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

```
1  <program> ::= <tile-definitions> <tiling-rules> <functions>
2  <tile-definitions> ::= <tile-definition> | <tile-definitions> <tile-definition>
3  <tile-definition> ::= "tile" <tile-name> "{" <cell-rows> "}"
4  <tile-name> ::= <identifier>
5  <cell-rows> ::= <cell-row> | <cell-rows> <cell-row>
6  <cell-row> ::= <cell> | <cell-row> <cell>
7  <cell> ::= "0" | "1"
8  <tiling-rules> ::= <tiling-rule> | <tiling-rules> <tiling-rule>
9  <tiling-rule> ::= <tile-name> "->" <tile-set>
10 <tile-set> ::= <tile> | <tile-set> <tile>
11 <tile> ::= <tile-name> | <tile-rotation> <tile>
12 <tile-rotation> ::= "R" | "L" | "U" | "D"
13
14 <functions> ::= <function> | <functions> <function>
15 <function> ::= "func" <func-name> "(" <parameters> ")" "{" <statements> "}"
16 <func-name> ::= <identifier>
17 <parameters> ::= <parameter> | <parameters> "," <parameter>
18 <parameter> ::= <type> <identifier>
19 <statements> ::= <statement> | <statements> <statement>
20 <statement> ::= <variable-declaration> | <function-call> | <loop>
21 <variable-declaration> ::= <type> <identifier> "=" <expression> ";"
22 <function-call> ::= <func-name> "(" <arguments> ")" ";"
23 <arguments> ::= <expression> | <arguments> "," <expression>
24 <loop> ::= "for" <identifier> "in" <range> "{" <statements> "}"
25 <range> ::= <expression> ".." <expression>
26 <expression> ::= <literal> | <variable> | <function-call> | <expression> <operator> <expression>
27 <operator> ::= "+" | "-" | "*" | "/"
```

```
28
29  <type> ::= "int" | "float" | "bool" | "string" | <tile-name>
30  <literal> ::= <int-literal> | <float-literal> | <bool-literal> | <string-literal>
31  <int-literal> ::= <digit> | <int-literal> <digit>
32  <float-literal> ::= <int-literal> "." <int-literal>
33  <bool-literal> ::= "true" | "false"
34  <string-literal> ::= '"' <characters> '"'
35  <characters> ::= <char> | <characters> <char>
36  <char> ::= Any printable ASCII character excluding double quotes(")
37
38  <identifier> ::= <alpha> | <identifier> <alpha> | <identifier> <digit>
39  <alpha> ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z" | "_"
40  <digit> ::= "0" | "1" | ... | "9"
41
42  <file> ::= <filename> | <filepath>
43  <filename> ::= <identifier> "." <extension>
44  <filepath> ::= <directory> | <filepath> <directory>
45  <directory> ::= <identifier> "/"
46  <extension> ::= <identifier>
47
48  <read-file> ::= "readFile" "(" <file> ")"
49  <built-in-function> ::= "print" | "readFile"
50  <function-call> ::= <func-name> | <built-in-function> "(" <arguments> ")" ";"
51
```

1.2   Explanation

1.2.1   Program (main)

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

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

2. `<tile-definitions> ::= <tile-definition> | <tile-definitions> <tile-definition>`

   (a) Tile definitions can be a single tile definition or multiple tile definitions.

3. `<tile-definition> ::= "tile" <tile-name> "{" <cell-rows> "}"`

   (a) A tile definition starts with the keyword tile, followed by a tile name, an opening curly brace, cell rows, and a closing curly brace.

   (b) Example: tile A { 010 101 010 }

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

5. `<cell-rows> ::= <cell-row> | <cell-rows> <cell-row>`

   (a) Cell rows can be a single cell row or multiple cell rows.

6. `<cell-row> ::= <cell> | <cell-row> <cell>`

   (a) A cell row can be a single cell or multiple cells.

7. `<cell> ::= "0" | "1"`

   (a) A cell can be either "0" or "1".

8. `<tiling-rules> ::= <tiling-rule> | <tiling-rules> <tiling-rule>`

    (a) Tiling rules can be a single tiling rule or multiple tiling rules.

9. `<tiling-rule> ::= <tile-name> "->" <tile-set>`

    (a) A tiling rule consists of a tile name, followed by the symbol "->", and then a tile set.

    (b) Example: A -> B R C L D U

10. `<tile-set> ::= <tile> | <tile-set> <tile>`

    (a) A tile set can be a single tile or multiple tiles.

11. `<tile> ::= <tile-name> | <tile-rotation> <tile>`

    (a) A tile can be a tile name or a tile rotation followed by a tile.

12. `<tile-rotation> ::= "R" | "L" | "U" | "D"`

    (a) A tile rotation can be "R" (right), "L" (left), "U" (upside-down), or "D" (down).

13. `<functions> ::= <function> | <functions> <function>`

    (a) Functions can be a single function or multiple functions.

14. `<function> ::= "func" <func-name> "(" <parameters> ")" "{" <statements> "}"`

    (a) 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.

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

15. `<func-name> ::= <identifier>`

    (a) A function name is an identifier.

