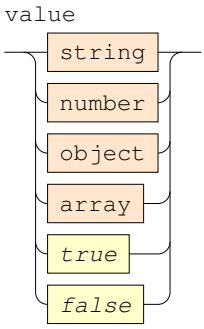


1.2 Context-Free Syntax Rules

Here is the Context-Free EBNF and Railroad Diagrams drawn from those rules, to demonstrate how the Lexical Tokens can be combined within JSON:

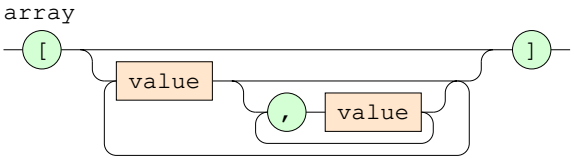
Values - Numbers, Strings, Arrays, Objects, True, False

```
value ::= string | number | object | array | $true$ | $false$ .
```



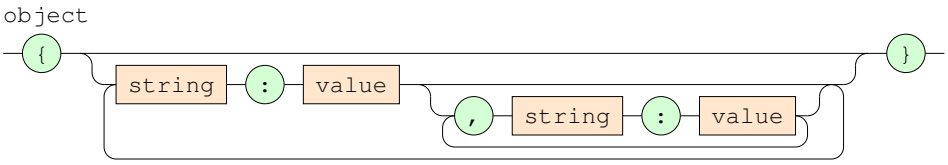
Arrays - A collection of any Values

```
array ::= "[" {value {"," value } } "]".
```



Objects - A (key:value) type data structure to store any type of Value

```
object ::= "{"  { string ":" value { "," string ":" value } } "}".
```



2 Task 2: Haskell JSON Data Type

A Recursive Algebraic Haskell Data Type has been designed to store JSON as seen here:

JSON Data Type

Note: Array and Object are Recursive so as to store any other Type of JSON.

```
data Json = Str String
    | Number Float
    | Object (String, Json)
    | Array [Json]
    | True
    | False deriving (Show)
```