Syntax

Sau P

20 April 2023

Contents

1 Syntax of our language

This language will be a domain specific language specialising in the manipulation of tiles.

1.1 Language specification in Backus-Naur form

```
<program> ::= {<statement>}

<statement> ::= <comment>
             | <tile-definition>
             | <variable-declaration>
             | <operation>
             | <iteration>
             | <print>
             | <if-statement>

<comment> ::= "//" {<character>}

<tile-definition> ::= "tile" <identifier> <matrix>

<matrix> ::= "[" {<row>} "]"

<row> ::= "[" {<cell-value>} "]"

<cell-value> ::= "0" | "1"
```

```

<variable-declaration> ::= "let" <identifier> [":" <tile-type>] "=" <expression>

<tile-type> ::= "Tile" <positive-integer> "x" <positive-integer>

<operation> ::= <rotate>
             | <hjoin>
             | <vjoin>
             | <equality>
             | <not-equal>
             | <greater-than>
             | <less-than>
             | <greater-than-or-equal-to>
             | <less-than-or-equal-to>
             | <add>
             | <subtract>
             | <multiply>
             | <divide>
             | <modulus>
             | <and>
             | <or>

<rotate> ::= "rotate(" <expression> "," <angle> ")"

<angle> ::= "90" | "180" | "270"

<hjoin> ::= "hjoin(" <expression> "," <expression> ")"

<vjoin> ::= "vjoin(" <expression> "," <expression> ")"

<equality> ::= <expression> "==" <expression>

<not-equal> ::= <expression> "!=" <expression>

<greater-than> ::= <expression> ">" <expression>

<less-than> ::= <expression> "<" <expression>

<greater-than-or-equal-to> ::= <expression> ">=" <expression>

<less-than-or-equal-to> ::= <expression> "<=" <expression>

<add> ::= <expression> "+" <expression>

<subtract> ::= <expression> "-" <expression>

<multiply> ::= <expression> "*" <expression>

<divide> ::= <expression> "/" <expression>

<modulus> ::= <expression> "%" <expression>

<and> ::= <expression> "&&" <expression>
```

```

<or> ::= <expression> "||" <expression>

<iteration> ::= "for" <identifier> "in" <range> "{" {<statement>} "}"

<range> ::= <positive-integer> ".." <positive-integer>

<print> ::= "print(" <expression> ")"

<if-statement> ::= "if" "(" <expression> ")" "{" {<statement>} "}" ["else" "{" {<statement>} "}"]

<expression> ::= <identifier>
               | <operation>
               | <matrix>
               | <positive-integer>
               | <true>
               | <false>
               | <not>

<identifier> ::= <letter> {<letter> | <digit>}

<letter> ::= "a" | ... | "z" | "A" | ... | "Z"

<digit> ::= "0" | ... | "9"

<positive-integer> ::= <digit> {<digit>}

<true> ::= "true"

<false> ::= "false"

<not> ::= "!" <expression>
```

1.2 Examples

1.2.1 Defining tiles

```
tile T1 [
  [1, 0],
  [0, 1]
]
```

1.2.2 Variables

```
let myTile = T1
```

1.2.3 Types

There are two variations of the types of tiles you can use. You can use the type that was defined above, or you can use a fixed size tile:

```
let myTile : Tile2x2 = T1
```

This represents the size of the tile, so this one is 2 by 2.

1.2.4 Operations (rotation, vertical and horizontal joining)

```
let rotatedTile = rotate(T1, 90)
let combinedTile = hjoin(T1, rotatedTile)
let stackedTile = vjoin(T1, rotatedTile)
```

1.2.5 Iteration

```
for i in 1..4 {
  let newTile = rotate(myTile, i * 90)
  // Do something with newTile
}
```

1.2.6 Example dummy program

```
// Define a 2x2 tile
tile T1 [
  [1, 0],
  [0, 1]
]

// Define another 2x2 tile
tile T2 [
  [0, 1],
  [1, 0]
]

// Declare a variable and store T1 in it
let myTile: Tile2x2 = T1

// Rotate T1 by 90 degrees
let rotatedTile = rotate(T1, 90)

// Join T1 and rotatedTile horizontally
let combinedTile = hjoin(T1, rotatedTile)

// Join T1 and rotatedTile vertically
let stackedTile = vjoin(T1, rotatedTile)

// Iterate over rotations of T2 and join them horizontally
let finalTile = T2
for i in 1..3 {
  let newTile = rotate(T2, i * 90)
  finalTile = hjoin(finalTile, newTile)
}

// Print the final result
print(finalTile)
```

2 Problems Solutions

2.1 Problem 1

```
// Declare A and B tiles
```

```
tile A [ [1] ]
tile B [ [0] ]

// Declare variable to store the checkerboard
let checkerboard: Tile64x64 = []

// Create the 64x64 checkerboard
for i in 1..32 {
    let tempRow: Tile64x1 = []
    for j in 1..32 {
        if (i % 2 == j % 2) {
            tempRow = hjoin(tempRow, A)
        } else {
            tempRow = hjoin(tempRow, B)
        }
    }
    checkerboard = vjoin(checkerboard, tempRow)
}
    // Done
```

2.2 Problem 2

2.2.1 Part 1

```
// Declare the input tile (tile1)
tile tile1 [
    [0, 0, 0, 1],
    [0, 0, 1, 1],
    [0, 1, 1, 1],
    [1, 1, 1, 1]
]

// Rotate tile1 in different directions
let tile1_90: Tile4x4 = rotate(tile1, 90)
let tile1_180: Tile4x4 = rotate(tile1, 180)
let tile1_270: Tile4x4 = rotate(tile1, 270)

// Create the output pattern
let topRow: Tile8x4 = hjoin(tile1, tile1_90)
let bottomRow: Tile8x4 = hjoin(tile1_270, tile1_180)

let output: Tile8x8 = vjoin(topRow, bottomRow)
    // Done
```