Syntax

Sau P

2023-04-17

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> ::= <tile-definitions> <tiling-rules> <functions>
<tile-definitions> ::= <tile-definition> | <tile-definitions> <tile-definition>
<tile-definition> ::= "tile" <tile-name> "{" <cell-rows> "}"
<tile-name> ::= <identifier>
<cell-rows> ::= <cell-row> | <cell-rows> <cell-row>
<cell-row> ::= <cell> | <cell-row> <cell>
<cell> ::= "0" | "1"
<tiling-rules> ::= <tiling-rule> | <tiling-rules> <tiling-rule>
<tiling-rule> ::= <tile-name> "->" <tile-set>
<tile-set> ::= <tile> | <tile-set> <tile>
<tile> ::= <tile-name> | <tile-rotation> <tile>
<tile-rotation> ::= "R" | "L" | "U" | "D"

<functions> ::= <function> | <functions> <function>
<function> ::= "func" <func-name> "(" <parameters> ")" "{" <statements> "}"
<func-name> ::= <identifier>
<parameters> ::= <parameter> | <parameters> "," <parameter>
<parameter> ::= <type> <identifier>
<statements> ::= <statement> | <statements> <statement>
<statement> ::= <variable-declaration> | <function-call> | <loop>
<variable-declaration> ::= <type> <identifier> "=" <expression> ";"
<function-call> ::= <func-name> "(" <arguments> ")" ";"
<arguments> ::= <expression> | <arguments> "," <expression>
<loop> ::= "for" <identifier> "in" <range> "{" <statements> "}"
<range> ::= <expression> ".." <expression>
<expression> ::= <literal> | <variable> | <function-call> | <expression> <operator> <expression>
<operator> ::= "+" | "-" | "*" | "/"
```

```

<type> ::= "int" | "float" | "bool" | "string" | <tile-name>
<literal> ::= <int-literal> | <float-literal> | <bool-literal> | <string-literal>
<int-literal> ::= <digit> | <int-literal> <digit>
<float-literal> ::= <int-literal> "." <int-literal>
<bool-literal> ::= "true" | "false"
<string-literal> ::= '"' <characters> '"'
<characters> ::= <char> | <characters> <char>
<char> ::= Any printable ASCII character excluding double quotes(")

<identifier> ::= <alpha> | <identifier> <alpha> | <identifier> <digit>
<alpha> ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z" | "_"
<digit> ::= "0" | "1" | ... | "9"

<file> ::= <filename> | <filepath>
<filename> ::= <identifier> "." <extension>
<filepath> ::= <directory> | <filepath> <directory>
<directory> ::= <identifier> "/"
<extension> ::= <identifier>