16. `<parameters> ::= <parameter> | <parameters> "," <parameter>`

    (a) Parameters can be a single parameter or multiple parameters separated by commas.

17. `<parameter> ::= <type> <identifier>`

    (a) A parameter consists of a type and an identifier.

18. `<statements> ::= <statement> | <statements> <statement>`

    (a) Statements can be a single statement or multiple statements.

19. `<statement> ::= <variable-declaration> | <function-call> | <loop>`

    (a) A statement can be a variable declaration, a function call, or a loop.

20. `<variable-declaration> ::= <type> <identifier> "=" <expression> ";"`

    (a) A variable declaration consists of a type, an identifier, an equals sign, an expression, and a semicolon.

    (b) Example: int a = 5;

21. `<function-call> ::= <func-name> "(" <arguments> ")" ";"`

22. A function call consists of a function name, arguments enclosed in parentheses, and a semicolon.

    (a) Example: add(3, 4);

23. `<arguments> ::= <expression> | <arguments> "," <expression>`

24. Arguments can be a single expression or multiple expressions separated by commas.

25. `<loop> ::= "for" <identifier> "in" <range> "{" <statements> "}"`

    (a) A loop consists of the keyword for, an identifier, the keyword in, a range, and a block of statements enclosed in curly braces.

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

26. `<range> ::= <expression> ".." <expression>`

    (a) A range is defined by two expressions separated by two dots.

27. `<expression> ::= <literal> | <variable> | <function-call> | <expression> <operator> <expression>`

    (a) An expression can be a literal, a variable, a function call, or a combination of expressions with an operator.

28. `<operator> ::= "+" | "-" | "*" | "/"`

    (a) An operator can be addition (+), subtraction (-), multiplication (*), or division (/).

29. `<type> ::= "int" | "float" | "bool" | "string" | <tile-name>`

    (a) A type can be an integer (int), a floating-point number (float), a boolean (bool), a string, or a tile name.

30. `<literal> ::= <int-literal> | <float-literal> | <bool-literal> | <string-literal>`

    (a) A literal can be an integer literal, a float literal, a boolean literal, or a string literal.

31. `<int-literal> ::= <digit> | <int-literal> <digit>`

    (a) An integer literal is composed of one or more digits.

    (b) Example: 42

32. `<float-literal> ::= <int-literal> "." <int-literal>`

    (a) A float literal is composed of an integer literal, a decimal point, and another integer literal.

    (b) Example: 3.14

33. `<bool-literal> ::= "true" | "false"`

    (a) A boolean literal can be either "true" or "false".

34. `<string-literal> ::= '"' <characters> '"'`

    (a) A string literal is composed of characters enclosed in double quotes.

    (b) Example: "hello"

35. `<characters> ::= <char> | <characters> <char>`

    (a) Characters can be a single character or multiple characters.

36. `<char> ::= Any printable ASCII character excluding double quotes(")`

    (a) A character can be any printable ASCII character, except for double quotes.

37. `<identifier> ::= <alpha> | <identifier> <alpha> | <identifier> <digit>`

    (a) An identifier is composed of letters, underscores, or digits, but it must start with a letter or underscore.

38. `<alpha> ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z" | "_"`

    (a) Alpha characters can be uppercase letters, lowercase letters, or an underscore.

39. `<digit> ::= "0" | "1" | ... | "9"`

    (a) A digit can be any number from 0 to 9.

40. `<file> ::= <filename> | <filepath>`

    (a) A file can be a filename or a filepath.

41. `<filename> ::= <identifier> "." <extension>`

    (a) A filename consists of an identifier, a period, and an extension.

    (b) Example: input.txt

42. `<filepath> ::= <directory> | <filepath> <directory>`

    (a) A filepath is composed of one or more directories.

    (b) Example: folder1/folder2/input.txt

43. `<directory> ::= <identifier> "/"`

    (a) A directory consists of an identifier followed by a forward slash.

44. `<extension> ::= <identifier>`

    (a) An extension is an identifier.

45. `<read-file> ::= "readFile" "(" <file> ")"`

    (a) Reading a file consists of the keyword readFile, followed by the file enclosed in parentheses.

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 Old Backus-Naur Form

```
 1   <program> ::= <tile-definitions> <tiling-rules>
 2   <tile-definitions> ::= <tile-definition> | <tile-definitions> <tile-definition>
 3   <tile-definition> ::= "tile" <tile-name> "{" <cell-rows> "}"
 4   <tile-name> ::= <identifier>
 5   <cell-rows> ::= <cell-row> | <cell-rows> <cell-row>
 6   <cell-row> ::= <cell> | <cell-row> <cell>
 7   <cell> ::= "0" | "1"
 8   <tiling-rules> ::= <tiling-rule> | <tiling-rules> <tiling-rule>
 9   <tiling-rule> ::= <tile-name> "->" <tile-set>
10   <tile-set> ::= <tile> | <tile-set> <tile>
11   <tile> ::= <tile-name> | <tile-rotation> <tile>
12   <tile-rotation> ::= "R" | "L" | "U" | "D"
13   <identifier> ::= <alpha> | <identifier> <alpha> | <identifier> <digit>
14   <alpha> ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z" | "_"
15   <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.