<read-file> ::= "readFile" "(" <file> ")"

```

1.2 Explanation

1.2.1 Program (main)

1. `<program> ::= <tile-definitions> <tiling-rules> <functions>`

   A program consists of tile definitions, tiling rules, and functions.

`<tile-definitions> ::= <tile-definition> | <tile-definitions> <tile-definition>` Tile definitions can be a single tile definition or multiple tile definitions.

<tile-definition> ::= “tile” <tile-name> “{” <cell-rows> “}” A tile definition starts with the keyword tile, followed by a tile name, an opening curly brace, cell rows, and a closing curly brace.

Example: tile A { 010 101 010 }

<tile-name> ::= <identifier> A tile name is an identifier.

<cell-rows> ::= <cell-row> | <cell-rows> <cell-row> Cell rows can be a single cell row or multiple cell rows.

<cell-row> ::= <cell> | <cell-row> <cell> A cell row can be a single cell or multiple cells.

<cell> ::= “0” | “1” A cell can be either “0” or “1”.

<tiling-rules> ::= <tiling-rule> | <tiling-rules> <tiling-rule> Tiling rules can be a single tiling rule or multiple tiling rules.

<tiling-rule> ::= <tile-name> “->” <tile-set> A tiling rule consists of a tile name, followed by the symbol “->”, and then a tile set.

Example: A -> B R C L D U

<tile-set> ::= <tile> | <tile-set> <tile> A tile set can be a single tile or multiple tiles.

<tile> ::= <tile-name> | <tile-rotation> <tile> A tile can be a tile name or a tile rotation followed by a tile.

<tile-rotation> ::= “R” | “L” | “U” | “D” A tile rotation can be “R” (right), “L” (left), “U” (upside-down), or “D” (down).

<functions> ::= <function> | <functions> <function> Functions can be a single function or multiple functions.

<function> ::= “func” <func-name> “(” <parameters> “)” “{” <statements> “}” A function is defined with the keyword func, followed by a function name, parameters enclosed in parentheses, and a block of statements enclosed in curly braces.

Example: func add(int a, int b) { int result = a + b; return result; }

<func-name> ::= <identifier> A function name is an identifier.

<parameters> ::= <parameter> | <parameters> “,” <parameter> Parameters can be a single parameter or multiple parameters separated by commas.

<parameter> ::= <type> <identifier> A parameter consists of a type and an identifier.

<statements> ::= <statement> | <statements> <statement> Statements can be a single statement or multiple statements.

<statement> ::= <variable-declaration> | <function-call> | <loop> A statement can be a variable declaration, a function call, or a loop.

<variable-declaration> ::= <type> <identifier> “=” <expression> “;” A variable declaration consists of a type, an identifier, an equals sign, an expression, and a semicolon.

Example: int a = 5;

<function-call> ::= <func-name> “(” <arguments> “)” “;” A function call consists of a function name, arguments enclosed in parentheses, and a semicolon.

Example: add(3, 4);

<arguments> ::= <expression> | <arguments> “,” <expression> Arguments can be a single expression or multiple expressions separated by commas.

<loop> ::= “for” <identifier> “in” <range> “{” <statements> “}” A loop consists of the keyword for, an identifier, the keyword in, a range, and a block of statements enclosed in curly braces.

Example: for i in 0..10 { print(i); }

<range> ::= <expression> “..” <expression> A range is defined by two expressions separated by two dots.

<expression> ::= <literal> | <variable> | <function-call> | <expression> <operator> <expression> An expression can be a literal, a variable, a function call, or a combination of expressions with an operator.

<operator> ::= “~~“ | ”-“ | ”*“ | ”/“ An operator can be addition (~~), subtraction (-), multiplication (*), or division (/).

<type> ::= “int” | “float” | “bool” | “string” | <tile-name> A type can be an integer (int), a floating-point number (float), a boolean (bool), a string, or a tile name.

<literal> ::= <int-literal> | <float-literal> | <bool-literal> | <string-literal> A literal can be an integer literal, a float literal, a boolean literal, or a string literal.

<int-literal> ::= <digit> | <int-literal> <digit> An integer literal is composed of one or more digits.

Example: 42

<float-literal> ::= <int-literal> “.” <int-literal> A float literal is composed of an integer literal, a decimal point, and another integer literal.

Example: 3.14

<bool-literal> ::= “true” | “false” A boolean literal can be either “true” or “false”.

<string-literal> ::= ”“’ <characters> ”” A string literal is composed of characters enclosed in double quotes.

Example: “hello”

<characters> ::= <char> | <characters> <char> Characters can be a single character or multiple characters.

<char> ::= Any printable ASCII character excluding double quotes(“) A character can be any printable ASCII character, except for double quotes.

<identifier> ::= <alpha> | <identifier> <alpha> | <identifier> <digit> An identifier is composed of letters, underscores, or digits, but it must start with a letter or underscore.

<alpha> ::= “A” | “B” | . . . | “Z” | “a” | “b” | . . . | “z” | “_” Alpha characters can be uppercase letters, lowercase letters, or an underscore.

<digit> ::= “0” | “1” | . . . | “9” A digit can be any number from 0 to 9.

<file> ::= <filename> | <filepath> A file can be a filename or a filepath.

<filename> ::= <identifier> “.” <extension> A filename consists of an identifier, a period, and an extension.

Example: input.txt

<filepath> ::= <directory> | <filepath> <directory> A filepath is composed of one or more directories.

Example: folder1/folder2/input.txt

<directory> ::= <identifier> “/” A directory consists of an identifier followed by a forward slash.

<extension> ::= <identifier> An extension is an identifier.

<read-file> ::= “readFile” “(” <file> “)” Reading a file consists of the keyword readFile, followed by the file enclosed in parentheses.

<built-in-function> ::= “print” | “readFile” <function-call> ::= <func-name> | <built-in-function> “(” <arguments> “)” “;”

1.2.2 Example: readFile(“input.txt”)

```
tile A {
  010
  101
  010
}

tile B {
  111
  000
  111
}

A -> B R
B -> A L

func add(int a, int b) {
  int result = a + b;
  return result;
}

func main() {
  int sum = add(3, 4);
  for i in 0..sum {
    print(i);
  }
  string content = readFile("input.txt");
  print(content);
}
```

1.3 Backus-Naur Form

```
<program> ::= <tile-definitions> <tiling-rules>
<tile-definitions> ::= <tile-definition> | <tile-definitions> <tile-definition>
<tile-definition> ::= "tile" <tile-name> "{" <cell-rows> "}"
<tile-name> ::= <identifier>
<cell-rows> ::= <cell-row> | <cell-rows> <cell-row>
<cell-row> ::= <cell> | <cell-row> <cell>
<cell> ::= "0" | "1"
<tiling-rules> ::= <tiling-rule> | <tiling-rules> <tiling-rule>
<tiling-rule> ::= <tile-name> "->" <tile-set>
<tile-set> ::= <tile> | <tile-set> <tile>
<tile> ::= <tile-name> | <tile-rotation> <tile>
<tile-rotation> ::= "R" | "L" | "U" | "D"
<identifier> ::= <alpha> | <identifier> <alpha> | <identifier> <digit>
<alpha> ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z" | "_"
<digit> ::= "0" | "1" | ... | "9"
```

1.3.1 Explanation

1. A program consists of tile-definitions and tiling-rules.
2. Tile-definitions allows you to compound multiple tiles together.
3. A tile-definition defines a tile with a tile-name and cell-rows.
4. A tile-name is an identifier that uniquely identifies a tile.
5. cell-rows consists of one or more cell-row.
6. A cell-row is a sequence of cell values.
7. A cell is either “0” or “1”, representing an empty or filled cell, respectively.
8. tiling-rules specifies how tiles can be combined to form larger tiles.
9. A tiling-rule maps a tile-name to a tile-set.
10. A tile-set consists of one or more tile.
11. A tile can be a tile-name or a rotated tile.
12. A tile-rotation specifies a rotation of a tile, with “R”, “L”, “U”, and “D” representing right, left, up, and down rotations, respectively.
13. An identifier is a sequence of one or more alphanumeric characters or underscores, starting with an alphabet character.
14. An alpha is an uppercase or lowercase alphabet character or underscore.
15. A digit is a number from 0 to 9.